View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by KnowledgeBank at OSU

Generating Disambiguating Paraphrases for Use in Crowdsourced Judgments of Meaning

RESEARCH THESIS

Presented in Partial Fulfillment of the Requirements for graduation with "Honors Research Distinction in Computer & Information Science" in the undergraduate colleges of The Ohio State University

By

Ethan A. Hill

The Ohio State University

May 2015

Committee:

Michael White, Advisor

Eric Fosler-Lussier

Jeremy Morris

Copyright by

Ethan A. Hill

2015

Abstract

Adapting statistical parsers to new domains requires annotated data, which is expensive and time consuming to collect. Using crowdsourced annotation data as a "silver standard" is a step towards a more viable solution and so in order to facilitate the collection of this data, we have developed a system for creating semantic disambiguation tasks for use in crowdsourced judgements of meaning. In our system here described, these tasks are generated automatically using surface realizations of structurally ambiguous parse trees, along with minimal use of forced parse structure changes.

Dedication

This document is dedicated to my friends and family who always forgave my time spent away working so many nights in the lab.

Acknowledgments

I would like to thank Dr. Michael White and Manjuan Duan for their instrumental role in guiding me through this work.

A warm thank you also to Dr. Eric Fosler-Lussier for patiently providing assistance throughout the thesis application process, along with Dr. Jeremy Morris, who despite short notice and a busy schedule, was honored to take part in the committee for the defense of this work.

A formal thank you as well to the National Science Foundation for providing funding for this project under grant IIS-1319318.

Table of Contents

List of Tables and Figures

List of Charts

1 Introduction

In parsing natural language, a good statistical parser requires training data which has been traditionally hand annotated by trained linguists as a gold standard. Hand annotation is an expensive and time consuming process and one that must be repeated if training data is adapted to new subject domains. This project aims to provide an alternative means for producing training data in new domains by automatically generating disambiguating paraphrases of syntactically ambiguous sentences which will later be used in experiments on crowdsourced judgments of meaning similarity. As part of the bigger picture, these judgments can then be used in retraining parsers to improve parsing accuracy in a new subject domain as well as language generation in broad coverage settings.

Previous work in the area of using crowdsourced annotations for corpus creation shows promise. A study by Jha et al. (2010) showed that annotators from the crowdsource platform Amazon Mechanical Turk (AMT) were able to make highly accurate judgements of prepositional phrase (PP) attachment ambiguity resolution when presented with the competing attachments. In this study AMT annotators were presented with possible attachments for prepositional phrases and were asked which attachment was correct (five annotations were collected for each attachment ambiguity). Jha et al. (2010) reported that annotators were capable of attachment choice accuracies of 97% in the best case (all five annotators of an ambiguity were in agreement) and 64% to 67% in

the worst case (a plurality of two and three annotators respectively for the attachment choices of an ambiguity). Our own initial study, as discussed in section 3 *Initial Study*, also shows some promise.

Our work involves collecting naturally occurring sentences from several domains from Wikipedia and analyzing them across an n-best list of parses for syntactically ambiguous structures. These ambiguous parses are then generated back into new sentences by OpenCCG's surface realizer and automatically analyzed for breakup in the original ambiguity. Finally, these new sentences are re-parsed in order to verify the original syntactic analysis.

Since not all ambiguous sentences are able to generate disambiguating paraphrases automatically, forced structural changes are applied to the parses of these sentences in order to increase coverage. These changes do not alter the original meaning of the sentence but are designed to force a generated sentence to demonstrate the intended meaning hidden by the ambiguity of the original sentence. These modifications involve forcing passive and/or cleft structures in the presence of some attachment ambiguities, or forcing verbosity in the presence of ambiguities occurring across coordinating conjunctions.

We describe first in section 2 some background of the tools and concepts referenced throughout this paper so the reader may have a firm understanding of the processes used within our system. In section 3, we briefly discuss some initial findings from an evaluation of this system using judgements from AMT annotators. Section 4 describes in depth the methodology and design of our system, along with several

illustrations and examples. We then share the results of our system on a corpus of data drawn from Wikipedia in section 5, followed by a discussion of these results and the peculiarities worth examining. A conclusion follows in section 7 which suggests further work and experimentation using this system.

2 Background

We use OpenCCG[1], an open source natural language processing library written in Java for Combinatory Categorial Grammar (Steedman 2000), as an integral part of our system and so it is beneficial to give a brief overview of this technology as requisite knowledge.

Combinatory Categorial Grammar (CCG) is a categorial grammar formalism capable of modeling a wide range of linguistic phenomena, and is defined by a lexicon of items associated with syntactic categories, which correspond to a semantic interpretation (Steedman and Baldridge 2005). Categories in CCG can act as functions, and because CCG is unification-based, structures can be formed from the combination of categories using an explicit set of CCG combinatory rules.

OpenCCG provides a CCG parser which can probabilistically produce the most likely dependency graph given some sentence and a CCG lexicon. OpenCCG uses a lexicon extracted from CCGBank (Hockenmaier and Steedman 2007), a corpus of CCG derivations derived from the Penn Treebank corpus. Boxwell and White (2008) have enhanced the version of CCGBank in use by OpenCCG by including PropBank's (Palmer et al. 2005) Penn Treebank verb argument and modifier semantic roles. This work was

[1] http://openccg.sourceforge.net

intended to facilitate broad coverage generation with CCG and improve the OpenCCG parser.

In addition to a CCG parser, OpenCCG also includes a CCG surface realizer. Surface realization is a language generation task that, when given an input logical form (semantic graph), produces a sequence of words constrained by a lexicon and grammar (in our case, the Propbank enhanced CCGBank). Because the number of probable realizations of a logical form can become explosive if the grammar permits a relatively free word ordering, a chart-based and statistical hybrid algorithm is used for more efficient realization (White 2006). Realizations from OpenCCG are ranked using a 5-gram model from the Gigaword corpus and/or a perceptron scoring model as described by White and Rajkumar (2009).

The system described hereafter in section 4 was built entirely using the Python v2.7[2] standard library (with the exception of a Wikipedia text extraction script described in section 4 below) and is intended to be an extension package for the OpenCCG library already mentioned.

[2] https://www.python.org

3 Initial Study

Our current iteration of this system has recently undergone a small test using Amazon's Mechanical Turk (AMT) in order to gauge the effectiveness of the paraphrases generated. AMT users are shown the original ambiguous sentences and one or two generated paraphrases from the top competing parses for these sentences. The task is to simply select the paraphrase which is closer in meaning to the original sentence from the competing parses. Alongside the judgments collected from AMT, we have also provided our own judgments of which parses are correct for sentences as a gold standard to measure the accuracy of the AMT similarity judgments. For each similarity judgement task, we also noted if none of the parses were correct so that the parses in that task are not considered in the AMT judgement accuracies.

In an initial trial of 92 sentences, we have observed encouraging results. For each similarity judgement task, we collected five judgments from AMT users and observed that for majority agreement cases (three or more users agree on a task), accuracy is much higher than chance level. For agreement of three or more AMT users on a sentence, we observe 67% accuracy over 69 sentences and for agreement of four or more users, we observe 71% accuracy over 51 sentences. In cases of total user agreement, we observe 77% accuracy over 26 sentences. This shows that our method of paraphrase generation is effective for creating crowd-sourced annotation tasks. In our gold standard annotations,

we found 23 sentences of the 92 to have incorrect parses and so were not considered in these reports of accuracy. For continued study, we are extending these trials to include judgments on 1000 total sentences to establish a better and deeper analysis.

4 Methodology and System Design

4.1 Wikipedia Data Extraction

For open domain data, text was collected from two article categories on Wikipedia, *Prehistoric Reptiles* and *Big 10 Conference Football.* The text from the articles in these categories (and related sub-categories) was extracted using the Python package Wikipedia[3] and then tokenized into sentences using another Python package, NLTK[4]. Of all downloaded Wikipedia sentences, only sentences with word counts ranging between five and twenty were selected. This word count limitation is in place in order to favor more simple parse trees in the hope that, when adapting a parser to the new domain, simple structures will generalize more easily. There is also a limitation on Unicode characters present in the sentences for convenience in handling the text later in Python (Python v2.7 has a different interface[5] for Unicode strings than other encodings like ASCII). Downloaded sentences are stored in files corresponding to the name of the article from which they came for convenience in later examining the sentence's context on the original Wikipedia source article. For our collection analysis, we will examine the

[3] https://wikipedia.readthedocs.org
[4] Natural Language Toolkit, http://www.nltk.org/
[5] https://docs.python.org/2/howto/unicode.html

amount of paraphrases available for ambiguous sentences and the distribution of paraphrase types based on the strategies which are described in this section.

4.2 Parsing and Ambiguity Analysis

To begin the process of generating disambiguating paraphrases, each downloaded sentence is parsed using the OpenCCG parser into an n-best parse list (with $n = 25$). All parses with more than one tree root are filtered out as these are considered to be broken parses. Next, a set of unlabeled dependency relationships is generated for each parse tree in order to detect structural ambiguities existing between competing parses of a sentence. This examination of unlabeled dependencies allows for minor labeling differences, such as part of speech tags, to be ignored between otherwise structurally equivalent parses. Any unlabeled dependency set of a parse containing a dependency whose dependent is the same as its head, i.e. (head = John, dependent = John), is ignored, as this is also indicative of a malformed parse.

After applying these filters to the list of parses, sentences are examined for structural ambiguities existing between its top parses. Ambiguities are identified by selecting the top parse of a sentence and its next best parse which is distinct from the top parse and therefore amenable to disambiguation via paraphrasing. Distinct parses, in the case of this system, have a non-empty symmetric difference between the unlabeled dependency sets of the parses, and this difference then passes through a number of other filters. These filters are designed to allow the system to ignore other minor differences between parses which don't contribute to interesting ambiguities.

The first of these filters simply checks if the symmetric difference of the parse's unlabeled dependency sets has only one head-dependent pair (i.e. the symmetric difference consists of only one element). It is not clear that this kind of ambiguity contributes any kind of meaningful difference in interpretations because the parses are identical with the exception of just one additional dependency relationship.

The next filter for the competing parses examines part-of-speech tags for the dependencies in the symmetric difference in order to identify possible auxiliary attachments. We define auxiliary attachments in the same way that de Marneffe and Manning (2008) define them in their Stanford dependencies manual. The symmetric difference is not considered sufficiently interesting as an ambiguity if a dependency in the difference is headed by a modal (PTB *tag = MD*) or forms of "be", "do", or "have", and the dependent has one of the following part-of-speech tags (PTB): *RB*, *JJ*, and *IN*. A sentence's top parses whose differences pass through these filters are considered to be distinct and structurally ambiguous enough that their interpretations may possibly be disambiguated by our system.

