Automated Generation of Questions from Factual, Natural Language Sentences

Tyler Yates

April 29, 2016

Abstract

This paper describes methods to examine factual, natural language sentences in order to generate factual, natural language questions whose answers are found in the sentences. These methods rely on rules that look for patterns in the syntactic structure, named entities, and grammatical relations of a sentence. The extraction of these pieces of information from a sentence is not the focus of this paper. Rather, this paper examines how these pieces of information can be used for the purpose of factual, natural language question generation. This paper also presents an implementation of these methods in the form of a computer program called Specimus. This paper describes the implementation of the system and presents an evaluation of the output of the system.
1 Introduction

1.1 Purpose

Automated Question Generation (AQG) is an important area of research in the fields of Artificial Intelligence (AI) and Natural Language Processing (NLP) and is a key part of help systems and learning environments [1]. The ability to generate factual, natural language questions from a corpus of factual, natural language sentences would allow Automated Question Answering (AQA) systems like IBM Watson to perform self-training. This would remove the need for humans to manually define the ground-truth by entering questions and marking the corresponding answers from the corpus [2]. Tutoring systems could also benefit by allowing end users to define the study material instead of human experts who manually write questions for specific topics that the tutoring system supports.

The difficulty in determining what questions are answered by a given sentence, however, is enormous due to the complex and irregular structure of natural language. This difficulty is compounded by the fact that a single sentence may hold the answer to several questions. Additionally, these answers may be contained within complicated syntactic constructions, making it difficult to automate the extraction of the information.

This paper describes a computer system called Specimus which examines factual, natural language sentences in order to generate factual, natural language questions whose answers are found in the sentences. Specimus tackles the problem of determining what questions are answered by a given sentence by first simplifying the given sentence using a set of simplification rules. Once the sentence has been simplified, Specimus applies rules to transform the simplified sentences into grammatical questions. Both sets of rules look for patterns in the syntactic structure, named entities, and grammatical relations of a sentence. To obtain these pieces of information, the system relies on the parser and Named Entity Recognizer (NER) from Stanford’s CoreNLP toolkit [3].
1.2 Definitions

The version of Stanford CoreNLP used by this paper is v3.6.0 released 2015-12-09. CoreNLP can be downloaded from the following URL:
http://stanfordnlp.github.io/CoreNLP/history.html

The version of SimpleNLG used by this paper is v4.4.3 released 2014-8-20. SimpleNLG can be downloaded from the following URL:
https://github.com/simplenlg/simplenlg/releases

A named entity is a word or phrase that has been classified into pre-defined, broad semantic categories. Stanford’s NER recognizes the following categories:

- PERSON (e.g. “George Washington”)
- ORGANIZATION (e.g. “Ford Motor Company”)
- LOCATION (e.g. “Virginia”)
- DATE (e.g. “June 8, 1778”)

A phrase structure tree is a representation of the syntactic structure of a sentence. This representation describes the phrases in the sentence and the hierarchy of those phrases. The leaves of a phrase structure tree are the actual words of the sentence arranged in the same order they appear in the sentence. Each leaf node has exactly one parent node which describes the Part of Speech (POS) of the word. All other nodes in the tree describe the phrases of the sentence. The root of a phrase structure tree for a whole sentence is the node ‘S’ which stands for ‘Sentence’. A phrase structure tree for the sentence “John is a man” is given below.
In this paper, phrase structure trees will follow the conventions of the Stanford CoreNLP toolkit, which are outlined in the Penn Treebank Project [4].

The Penn Treebank Project uses acronyms for phrase and POS tags. Some examples are NP (Noun Phrase) and VBZ (3rd person singular present verb). A complete listing of the acronyms used by the Penn Treebank Project can be found at the following URL: http://web.mit.edu/6.863/www/PennTreebankTags.html

The grammatical relations mentioned in this paper are defined in the UniversalEnglishGrammaticalRelations class in the Stanford CoreNLP toolkit [5]. Every grammatical relation is a binary relationship between two words in the sentence. The first word is called the governor and the second word is called the dependent [6]. For the sentence

| Washington, the first president, was born in Virginia. |

CoreNLP identifies several grammatical relations. One of these relations is appos(Washington, president) which means that “Washington” is the governor of an appositive relation with the dependent “president”.

Figure 1 – The phrase structure tree for the sentence "John is a man" as constructed by the CoreNLP parser.
1.3 Implementation

The sub-systems in Specimus are described at length in the sections below. Each section contains an implementation sub-section describing how each sub-system was implemented. For more detail, the entire source of the Specimus implementation can be accessed at the following URL: https://github.com/Tyler-Yates/TuringThesis.

1.4 Acknowledgements

I would like to thank the many people who have helped me throughout the process of completing this thesis. The first is my advisor, Dr. Bruce Porter, who has provided outstanding mentorship and guidance since the inception of this project. I would also like to thank Samantha Huang from the department of linguistics. Her presence at every weekly thesis meeting and her knowledge in the field of linguistics greatly benefitted the breadth and depth of this thesis. Additionally, I would like to thank the Department of Computer Science, specifically the Turing Scholars Program and its director Dr. Calvin Lin, for providing the knowledge and experience that enabled me to complete this thesis.

1.5 Outline

The rest of the paper is organized as follows: The second section describes related work. The third section describes text generation. The fourth section describes the sentence simplification system. The fifth section describes a structured language created to standardize the process of writing rules for the question generation system. This language was written for Specimus but can be used outside of the system as well. The sixth section describes the question generation system. The seventh section describes the test set used to measure progress during the implementation of Specimus. Additionally, the seventh section discusses results obtained by asking human judges to rate the output of Specimus on an evaluation set of factual, natural language sentences. The eighth section describes limitations of Specimus, CoreNLP, and SimpleNLG. The ninth section describes future work.
2 Related Work

Heilman produced a system that addresses many of the problems facing AQG [7]. His focus, however, was on generating questions for the purpose of educational instruction. As such, the focus of his paper was on quality over quantity. Thus, the sentence simplification system outlined in the paper simply removes some complex syntactic constructions, which could be used to generate questions. For AQA systems like IBM’s Watson, this behavior is suboptimal. The goal of defining the ground-truth in an AQA system is to cover the corpus material as widely as possible, so the quality of generated questions can be less of a concern. This paper outlines ways to modify Heilman’s sentence simplification system in order to potentially generate more questions from the same sentences. Additionally, the sentence simplification system outlined in Heilman is described at a very high level. Heilman’s pseudo-code algorithms do not describe how the syntactic constructions are identified in a given sentence, making it difficult to reproduce an actual implementation of his work. This paper details how each syntactic construction is identified in a sentence, and how each construction can potentially be used for generating additional simplified sentences.

Ali, Chali, and Hasan produced a system to generate questions from complex sentences [1]. Like Heilman, complex sentences are passed through an Elementary Sentence Extraction System (ESES) in order to simplify the parsing of the sentence. The paper does not give concrete information about how the ESES works but it did reinforce the idea that simplification can be beneficial to an AQG system.

