

The Preposition Corpus in the Sketch Engine

Ken Litkowski

CL Research

9208 Gue Road

Damascus, MD 20872 USA

ken@clres.com

Abstract

Corpora from the Pattern Dictionary of English Prepositions (PDEP) provide the basis for in-depth examination of preposition behavior. At the main PDEP site, the organization is designed primarily to investigate individual prepositions and their senses. We have uploaded these corpora, in the form of their dependency parses, into the word sketch engine to enable their examination across prepositions and holistically (e.g., all at one time). We have annotated these corpora with additional information that provides an even richer set of data to examine. These include labeling the individual sentences with their senses (linked to the PDEP sense description), the PDEP class and the PDEP subclass, and supersense tags for the preposition complements and governors (enabling semantic word sketches of preposition behavior).

We describe in detail how the corpora were prepared for the sketch engine, involving several scripts used in PDEP to access its databases. Some of these scripts provide additional entry points into the PDEP data, particularly for the class and subclass. We describe the use of WordNet noun, verb, adjective, and adverb supersenses to tag the complements and governors. The resultant preposition data within the sketch engine provides a perspective different from the usual focus on the main parts of speech, so we describe the unique aspects enabled for the PDEP corpora. In particular, we describe several corpus query language (CQL) queries that provide useful perspectives on preposition behavior.

1. Introduction

The [Pattern Dictionary of English Prepositions](#) (PDEP) (Litkowski, 2014; Litkowski, 2017) provides a comprehensive set of data for describing preposition behavior. This behavior is captured in individual patterns that generally correspond to senses in the [Oxford Dictionary of English](#) (Stevenson and Soames, 2003). The Preposition Project (TPP) (Litkowski and Hargraves, 2005; Litkowski, 2013) provided sentences exemplifying preposition behavior and were tagged with senses; in many cases, this tagging resulted in an expansion of the sense inventory. The tagged instances have been used as the basis for characterizing preposition behavior in individual patterns, containing up to 20 properties for each sense. PDEP provides several online routines to examine the properties of each sense. Importantly, the tagging has been used for the development of support-vector machine models for preposition disambiguation. These models have suggested shortcomings in disambiguation and has led to more detailed examination of the importance of individual features (Litkowski, 2017). An important aspect of these studies has been

the need to examine features across prepositions in the same class and subclass. Facilities available in the sketch engine provide a further mechanism for detailed examination.¹

In [section 2](#), we describe the corpus as it exists in the sketch engine and the procedures by which it was generated, details of the Python script used to generate the vertical file are given in the [appendix](#). [Section 3](#) describes the idiosyncrasies in the sketch engine from having a corpus that focuses on prepositions. [Section 4](#) details procedures for systematic examination of the corpus, particularly for assisting in characterizing the behavior of individual senses and for examining characteristics of PDEP classes and subclasses. [Section 5](#) describes some additional results about preposition behavior afforded by using the sketch engine.

2. Source of the SkE Preposition Corpus

Each sentence in the TPP corpora was parsed using the Tratz parser ([Tratz and Hovy, 2011](#)), with output in the CoNLL-X format. This format consists of a line for each token in a sentence. Each token is characterized by 14 tab-separated fields, of which only six are used in the parse output. In the CoNLL-X format, a blank line is used to separate the sentences. The fields that are present in the parse output are the token number (starting at 1 for each sentence), the token itself, the lowercased lemma for the token with an appended one-character identification of the major part of speech, a part of speech tag for the token (slightly modified from the Penn Treebank tag set), the token number of the token upon which the instant token is dependent, and the dependency relation for the token. Files in this format are called vertical files.

For PDEP, a vertical file was created for each preposition in each TPP corpus². These files form the basis for the construction of the vertical files uploaded to the sketch engine. The source parse data, in the vertical file format, contains no identifier information. To provide such linkages, additional PDEP data was used from two scripts, both containing identifying numbers, one obtaining the raw sentences with the location of the focus preposition and one obtaining the locations of the complement and the governor associated with the sentence. This information was used to generate a new vertical file, with some added attributes for each sentence, in a [Python script](#), described in Algorithm 1.