4.3 Reverse Realization and Verification

4.3.1 Breaking up the Ambiguity

In order to automatically generate disambiguating paraphrases for ambiguous sentences and verify these paraphrases have the same interpretation structurally as the parses from which they were generated, we have devised a process of **reverse realization** (parse a sentence and realize sentence back from its parse) with a **verification** step. This

process begins where the previous section had left off; once a top parse and next best parse are identified as ambiguous for a sentence, these parses are then realized using the OpenCCG surface realizer. This produces an n-best list of realizations for each parse (for the current system, *n=25* realizations). Here, the goal in generating a list of new sentences from a parse is to find a disambiguating paraphrase of the original parsed sentence. In realizing a parse, it is possible that words in the original ambiguous sentence can be expressed in different orders in the realization, and this new ordering can lead to an unambiguous interpretation of the sentence. By moving down the n-best list of realizations and comparing each realization to the original sentence, we can select a realization for each parse which appears to accomplish disambiguation. To illustrate this further, consider Sentence 1 drawn from the *Prehistoric Reptiles* corpus (specifically from an article about the 1997 movie *The Lost World: Jurassic Park*) and its competing parse trees (Figure 1 and Figure 2) pictured on the next pages.

The two adult T-Rex and their baby are shown to have been returned safely.

Sentence 1 as sample input to our system

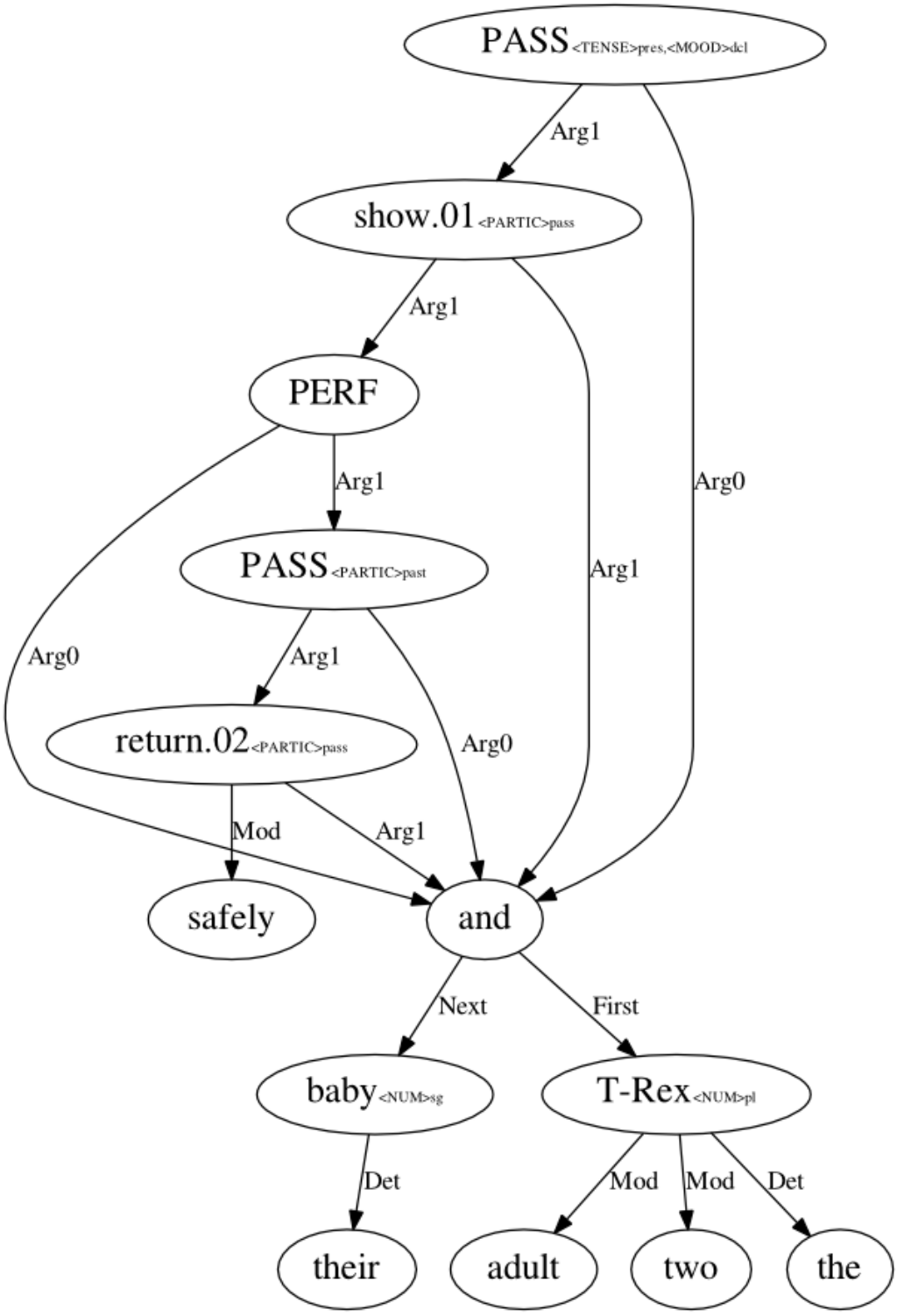

Figure 1: The top parse for Sentence 1 as produced by the OpenCCG parser

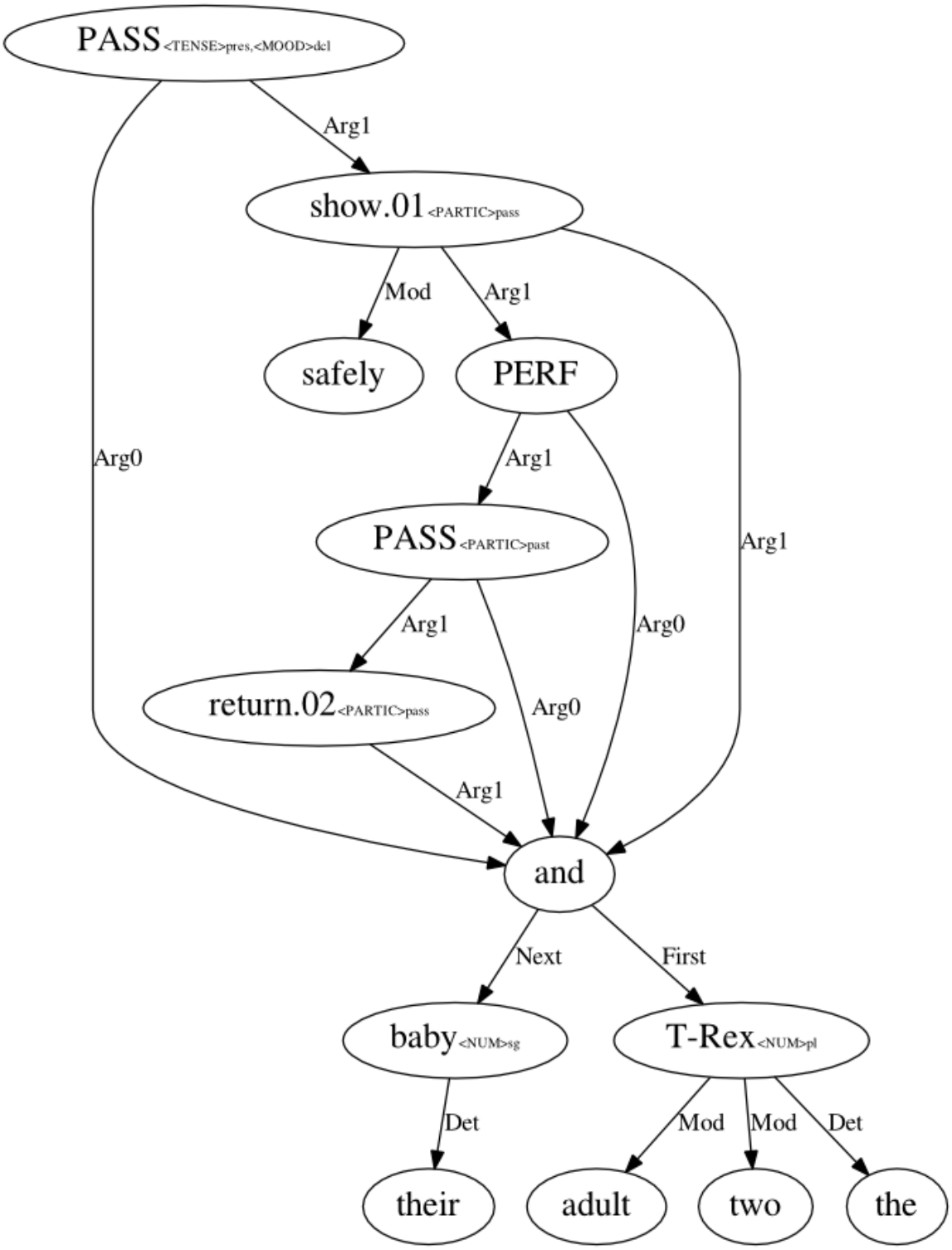

Figure 1: The next best parse for Sentence 1 as produced by the OpenCCG parser and selected by the filters described in section 3.2

As can be seen in Figure 1 and Figure 2, the ambiguity of the original sentence involves the adverb “safely” attaching to either of the verbs “shown” or “return”. Table 1 and Table 2 show the resulting realizations which are produced given those parse trees.

1) *The two adult T-Rex and their baby are shown to have been returned safely.*
2) *The adult two T-Rex and their baby are shown to have been returned safely.*
3) ***The two adult T-Rex and their baby are shown to have been safely returned.***
4) *The two adult T-Rex, and their baby are shown to have been returned safely.*

….

24) *The two adult T-Rex, and their baby are shown to have been safely returned.*
25) *The two adult T-Rex and their baby am shown to have been returned safely.*

Table 1: The output realizations resulting from the parse tree in Figure 1 as input to the OpenCCG realizer, with the selected realization in bold.

1) *The two adult T-Rex and their baby are shown to have been returned safely.*
2) ***The two adult T-Rex and their baby are shown safely to have been returned.***
3) *The two adult T-Rex and their baby are safely shown to have been returned.*

…

24) *The two adult T-Rex and their baby are shown safely na have been returned.*
25) *The adult two T-Rex, and their baby are shown to have been returned safely*.

Table 2: The output realizations resulting from the parse tree in Figure 2 with the selected realization in bold.

Of note are the following realizations from Table 1 and Table 2.

The two adult T-Rex and their baby are shown to have been safely returned.

Realization 1: A selected realization which breaks up the structural ambiguity appearing in Sentence 1 and corresponds to the parse tree from Figure 1

The two adult T-Rex and their baby are shown safely to have been returned.

Realization 2: A selected realization which breaks up the structural ambiguity appearing in Sentence 1 and corresponds to the parse tree from Figure 2

From Realization 1 and Realization 2 one can clearly see the differences in interpretation for the two parses, but this intuition needs to be automated somehow. In this system, realizations of parses are examined for any breakup of the ambiguity in the original sentence. The words from the unlabeled dependencies which make up the ambiguity between the two parses are used to form what we call an **ambiguous span** in the original sentence.

