# RoBox: CCG with Structured Perceptron for Supervised Semantic Parsing of Robotic Spatial Commands

**Kilian Evang**
University of Groningen
k.evang@rug.nl

**Johan Bos**
University of Groningen
johan.bos@rug.nl

## Abstract

We use a Combinatory Categorial Grammar (CCG) parser with a structured perceptron learner to address Shared Task 6 of SemEval-2014, Supervised Semantic Parsing of Robotic Spatial Commands. Our system reaches an accuracy of 79% ignoring spatial context and 87% using the spatial planner, showing that CCG can successfully be applied to the task.

## 1 Introduction

When interpreting utterances, humans use world knowledge whereas most semantic parsers to date rely purely on linguistic clues. Shared Task 6 in the SemEval 2014 campaign for semantic evaluation aims to integrate reasoning about microworlds with semantic parsing. In this task, a system is given an instruction for a robot and has to produce an executable semantic representation in Robot Control Language (Dukes, 2013a, RCL). The Robot Commands Treebank (Dukes, 2013b) is used for training and evaluation. We participated in this shared task with a system rooted in Combinatory Categorial Grammar (CCG). In particular, we were interested in finding out whether existing techniques for automatically deriving categorial grammars with semantics could be moved easily to the new domain of robot commands and integrated with the provided spatial reasoning component. In this paper we outline our method and present the results for this shared task. [1]

## 2 Extracting a CCG from RCL

CCGs (Steedman, 2001) use a small set of atomic constituent categories such as $S$ (sentence), $NP$ (noun phrase) or $PP$ (prepositional phrase). Constituents that take other constituents as arguments have complex categories describing their combinatory potential. For example, an intransitive English verb has category $S\backslash NP$, meaning that it forms a sentence by combining with an NP to its left. Similarly, *modifiers* also have complex categories. For example, a pre-sentential adverb might have category $S/S$ because it combines with a sentence to its right to form a modified sentence. The *combinatory rules* that license these example combinations are called *backward application* and *forward application*. They and other combinatory rules also allow for constituents to be associated with semantic expressions, and specify how to form a combined semantic expression for the derived larger constituent.

In this section, we describe a process that takes an RCL corpus as input and produces a set of CCG lexical entries, i.e. natural-language words paired with categories and semantic expressions. The goal is for these lexical entries to produce the correct semantics under CCG combinatory rules also for unseen robotic commands.

### 2.1 Transforming the Trees

RCL expressions are rooted ordered trees whose nodes are labeled with *tags*. We will write them in the form $(t{:}\mathbf{h})$ where $t$ is the root tag and $\mathbf{h}$ is the sequence of subtrees of the root's children. Leaves are abbreviated as just their tags. In each training example, each pre-terminal (parent of a leaf) can be aligned to one or more words in the corresponding natural language expression. An example is shown in Figure 1. Since the alignments to words are not crossing, we can interpret the RCL tree as a phrase structure tree for the sentence and use the algorithm of (Hockenmaier and Steedman, 2007) to translate it to CCG. We extend the algorithm with a semantic step that makes sure the derivations would produce the original RCL expressions.



[1] Our code is available at `http://www.let.rug.nl/evang/RoBox.zip`



*Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 482–486,
Dublin, Ireland, August 23-24, 2014.

$(event{:}(action{:}move)(entity{:}(color{:}green)(type{:}prism))(destination{:}(spatial\text{-}relation{:}(relation{:}within)(entity{:}(indicator{:}back)(indicator{:}left)(type{:}corner))))$