Algorithm 1 Sketch Engine Vertical Files

Input: List of parse files

Input: Class, subclass, and supersense dictionaries

Output: Sketch engine vertical files

1: Read list of parse files

2: For each file, **do**

3: Get dependencies for preposition

4: Get content for each dependency

5: Get sentences for corpus, preposition, sense

6: Match sentence in corpus vertical file

7: Note position for each dependency

8: Create new vertical file with annotations

9: Get positions of preposition, complement, governor

10: Get class and subclass for preposition sense

¹ <https://www.sketchengine.co.uk/>

² These files are available at <http://www.cires.com/db/parses/>.

- 11: Annotate lemma part of speech
 - 12: Identify supersense tag for complement, governor
 - 13: Print vertical file for sentence
-

The subdirectories at the parse link provide a list of the parse files for each corpus; each subdirectory was saved as a list and used to identify the corpus name and each preposition with a parse file in that corpus. This is line 1 in Algorithm 1. Each preposition sense in PDEP has a TPP class and subclass. Since each sentence in the corpora has a sense tag, the class and subclass will be added to the sentence structure in the sketch engine files. This data is obtained from a PDEP script³ and placed in a dictionary used during the creation of the sketch engine vertical file. This dictionary is part of line 2 of the algorithm.

Following [McCarthy et al.](#) (2014), we have used the resources of the SuperSense Tagger (SST, [Ciaramita and Altun](#), 2006) to annotate the content words of the complements and governors in the PDEP data. These tags identify the WordNet lexicographer class, a set of 41 classes used for organizing WordNet synsets. As used in SST, the most frequent WordNet class is used as the tag. SST essentially performs a coarse word sense disambiguation. As McCarthy notes, accurate WSD is not critical, and it is likely that individual errors in disambiguation will be filtered out as noise when examining the tags in the sketch engine. For this tagging, we created a dictionary (part of line 2) of the lemmas in each of four “gazetteer” files from SST (one for each major part of speech), where we appended a part of speech code to each lemma (e.g., to distinguish nouns from verbs) to facilitate lookup for a lemma-POS combination in the creation of the sketch engine vertical files.

Steps 1 and 2 of the algorithm read the list of files for a corpus and process each one in turn, creating a vertical file for the entire set of prepositions in each corpus. This yields three vertical files, each of which is uploaded into the sketch engine. We describe the major steps of the algorithm in the following subsections.

2.1. Getting the Dependencies for the Preposition

Step 3 gets the dependencies for the instances of the preposition in each corpus, using a PDEP script.⁴ This script is used in PDEP to highlight in color the complements and governors of each sentence. This script is important here because it identifies (in JSON format) the sense, the instance number, and the starting positions and lengths of the complement and the governor when these are available (approximately 92 percent of the time). The instance number is most important because it provides a linking mechanism used in subsequent steps. The locations of the complement and the governor will be used in a subsequent step in identifying where to put structure tags for these elements in the new vertical file.

The list of dependencies provides the iterate for getting the content (step 4) for each instance. This first entails getting the sentences (step 5) for the specific sense of the preposition in the given corpus. This step also makes use of a PDEP script.⁵ This script returns (in JSON format) the preposition, the source, the sense, the instance number, the sentence, and the 0-based position of the preposition. This script is used in

³ <http://www.clres.com/db/prepclas.php>

⁴ An example is <http://www.clres.com/db/getDeps.php?corp=FN&prep=about>

⁵ An example is [http://www.clres.com/db/prepresents.php?source=FN&prep=above&sense=1\(1\)](http://www.clres.com/db/prepresents.php?source=FN&prep=above&sense=1(1))

PDEP to display the corpus instances for a given preposition, corpus, and sense. Here, the instance numbers between the dependencies and the sentences are linked, serving as the basis for further processing.

2.2. Matching the Sentence in the Corpus Vertical File

As indicated above, the parse files in CoNLL-X format do not identify the sentence instance numbers. All sentences in a preposition's parse file are read into tokenized structures corresponding to the fields in these files. Step 6 of the algorithm examines the "plain sentence" corresponding to the tokenized structure, i.e., the words are concatenated into a single string with no spaces. The matching process looks at the sentences from step 5, correspondingly stripped of all spaces, until it finds the one that matches the plain sentence.