Klebanov, Knight, and Marcu define criteria for determining whether a string is simple enough for a computer system to extract information from it easily [8]. Strings that are simple enough are called Easy Access Sentences (EASs). Specifically, an EAS must be a grammatical sentence, have one finite verb, not make any claims that were not in the original text, and have as many named entities as possible. Klebanov, Knight, and Marcu then describe a system that transforms text into EASs in order to make it simpler for information retrieval applications to pull information from the texts. They also provide a high-level overview of their algorithm to transform a given text into EASs. The simplification system outlined in this paper does not use the algorithm outlined in Klebanov, Knight, and Marcu’s paper, but rather uses the EAS criteria to create additional simplifications that augment Heilman’s simplification system.
Graesser, Rus, and Cai defined sixteen categories of questions and empirically analyzed the difficulty in answering each kind of question [9]. The three levels of difficulty defined in the paper were simple/shallow, intermediate, and complex/deep. For the purpose of defining what kinds of questions Specimus would produce, only the simple/shallow question categories were considered. Graesser, Rus, and Cai define four simple/shallow question categories:

1. Verification – yes/no answer
2. Disjunctive – Is X, Y, or Z the case?
4. Example – What is an example of X?

Specimus does not generate verification questions because of the nature of the answer. Verification questions expect a single word response, “yes” or “no”, but these two words do not necessarily show up in the original sentence. Specimus is only able to link a question with the sentence that it was generated from, so this question category is not well suited to the system.

Specimus does not generate disjunctive questions because the corpus for the system (individual factual sentences) might not lend itself to disjunctive questions. For instance, generating a disjunctive question for the sentence:

George Washington was the first president of the United States.

would require information from outside the sentence. For example, one might ask the following disjunctive question about the sentence above:

Was George Washington the first president of the United States, Canada, or Great Britain?

“Canada” and “Great Britain” are never mentioned in the original sentence and require knowledge on the part of the question asker to put them into the question. Specimus does not have this knowledge so it cannot generate questions of this type.

Specimus does generate concept completion questions because concept completion questions are applicable to a wide range of factual sentences, often have non-trivial answers, and can be posed in a way that does not require outside information to generate them.
Specimus does not generate example questions because of the difficulty in determining what situations warrant an example question. For example, the sentence:

| The Bible is the best-selling book of all time. |

does not lend itself to an example question. One would not ask the question:

| What is an example of the best-selling book at all time? |

because of the definite article in the original sentence. There is only one best-selling book of all time so it does not make sense to ask for an example of it.

# 3 Generation & Alteration of Text

## 3.1 Overview

Both sentence simplification and question creation require the generation of new text and the alteration of existing text.

Generation is the synthesis of new text to explicitly describe certain semantic information. As humans, we internally perform generation as we read. Parentheticals are a good example of this, as readers are often able to draw complex semantic information from limited syntax.

| George Washington (1732 – 1799) was the first president. |

Readers are able to understand that the above sentence describes the year of both Washington’s birth and death even though the verbs “born” and “died” never appear in the sentence. Specimus uses generation to concretely state the semantic meaning of a sentence. For the sentence above, Specimus would generate the sentence:

| George Washington was born in 1732. |

This new sentence requires the construction of a brand new VP (“born in 1732”) that was not contained in the original sentence.

Existing text can be altered to take on new lexical features (such as passive voice or past tense). Alteration is used in Specimus to attain grammatical correctness when
transforming a sentence into a question. For instance, a VP may need to be changed to passive voice during simplification.

```
George Washington, born in 1732, was the first president.
```

It would be grammatically incorrect to simply combine the NP “George Washington” and the VP “born in 1732”.

```
George Washington born in 1732.
```

Changing the VP to passive voice fixes the grammatical incorrectness.

```
George Washington was born in 1732.
```

### 3.2 Implementation

Synthesis of new text is hard-coded into the system. For example, the birth-death-date parenthetical case mentioned above simply uses the string literals “was born in” and “died in” to generate the simplified sentences.

```
String birthQuestion = npString + "was born in" + birthDate;
String deathQuestion = npString + "died in" + deathDate;
```

Alteration of existing text is accomplished using the SimpleNLG system [10]. SimpleNLG is a realization engine designed for the English language. The engine supports realization of text with defined lexical features. For instance, SimpleNLG supports the realization of passive voice VPs, allowing the VP “born in 1732” to be turned into “was born in 1732”. The following code is an example of turning a VP into a past-tense, passive-voice string:

```
VPPhraseSpec vpPhraseSpec = nlgFactory.createVerbPhrase(lemmatizedVpString);
vpPhraseSpec.setFeature(Feature.PASSIVE, true);
vpPhraseSpec.setFeature(Feature.TENSE, Tense.PAST);
return realiser.realise(vpPhraseSpec).toString();
```
4 Sentence Simplification

4.1 Overview

Factual sentences often contain multiple pieces of information within various syntactic constructions [7]. By extracting these syntactic constructions into separate sentences, Specimus is able to create sentences that are syntactically simpler. This in turn allows us to create question generation rules that are simpler and more correct. As such, all sentences that Specimus takes in as input are passed through a sentence simplification system first. The resulting simplified sentences outputted by the sentence simplification system are then passed on to the question generation system.

The basis for the sentence simplification system in Specimus comes from Section 3.2.1 of Heilman, which describes several syntactic constructions that can be extracted from a complex sentence [7]. Section 2.3.1 of Heilman also describes how conjunctions can be split apart to simplify sentences. The sentence simplification performs some additional simplifications in addition to those outlined by Heilman. These additional simplifications were inspired by the criteria outlined in Klebanov, Knight, and Marcu’s paper [8].

4.2 Heilman’s Syntactic Constructions

Section 3.2.1 of Heilman outlines six syntactic constructions recognized by his sentence simplification system [7].

- **non-restrictive appositives** (e.g., Jefferson, the third U.S. President, …)
- **non-restrictive relative clauses** (e.g., Jefferson, who was the third U.S. President, …)
- **parentheticals** (e.g., Jefferson (1743–1826) …)
- **participial modifiers of noun phrases** (e.g., Jefferson, being the third U.S. President, …)
- **verb phrase modifiers offset by commas** (e.g., Jefferson studied many subjects, like the artist DaVinci. …)
The following sections describe how the sentence simplification system identifies and handles each of the syntactic constructions listed above.

4.2.1 Non-restrictive Appositives

Appositives are identified by the Stanford CoreNLP parser with the grammatical relation appos. The system first searches for all appos relations in the given sentence. Next, the system checks to see if the appositive is non-restrictive. The dependent of non-restrictive appositives is always bounded by commas or a comma and the end of the sentence [11]. Thus, for each appositive relation, the dependent of the relation is identified. The system then checks to see if the dependent is bounded by commas or a comma and the end of the sentence and that the governor of the relation is not within the bound. If these checks pass, the bounded part of the sentence is extracted and used to generate a new simplified sentence.