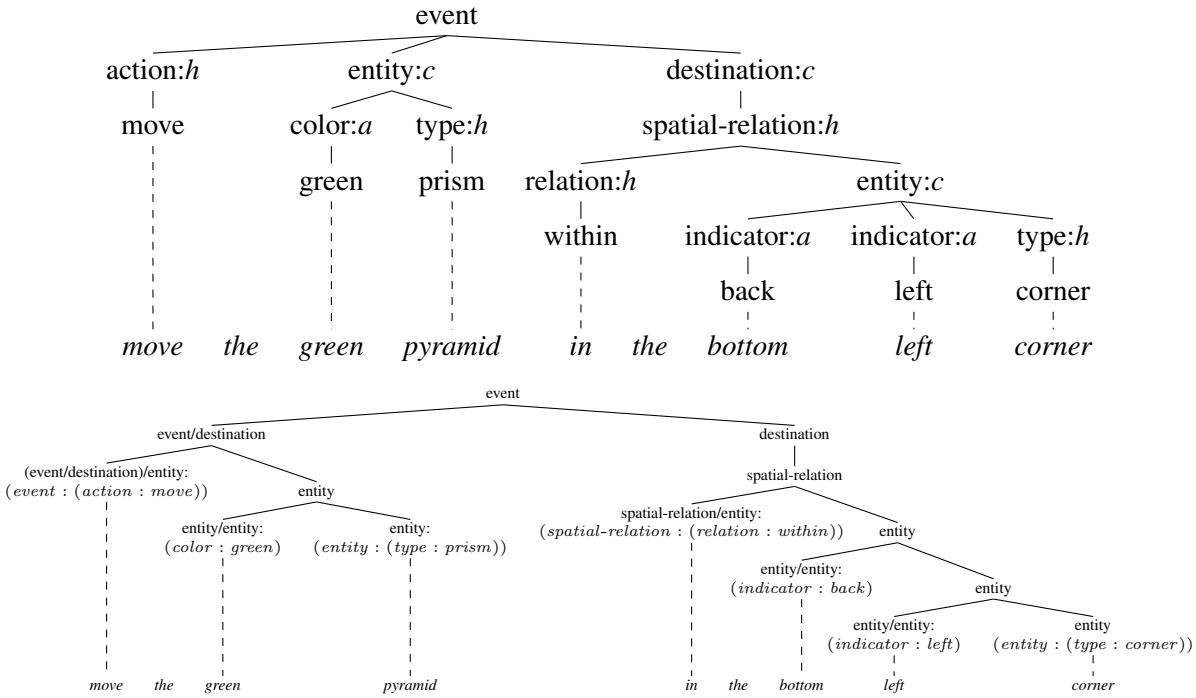


Figure 1: Top: an RCL expression. Middle: its representation as a tree diagram. Internal nodes are annotated with constituent types. Pre-terminals are aligned to words in a corresponding natural-language expression. Bottom: result of the CCG transformation.

The procedure is as follows:

**1. Determine constituent types.** We treat $action$, $relation$ and $type$ constituents as *heads*, $entity$s and $destination$s as *complements* (i.e. arguments) and $cardinal$s, $color$s, $indicator$s, $measure$s and $spatial\text{-}relation$s as *adjuncts* (i.e. modifiers). For $sequence$ nodes that have multiple $event$ children, we treat the first as head and the rest as adjuncts. A corresponding *constituent type label* $h$, $a$ or $c$ is added to the label of each internal node (cf. Figure 1, middle).

**2. Assign lexical semantics.** To the label of each pre-terminal, add an RCL expression which is a copy of a connected subgraph of the tree itself (without the constituent type labels). For *a*-type and *c*-type pre-terminals, the subgraph includes only the pre-terminal and its daughter. For *h*-type pre-terminals the parent is also included, as well as any subtrees with root tag $id$ or $reference\text{-}id$ the parent may have.To illustrate, the label of the action:*h* node in our example becomes action:*h*:$(event : (action : move))$, and color:*a* becomes color:*a*:$(color{:}green)$. The leaves are now no longer needed, so we remove them.

**3. Add sequence nodes.** If the root is tagged $sequence$, add an additional node tagged $sequence$ between each child and the root.

**4. Binarize the tree.** Each local tree with more than two daughters is binarized by inserting dummy nodes, provisionally labeled $C : h$ where $C$ is the tag of the parent. Left adjuncts (such as the first $indicator$ in Figure 1) are split off first, followed by right adjuncts (such as the $destination$ in Figure 1), left complements and right complements.

