ShrdLite: Semantic Parsing Using a Handmade Grammar

Peter Ljunglöf
Department of Computer Science and Engineering
University of Gothenburg and Chalmers University of Technology
Gothenburg, Sweden
peter.ljunglof@cse.gu.se

Abstract

This paper describes my approach for parsing robot commands, which was task 6 at SemEval 2014. My solution is to manually create a compact unification grammar. The grammar is highly ambiguous, and relies heavily on filtering the parse results by checking their consistency with the current world.

The grammar is small, consisting of not more than 25 grammatical and 60 lexical rules. The parser uses simple error correction together with a straightforward iterative deepening search. Nevertheless, with these very basic algorithms, the system still managed to get 86.1% correctness on the evaluation data. Even more interesting is that by making the parser slightly more robust, the accuracy of the system rises to 93.5%, and by adding one single word to the lexicon, the accuracy is boosted to 98.0%.

1 Introduction

SemEval 2014, task 6, was about parsing commands to a robot operating in a blocks world. The goal is to parse utterances in natural language into commands in a formal language, the Robot Control Language (RCL). As a guide the system can use a spatial planner which can tell whether an RCL command is meaningful in a given blocks world.

The utterances are taken from the Robot Commands Treebank (Dukes, 2013), which pairs 3409 sentences with semantic annotations consisting of an RCL command together with a description of a world where the command is meaningful. The corpus was divided into 2500 training sentences and 909 evaluation sentences.

The system that is described in this paper, together with the evaluation data, is available online from GitHub:

https://github.com/heatherleaf/semeval-2014-task6

2 System Description

The Shrdlite system is based on a small unification grammar, together with a naive robust parser implemented using iterative deepening. After parsing, the resulting parse trees are modified according to six extra-grammatical semantic rules.

The grammar and the semantic rules were handcrafted using manual analysis of the available 2500 training sentences, and an incremental and iterative process to select and fine-tune the grammar. The total amount of work for creating the grammar consisted of about 3–4 days for one person. This excludes programming the robust parser and the rest of the system, which took another 2–3 days.

I did not have any access to the 909 evaluation sentences while developing the grammar or the other parts of the system.

2.1 Grammar

The grammar is a Prolog DCG unification grammar (Pereira and Warren, 1980) which builds a semantic parse tree during parsing. The core grammar is shown in figure 1. For presentation purposes, the DCG arguments (including the semantic parse trees) are left out of the figure. The lexicon consists of ca 150 surface words (or multi-word units) divided into 23 lexical categories. The lexical categories are the lowercase italic symbols in the core grammar.

Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014), pages 556–559,
Dublin, Ireland, August 23-24, 2014.

```
Main          ⟶  Event ?periods
Event         ⟶  take-verb Entity take-suffix
                |  drop-verb Entity drop-suffix
                |  move-verb Entity ?commas move-suffix Destination
                |  take-verb Entity take-suffix and-then move-verb RefEntity move-suffix Destination
RefEntity     ⟶  ?reference-pronoun
Destination   ⟶  SpatialRelation
SpatialRelation ⟶ (Measure | entity-relation | Measure entity-relation) Entity
RelativeClause ⟶ ?commas relative-pronoun SpatialRelation
Measure       ⟶  Entity
Entity        ⟶  ?determiner BasicEntity
BasicEntity   ⟶  (cardinal | indicator | color | cube-group-indicator) BasicEntity
                |  (type | one) ?RelativeClause
                |  ?(its | the) region-indicator ?of-board
```

Figure 1: The core grammar. Lowercase italic symbols are lexicon entries, and a question mark indicates that the following symbol is optional. DCG arguments (semantic trees and syntactic coordination) are left out for presentation reasons.

2.2 Semantic Modifications After Parsing

After parsing, each resulting semantic parse tree is modified according to the following rules. Most of these rules should be possible to implement as grammar rules, but I felt that this would make the grammar unnecessarily complicated.