(shown, safely) (returned, safely)

Figure 3: The unlabeled dependencies that form the ambiguity between the parses in Figure 1 and Figure 2

For example, in Figure 3 above, the ambiguous span is *{safely, shown, returned}* for Sentence 1. We then establish a one-to-one correspondence for the words in the span between the original sentence and the realization being examined. To ensure an exact correspondence of words for this system, we store the paths in the sentence's parse trees

from the root to the nodes of the words in the ambiguous span. The OpenCCG realizer maintains as its output, the same parse tree of the generated sentence as the original parse with updated word index (sentence offset) values, so we can use these stored paths to retrieve the new offset positions of the words from the ambiguous span as they appear in the generated sentence. This approach removes any possibility of incorrectly examining words which are the same as those appearing in the ambiguous span, but are not contributing to the ambiguity.

After establishing a correspondence between the original sentence and a generated sentence, we measure the relative word distances between each word in the ambiguous span. These distances are bidirectional and are measured for both the original sentence and the generated sentence.

Word	Other Word	Relative Distance
shown	safely	5
safely	shown	-5
returned	safely	1
safely	returned	-1

Table 3: The relative distances between the words in the ambiguous span for Sentence 1

Word	Other Word	Relative Distance
shown	safely	4
safely	shown	-4
returned	safely	-1
safely	returned	1

Table 4: The relative distances between the words in the ambiguous span as they appear in Realization 1

With these measurements, one can observe movement of the words in the ambiguous span from the original sentence to the generated sentence and ignore inconsequential movement of the other words not contributing to the ambiguity. Any change in this relative distance between the words in the ambiguous span is considered a

breakup of the ambiguity, and the realization providing this breakup is then selected for **verification**. This process is repeated for both of the competing parses of an ambiguous sentence and subsequently for all ambiguous sentences in the corpora.

4.3.2 Verifying a Realization

Selected realizations which breakup ambiguities (as described in the previous section) go through a process of verification intended to determine whether a realization is structurally representative of the parse from which it was generated (and subsequently its interpretation). This process is inspired by work in self-monitoring for avoiding structural ambiguities in realizations by examining their parses as described by Duan and White (2014). In order to achieve this, the realizations themselves are parsed like their original counterparts using the OpenCCG parser. The new set of unlabeled dependencies from the top parses of the realizations are then compared to the dependencies of the original parses from which the realizations were generated. If the realizations' dependency set contains the subset of dependencies from the original parse which contributed to the ambiguity, this is indicative of a realization which structurally shares the original interpretation of the parse from which it was generated. We consider these realizations to be verified paraphrases and are called **reversals** in this system.

4.4 Parse Structure Rewrites

4.4.1 Passivization and Clefting

Taking a step back in the control flow of this system for a moment, there are a few more processes which are applied to the parses of ambiguous sentences before reverse realization and verification, as described in the previous section. There are cases of ambiguous sentences in which it is not possible to have reversals that disambiguate the interpretations for both parses. As a simple example, one can look at Sentence 2 and its parses in Figure 4 and Figure 5

He stopped Godzilla with the laser.

Sentence 2: Sample input for passive and cleft rewrites

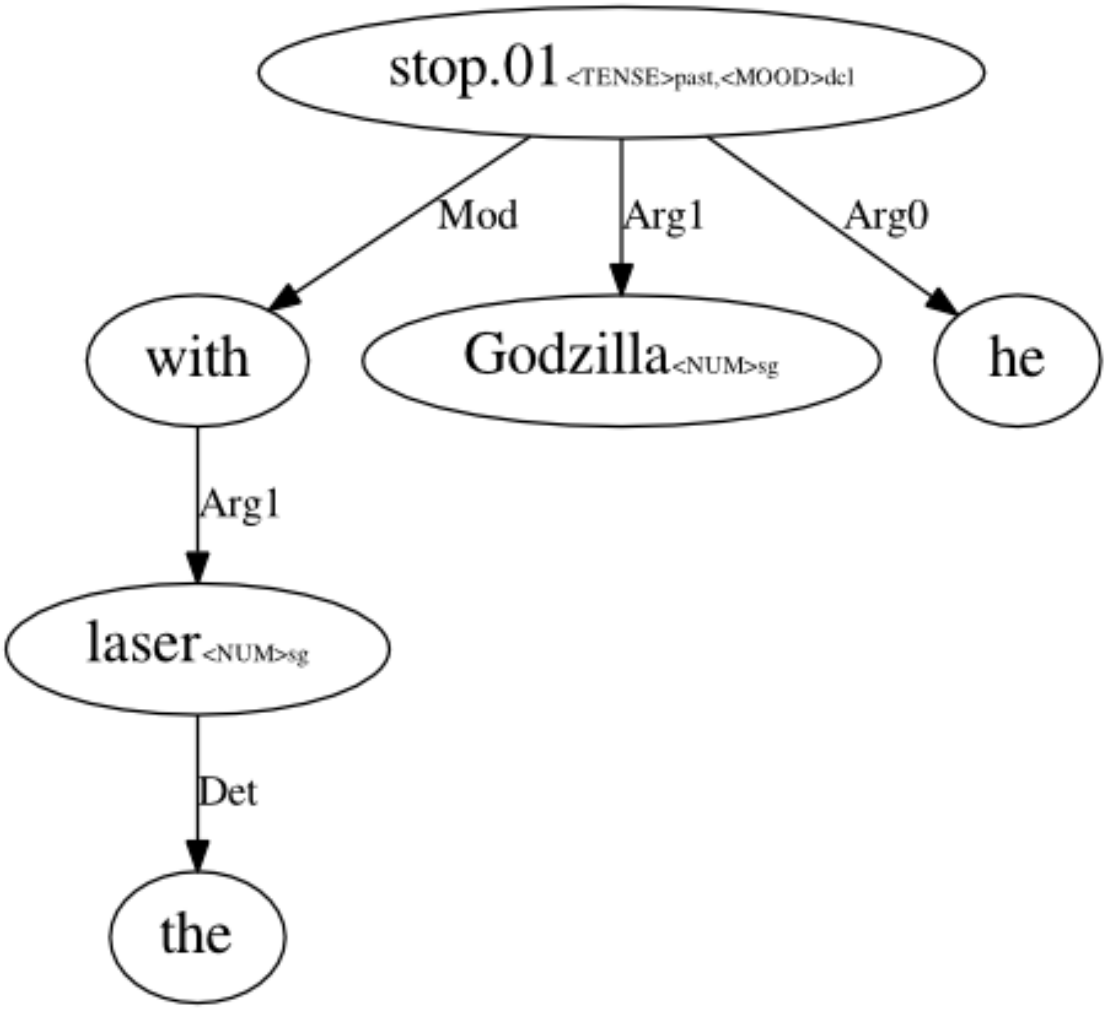

Figure 4: Top parse tree for Sentence 2 as output from the OpenCCG parser

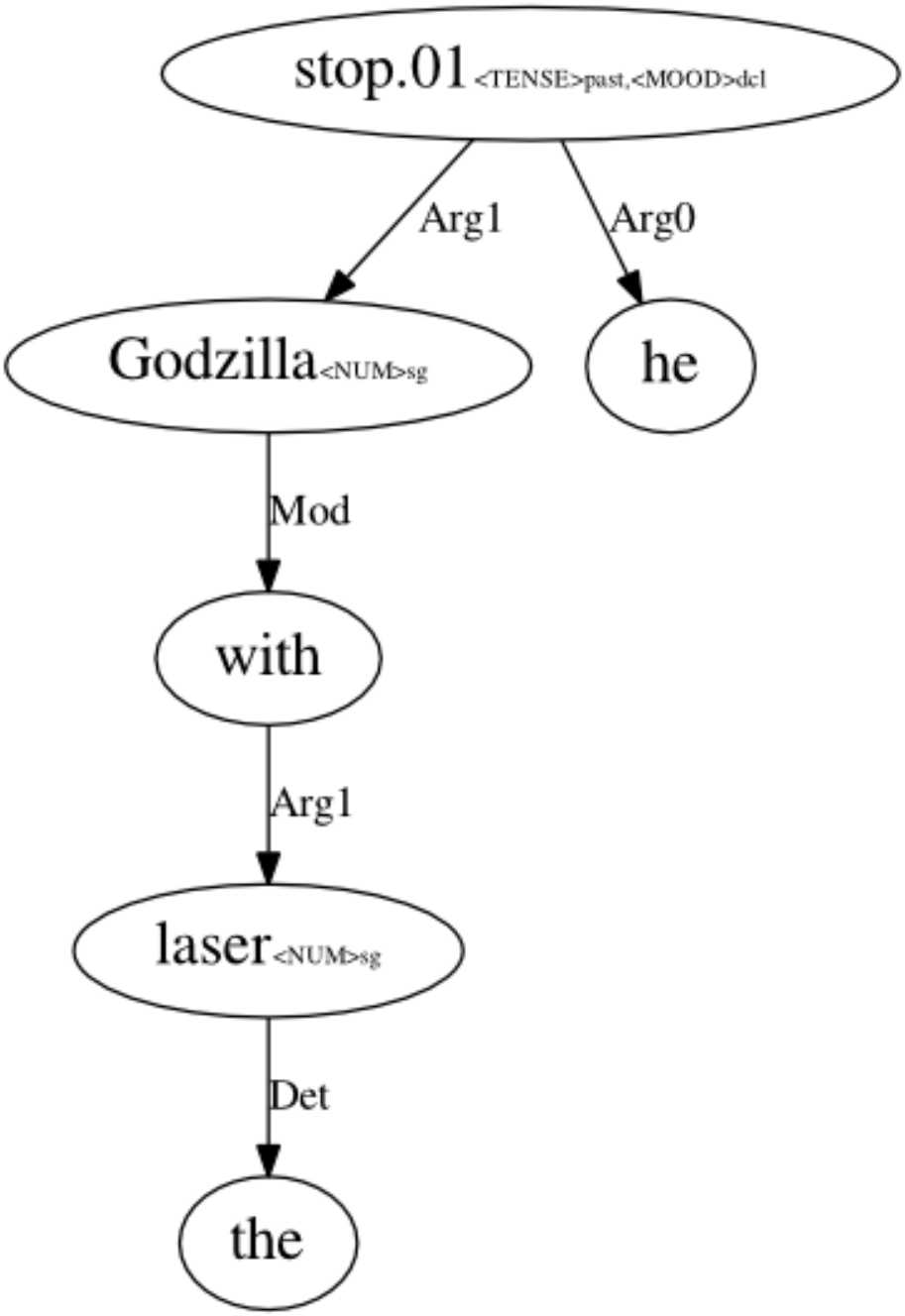

Figure 5: The next best parse for Sentence 2 as selected by our system

When selecting reversals for these two parses, we observe the following:

1) He stopped with the laser Godzilla.
2) *He stopped Godzilla with the laser.*
…
25) He stopped with the laser Godzilla

Table 5: Realizations corresponding to the parse tree in Figure 4

1) He stopped Godzilla with the laser.
2) He stopped Godzilla with the laser.
3) He stopped with the laser Godzilla.
…
25) *He stopped Godzilla with the laser.*

Table 6: Realizations corresponding to the parse tree in Figure 5

It is clear that no amount of word reordering is going to produce a sentence which is structurally representative of the interpretation for the second parse. To overcome this, we force structure changes in the parse trees which, when realized, more sufficiently demonstrate the parse's interpretation without changing the original meaning of the sentence. The realizations resulting from these structure changes are referred to as **rewrites** in this system.