**5. Assign CCG categories.** Starting from the root, the tag of each node is replaced by a CCG category. For simplicity, we directly use RCL tags as atomic categories rather than mapping them to standard CCG categories:

The root gets its tag ($event$ or $sequence$) as category.

$c$-type nodes get their tag as category. Their sibling gets category $P/T$ if it is on the left and $P\backslash T$ if it is on the right, where $T$ is the tag of the $c$-type node and $P$ is the category of the parent. For example, the $destination$ node in Figure 1 gets $destination$ as category, and its left sibling there-

fore gets $event/destination$ because the parent's category is $event$.

$a$-type nodes such as the two $indicator$s in Figure 1 get category $P/P$ if they are on the left of their sibling and $P\backslash P$ if they are on its right, where $P$ is the category of their parent. The sibling gets category $P$.

Nodes without siblings get their tag as category.

Constituent type labels are dropped. The result for our example is shown at the bottom of Figure 1.

### 2.2 The Lexicon

For each leaf in the transformed corpus that is aligned to one or more words, a lexical item is extracted containing the words, category and RCL. For single-word items, we also add part-of-speech tags, obtained using the C&C POS tagger (Curran and Clark, 2003), to reduce overgeneration. Examples of lexical items are:

- $\langle$block/NN$\rangle$ $\vdash$ entity : $(entity : (type : (block))\rangle$
- $\langle$on, top, of$\rangle$ $\vdash$ spatial-relation/entity : $(spatial\text{-}relation : (relation : above))$

### 2.3 Combinatory Rules

Given the extracted lexical items, the corpus derivations are licensed by standard CCG rules (Steedman, 2001), using a modified semantics that keeps things simple and ensures that the semantics of (most) intermediate constituents are themselves RCL subexpressions, which is important for interfacing with the spatial planner during parsing. The most important two rules are forward and backward application:

$$(X/Y){:}f \quad Y{:}g \Rightarrow X{:}\textsc{fapp}(X/Y, f, g) \quad (>)$$
$$Y{:}g \quad (X\backslash Y){:}f \Rightarrow X{:}\textsc{bapp}(X\backslash Y, g, f) \quad (<)$$

where FAPP and BAPP are defined as follows:

$$\textsc{fapp}(X/Y, \mathbf{a}, (t{:}\mathbf{h})) = (t{:}\mathbf{ah}) \text{ if } X = Y$$
$$\textsc{fapp}(C, (t{:}\mathbf{h}), \mathbf{c}) = (t{:}\mathbf{hc}) \text{ otherwise}$$
$$\textsc{bapp}(X\backslash Y, (t{:}\mathbf{h}), \mathbf{a}) = (t{:}\mathbf{ha}) \text{ if } X = Y$$
$$\textsc{bapp}(C, \mathbf{c}, (t{:}\mathbf{h})) = (t{:}\mathbf{ch}) \text{ otherwise}$$

In words, the semantics of the adjunct or complement is added as a subtree under the root of the semantics of the head.

We also use a restricted form of the CCG rule *forward composition* to form chains of entity adjuncts:

$$(\text{entity}/\text{entity}){:}\mathbf{a} \quad (\text{entity}/\text{entity}){:}\mathbf{b} \Rightarrow (\text{entity}/\text{entity}){:}\mathbf{ab} \quad (>_{\mathsf{B}})$$

This is motivated by our use of the spatial planner. Without forward composition, we would, e.g., not be able to build a constituent with the semantics $(entity : (color : green)(color : red)(type : cube\text{-}group))$ in the context of a stack consisting of green and red cubes, but no stack consisting exclusively of red cubes – the planner would filter out the intermediate constituent with semantics $(entity{:}(color{:}red)(type{:}cube\text{-}group))$.