- If a color is specified before an indicator, change the order between them.
- If an entity of type CUBE is described using two colours, its type is changed to CUBE-GROUP.
- Relation words such as *"above", "opposite", "over"*, etc., correspond to the relation WITHIN() if the entity is of type CORNER or REGION; for all other entities the relation will be ABOVE().
- The relation FRONT() is changed to FORWARD() if the entity is of type TILE.
- Add a reference id to the subject of a TAKE-AND-DROP command sequence, unless it already has a reference id.
- If the destination of a move command is of type TYPE-REFERENCE or TYPE-REFERENCE-GROUP, add a reference id to the subject; unless the subject is of type PRISM and it has a spatial relation, in which case the reference id is added to its spatial relation instead.

2.3 Robust Parsing

The parser is a standard Prolog recursive-descent parser, augmented with simple support for handling robustness. The algorithm is shown in figure 2.

2.3.1 Misspellings and Junk Words

The parser tries to compensate for misspellings and junk words. Any word can be recognized as a misspelled word, penalized using the Levenshtein edit distance (Levenshtein, 1966), or it can be skipped as a junk word with a fixed penalty.[1] The parser first tries to find an exact match of the sentence in the grammar, then it gradually allows higher penalties until the sentence is parsed. This is done using iterative deepening on the edit penalty of the sentence, until it reaches the maximum edit penalty – if the sentence still cannot be parsed, it fails. In the original evaluation I used a maximum edit penalty of 5, but by just increasing this penalty, the accuracy was boosted considerably as discussed in sections 2.4 and 3.1.

2.3.2 Filtering Through the Spatial Planner

The parser uses the spatial planner that was distributed together with the task as a black box. It takes a semantic parse tree and the current world configuration, and decides if the tree is meaningful in the given world.

When the sentence is recognized, all its parse trees are filtered through the spatial planner. If

[1] The penalty of skipping over a word is 3 if the word is already in the lexicon, and 2 otherwise.

```
function robust-parse(sentence, world):
    for penalty in 0...5:
        trees = { t' | t ∈ parse-dcg(sentence, edit-penalty=penalty),
                       t' = modify-tree(t),
                       spatial-planner(t', world) = MEANINGFUL }
        if trees ≠ ∅:
            return min(trees, key=treesize)
    return FAILURE
```

Figure 2: The robust parsing algorithm.

none of the trees are meaningful in the world, the parser tries to parse the sentence with a higher edit penalty.

2.3.3 Selecting the Best Tree

If there is more than one possible semantic tree, the system returns the tree with the smallest number of nodes.

2.4 Minor Modifications After Evaluation

As explained in section 3.1, the error analysis of the final evaluation revealed one construction and one lexical item that did not occur in the training corpus:

- Utterances can start with 2–3 periods. The reason why this was not caught by the robust parser is that each of these periods are considered a word of its own, and as mentioned in section 2.3.1, the penalty for skipping a lexicon word is 3 which means that the penalty for parsing a sentence with 2–3 initial periods is 6 or 9. Unfortunately I had chosen a maximum penalty of 5 which meant that the original evaluation missed all these sentences.

  By just increasing the maximum penalty from 5 to 9, the accuracy increased from 86.1% to 93.5%.

- The word *"cell"* occurs in the evaluation data as a synonym for the entity type TILE, in addition to the existing tile words *"square", "grid", "space"*, etc. Unfortunately, the parser tries to correct *"cell"* into the Levenshtein-similar *"cube"*, giving the wrong semantics.

  By adding *"cell"* to the lexicon, the accuracy increased further from 93.5% to 98.0%.

The results of these minimal modifications are substantial, and are discussed further in section 3.2.

3 Evaluation

The system was evaluated on 909 sentences from the treebank, and I only tested for exact matches. The result of the initial evaluation was that 86% of the sentences returned a correct result, when using the spatial planner as a guide for selecting parses. Without the planner, the accuracy was only 51%. The results are shown in the top rows in tables 1 and 2.