When the match is found, the positions of the preposition, the complement, and the governor are annotated into the matching raw sentence (as obtained from step 5). First, the sense from the raw sentence is noted (for later use), along with the location of the preposition and its length. Next, new sentences are created for each of the three components; making use of the location and length information, structure tags are added to the raw sentence, i.e., <prep>, <compl>, and <gov>. Along with the plain sentence and preposition sense, these three tagged sentences are passed to the routine to create the vertical file entry for this sentence (step 8).

2.3. Identifying the Token Positions of the Preposition, the Complement, and the Governor

The first step in creating the vertical file entry for a sentence involves an attempt to locate the starting and ending token positions of the preposition, the complement, and the governor. Each of these uses the same routine to identify where to include the component. This routine (step 9) strips each word from the plain sentence and the tagged sentence until a tag is encountered, enabling the return of the starting and ending positions of the tag.

2.4. Annotating the Tokens (Class, Subclass, and Part of Speech)

After finding the positions of the three components, we next look up the preposition and sense in the class and subclass dictionary to obtain the values to be placed in the new vertical file (step 10). Using the part of speech tags in the CoNLL-X vertical file, we append a one-letter part of speech code to each noun, verb, adjective, or adverb lemma (step 11). These are used when the new vertical file is printed.

2.5. Printing the New Vertical File

Step 13 prints the new vertical file, which includes all the information from the original CoNLL-X vertical files, but with various added information. The new information adds XML structures, sometimes with attributes, in creating the new files. Each preposition is contained in a <doc> structure; the opening tag contains a **corpus** attribute that identifies the corpus and a **preposition** attribute that identifies the preposition. Each sentence is contained in a <s> structure. This structure includes (1) a **sense_label** attribute containing the PDEP sense number (e.g., "1(1)"), (2) a **class** attribute containing the TPP class (e.g., "Activity"), (3) a **subc** attribute containing the TPP subclass (e.g., "Proposed"), (4) an **inst** attribute containing the instance number of the sentence in the corpus, and (5) a **sense_desc** attribute containing a

link to the PDEP pattern⁶ for the given sense (allowing a sketch engine user to examine the behavioral properties for the sense). The information in these structures will allow more targeted examination of a preposition's properties in the sketch engine.

After the identifying information, the new vertical file contains a line for each token in the sentence, corresponding to these tokens in the original CoNLL-X vertical files. Each line is collapsed slightly from the original, containing six fields (as identified [above](#)). Additional lines are interleaved into the original lines to provide structure tags for the three main components: (1) a preposition tag (<**prep**>) surrounding the lines for the preposition (which may include more than one line for phrasal prepositions), (2) a complement tag (<**compl**>) around the line identifying the preposition complement, and (3) a governor tag (<**gov**>) around the line identifying the preposition governor.

The complement tag and the governor tag may include an **sst** (supersense tag) attribute. This tag is identified in step 12 of the algorithm, where possible. For example, a complement “withdrawing” with the part of speech VBG (a verb gerund) has the lowercased **lempos** field “withdraw-v”; accessing the supersense dictionary for this sense yields the tag “verb.motion” which is recorded as the **sst** attribute for the **compl** structure. Many complements and governors will not have **sst** attributes; e.g., personal pronouns will not have an **sst** attribute.

Running the Python script for the three corpora results in three vertical files, one for each corpus. These are uploaded to the sketch engine and compiled into a form ready for searching and other sketch engine functions. The script generates 80,369 sentences, compared to 81,509 sentences listed in PDEP. The Python script identifies problematic cases, written to separate files; these have not yet been examined in detail.

3. Examining the Preposition Corpus

Since prepositions are not normally a focus of investigation in the sketch engine, we describe additional perspectives on preposition behavior made possible with this corpus. In general, the functionality in Sketch Engine is accompanied by help pages which provide considerable detail. These are provided by means of a help icon containing a question mark; clicking on the icon opens a help page in a separate window. These help pages will supplement what is described below for the preposition corpus.