Specifically, the changes applied to parse trees involve altering the voice of the sentences to passive and/or cleft structures. As of now, these changes are applied in the presence of prepositional phrase (*PP*) attachment ambiguities but could be extended to cover cases of adverbial attachment ambiguities as well.

To detect the presence of a PP attachment ambiguity, we simply look to the part of speech tags of the words in the ambiguous span for any prepositions (PTB *tag = IN*). Next, we verify that the sentence has a voicing which can be passivized/clefted by iteratively moving up the nodes of the parse trees, starting from the PP's root node, to find the verb of the sentence. Once a verb above the PP is found in a parse, we repeat this process for the other parse and compare the verbs found above the PP. If the verbs are not the same, we consider the verb which has the other verb as a descendant. We do this in order to capture the whole verb phrase, which might be modified by the PP, in a passive/cleft structure. Then, we find the subject and object(s) just below the verb (in our system they are Arg0 and Arg1 of the verb).

For a passive rewrite, we create a passive (PASS) node with the same tense as the original sentence in the dependency graph above the verb. We then make the object into the subject (Arg0) and the verb itself into Arg1 under the PASS node. The original subject of the verb is replaced with "by SUBJECT" sequence. If the subject was a pronoun, we replace it with the object pronoun (she-> her).

For a cleft rewrite, we create a "be" verb node, again with the same tense as the original sentence, above the verb we are considering and make the object into the Arg0 subject. Where the cleft rewrite differs is in the treatment of the original subject and verb. We move the verb (and consequently the subject) under a "what" reference node (x1), and in the place of the object under the original verb, we create a second reference (x2). This encodes the "what SUBJECT VERB" sequence of a cleft structure.

An example of the parses from Figure 4 and Figure 5 with their passive and cleft rewrites are shown in Figure 6, Figure 7, Figure 8, and Figure 9 on the next pages. For further illustration of the effect of a rewrite on a parse tree, the rewrite realizations are also included in these figures.

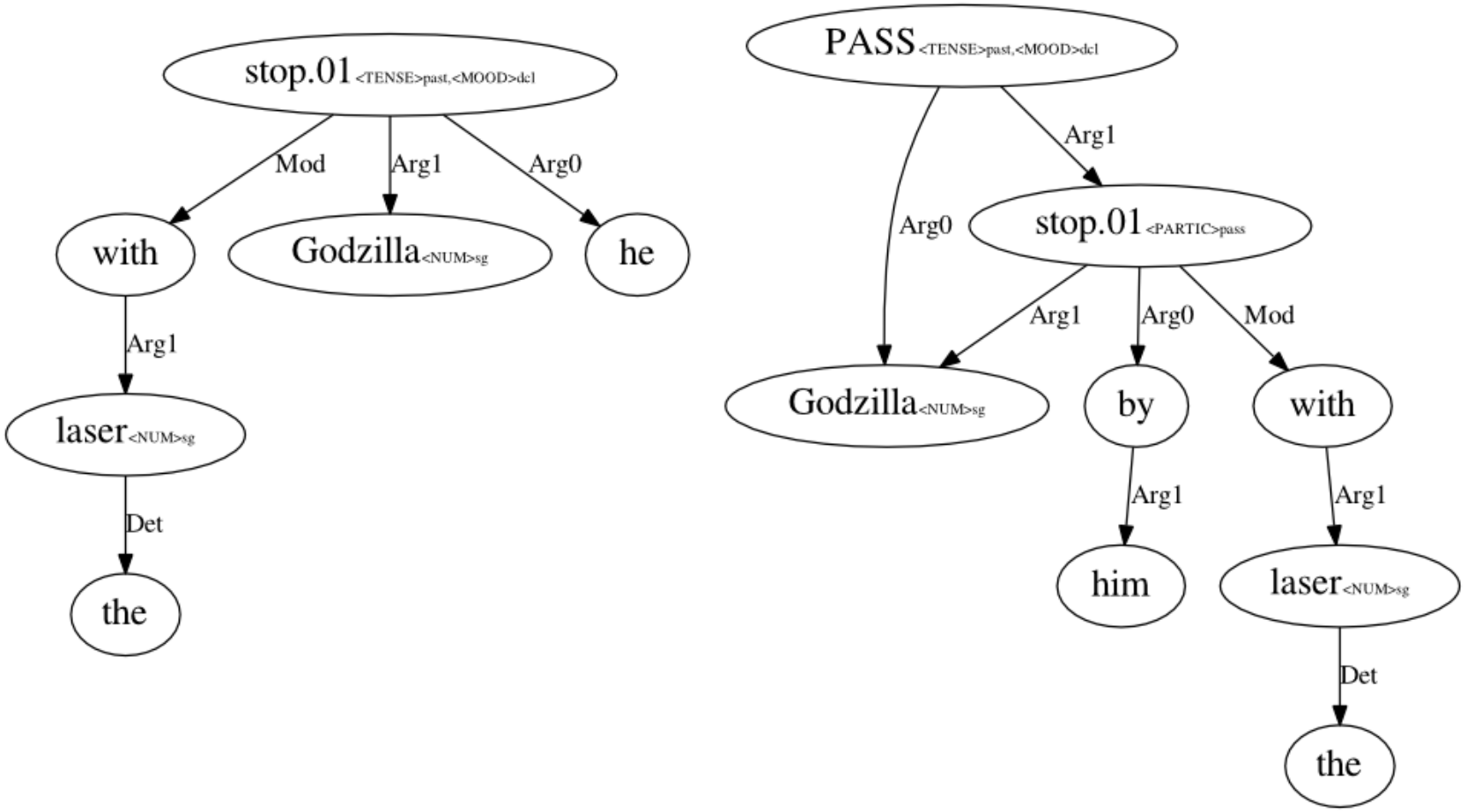

Figure 6: Pictured left is the parse tree from Figure 4 and pictured right is the same parse tree after a passive rewrite was applied to it. The parse tree on the right has the realization, *Godzilla was stopped by him with the laser*.

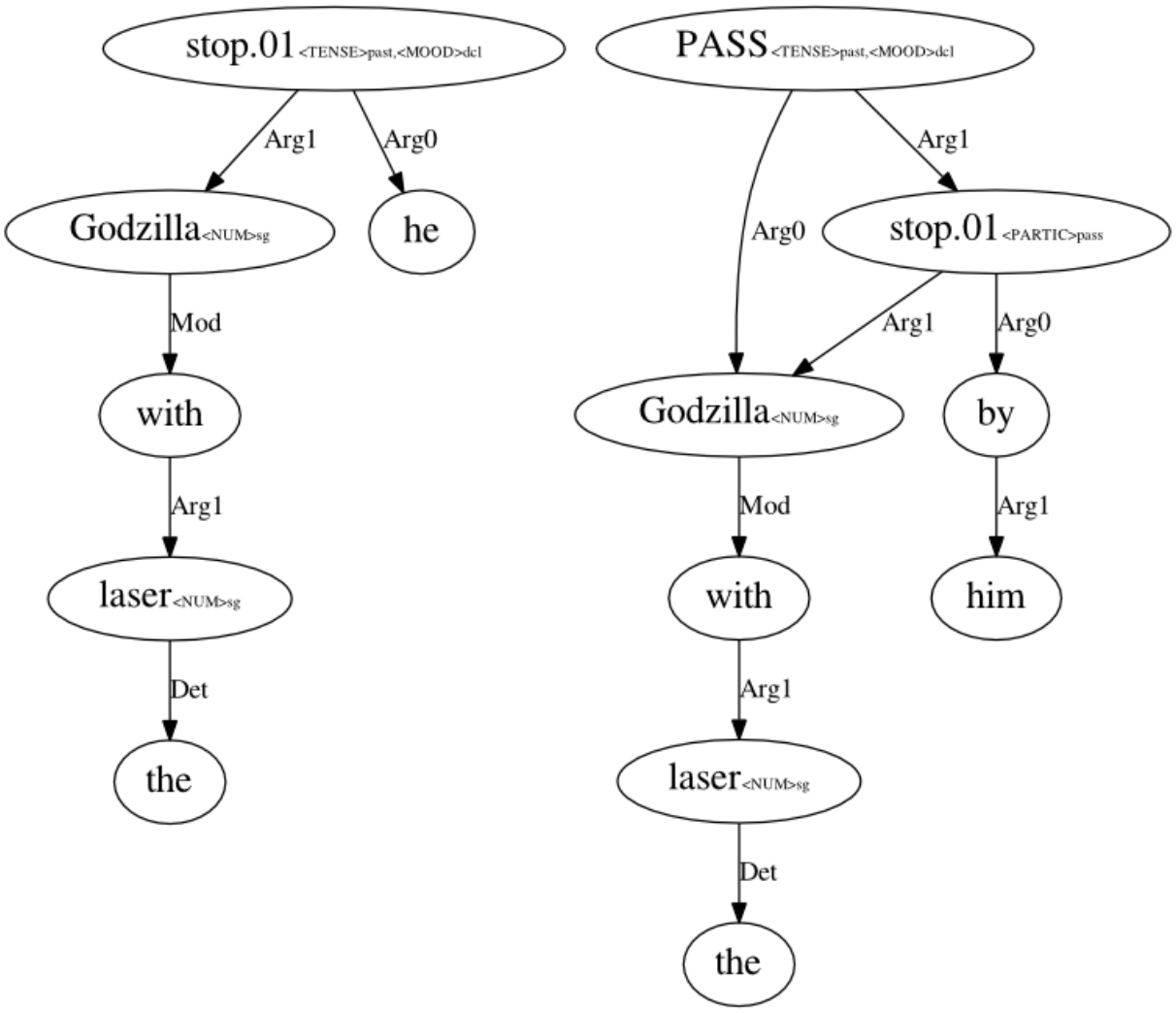

Figure 7: Pictured left is the parse tree from Figure 5 and pictured right is the same parse tree after a passive rewrite was applied to it. The parse tree on the right has the realization, *Godzilla with the laser was stopped by him.*