For example, the following sentence has two non-restrictive appositives:

George Washington, the first president, was friends with Ben Franklin, a noted inventor.

The two appositive relations are appos(Washington, president) and appos(Franklin, inventor). Both of them are non-restrictive appositives because the dependent is bounded by commas or a comma and the end of the sentence and the governor is not within the bound. As such, the system removes both non-restrictive appositives from the sentence to yield the simplified sentence below.

George Washington was friends with Ben Franklin.

Specimus also generates a simplified sentence for each appositive that is removed. For the relation appos(Washington, president) Specimus generates the following simplified sentence:

---

1 When determining whether syntactic constructs are bounded by commas, the sentence simplification system excludes commas that are within a date or between words tagged as locations.
George Washington was the first president.

This sentence is generated by linking the noun-phrase (NP) containing the governor of the relation (“George Washington”) and the NP containing the dependent (“the first president”) with either the past or present tense form of the verb “be”. To determine the tense that “be” should take, the sentence simplification system examines the POS tags for each word in the sentence and uses the past tense if it finds a VDB (past-tense verb) or a VBN (past participle) in the sentence. Otherwise, the present tense is used.

### 4.2.2 Non-restrictive Relative Clauses

Relative clauses are very similar to appositives. The only difference in terms of identification is that the Stanford CoreNLP parser identifies relative clauses with the grammatical relation *acl:relcl*. The same rules for detecting whether an appositive is non-restrictive apply to relative clauses as well. As with before, the non-restrictive relative clauses that are removed from the original sentence are used to generate a new simplified sentence.

For example, the following sentence has a non-restrictive relative clause:

Jefferson, who was the third president, was born in 1743.

The relative clause relation is *acl:relcl(Jefferson, president)*. It is a non-restrictive relative clause because the dependent is bounded by commas and the governor is not within the bound. As such, the system removes the non-restrictive relative clause to yield the simplified sentence below.

Jefferson was born in 1743.

Specimus also generates a simplified sentence for each relative clause that is removed. For the relation *acl:relcl(Jefferson, president)* Specimus generates the following simplified sentence:

Jefferson was the third president.

This sentence is generated by linking the noun-phrase (NP) containing the governor of the relation (“Jefferson”) and the NP containing the dependent (“who was the third president”) with either the past or present tense form of “be”. The NP containing the dependent will always start with a relative pronoun or adverb [11]. For the sentence
above, the first word is “who”. This word is deleted by the system when generating the simplified sentence in order to make the sentence grammatically correct.

### 4.2.3 Parentheticals

Parentheticals can be identified without the use of grammatical relations, since they are always bounded by open- and closed-parentheses. Thus, the system identifies parentheticals by searching through each leaf in the phrase structure tree of the sentence. Punctuation is always its own leaf in the tree, so identifying a parenthetical is as simple as finding an open-parenthesis leaf and a close-parenthesis leaf. The Stanford CoreNLP parser converts “(“ and “)” into “-LRB-“ and “-RRB-“ respectively in the phrase structure tree in order to avoid confusion with parse brackets [12]. All leaves in-between these two leaves are the content of the parenthetical since leaves of the phrase structure tree are in sentence order.

For example, the following sentence has one parenthetical which contains two years separated by a dash:

```
George Washington (1732 – 1799) was the first President.
```

The phrase structure tree for the sentence is given below.
Figure 2 – The phrase structure tree for the sentence “George Washington (1732 – 1799) was the first President” as constructed by the CoreNLP parser.

Removing the parenthetical yields the following simplified sentence:

| George Washington was the first President. |

Heilman’s system did not use any further processing to try and create additional simplified sentences from the parenthetical. Heilman explained that doing so “would likely require components to identify time expressions and to generate appropriately formatted output.” [7] The simplification system relies on the Stanford CoreNLP NER to perform this identification.

The first check the system performs is on the word immediately before the parenthetical in the sentence. If this word is labelled as a PERSON by CoreNLP’s NER, the system examines the contents of the parenthetical. If the parenthetical consists of two DATEs, it assumes that the dates are in reference to the person’s birth and death, since this is a very common convention in English. Two simplified sentences are then generated using the NP containing the PERSON word and the two DATE NPs. For the sentence above, the PERSON NP is “George Washington” and the DATE NPs are “1732” and “1799”. Thus, two simplified sentences can be generated:
George Washington was born in 1732.

George Washington died in 1799.

Another common convention in English is to include a single verb phrase in a parenthetical. For example:

Gerald Ford (born Leslie Lynch King) was an American politician.

The system identifies this construct by looking at the POS of the first word in the parenthetical. If the first word is a verb\(^2\), it then finds the VP containing that verb by moving up towards the root of the phrase structure tree from the leaf representing the verb. The system then finds the NP immediately preceding the parenthetical. By combining the NP and VP together, the system generates a new simplified sentence:

Gerald Ford was born Leslie Lynch King.

SimpleNLG handles the alteration of the VP to be grammatically correct.

4.2.4 Participial Modifiers of Noun Phrases

Participial phrases are verb phrases that begin with a participle. Participles are tagged by the CoreNLP parser as either VBG or VBN for present and past tense respectively. Thus, identifying a participial phrase is as simple as looking for a VP with the verb tagged as VBG or VBN by the CoreNLP parser. The simplification system will only simplify participial phrases that are offset by commas in order to preserve the semantic meaning of the sentence.

Participial phrases are often used to modify noun phrases. The sentence below is an example of a participial phrase that modifies a noun phrase.

The horse **trotting up to the fence** loves to run.

The phrase “trotting up to the fence” is a participial phrase which describes the horse. In effect, the participial phrase is serving as an adjective. For this sentence, CoreNLP gives “trotting” the POS tag VBG. We can determine what the participial phrase is modifying by looking for the grammatical relation with “trotting” as its dependent. For this sentence, CoreNLP identifies the relation acl(horse, trotting).

\(^2\) There are several POS tags for verbs, but all start with the string “VB”.
However, participial phrases can also describe verb phrases.

| The woman improved her health **running five miles a day**. |

In the above sentence, “running five miles a day” describes the verb “improved”. In this regard, the participial phrase is serving as an adverb. CoreNLP identifies the relation `xcomp(improved, running)` for the above sentence.

Heilman uses this distinction to generate new simplified sentences. When a participial phrase modifies a noun, his system generates a new sentence using just the noun phrase and the participial phrase. For instance, examine the following sentence:

| Jefferson, being the third president, loved white houses. |

Extracting the participial phrase and the noun it modifies can create a new simplified sentence:

| Jefferson was the third president. |

After passing many sentences with participial phrases to CoreNLP, it was discovered that the majority of the participial phrases were identified as modifiers of verb phrases and thus not in the scope of Heilman’s syntactic construction. Even examples of participial phrases that did modify noun phrases were tagged as verb phrase modifiers (see section 8 Limitations for more information). As such, Specimus just removes all participial phrases that are offset by commas without generating additional sentences.