The preposition corpus is described in SkE in the menu item “**Corpus info**”. This shows that it was generated using 571 files from three corpora and 289 prepositions. The corpus contains 80,369 sentences, 2.43 million tokens, and 2.14 million words. The corpus contains 95,272 distinct words (84,058 distinct lowercase words) and 61,330 distinct lemmas. The corpus contains 46 distinct grammatical tags and 50 distinct dependency relations (deprels). This page also has a link to a corpus description (this document) and to a description of the dependency relations (the syntactic dependency guide appendix in [Tratz \(2011\)](#)).

Each sentence structure (<**s**>) contains one of 14 class labels, one of 67 subclass (**subc**) labels, an instance number, the PDEP sense label, and a sense description (**sense_desc**) which provides a link to the PDEP pattern for the sense. In most of the sentences, a structure is given for the preposition (<**prep**>, in 80,363 sentences), the preposition complement (<**compl**>, 73,692 sentences), and the governor of the

⁶ An example is [http://clres.com/db/TPPpattern.php?prep=on%20the%20point%20of&sense=1\(1\)](http://clres.com/db/TPPpattern.php?prep=on%20the%20point%20of&sense=1(1))

prepositional phrase (<gov>, 74,900 sentences). Where available, the **compl** and **gov** structures contains a supersense (**sst**) tag, 45 possible values for the **compl** and 46 possible values for the **gov**.

An important consideration in examining the preposition corpus is an understanding of its representativeness. The corpus is described in detail in Litkowski (2013). The whole corpus is not representative. Each of the constituent corpora must be considered on its own. The CPA corpus is the most representative, but only for its individual prepositions. For example, both *amid* and *after* have 250 instances, but these correspond to 681 and 42366 instances in the British National Corpus, respectively. The OEC corpus attempts to include 20 instances for each sense in ODE, but these are not representative for the particular sense and the full sense inventory for a single preposition cannot be considered representative. The FrameNet corpus is drawn from instances intended to illustrate particular frames and frame elements, which are frequently skewed in the number of instances for particular frames. Notwithstanding these concerns, and taking them into account while performing searches, the results from examining are likely to yield important behavioral characteristics.

There are several mechanisms in the SkE for examining the corpora. These include [Search](#), [Word list](#), [Word sketch](#), [Thesaurus](#), and [Sketch difference](#). We focus on aspects that are particularly useful for examining preposition behavior.

3.1. Searching the Corpus

SkE provides a form for specifying the parameters of a search. The bare form consists only of a text box for entering a **simple query**, as shown in Figure 1. The form can be expanded to enable a more advanced specification for **Query types** (as shown), [Context](#), and [Text types](#). For the most part, the **Context** and **Text types** details will not be used, although the latter will frequently be used in CQL queries.



The screenshot shows a web-based search interface. At the top, there is a text input field labeled "Simple query:" followed by a "Make Concordance" button. Below this, there are links for "Query types", "Context", and "Text types". The "Query type" section has radio buttons for "simple" (selected), "lemma", "phrase", "word", "character", and a checkbox for "CQL". There are also input fields for "Lemma:", "Phrase:", "Word form:", and "Character:". A "match case" checkbox is present. At the bottom, there is a "CQL:" input field, a "Default attribute: word" dropdown menu, and two buttons: "Make Concordance" and "Clear All".

Figure 1. Query Form

When a query is performed, the result is a concordance that identifies all occurrences of a word or a phrase, as shown in Figure 2. The default displays 20 occurrences per page, each of which highlights the matches to the query. Above the instances is a simple identification of the query and the number of hits. (In parentheses, there is an identification of the number of hits per million; this is computed as the number of occurrences of the search term, divided by the number of tokens in the corpus. This denominator is used for phrases as well, i.e., not considering that the phrase consists of multiple words. As mentioned above, the number of hits should not be taken as representative.) Finally, there is a circled “i”, which, when clicked, gives a detailed specification of the query parameters. The specification is linked and will bring up the query form with the specification in the CQL field.