Finally, we use type-changing rules, which is standard practice in CCG parsing (Clark and Curran, 2007; Zettlemoyer and Collins, 2007). They are automatically extracted from the training data. Some of them account for unary productions within RCL expressions by introducing an additional internal node, such as the $destination$ node in Figure 1. For example:

$$\text{sp-relation}{:}\mathbf{h} \Rightarrow \text{destination}{:}(destination{:}\mathbf{h}) \quad (*_1)$$

Others account for RCL leaves that are not linked to any words. For example, the RCL expression for the command *take the light blue prism from the blue cube* renders the *from*-phrase as an adjunct to the $prism$ node: $(spatial\text{-}relation{:}(relation{:}above)(entity{:}(color{:}blue)(type{:}cube)))$, where $above$ is not linked. Rules like the following deal with this by not only introducing an internal node, but also a branch leading to the unlinked leaf:

$$\text{entity}{:}\mathbf{h} \Rightarrow \text{entity}/\text{entity}{:} (sp\text{-}relation{:}(relation{:}above)\mathbf{h}) \quad (*_2)$$

### 2.4 Anaphora

Anaphora are marked in RCL $entity$ expressions by the subexpression $(id : 1)$ for antecedent entities and $(reference\text{-}id : 1)$ for anaphoric entities. The latter have the special type $reference$, in which case they are typically linked to the word *it*, or $type\text{-}reference$, in which case they are typically linked to the word *one*, as in *the yellow one*. More than one anaphoric relation in a command, and thus, other IDs than 1, are possible, but extremely rare. We do not explicitly try to resolve

anaphora, but merely generate versions both with and without the $id$ subexpression for each $entity$ lexical item seen in training as an antecedent. We then rely on the parser and spatial planner to find a parse with the correct item marked as antecedent. If the spatial planner rejects a subexpression because it contains an unknown reference ID, we accept it anyway because the expression can later combine with another one that contains the antecedent. However, at the level of complete parses, those containing a $reference\text{-}id$ expression but no $id$ expression – or vice versa – are rejected. As a heuristic, we also reject parses where $reference\text{-}id$ precedes $id$ because we found this to be a noticeable source of errors, and no cataphora in the training data.

## 3 Training and Decoding

Following (Zettlemoyer and Collins, 2007), we use a CKY CCG parser in combination with simple perceptron updates: iterate over the training corpus $T$ times, for each sentence producing all parses. Each parse is characterized by a number of features and scored using a global weight vector. The weight vector is updated by subtracting the feature vector of the highest-scoring parse and adding the feature vector of the highest-scoring correct parse. No update is performed if the highest-scoring parse is correct, or no correct parse was found. Since for the present task the training data already induces a lexicon, we treat the lexicon as fixed and perform no lexical update. We parallelize training using iterative parameter mixing (McDonald et al., 2010) with 12 shards.

### 3.1 Semantically Empty and Unknown Words

The parser initially considers each contiguous subsequence of words in the sentence and adds all matching lexical items to the chart. In order to allow for words that are not linked to the semantics, we simply add two additional lexical items to the chart for each word $w$ in the sentence: $\langle w \rangle \vdash X/X : nil$ and $\langle w \rangle \vdash X\backslash X : nil$ where $X$ is a variable that can be bound to any category during rule application. We modify the combinatory rules above to require that at least one of the input items has non-$nil$ semantics and to use that as output semantics if the other is $nil$.

In decoding, the parser also has to deal with words not seen in training. For one, there are the *nil* items, so it is possible to treat the unknown words as semantically empty. In addition, we look at other single-word lexical items with the same POS tag and generate corresponding lexical items for the unknown word on the fly, hoping that features and the spatial planner will guide the parser to the right choice. To limit the search space, this is currently only done for nouns since we found the greatest lexical variance to occur with them.