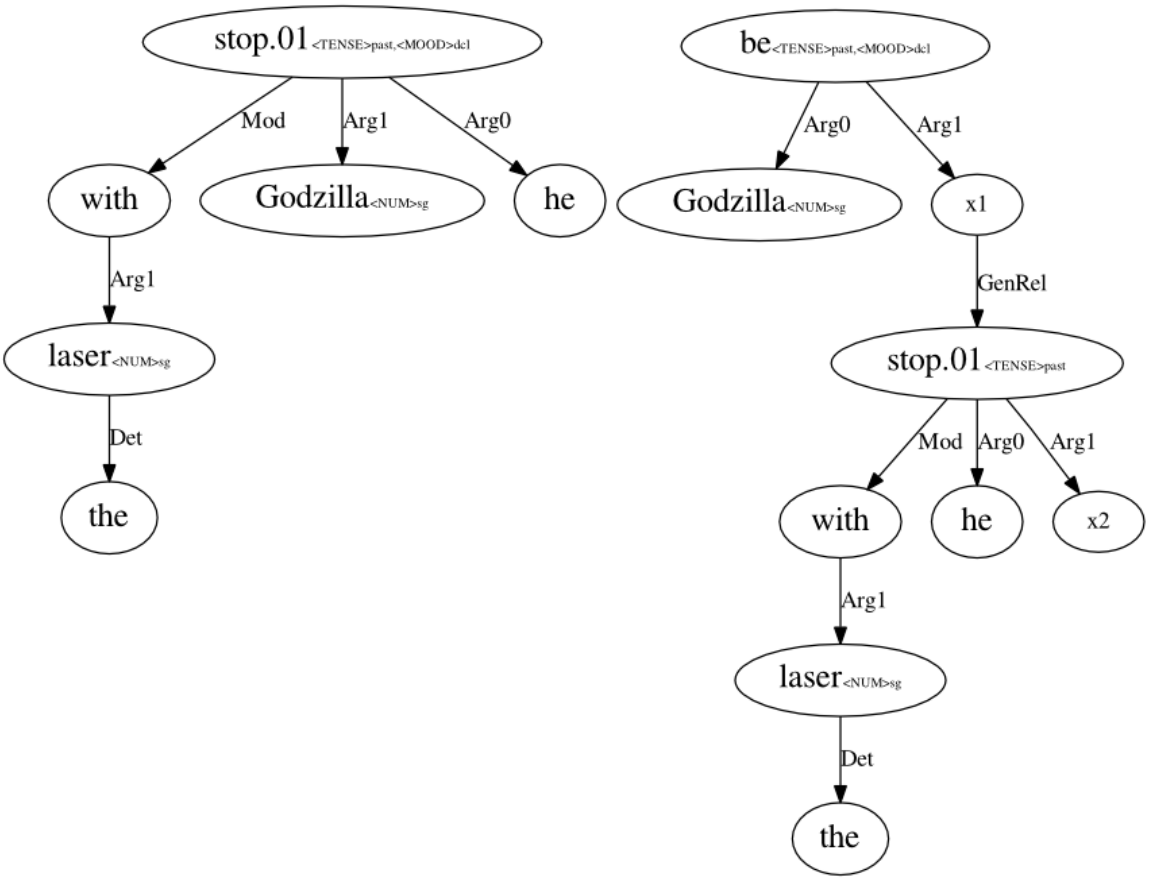

Figure 8: Pictured left is the parse tree from Figure 4 and pictured right is the same parse tree after a cleft rewrite was applied to it. The parse tree on the right has the realization, *Godzilla was what he stopped with the laser*.

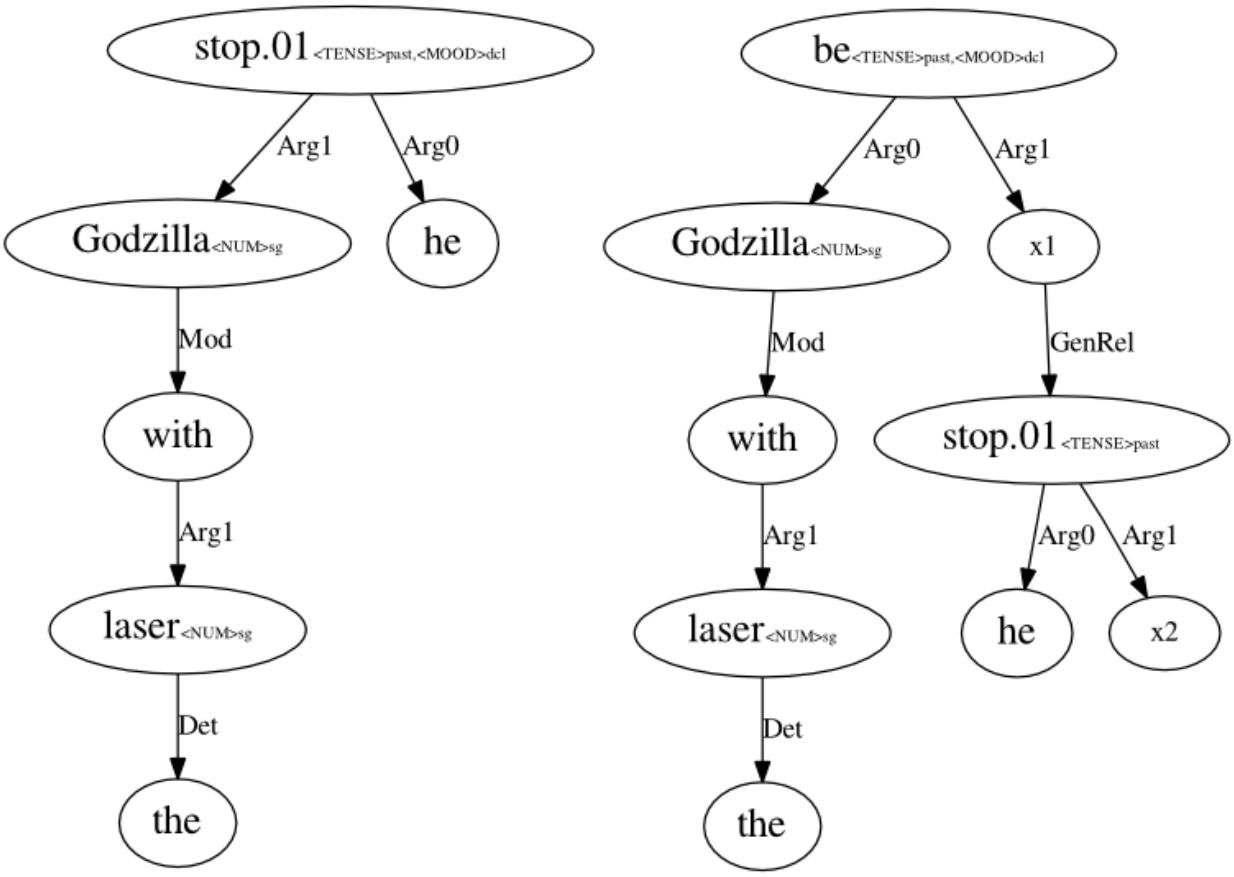

Figure 9: Pictured left is the parse tree from Figure 5 and pictured right is the same parse tree after a cleft rewrite was applied to it. The parse tree on the right has the realization, *Godzilla with the laser was what he stopped.*

After all changes have been applied to the parse trees, they are realized and checked for breakup of the ambiguous span just as in the reversals, with one small difference in the procedure. Because large changes were made to the parse trees, a record of these changes must be kept in order to establish a one-to-one correspondence between the original sentence and the rewrites. With these changes, we do not apply the verification to the rewrites as we do for reversals because we assume that the changes

made to the parse structure in a rewrite will properly demonstrate the intended interpretation of the parse.

4.4.2 Verbosity for Coordination Ambiguities

In order to increase our coverage of other ambiguities, we have also extended our rewrite process to include changes for coordination ambiguities. Just like the passivization and cleft rewrites, coordination rewrites look to the part of speech tags of the words appearing in the ambiguous span for a coordinating conjunction (PTB *tag* = *CC*) in order to determine if a rewrite may be applied. Once a coordination ambiguity is identified, the rewrite involves forcing verbosity in the parse tree wherein any modifiers and determiners applied across the conjunction are moved to its individual arguments on both sides of the conjunction, as in Figure 11. The order of the arguments for the conjunction are then reversed, so as to ensure that any modifiers of the first position argument now appear after the conjunction, as in Figure 10. Following these changes, the exact same process of checking for ambiguity breakup in passive and cleft rewrites is applied to coordination rewrites.

He was also selected to play in East/West Shrine game and Hula bowl.

Sentence 3: Sample input for a coordination rewrites depicted in Figure 10 and Figure 11

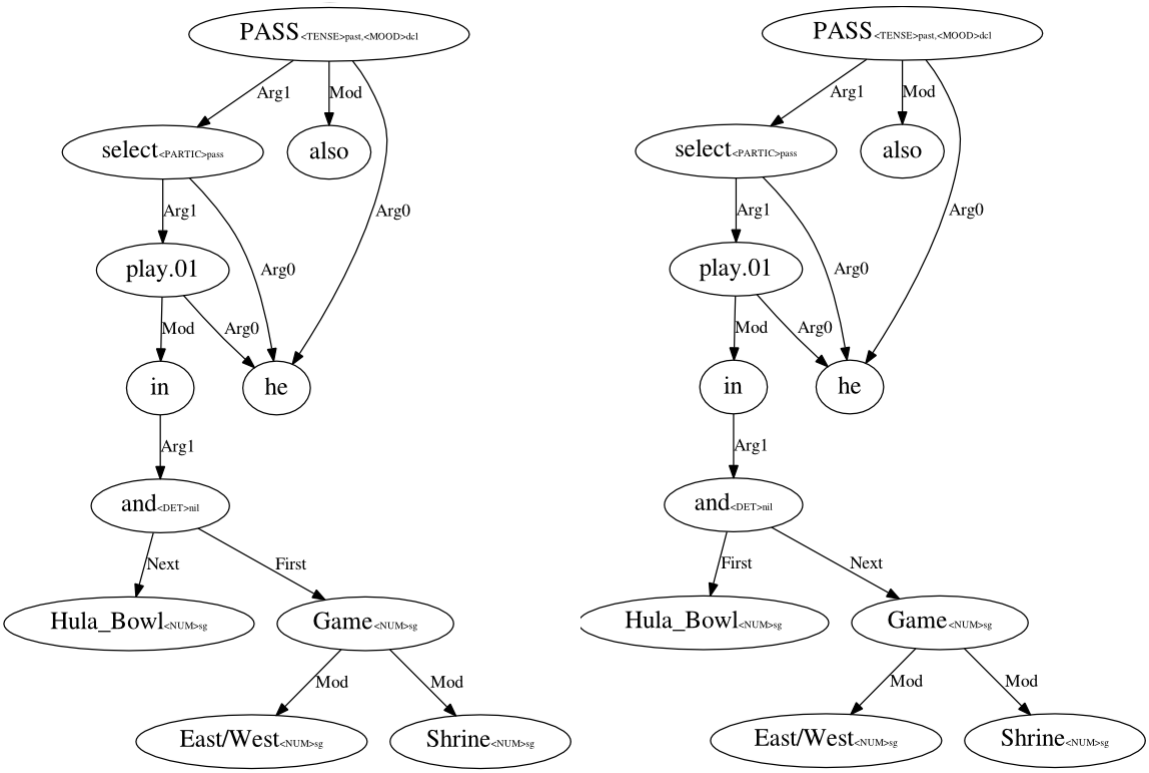

Figure 10: Pictured left is the top parse tree of Sentence 3 and pictured right is the same parse tree after a coordination rewrite was applied to it. Note the order of the arguments for the coordinating conjunction. The parse tree on the right has the realization, *He also was selected to play in Hula bowl and East/West Shrine game.*