The grammar is ambiguous and the system relies heavily on the spatial planner to filter out candidates. Without the planner, 42% of the utterances are ambiguous returning between 2 and 18 trees, but with the planner, only 4 utterances are ambiguous (i.e., 0.4%).

3.1 Error Analysis

As already mentioned in section 2.4, almost all of the errors that the system makes are of two forms that are very easy to correct:

- None of the training sentences start with a sequence of periods, but 58 of the evaluation sentences do. This was solved by increasing the maximum edit penalty to 9.

- The word *"cell"* does not occur in the training sentences, but in does appear in 45 of the evaluation sentences. To solve this error I just added that word to the lexicon.

3.2 Evaluation Results

As already mentioned, the accuracy of the initial grammar was 86.1% with the spatial planner. The two minor modifications described in section 2.4 improve the results significantly, as can be seen in table 1. Increasing the maximum edit penalty solves 67 of the 126 failing sentences, and adding the word *"cell"* solves 41 of the remaining sentences. These two improvements together solve 108 sentences, leaving only 18 failing sentences.

	Max. penalty	Correct			Incorrect			
		Unique	Ambiguous	Total	Ambiguous	Miss	Fail	Total
Original grammar	5	782	1	86.1%	0	19	107	13.9%
Original grammar	9	845	5	93.5%	0	50	9	6.5%
Adding *"cell"*	9	886	5	98.0%	0	10	8	2.0%

Table 1: Evaluation results *with* the spatial planner.

	Max. penalty	Correct			Incorrect			
		Unique	Ambiguous	Total	Ambiguous	Miss	Fail	Total
Original grammar	5	450	18	51.5%	315	64	62	48.5%
Original grammar	9	493	20	56.4%	330	65	1	43.6%
Adding *"cell"*	9	498	24	57.4%	366	20	1	42.6%

Table 2: Evaluation results *without* the spatial planner.

The final accuracy was therefore boosted to an impressive 98.0%.

The columns in the result tables are as follows: *Unique* are the number of sentences for which the system returns one single tree which is correct. *Ambiguous* are the number of sentences where the parser returns several trees, and the correct tree is among them: if the tree that the system selects (i.e., the smallest tree) is correct, it is counted as a correct ambiguous sentence, otherwise it is counted as incorrect. *Miss* are the number of sentences where all the returned trees are incorrect, and *Fail* are the sentences for which the system could not find a tree at all.

Table 2 shows that the modifications also improve the accuracy when the spatial planner is not used, but the improvement is not as impressive. The reason for this is that many of the failed sentences become ambiguous, and since the planner cannot be used for disambiguation, there is still a risk that the returned tree is not the correct one. The number of sentences for which the system returns the correct tree somewhere among the results is the sum of all unique and ambiguous sentences, which amounts to $450+18+315=783$ (i.e., 86.1%) for the original grammar and $498+24+366=888$ (i.e., 97.7%) for the updated grammar. Note that these are almost the same results as in table 1, which is consistent with the fact that the system uses the planner to filter out incorrect interpretations.

4 Discussion

In this paper I have showed that a traditional symbol-based grammatical approach can be as good as, or even superior to, a data-based machine learning approach, in specific domains where the language and the possible actions are restricted. The grammar-based system gets an accuracy of 86.1% on the evaluation data. By increasing the penalty threshold the accuracy rises to 93.5%, and with a single addition to the lexicon it reaches 98.0%.

This suggests that grammar-based approaches can be useful when developing interactive systems for limited domains. In particular it seems that a grammar-based system could be well suited for systems that are built using an iterative and incremental development process (Larman and Basili, 2003), where the system is updated frequently and continuously evaluated by users.

References

Kais Dukes. 2013. Semantic annotation of robotic spatial commands. In *Proceedings of LTC'13: 6th Language and Technology Conference*, Poznań, Poland.

Craig Larman and Victor R. Basili. 2003. Iterative and incremental development: A brief history. *Computer*, 36(6):47–56.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Fernando C. N. Pereira and David H. D. Warren. 1980. Definie clause grammars for language analysis. *Artificial Intelligence*, 13:231–278.