### 3.2 Features

Each chart edge is characterized by the following local features:

- each lexical item $\mathbf{w} \vdash c{:}s$ used.
- each instance of a combinatory rule used, e.g. $>$.
- $\langle p, c, s \rangle$ for each lexical item used where $p$ is the POS tag (or empty for multiwords). This allows to learn correlations between category/semantics pairs and particular parts of speech, primarily for unknown words.
- each instance of a type-changing rule used, together with the semantic head word of the constituent it roots, e.g. $\langle *_1, \text{in} \rangle$. This helps to learn not to use type-changing rules where they don't make sense. E.g. the word *squares* often heads entity descriptions that type-change into $measure$ phrases but the word *cube* doesn't.
- the root tag of the semantics of each constituent, together with the word to its immediate left, e.g. $\langle destination, \text{from} \rangle$. This example feature is indicative of typical erroneous parses where spatial adjuncts corresponding to *from*-phrases are misparsed as destination complements. The word *from* provides a strong clue against such a parse but would be ignored without such a feature because it is not aligned to any RCL node.
- the root tag of the semantics of each constituent, together with the first word in it, e.g. $\langle spatial\text{-}relation, \text{above} \rangle$.

### 3.3 The Spatial Planner

The spatial planner provided together with the treebank provides access to the context in which each command is to be interpreted, consisting of a current arrangement of bodies on a board and in

the gripper of the robot being instructed. It can tell us for some RCL subexpressions, chiefly *entity* descriptions, whether they "make sense" given the context. For example, if the parser builds an edge with semantics $(entity:(type:cube)(color:red))$ but there is no red cube anywhere on the board, we can immediately reject the edge (provided no negations or hypothetical descriptions are used, which is the case for the commands in this task) and thereby avoid errors and reduce the search space. The planner also helps resolve attachment ambiguities early: in the command *put the prism on the cube*, a constituent with semantics $(entity:(type:prism)(spatial\text{-}relation:(relation:above)(entity:(type:cube))))$ is a possible but incorrect parse. If we are lucky enough that no prism is actually sitting on a cube in the microworld, the planner will weed it out.

We have not yet explored making the fullest possible use of the spatial planner for checking the validity of *event* or *sequence* expressions, which would involve simulating changing the state of the world as a sequence of *event* instructions is carried out. Currently we only filter out initial *event* instructions with action *drop* for scenes in which there is nothing initially in the robot's gripper to be dropped. RCL requires the action *move* here instead, a distinction which is often not made in the natural language commands.

## 4 Experiments and Results

We carried out two experiments, one using the spatial planner and one not using it. In each case, we trained on training examples shorter than 16 words to speed up training and evaluated on the full test set. In both training and decoding, a beam search strategy keeps only the 60 highest-scoring edges per chart cell. The weights of non-*nil* lexical items were initialized to 1, those of *nil* items to 0.5, all other feature weights to 0. The number of training epochs $T$ was set to 3. These values were chosen experimentally using 80% of the training data and another 10% for testing.

Of the 909 test sentences, 720 (79.21%) were parsed exactly correctly when not using the planner, and 789 (86.80%) when using it, making third place among the six participating systems. The result shows that standard CCG-based techniques for semantic parsing can be successfully applied to the domain of robotic spatial commands and profit from the integration of a spatial planner.

A preliminary analysis suggests most errors are related to pronoun ellipsis, the ambiguous word *one*, anaphora or attachment ambiguity. We believe some further careful feature engineering and extended use of the spatial planner could go a great length to improve accuracy further.

## References


Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

James R. Curran and Stephen Clark. 2003. Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the 11th Meeting of the European Chapter of the Association for Computational Linguistics (EACL-03)*, pages 91–98, Budapest, Hungary.

Kais Dukes. 2013a. Semantic annotation of robotic spatial commands. In *Language and Technology Conference (LTC)*, Poznan, Poland.

Kais Dukes. 2013b. Train robots: A dataset for natural language human-robot spatial interaction through verbal commands. In *International Conference on Social Robotics (ICSR). Embodied Communication of Goals and Intentions Workshop*, Bristol, United Kingdom.

J. Hockenmaier and M. Steedman. 2007. CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Ryan McDonald, Keith Hall, and Gideon Mann. 2010. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 456–464, Stroudsburg, PA, USA.

Mark Steedman. 2001. *The Syntactic Process*. The MIT Press.

Luke S. Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL-2007)*, pages 678–687.