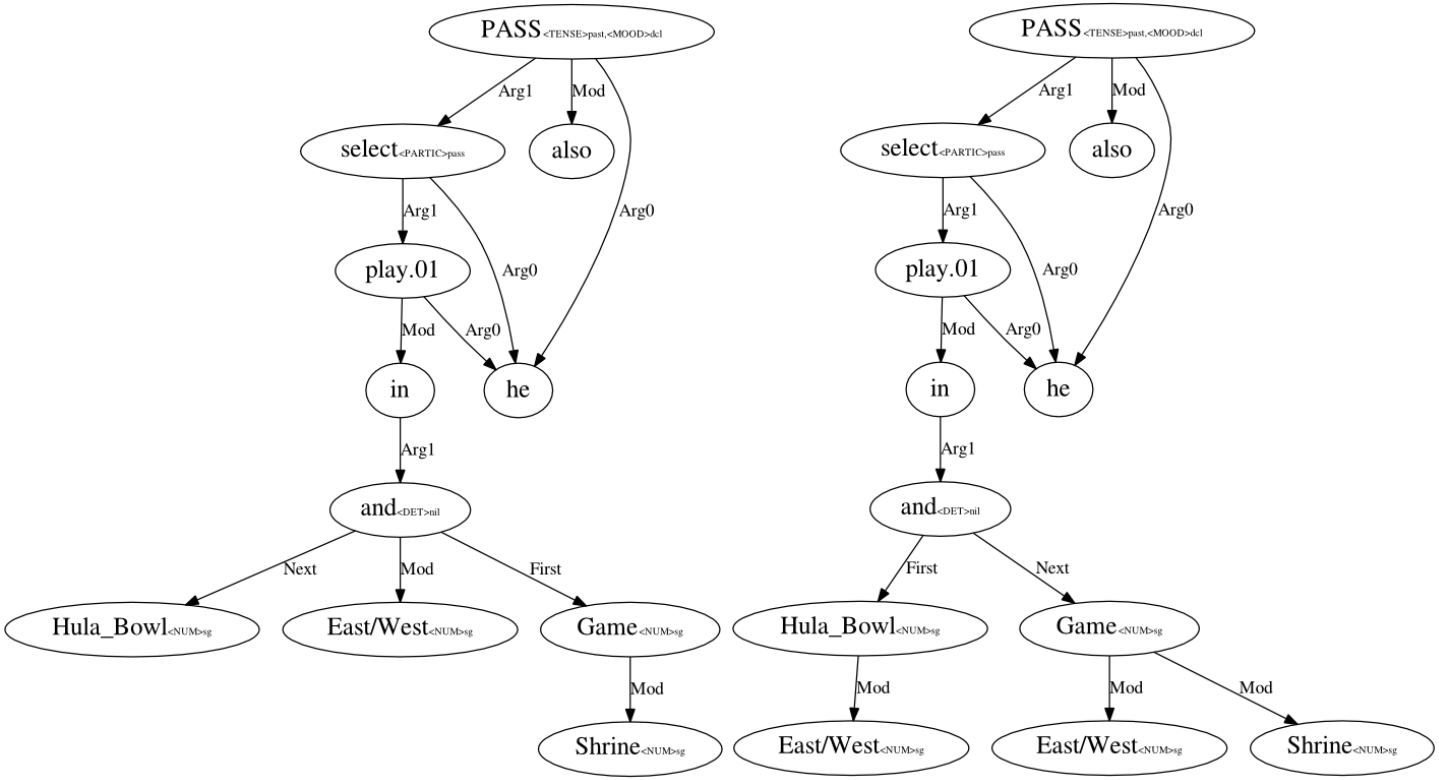

Figure 11: Pictured left is the next best parse tree of Sentence 3 and pictured right is the same parse tree after a coordination rewrite was applied to it. Note the order of the arguments for the coordinating conjunction as well as the modifier *East/West having* shifted. The parse tree on the right has the realization, *He also was selected to play in East/West Hula bowl and East/West Shrine game.*

5 Results

In this section, several charts have been included which give various breakdowns of the data extracted from Wikipedia and processed by our system. For real valued illustrations of the data shown in these charts, please refer to appendix items B through J.

A total of 4769 articles were downloaded from Wikipedia using the text extraction script described in section 3.1: 1965 articles from *Prehistoric Reptiles* and 2804 from *Big 10 Conference Football*. From those articles, a total of 17,097 sentences were extracted: 6525 from articles related to *Prehistoric Reptiles* and 10,572 from articles related to *Big 10 Conference Football*. Many of these sentences were actually repeated several times across articles from the *Big 10 Conference Football* corpus (and to a much lesser extent, the *Prehistoric Reptiles* corpus as well) and so the actual number of unique sentences extracted from these Wikipedia articles were much lower: 6,335 sentences from *Prehistoric Reptiles* (2.9% reduction in total), 7,779 from *Big 10 Conference Football* (26.4% reduction in total), and a combined total of 14,114 sentences (17.45% reduction). The remaining numbers shown here will be in relation to this corpus of unique sentences.

After these sentences were processed in our system by the methods described in Section 3, a great deal of the sentences were found to be unambiguous: 3,877 sentences from the *Prehistoric Reptiles* articles (61.2% of total sentences), 5,174 sentences from *Big 10 Conference Football* (66.5% of total sentences), with a combined total of 9,051 unambiguous sentences (64.1% of total sentences).

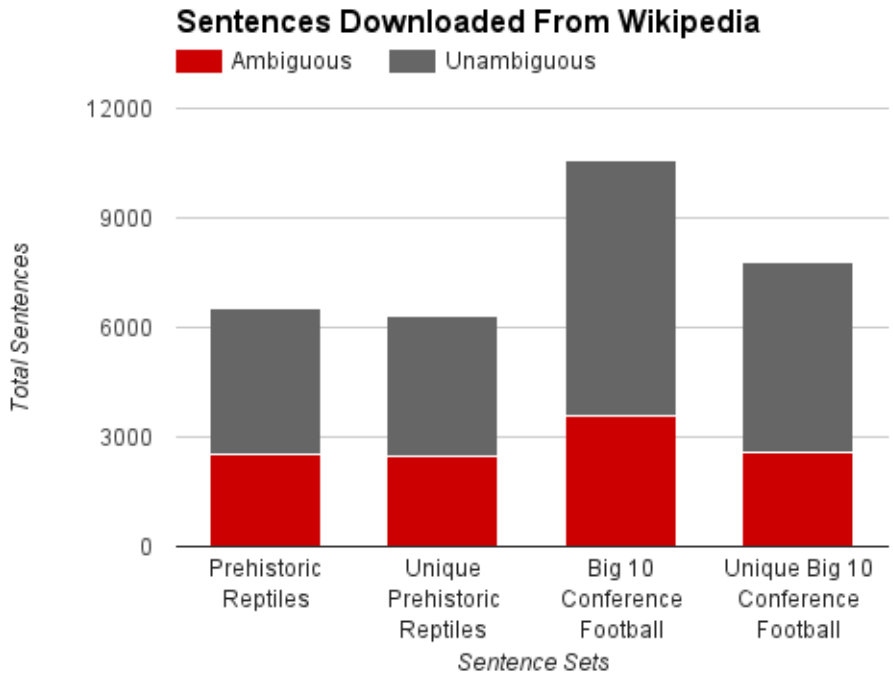

Chart 1: The number of sentences which are ambiguous vs. unambiguous across two corpora, including the same comparison for the subsets of unique sentences from the same corpora.

It is now pertinent to introduce a term we will use throughout this section concerning the nature of the paraphrases generated by this system. Ideally, every ambiguous sentence processed by our system will generate two disambiguating paraphrases, one for each competing parse tree. However, because these paraphrases must pass through a verification phase, not all ambiguous sentences will have two paraphrases. We call a paraphrase that corresponds only to one of the competing parse trees of an ambiguous sentence a one sided paraphrase, and similarly we call paraphrases that correspond to both competing parse trees a two sided paraphrase.

Of the 5,063 ambiguous sentences processed by our system, 1,458 sentences have

no paraphrases generated for them (28.8% of the total ambiguous sentences), 2,399 sentences have one sided paraphrases (47.4% of the total ambiguous sentences), and 1,206 sentences (23.8% of the total ambiguous sentences) have two sided paraphrases.

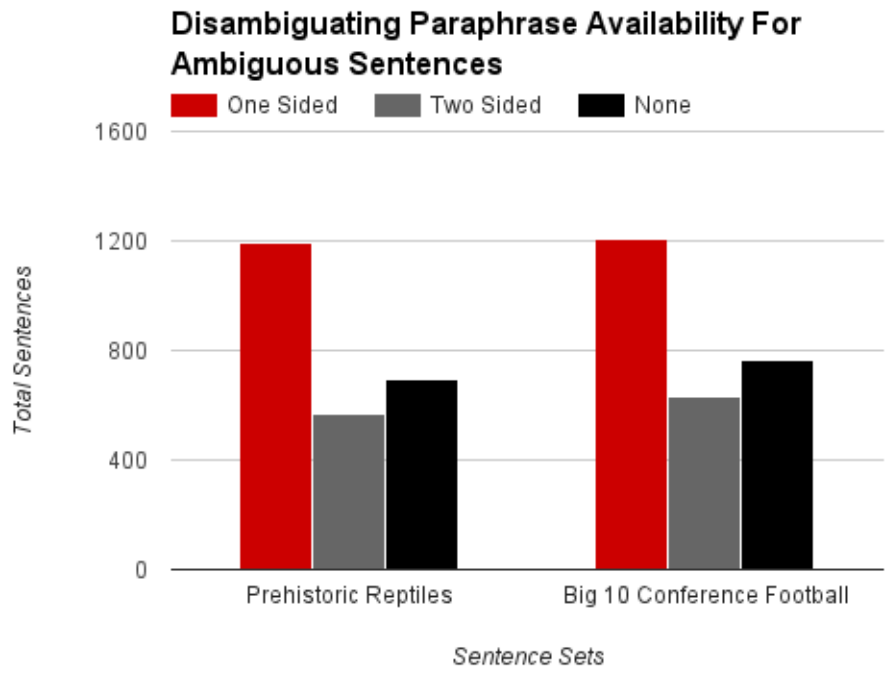

Chart 2: A breakdown of the number of sentences per corpus which have one sided paraphrases, two sided paraphrases, or no paraphrases at all generated for them.