### 4.2.5 Verb Phrase Modifiers Offset by Commas

Heilman does not specify which specific modifiers he looks for when identifying this syntactic construct. The example sentence provided by Heilman was used to try and determine how CoreNLP identifies the verb phrase modifier.

| Jefferson studied many subjects, **like the artist Da Vinci**. |

Passing the example sentence to CoreNLP yielded the grammatical relation `nmod(studied, Vinci)`. The relation `nmod` refers to a nominal modifier, which, in English, is a prepositional complement [5].
Since prepositional complements can modify more than just verbs, Specimus first checks to see if the governor of the nmod relation is actually a verb. If it is, it then checks to see if the dependent of the relation is bounded by commas or the end of the sentence. If so, the system removes the bounded part and returns the new sentence. For example, the sentence above would be simplified to:

```
Jefferson studied many subjects.
```

No way could be determined in order to use the information in the verb phrase modifier to generate an additional simplified sentence. As such, the simplification system simply removes the bounded modifier without generating additional sentences.

### 4.2.6 Modifiers that Precede the Subject

Of the six syntactic constructions outlined by Heilman, this is the broadest. There are many classes of modifiers in English, and Heilman does not outline which modifiers his system looks for when identifying this syntactic construction. By examining the corpus of sentences from the test set system, it was determined that prepositional phrases occasionally are used before their subject. For instance, some PPs occur at the start of the sentence:

```
During his presidency, George Washington lived in New York.
```

Additionally, PPs can occur between an NP and VP:

```
George Washington, during his presidency, lived in New York.
```

In both cases, the PP can be moved to the end of the sentence while still preserving semantic meaning. Additionally, PPs that occur before their subject are always bounded with commas or a comma and the beginning of the sentence [11]. By moving the PP to the end of the sentence, the punctuation and the phrase structure tree are simplified:

```
George Washington lived in New York during his presidency.
```

To perform this simplification, the system looks for a PP node bounded by commas or the beginning of the sentence and a comma. The system then removes the PP and the comma(s) around the PP. The PP is then moved to the end of the sentence.
There are more modifiers that can potentially occur before the subject, but PPs are the only ones that the simplification system looks for.

4.3 Conjunctions

4.3.1 Verb Phrases

Heilman outlines conjunctions of verb phrases as another area for simplification. Verb phrases conjoined by a coordinating conjunction can be split into new sentences to simplify the phrase structure tree of the sentence.

"Dark Souls was developed by FromSoftware and published by Sony."

The sentence above has two conjoined verb phrases: “developed by FromSoftware” and “published by Sony”. Both verb phrases share the same subject “Dark Souls”. When constructing the phrase structure tree for a sentence containing conjoined verb phrases, the CoreNLP parser creates a VP node with a coordinating conjunction (CC) and each of the conjoined VPs as its children.
Figure 3 – The phrase structure tree for the sentence “Dark Souls was developed by FromSoftware and published by Sony” as constructed by the CoreNLP parser.

Care must be taken when performing this simplification in order to obey Klebanov, Knight, and Marcu’s criterion that simplified sentences not make any claims that were not in the original text. Specifically, Specimus only performs conjoined verb simplification when the CC is the word “and”. For instance, examine the following sentence:

I will run to the park or walk the dog.

The VPs “run to the park” and “walk the dog” are conjoined by the CC “or”, so without the previously mentioned restriction Specimus would generate two simplified sentences:

I will run to the park.
These simplified sentences are not necessarily correct, however, since “or” is exclusive. Therefore, only one of these actions could have happened. Had the speaker used “and” as the CC, the simplification would be correct.

Thus, conjoined VPs eligible for simplification can be identified by looking at each VP node in the phrase structure tree and checking to see if it has VP children and a CC child with the word “and” Once the parent VP node of the conjoined VPs has been identified, the system can replace the parent VP node with each of its VP children to get simplified sentences. For the sentence above, this yields two simplified sentences:

- Dark Souls was developed by FromSoftware.
- Dark Souls was published by Sony.

These simplified sentences make it easier to generate questions since the phrase structure tree for the simplified sentence is now less complicated than the tree for the original sentence.

Klebanov, Knight, and Marcu’s criterion that an EAS must have only one finite verb was an inspiration for this simplification rule. Though this simplification does not guarantee that the resulting simplified sentences have only a single verb, it does make a substantial impact with regards to reducing the number of verbs per sentence in the test set.

### 4.3.2 Individual Verbs

There are also cases where individual verbs are conjoined.

Dark Souls was developed and published in 2011.

The words “developed” and “published” are past participles which have VBN as their POS tag. Unlike the previous example, these words are not nested under VP children; the parent VP node links directly to their POS tag node.
Figure 4 – The phrase structure tree for the sentence “Dark Souls was developed and published in 2011” as constructed by the CoreNLP parser.

The principle used for the conjoined verb phrases will still work for this case: Each child verb takes its turn as the sole verb under the parent VP. Thus, the system can simply expand the previously outlined verb phrase conjunction simplification algorithm by looking for a VP node with VP or VB__ (any verb POS tag such as VBG, VBN, VBZ, etc.) children in addition to a CC child with the word “and”.

4.3.3 Corner Cases

A caveat to this simplification is the potential for child nodes other than VPs and CCs under the parent VP node. The sentence in Figure 4, for example, has a PP child in addition to the VP children and CC child. In this case, it is desirable for the PP to be attached to each of the conjoined verbs when the system performs simplification since the PP semantically modifies each conjoined part.
To handle this case, the system looks for non-verbal children under the parent VP and inserts them after each verbal child when performing simplification. For the sentence in Figure 3, the system holds onto the PP “in 2011” and attaches it after “developed” and “published” to generate the following simplified sentences.

<table>
<thead>
<tr>
<th>Dark Souls was developed in 2011.</th>
</tr>
</thead>
</table>

| Dark Souls was published in 2011. |

If the system did not handle this case, the system would lose information from the original sentence, effectively deleting entire subtrees from the phrase structure tree.

4.4 Noun Phrase Modifiers Under Copulas

A common structure in English sentences is to have a VP nested under a copula.

| The Hobbit is a book written by J.R.R. Tolkien. |

For this sentence, the VP “written by J.R.R. Tolkien” is nested under the copula VP with the verb “is” and serves to modify the NP “a book”. The phrase structure tree for the sentence is given below.
In order to make an effort towards keeping Klebanov, Knight, and Marcu’s criterion that an EAS must have only one finite verb, the VP nested under the copula is extracted into its own sentence:

<table>
<thead>
<tr>
<th>The Hobbit is a book.</th>
</tr>
</thead>
</table>

| The Hobbit is written by J.R.R. Tolkien. |

This simplification makes it easier for the question generation system to generate good questions because the auxiliary verb “is” is immediately before the verb “written” in the simplified sentence instead of being farther away at the beginning of the original sentence.
Specimus performs this simplification by first looking for a copula VP (a VP starting with the lemma “be”). If it finds a copula VP, it checks for any VP nodes under the copula VP. If it finds a sub-VP, it takes the subject NP of the copula VP and creates a new sentence by combining that NP with the extracted sub-VP. The subject NP of the copula is assumed to be the first NP preceding the VP.