The available query types are **simple**, **lemma**, **phrase**, **word**, **character**, and **CQL** (corpus query language specification). The **simple** and **phrase** queries perform essentially the same, finding all occurrences of what is entered. Since prepositions are not inflected, use of **lemma**, **word**, and **character** do not seem to provide any benefit, although they may be useful when examining other aspects of the surrounding context. When a preposition is entered as a simple query, without any further specification, the entire corpus is searched, without regard to where it occurs, frequently beyond the sentences that were identified in the preposition corpora for the specific preposition. A frequency analysis on the document corpus or preposition will show the extent to which the preposition is distributed across the corpora and the preposition files. The CQL query will be described in more detail below.



Figure 2. Search Results

When a search is performed, a set of menu options is displayed. For the preposition corpus, **View options**, **Frequency**, and **Collocations** are the most relevant. In **View options** (Figure 3), setting the display to view sentences allows a view of the full corpus instances. Other **View options** will identify what attributes, structures, and references are displayed, as well as indicating whether attributes are displayed for KWIC tokens, for each token, and as tooltips. In Figure 2, the blue script to the left of a sentence displays the references selected in Figure 3. When a concordance line is selected, a full display of the references is given, as in Figure 4. Several of these items are significant: (1) the last two lines identify the preposition and the corpus from which this sentence was taken; (2) the first line identifies the sense number with which this sentence has been tagged; (3) the second line provides a clickable link to the PDEP details for this sense; and (4) the **class** and **subc** lines identify the class and the subclass to which this sentence has been assigned.

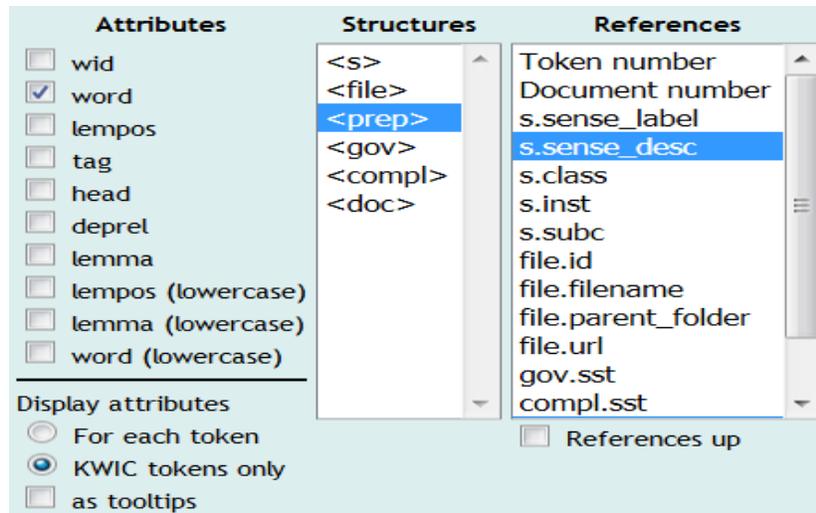


Figure 3. View Options

| | |
|------------------------|---|
| s.sense_label | 1(1) |
| s.sense_desc | http://clres.com/db/TPPpattern.php?prep=atop&sense=1(1) |
| s.class | Spatial |
| s.inst | 2 |
| s.subc | Above |
| file.id | file4071902 |
| file.filename | preps_cpa3.vert |
| doc.preposition | atop |
| doc.corpus | cpa |

Figure 4. Concordance References

Under the **Frequency** option, the **Text type frequency distribution** can be used to examine the distribution of the search result over various structure attributes. For this corpus, **s.class**, **s.subc** (subclass), **gov.sst** (governor supersense tag), **compl.sst** (complement supersense tag), **doc.preposition**, and **doc.corpus** may provide useful information. For the supersense tags, the frequencies will be generated only when the search is looking at the complements or the governors. The resulting table has three columns: the structure attribute, the frequency, and the relative frequency. The relative frequency of the search result compares the frequency of the specific text type to the whole corpus. Since this corpus is not subdivided into the usual text types involved in this type of analysis (e.g., “spoken” versus “written”), the relative frequency is not a useful statistic. As an example, Figure 5 shows the frequency of the supersense tags occurring two or more times for the 53 complements of *atop* in the subclass “Above”.