Focusing for a moment on the paraphrases themselves, it is important to examine the distribution of the available paraphrase generated by the strategies described in section 3. A total of 1,399 two sided paraphrases were generated (649 *Prehistoric Reptiles*, 750 *Big 10 Conference Football*) and 2,520 one sided paraphrases were generated (1,262 *Prehistoric Reptiles*, 1,258 *Big 10 Conference Football*). Of the 1,399 two sided paraphrases: 657 are reversals, 260 are cleft rewrites, 135 are passive rewrites,

and 347 are coordination rewrites. Of the 2,520 one sided paraphrases: 2,184 are reversals, 62 are cleft rewrites, 39 are passive rewrites, and 235 are coordination rewrites.

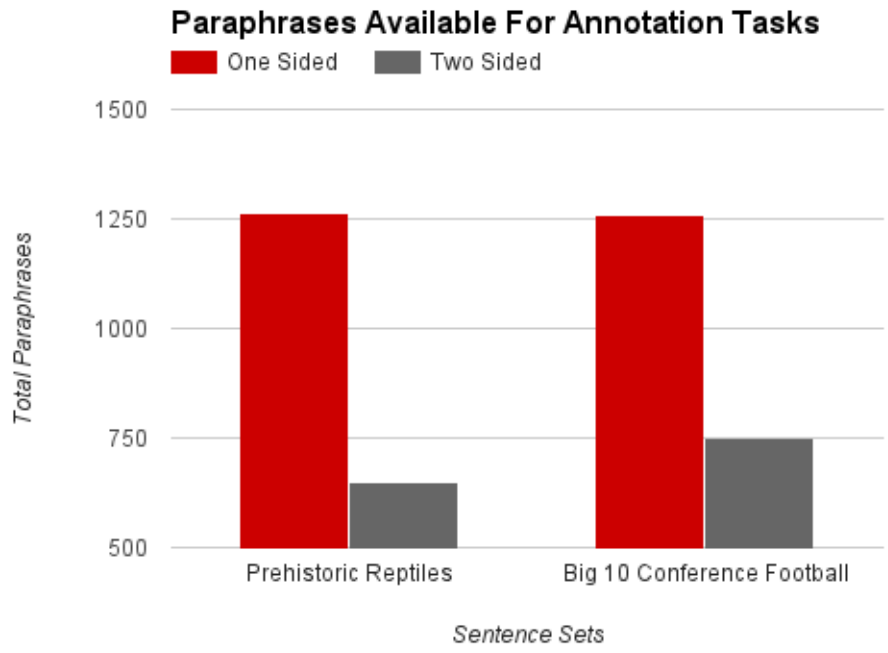

Chart 3: A depiction of the overall number paraphrases generated for each corpus

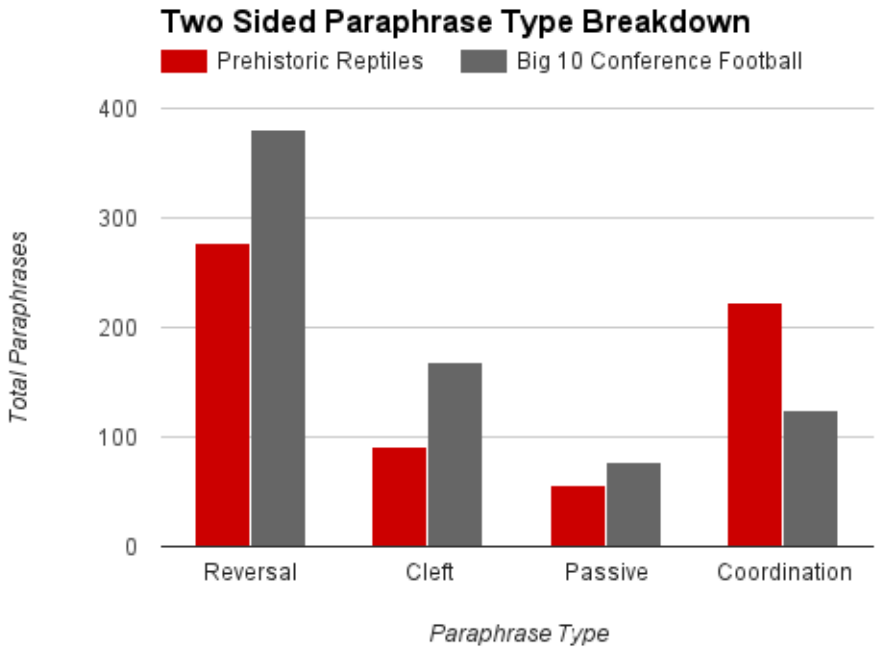

Chart 4: A distribution of the available paraphrases by paraphrase type for two sided paraphrases

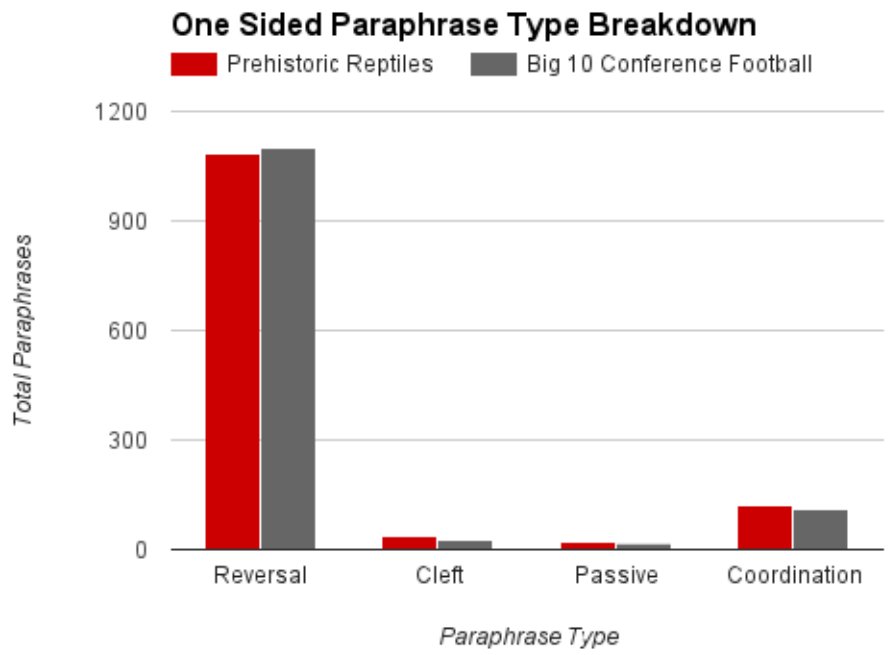

Chart 5: A distribution of the available paraphrases by paraphrase type for one sided paraphrases

Returning to the focus on sentences, we observed the following with regard to reversal and rewrite paraphrases: 650 sentences (277 *Prehistoric Reptiles*, 124 *Big 10 Conference Football*) have a two sided reversal, 2,554 sentences (1,276 *Prehistoric Reptiles*, 1278 *Big 10 Conference Football*) have a one sided reversal, and 1,852 sentences (905 *Prehistoric Reptiles*, 947 *Big 10 Conference Football*) have no reversals at all.

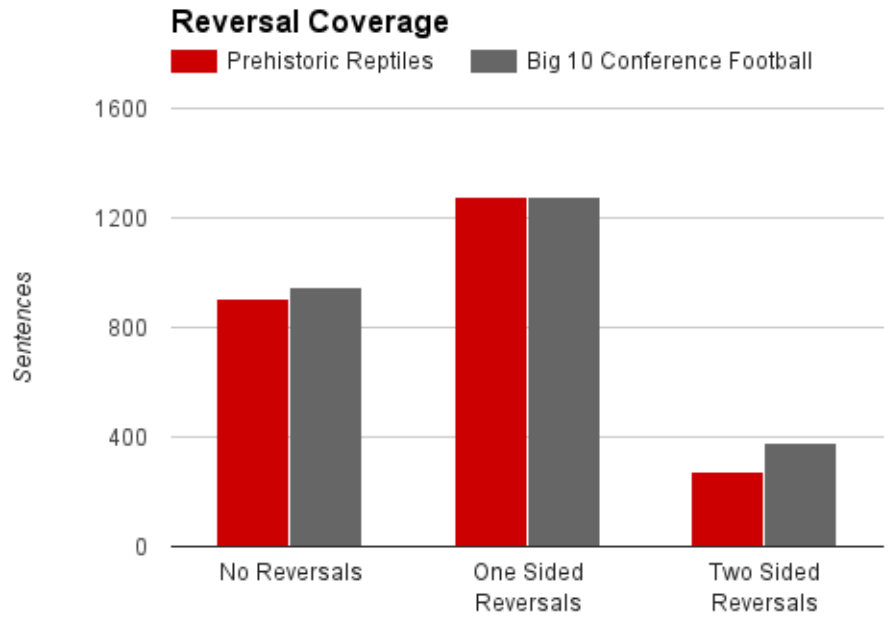

Chart 6: The coverage of reversals for sentences in the corpora

Of those sentences which either had no reversal generated, or a one sided reversal generated, 549 sentences (294 *Prehistoric Reptiles*, 255 *Big 10 Conference Football*) had a two sided rewrite. Of the 1,852 sentences with no reversals generated, 243 sentences (128 *Prehistoric Reptiles*, 115 *Big 10 Conference Football*) had a two sided rewrite generated and 215 sentences (107 *Prehistoric Reptiles*, 108 *Big 10 Conference Football*) had a one sided rewrite generated. Of the 2,554 sentences with one sided reversals generated, 306 sentences (166 *Prehistoric Reptiles*, 140 *Big 10 Conference Football*) had a two-sided rewrite generated.