4.5 SBAR

SBAR is defined by the Penn Treebank as “A clause introduced by a (possibly empty) subordinating conjunction.” [13] SBARs can be used to generate new simplified sentences. The system only looks for SBARs that start with a wh-determiner or a wh-pronoun. The following sentence contains an SBAR that can be used by the system.

The Hobbit is a book that has sold 140 million copies.

The phrase structure tree for the sentence is given below.

---

3 Prepositions are ignored when determining whether an SBAR starts with a wh-determiner or a wh-pronoun. For example, the SBAR “in which the book was found” will still be simplified.
The SBAR in the sentence begins with a wh-determiner (“that”) which means the system will generate a new sentence. The new sentence is generated by combining the first NP from the sentence with everything in the SBAR after the wh-determiner or wh-pronoun. For the sentence above, the system would generate the sentence below.

The Hobbit has sold 140 million copies.

Initially, the simplification system would also remove the SBAR from the original sentence. This behavior is not always correct however; as the SBAR can be a restrictive
clause which means that it is integral to the meaning of the sentence. An example of a restrictive SBAR is given below.

| Hero of Belarus is the highest title that can be bestowed on a citizen of Belarus. |

Removing the SBAR from the sentence would lead to the sentence:

| Hero of Belarus is the highest title. |

which is not entailed by the original sentence. Thus, removing SBARs from the original sentence can violate Klebanov, Knight, and Marcu’s criterion that an EAS must not make any claims that were not in the original text. A consistent and simple way to identify whether an SBAR is restrictive or non-restrictive⁴ could not be found. For this reason, SBARs are left in the original sentence.

## 4.6 Dummy Subjects

A dummy subject is a word that occupies the subject position of a verb but has no semantic meaning [11]. In English, “there” and “it” are often used as dummy subjects. The following sentences are examples of dummy subjects.

| It is raining. |

| There will be snow. |

Sentences with dummy subjects are difficult to generate concept completion questions for. Questions that are answered by a sentence containing a dummy subject often start with “Will”, “Is”, or “Was”:

| Is it raining? |

| Will there be snow? |

---

⁴ Conventions do exist for syntactically denoting whether a clause is restrictive or non-restrictive [14] but these conventions did not appear to be used consistently in the test set.
Thus, Specimus does not attempt to generate questions for sentences with dummy subjects. If the simplification system sees a sentence containing a dummy subject, it returns an empty set of simplified sentences in order to prevent questions from being generated.

CoreNLP identifies dummy subjects with the POS tag “EX”. Thus, identifying whether a sentence contains a dummy subject is as simple as looking for the label “EX” in the list of POS tags for the sentence.

4.7 Quoted Text Grouping

Some classes of NPs are often wrapped in quotes. Song titles are one example of this:

```
“All You Need Is Love” is a song by the Beatles.
```

The quotes allow the reader to understand that the words “All You Need Is Love” constitute a single NP. CoreNLP, however, does not possess this understanding. For the sentence above, CoreNLP will identify “Need” and “Is” as verbs. This is not correct and leads to the creation of invalid questions for the sentence.

To work around this, the simplification system looks for phrases surrounded by double quotes and replaces the spaces between each word with underscores. Thus, the sentence above would become:

```
All_You_Need_Is_Love is a song by the Beatles.
```

Transforming the sentence in this way leads to correct parsing in CoreNLP: “All_You_Need_Is_Love” is identified as a single NP by the parser. At the very end of Specimus’ pipeline, all underscores are replaced with spaces to yield correct English text.

4.8 Implementation

The simplifications outlined in the previous section are each implemented in their own Java class using the functions provided by Simple CoreNLP’s Sentence class. The source for each class can be accessed at the following URL: https://github.com/Tyler-Yates/TuringThesis/tree/master/src/main/java/simplification
4.8.1 Phrase Structure Tree

The following code is used to retrieve the root node of the phrase structure tree for a sentence.

```java
final Tree root = sentence.parse();
```

The following code is used to iterate over every node in the phrase structure tree. Each node in the phrase structure tree is a subtree of the full phrase structure tree.

```java
for (int i = 1; i < root.size(); i++) {
    final Tree node = root.getNodeNumber(i);
}
```

The following code retrieves the label of a phrase structure tree node. The label is a string.

```java
tree.label().value()
```

The following code is used to retrieve the parent of a phrase structure tree node and traverse its children.

```java
final Tree parent = node.parent(root);
for (final Tree child : node.getChildrenAsList()) {

}
```

The following code is used to get the leaf nodes of a phrase structure tree node.

```java
for (final Tree leaf : node.getLeaves()) {
}
```

4.8.2 Named Entity Recognition

CoreNLP’s NER works on a per-word basis. As such, it is important to use the head word of a phrase when checking for named entities as the phrase may contain other named entities that are not correct for the phrase as a whole. For instance, given the following NP

| Sigismund of Austria |

CoreNLP’s NER will assign the named entity PERSON to “Sigismund” and LOCATION to “Austria”. Both named entities are correct for their given word but the NP as a whole is referencing the person known as Sigismund of Austria. The head of the NP is “Sigismund” so looking at the named entity assigned to that word gives the
correct named entity (PERSON) for the entire NP. The following code is used to retrieve the head word of a phrase represented by a node in the phrase structure tree. The head finder returns the phrase structure tree node representing the head of the given phrase.

```java
private static final HeadFinder HEAD_FINDER = new CollinsHeadFinder();
HEAD_FINDER.determineHead(tree);
```

The following code is used to retrieve the named entity assigned to the word with the given index. Words in a sentence are 0-indexed. The named entity is a string.

```java
sentence.nerTag(index)
```

CoreNLP’s NER does not recognize the pronouns “he” or “she” as the named entity PERSON even though these words do in fact refer to people. Thus, Specimus has its own function that checks if a word is “he” or “she”. If the check returns true, the word is treated as though it were tagged as the named entity PERSON.

### 4.8.3 Dependencies

The following code is used to retrieve all dependencies of a given type (in this case, appositives). The constant APPOSITIONAL_MODIFIER comes from the CoreNLP class UniversalEnglishGrammaticalRelations.

```java
final SemanticGraph dependencies = sentence.dependencyGraph();
final List<SemanticGraphEdge> appositives = new ArrayList<>();
appositives.addAll(dependencies.findAllRelns(APPOSITIONAL_MODIFIER));
```

The following code is used to get the governor and dependent of a SemanticGraphEdge. It is important to note that IndexedWord is 1-indexed not 0-indexed.

```java
final IndexedWord governor = edge.getGovernor();
final IndexedWord dependent = edge.getDependent();
```

### 5 Structured Rule Language

#### 5.1 Introduction

The structured language described in this section is used to define rules for the question generation system. Each rule has four parts: a phrase structure diagram, function definitions, preconditions, and generation specification.
5.2 Phrase Structure Diagram

Each rule has its own phrase structure diagram. The diagram describes what the rule is looking for as it traverses the sentence’s phrase structure tree. The rule will activate every time it finds a subtree in the sentence’s phrase structure tree that matches its own phrase structure diagram. The phrase structure diagram follows the same principles as the phrase structure tree: child nodes occur under their parent and are linked together with a line.