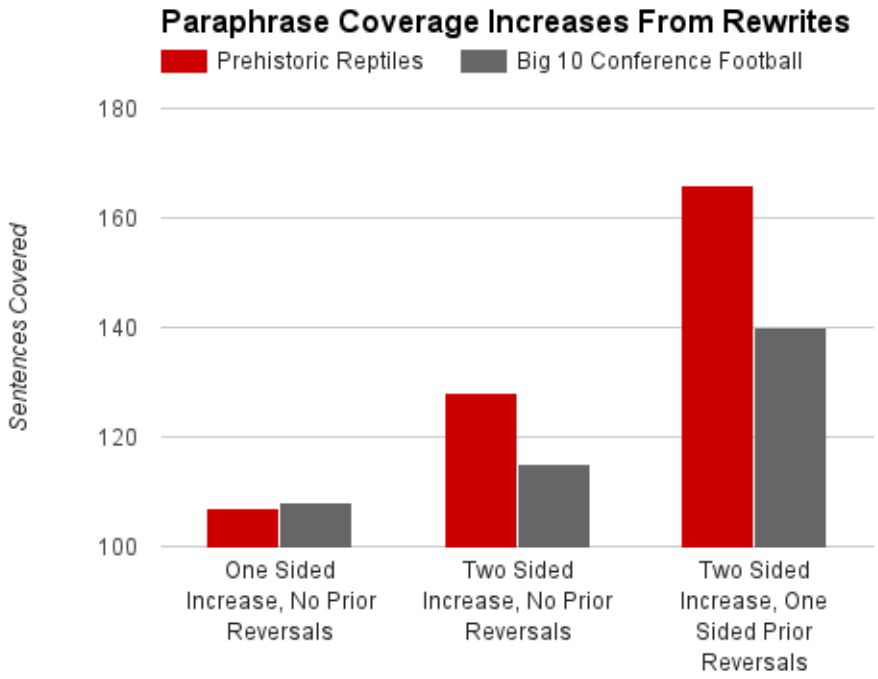

Chart 7: Rewrite coverage increases for sentences with either no prior reversals generated, or at most, one sided reversals generated

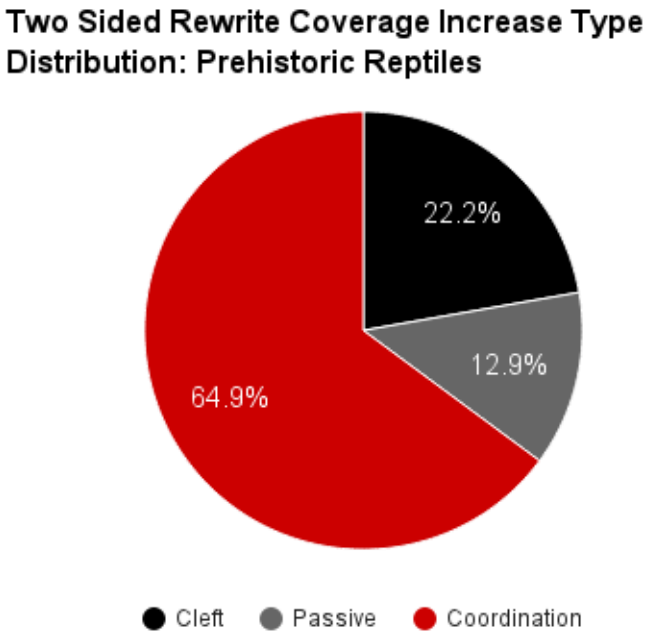

Chart 8: A distribution of the rewrite strategy types in the cases of *Prehistoric Reptiles* sentences with at most a one sided reversal

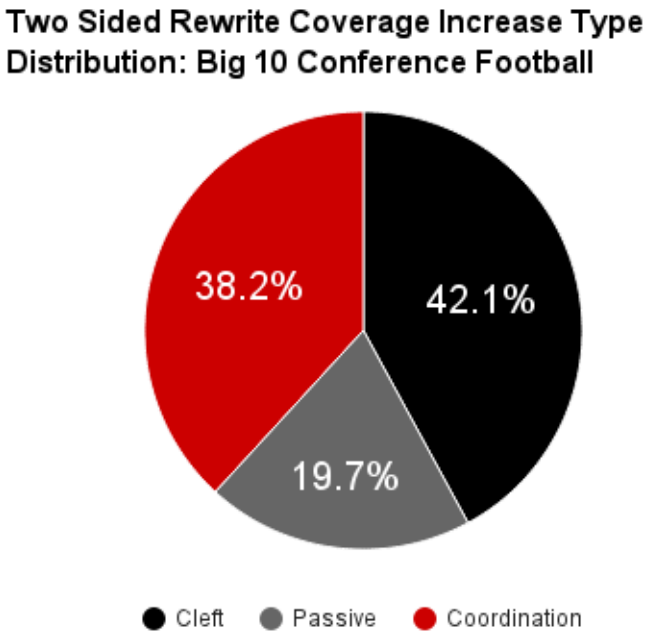

Chart 9: A distribution of the rewrite strategy types in the cases of *Big 10 Conference Football* sentences with at most a one sided reversal

6 Discussion and Suggestions for Future Work

The data that we have gathered is encouraging for our system for a number of reasons. The large number of sentences which were deemed to be unambiguous by our system, as seen in Chart 1 from the previous section, is an indication that there were not meaningful differences between the parses of these sentences and that, despite not being trained on open domain data, the OpenCCG parser may perform well on these sentences (or at least is reasonably confident about its incorrect parses, if it does not perform well).

It is also encouraging that a reasonably high percentage of the ambiguous sentences have paraphrases generated for them, as seen in Chart 2 from the previous section, though a huge majority of the paraphrases generated are one sided paraphrases. This is not necessarily a bad result however, as it is possible to still use a one sided paraphrase in a meaning similarity judgement task. Rather than presenting a user with two paraphrases for an ambiguous sentence, a single paraphrase is presented and a user may be prompted to decide whether or not this paraphrase has the same interpretation as the original sentence. An affirmative choice in this case this may still present an opportunity to learn the parse tree which generated that paraphrase in parser retraining.

So why are there so many one sided paraphrases? It is worth noting that because our methods involve moving down an n-best list of realizations for paraphrases, the paraphrases can become awkward and less fluent for a reader, human or otherwise. For a parser, less fluency may mangle the interpretation which, in the verification phrase for reversals, will cause the paraphrase to get filtered. It is also possible that many of the paraphrases simply were not capable of breaking up the ambiguous span of the sentence

and so also fail the verification phase.

Another encouraging aspect of this data is the significant number of rewrites which increased our coverage of paraphrases for ambiguous sentences.

Something to note, based on the information in Chart 8 and Chart 9, is the higher frequency of coordination rewrites compared to the other rewrite strategies. This does not necessarily suggest that the OpenCCG realizer is weak in terms of its ability to generate disambiguating realizations in the presence of coordination ambiguities. Rather, this simply shows that the realizer prefers fluency over verbosity, which is something that we must force if an unambiguous paraphrase is to be generated. It is also somewhat surprising that the cleft strategy tends to generate more paraphrases than the passivization strategy, given that they are built to disambiguate the same type of ambiguity, PP attachment. It would be worthwhile to explore other rewrite strategies that would further increase our coverage paraphrases for ambiguous sentences

It is not yet clear what the intersection of the words appearing in the unambiguous sentences and the words appearing in the Penn Treebank would be like, but this should be investigated in later studies.

7 Conclusions

We have successfully built a system for generating disambiguating paraphrases which we hope proves useful, as our initial study discussed in section 3 suggests, in collecting large numbers of similarity judgement annotations through crowdsourcing. Our system, while not able to find ambiguities for every sentence in open domain corpora, is able to successfully generate a significant number of disambiguating paraphrases for the ambiguities it does identify.

8 References

Mukund Jha, Jacob Andreas, Kapil Thadani, Sara Rosenthal, and Kathleen McKeown, (2010). Corpus creation for new genres: A crowdsourced approach to pp attachment, In: Workshop on Creating Speech and Text Language Data with Amazon's Mechanical Turk at NAACL-HLT, Los Angeles, California

Mark Steedman, (2000). *The Syntactic Process*. The MIT Press, Cambridge, MA

Mark Steedman and Jason Baldridge, (2011). Combinatory Categorial Grammer, In: Non-Transformational Syntax: Formal and Explicit Models of Grammar, Robert Borsley and Kersti Börjars, Wiley-Blackwell, pp. 181-224.

Julia Hockenmaier and Mark Steedman, (2007). CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank, In: Computational Linguistics, 33:3, pp. 355-396.

Stephen A. Boxwell and Michael White, (2008). Projecting Propbank Roles onto the CCGbank, In: Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC-08).

Martha Palmer, Dan Gildea, Paul Kingsbury, (2005). The Proposition Bank: An Annotated Corpus of Semantic Roles, In: Computational Linguistics, 31:1, pp. 71-105.

Michael White, (2006). Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar, In: Research on Language and Computation, 4:1, pp. 39–75.

Michael White and Rajakrishnan Rajkumar, (2009). Perceptron Reranking for CCG Realization, In: Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2009). Singapore. pp. 410-419.

Marie-Catherine de Marneffe and Christopher D. Manning, (2008). Stanford typed dependencies manual, Technical report, Stanford University.

Manjuan Duan and Michael White, (2014). That's Not What I Meant! Using Parsers to Avoid Structural Ambiguities in Generated Text, In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics. Baltimore, Maryland. pp. 413-423

Appendix

Item A

Corpus	Articles Downloaded
Prehistoric Reptiles	1965
Big 10 Conference Football	2804

Item B: Chart 1: The number of sentences which are ambiguous vs. unambiguous across two corpora, including the same comparison for the subsets of unique sentences from the same corpora.)

	Prehistoric Reptiles	Unique Prehistoric Reptiles	Big 10 Conference Football	Unique Big 10 Conference Football
Ambiguous	2534	2458	3593	2605
Unambiguous	3991	3877	6979	5174

Item C: Chart 2: A breakdown of the number of sentences per corpus which have one sided paraphrases, two sided paraphrases, or no paraphrases at all generated for them.

Paraphrases	Prehistoric Reptiles: # of Sentences	Big 10 Conference Football: # of Sentences
One Sided	1191	1208
Two Sided	571	635
None	696	762

Item D: Chart 3: A depiction of the overall number paraphrases generated for each corpus

Paraphrases	Prehistoric Reptiles: # of Paraphrases	Big 10 Conference Football: # of Paraphrases
One Sided	1262	1258
Two Sided	649	750

Item E: Chart 5: A distribution of the available paraphrases by paraphrase type for one sided paraphrases

Paraphrase Type	Prehistoric Reptiles: # of One Sided	Big 10 Conference Football: # of One Sided
Reversal	1084	1100
Cleft	35	27
Passive	21	18
Coordination	122	113

Item F: Chart 4: A distribution of the available paraphrases by paraphrase type for two sided paraphrases

Paraphrase Type	Prehistoric Reptiles: # of Two Sided	Big 10 Conference Football: # of Two Sided
Reversal	277	380
Cleft	92	168
Passive	57	78
Coordination	223	124

Item G: Chart 6: The coverage of reversals for sentences in the corpora

	Prehistoric Reptiles: # of Sentences	Big 10 Conference Football: # of Sentences
No Reversals	905	947
One Sided Reversals	1276	1278
Two Sided Reversals	277	380

Item H: Chart 7: Rewrite coverage increases for sentences with either no prior reversals generated, or at most, one sided reversals generated

Coverage Increase Label	Prehistoric Reptiles: # of Sentences	Big 10 Conference Football: # of Sentences
One Sided Increase, No Prior Reversals	107	108
Two Sided Increase, No Prior Reversals	128	115
Two Sided Increase, One Sided Prior Reversals	166	140

Item I: Chart 8: A distribution of the rewrite strategy types in the cases of *Prehistoric Reptiles* sentences with at most a one sided reversal

Rewrite Type	Prehistoric Reptiles: # of Paraphrases
Cleft	74
Passive	43
Coordination	216

Item J: Chart 9: A distribution of the rewrite strategy types in the cases of *Big 10 Conference Football* sentences with at most a one sided reversal

Rewrite Type	Big 10 Conference Football: # of Paraphrases
Cleft	130
Passive	61
Coordination	118