Each node in the diagram has a specific shape. This shape represents what kind of node in the phrase structure tree the rule is looking for. Rectangles represent phrase or POS nodes (non-leaves) such as NP, VP, and VBD nodes. Phrase or POS nodes must be named after the label they represent. Thus, a phrase node named “NP” in the diagram corresponds to a noun phrase node in the phrase structure tree of the sentence. Ovals represent lemma nodes (leaves) such as “be” and “go”. CoreNLP’s phrase structure trees do not contain lemma nodes; instead each leaf node is the actual word that occurred in the sentence. Thus, the phrase structure tree for the sentence “George Washington died in 1799.” would contain a leaf node with the label “died” instead of the lemma “die”. CoreNLP does, however, provide a method to retrieve the lemma of a given leaf node. This method is used in order to evaluate lemma nodes in the diagram.

If the outline of a node’s shape is dashed, the node is optional. Rules will activate if all non-optional nodes are present in the sentence’s phrase structure tree in the diagram’s given arrangement but will take optional nodes into account if they exist.

Each node in the diagram must have a unique name. Thus, if there are two NP phrase nodes in the diagram, one could be named “NP-1” and the other “NP-2”.

5.3 Function Definitions

A rule can have zero or more defined functions. Each function can take in zero or more inputs in order to generate a string as output. Each function describes how it uses the inputs, if there are any, to generate its output. The description is in plain English in order to allow for complex operations to be performed using CoreNLP. It is up to the programmer to implement the function’s description of its operation.
5.4 Preconditions

Each function can specify zero or more preconditions that must all be true in order for the rule to fire. As with functions, preconditions are described in plain English. Preconditions must be separated by a blank line.

5.5 Question Generation Specification

Each rule must specify how it uses its functions and the nodes from its phrase structure diagram in order to generate one or more natural language questions. The question generation specification must be a concatenation of function calls and/or phrase structure nodes. Phrase structure nodes in the generation specification act as functions that simply print out the words contained within the given phrase. For instance, if the node NP-1 represents the phrase “George Washington” in the sentence, then using NP-1 in the generation specification will output “George Washington” at that location in the question.

Each function call or phrase structure node must be on its own line. Each line must end with the concatenation symbol ‘+’ if there is another line below it. The concatenation symbol will insert spaces between non-empty strings.

If a rule generates more than one question, each question’s generation specification must be separated by a blank line.

6 Question Generation

6.1 Overview

As mentioned earlier, Specimus only generates concept completion (who, what, when, and where) questions. While narrow in scope, a single input sentence can generate several concept completion questions.

After a sentence is passed through the sentence simplification system, the resulting simplified sentences are passed to the question generation system. The question generation system uses defined rules in order to potentially generate questions.
for each sentence. Each rule is written using the structured language defined in section 5. Every rule is a Java class that takes in a sentence and returns a set of natural language questions. The returned set may be empty if the rule never activates or passes its preconditions. Thus, in order to retrieve all questions that a given sentence answers, the sentence is passed to each rule and all of the resulting sets are combined into a single answer set. By using a set instead of a list, the system does not need to worry about duplicate questions being generated by the various rules since a set does not allow duplicate elements.

The question generation system was designed this way in order to maximize the question coverage of the system: Because every rule is given a chance to generate questions for each input sentence, the system does not have to worry about questions being potentially missed. The downside to this approach is reduced performance as every rule is given the chance to examine each input sentence.

6.2 Rule Definitions

6.2.1 NP-VP Rule

Function Definitions

wh(x) – Generates ‘who’ if the head of x refers to a person, ‘what’ otherwise

passiveIfNecessary(v) – Transforms the given VP into passive voice, if possible, if the VP contains the preposition ‘by’ before the first NP.

Preconditions

None

Generate

wh(NP-1) + passiveIfNecessary (VP-1)
6.2.2 VP Rule

![Diagram](image)

**Function Definitions**

- `wh(x)` – Generates ‘who’ if the head of x refers to a person, ‘what’ otherwise
- `do(x)` – Generates ‘did’ if the given VP is past-tense, ‘do’ otherwise
- `lemma(v)` – Generates the lemma of the head verb of the given VP
- `firstNp()` – Generates the first NP of the sentence

**Preconditions**

None

**Generate**

`wh(NP-1)` + `do(VP-1)` + `firstNp()` + `lemma(VP-1)`

6.2.3 Equative Copula Rule

![Diagram](image)
Function Definitions
wh(x, y) – Generates ‘who’ if the head of x or y refers to a person, ‘what’ otherwise

originalWord(z) – Generates the original word for the given lemma from the sentence

Preconditions
None

Generate
wh(NP-1, ADJP-1) +
originalWord(“be”) +
NP-1

6.2.4 Attributive Copula Rule

Function Definitions
wh(x, y) – Generates ‘who’ if the head of x or y refers to a person, ‘what’ otherwise

originalWord(z) – Generates the original word for the given lemma from the sentence

Preconditions
None

Generate
wh(NP-1, ADJP-1) +
originalWord(“be”) +
NP-1
wh(NP-1, ADJP-1) +
originalWord(“be”) +
ADJP-1

6.2.5 PP Rule

Function Definitions
wh(x) – Generates ‘where’ if the head of x refers to a location, ‘when’ otherwise

auxOrDo(x, y) – Generates the first auxiliary verb of x if one exists. If one does not exist, ‘did’ is generated if x is past-tense, otherwise ‘do’ or ‘does’ is generated if y is plural or singular respectively.

allButFirstAux(x) – Generates x except for the first auxiliary verb if one exists.

Preconditions
NP-3 represents a LOCATION or a DATE

The preposition for PP-1 is not “by”

Generate
wh(NP-3) +
auxOrDo(VP-1, NP-1) +
NP-1 +
allButFirstAux(VP-1) +
NP-2 +
ADJP-1
6.3 Implementation

The rules described above are implemented using the functions provided by Simple CoreNLP’s Sentence class. Refer to section 4.8 for details on these functions.

In the implementation of Specimus, each rule described above is its own Java class. The full source for each class can be accessed at the following URL: https://github.com/Tyler-Yates/TuringThesis/tree/master/src/main/java/question

7 Test Set & Evaluation

7.1 Introduction

The sentences used to test and evaluate Specimus were topic sentences from Wikipedia feature articles. The reason these specific sentences were chosen is twofold:

1. Featured article candidates on Wikipedia undergo a rigorous evaluation by editors before selection. Featured articles are required to be “well-written and of a professional standard” [15]. As such, the topic sentences obtained from featured articles should be syntactically sound, which Specimus requires.
2. A coreference is when two or more phrases refer to the same entity but with different text. An example of this is anaphora.

   The **apple** was so red **it** glowed.

   This is an example of single-sentence anaphora; ‘it’ refers to ‘apple’. Coreferences can also appear between sentences.

   The **apple** was red. **It** tasted great.

Since Specimus operates on individual sentences, coreferences like the one above are impossible for the system to handle. Topic sentences, however, do not exhibit this kind of coreference. Because the topic sentence is the first sentence of the article, there is no possibility of a cross-sentence coreference. Single-sentence coreferences can still occur, as shown in the first example above, but the information necessary to disambiguate the coreference is still contained within the topic sentence.
In order to have a data set to pull topic sentences from, a bot was written to access all featured articles on Wikipedia. The bot ran on 2016-3-9 and accessed all 4,721 articles that were listed as featured articles at that time. The bot used the Wikipedia API to pull text from each article. The API only supported paragraph granularity when pulling out text from an articles, so the first paragraph of each featured article was retrieved and saved to disk.

Since Specimus requires its input to be a single sentence, the paragraphs would need to be modified before being passed to the program. Specifically, the first sentence from each paragraph needed to be identified and extracted. This process is known as sentence boundary disambiguation. In English, the period is the sentence boundary character, but as with many languages, the period can serve many functions and is thus an ambiguous boundary [16]. For instance, the following sentence has four periods, but only the last period represents the sentence boundary.

| Bob Jones Jr. (born c. 1872) could lift 482.3 pounds. |

There are systems that attempt to solve the problem of sentence boundary disambiguation [17] but it was found that the output of those systems still required human supervision in certain cases (the above sentence was not parsed correctly for instance). As such, the first sentence was extracted manually from the paragraphs as needed.

### 7.2 Test Set

At the start of the project, one hundred topic sentences were chosen randomly from the 4,721 featured articles to serve as the test set during the development of the system. Every successive change to the simplification or question generation systems would run through all one hundred topic sentences and output the results of the simplification and generation to disk. By differencing the output file with the previous output file before the current change, the effects of the current change could be analyzed. This allowed for an iterative approach to building the simplification and generation systems.
7.3 Evaluation

7.3.1 Hypotheses & Setup

The evaluation of Specimus focused on two hypotheses:

1. The generated questions are grammatically correct.
2. The original sentence answers the generated questions.

In order to test these hypotheses, human judges were asked to rate the output of Specimus on an evaluation set of sentence-question pairs. Fifty topic sentences were randomly chosen from the 4,721 featured articles to serve as the sentences for the evaluation set. None of the sentences in the evaluation set were contained in the test set. The fifty sentences were then given to Specimus and the generated questions were aggregated into a single set. In total, 284 questions were generated from the fifty sentences. Of the 284 generated questions, eighty were randomly chosen to be in the evaluation set.

Twelve human judges were each given twenty sentence-question pairs in order to evaluate the two hypotheses. This allowed each sentence-question pair to be evaluated by three different judges. The judges were instructed to use a 1-5 numeric scale when evaluating the two hypotheses. The scales for each of the hypotheses are given below.

**Hypothesis 1** - The generated question is grammatically correct.

5 - **Very Good**: question does not have any grammar errors

4 - **Good**: improper capitalization or punctuation that does not compromise the semantic meaning of the question

3 - **Okay**: incorrect tense or word choice that does not compromise the semantic meaning of the question

2 - **Bad**: semantic meaning of the question hard to determine

1 - **Very Bad**: semantic meaning of the question totally compromised
Hypothesis 2 - The original sentence answers the generated question.

5 - Very Good: sentence answers question in its entirety

4 - Good: sentence answers question broadly

3 - Okay: sentence hints at answer but does not provide concrete details

2 - Bad: sentence references entities in the question but provides no information to concretely answer the question

1 - Very Bad: sentence and question share almost no similarities

The instructions that the human judges were given at the start of the evaluation can be found in the Appendix.

7.3.2 Results

Hypothesis 1 - The generated question is grammatically correct

<table>
<thead>
<tr>
<th>Rating</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 – Very Good</td>
<td>147</td>
</tr>
<tr>
<td>4 – Good</td>
<td>34</td>
</tr>
<tr>
<td>3 - Okay</td>
<td>38</td>
</tr>
<tr>
<td>2 - Bad</td>
<td>17</td>
</tr>
<tr>
<td>1 – Very Bad</td>
<td>4</td>
</tr>
</tbody>
</table>

Hypothesis 2 - The original sentence answers the generated question

<table>
<thead>
<tr>
<th>Rating</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 – Very Good</td>
<td>117</td>
</tr>
<tr>
<td>4 – Good</td>
<td>64</td>
</tr>
<tr>
<td>3 - Okay</td>
<td>38</td>
</tr>
<tr>
<td>2 - Bad</td>
<td>17</td>
</tr>
<tr>
<td>1 – Very Bad</td>
<td>4</td>
</tr>
</tbody>
</table>

With regards to hypothesis 1, Specimus performed very well, with only 8.8% of the generated questions in the evaluation set scored as “Bad” or “Very Bad”. This is an important cutoff because questions rated as “Bad” or “Very Bad” have grammatical
errors that actually compromise the semantic meaning of the sentence. Additionally, 61.3% of the generated questions were rated as “Very Good” meaning that they had no grammatical errors.

With regards to hypothesis 2, Specimus also performed very well. 75.4% of the sentence-question pairs were scored as “Very Good” or “Good” meaning that the sentence answers the question if only in a broad sense. Only 1.7% of the sentence question pairs were scored as “Very Bad” which indicates that the sentence shares almost no similarities with the generated question.

With regards to inter-rater reliability, Krippendorff’s alpha coefficient [18] for the evaluation was calculated as $\alpha = 0.5873$.

8 Limitations

SimpleNLG’s passive voice realization is not always correct. Intransitive verbs cannot take on passive voice since they do not have an object. SimpleNLG does not respect this rule, however, leading to the creation of VPs like:

| was died in 1799 |

To mitigate this, a set of common intransitive verbs was compiled and hard-coded into the system. Whenever these verbs are encountered, the system disables the passive voice feature in SimpleNLG. This set of intransitive verbs is not exhaustive, and as such the system will not be able to correctly generate text for all verbs. Additionally, some verbs are ambitransitive, meaning they can be transitive or intransitive depending on the context of the sentence [19]. This means that context analysis is required in order to correctly determine if passive voice is allowable in all circumstances. Context analysis is beyond the scope of this project.

As mentioned in section 4.2.4, participial phrases can modify either noun phrases or verb phrases, in which case the participial phrase serves as an adjective or an adverb respectively. Unfortunately these two different roles (adjective and adverb) are often difficult for CoreNLP to identify. For instance, the example sentence that Heilman provides for a participial modifier of a noun phrase is actually identified by CoreNLP as a verb phrase modifier.
Jefferson, being the third U.S. President, loved white houses.

Heilman asserts that “being the third U.S. President” describes the noun phrase “Jefferson”. The CoreNLP parser, however, identifies the phrase as an adverbial clause modifier of “loved”. While Heilman’s interpretation of the sentence is certainly correct, CoreNLP’s interpretation is not wrong. One interpretation of the sentence is that Jefferson loved white houses because he was the third U.S. President, in which case the participial phrase functions as an adverb explaining the cause of Jefferson’s love.

CoreNLP’s parser identifies an incorrect grammatical relation when it parses sentences containing locations with words separated by commas. For instance, CoreNLP’s parse of the sentence:

I was born in Austin, Texas.

would contain the grammatical relation appos(“Austin”, “Texas”). This is incorrect because “Austin” and “Texas” are not appositives.

CoreNLP does not identify dummy subjects with great accuracy. Specifically, the dummy subject “it” is often not identified. In the sentence:

It is raining.

CoreNLP identifies “it” as a regular pronoun (PRP) which, while not technically incorrect, is not detailed enough for Specimus’ simplification system.

9 Future Work

Natural language questions can often be rephrased while still preserving the underlying semantic meaning:

“Where was George Washington born?”

“In which state was George Washington born?”

AQA systems like IBM’s Watson need to take this into account since users of the system are likely to ask the ‘same’ question in many different forms. Specimus does not attempt to perform any question rephrasing and as such is not as robust for training. A future
An improvement to the system would be to create more rules for the question generation system in order to generate more than one form of a question. Additionally, a separate system could be created in order to take questions as input and provide rephrased questions as output.

As described in section 4, the simplification system needs to determine whether a given comma marks the boundary between syntactic constructs. For instance, in the following sentence:

```
The book, written in June, 1942, was a best-seller.
```

the phrase “written in June, 1942” is one syntactic construction that can be extracted into a new simplified sentence. Thus, it is important for the simplification system to know that the comma between “June” and “1942” is not a boundary marker but a part of the phrase. Currently the simplification system uses several heuristics in order to determine whether a comma is a boundary comma or not. A future improvement would be to research ways in which to improve the detection of boundary commas.

As described in section 4.2.3, Specimus recognizes parenthetical birth and death dates for NPs that are tagged as a PERSON named entity. The simplified sentences that are generated contain the preposition “in” immediately before the date. This behavior is not correct in all cases however. For instance, one would not say:

```
Washington was born in February 22, 1732.
```

Instead, the preposition “on” would be used instead of “in”. A future improvement would be to tweak the simplification system to use “in” or “on” based on the date that it is referencing.

Specimus does not generate any questions for sentences with dummy subjects. A future improvement would be to devise special question generation rules for sentences with dummy subjects. For instance, for the sentence:

```
There will be cake at Bob’s house.
```

a possible question would be:
What will be at Bob’s house?

Additional evaluations could be carried out on Specimus. One particular area of interest is coverage: how many questions that are answerable by the original sentence did Specimus fail to generate? For instance, examine the following sentence.

Bob and Sue are cousins.

One question that may be asked of the sentence is:

What familial relationship do Bob and Sue have with one another?

If Specimus fails to generate the above question, then it does not have complete coverage for the original sentence. One way to quantitatively measure the coverage of Specimus is to give sentences to human judges and ask them to come up with all questions that are answered by the sentences. The questions from the judges and the questions from Specimus are then compared. Coverage can then be determined by counting how many questions that the judges came up with were not produced by Specimus.

Specimus only generates concept completion questions. Future work could expand the scope of Specimus to other question categories [9].

As mentioned in section 4.2.6, PPs are the only modifiers that the simplification system looks for when checking for modifiers that precede their subject. Future work could broaden the scope of this simplification to include more modifiers that can potentially occur before their subject.

10 Conclusion

This paper presented a system called Specimus which tackles the problem of AQG. Specimus uses two sub-systems to accomplish its goal: a simplification system and a question generation system. The simplification system looks for various syntactic constructions in the input sentence in order to generate one or more simplified sentences. Each simplified sentence is then passed to the question generation system. The question generation system uses rules written in a structured language presented in
this paper to transform the simplified sentences into grammatically correct questions answered by the input sentence. Specimus was evaluated by human judges and performed well with regard to both the grammaticality of the generated questions and the answerability of the questions given the original sentence.
Appendix

Evaluation Instructions

Thank you for your participation! Please read the following instructions on how to fill out the survey.

This survey will ask you to evaluate the quality of the output of a computer system. The system is designed to take an English sentence as input and generate English questions whose answers are found in the given sentence as output.

Only “who”, “what”, “when”, and “where” questions may be generated by the system. The sentences and questions in this survey have not been examined by human judges so no guarantees about the content of the sentences or generated questions can be made.

You will be presented with 20 sentence-question pairs in this survey. Each page will contain two questions with two responses. The responses will ask you to rate the grammaticality of the question and how well the sentence answers the question created from that sentence on a 1-5 scale. Use the following document to guide your judgements about the questions:

Grammaticality

Scale

5 - Very Good: question does not have any grammar errors

4 - Good: improper capitalization or punctuation that does not compromise the semantic meaning of the question

3 - Okay: incorrect tense or word choice that does not compromise the semantic meaning of the question

2 - Bad: semantic meaning of the question hard to determine

1 - Very Bad: semantic meaning of the question totally compromised

Examples
Very Good

Question: “When was George Washington born?”
Question: “Who wrote The Lord of the Rings?”

Good

Question: “What did My son see?”
Question: “What did, my son see?”
Question: “Do you like my daughter ’s cat?”

Okay

Question: “Who die on December 14, 1799?”
Question: “When was George Washington die?”

Bad

Question: “Who is a song written by Daughtry?”
Question: “What naval engagement off shores of Algiers?”

Very Bad

Question: “Who a part?”
Question: “What eat write during Wisconsin?”

Answerability

Scale

5 - Very Good: sentence answers question in its entirety
4 - Good: sentence answers question broadly
3 - Okay: sentence hints at answer but does not provide concrete details
2 - Bad: sentence references entities in the question but provides no information to concretely answer the question
1 - **Very Bad**: sentence and question share almost no similarities

**Examples**

**Very Good**

Sentence: “George Washington was born on February 22, 1732.”

Question: “When was George Washington born?”

Question: “Who was born on February 22, 1732?”

**Good**

Sentence: “George Washington, having been born in the New World, loved wide-open spaces.”

Question: “What was George Washington’s birthplace?”

**Okay**

Sentence: “Only two NBA players have over 20,000 career rebounds.”

Question: “Who have over 20,000 career rebounds?”

**Bad**

Sentence: “Jane Goodall studied chimpanzees in Tanzania.”

Question: “When did Jane Goodall start studying chimpanzees?”

**Very Bad**

Sentence: “The University of Texas at Austin was founded in 1883.”

Question: “What was the first animal in space?”
References