Frequency list

Frequency limit:

| P N | compl.sst | Frequency | Rel [%] ? |
|---------------------------------------|---------------------------|---------------------------|---|
| P N | noun.artifact | 18 | 10,048.70 |
| P N | noun.object | 7 | 18,552.50 |
| P N | noun.group | 4 | 3,562.20 |
| P N | noun.plant | 3 | 23,881.10 |
| P N | noun.shape | 2 | 34,565.60 |
| P N | noun.person | 2 | 1,543.90 |
| P N | noun.body | 2 | 5,560.10 |
| P N | noun.animal | 2 | 14,182.10 |

Figure 5. Complement Supersenses for "atop"

Under the **Frequency** option, the **Multilevel frequency distribution** can be used to examine various phrasal combinations that occur in conjunction with the search result. This can include an examination of the word, lemma, tag, or dependency relation (deprel) in up to four positions relative to the node result.

Similar results can be examined under the **Collocations** option, which will also rank each collocation candidate with a statistic. When the SkE has generated a concordance, it can be examined in more detail via the **Collocations** option. This form provides options to specify the attribute, the range, the minimum frequency, the scoring functions, and the score used to sort the results for collocation candidates. The attribute can be the **word**, the **lemma**, the **tag**, the **head**, the **deprel** (dependency relation), the lowercased **word**, and the lowercased **lemma**. Of these, the **deprel** seems most informative for the preposition corpus, along with the use of the **logDice** score for sorting the results. The logDice score is usually considered the most useful. This statistic measures the occurrence of a particular attribute within the search result compared to the occurrence of the attribute within the total corpus. Even though the total preposition corpus is not considered representative, it is likely that the results for the individual attributes are reasonably meaningful. (Similar results can be obtained from the [Word sketch](#) option, although it will not be able to examine collocations for phrasal prepositions.)

3.1.1. Corpus Query Language (CQL)

The CQL search option provides an ability to perform more detailed searches. A description of how a CQL search is specified is available to the Sketch Engine site.⁷ The following bullets identify useful patterns; further patterns will be illustrated below.

- **<prep> [] </prep>**: This is the most general global search. It will create a concordance of all the instances that have been tagged with **prep**. However, this tag does not surround phrasal prepositions, so it only forms the concordance for single preposition tokens (55,555 sentences). Adding "{1,4}" after the brackets will retrieve phrasal prepositions as well (81,125 sentences), but will cross sentence boundaries (which can be eliminated by specifying "**within <s />**". We

⁷ <https://www.sketchengine.co.uk/documentation/corpus-querying/>

can enter specific words inside the brackets to retrieve just the sentences for a specific preposition; we can also enter Boolean expressions to retrieve the sentences for multiple prepositions, e.g., “<prep> [word="in" | word="after"] [word="the"] [word="fashion"] [word = "of"] </prep> within <s />”. We can enter a **sense_label** in the <s> tag and we can enter another **within** statement to limit the concordance to a specific **corpus**. (See [findings](#).)

- **[lemma_lc="aboard"] within <s sense_label="1(1)" /> within <doc corpus="cpa" & preposition="aboard" />**: This query will retrieve just those instances having the specified sense within a preposition and a corpus, suitable for investigations of just that sense
- **<gov> [] </gov> []{2,5} <prep> [] </prep> []{2,5} <compl> [] </compl>**: This search identifies the prototypical prepositional phrase pattern. It looks for the governor, followed by the preposition and the preposition complement.

3.1.2. Context

The **Context** option in the search query allows a specification of lemmas in or not in the context of the search term. For the most part, this option is not likely to be used in investigating the preposition corpus. However, when a preposition is entered as a simple query, without any further specification, the entire corpus is searched, without regard to where it occurs, frequently beyond the sentences that were identified in the preposition corpora for the specific preposition. In such cases, it may be useful to examine the occurrence of the preposition in contexts beyond those sentences tagged for the specific preposition. Such cases may also be identified by performing frequency analyses for the concordances that are generated, as described above.

3.1.3. Text types

The **Text types** section provides the ability to specify portions of the corpus to be used in generating a concordance. In general, the types included here identify the various structures in the corpus. Several of these will be relevant for the preposition concordance analyses:

- the document preposition (**doc.preposition**) will limit the search to the sentences for a particular preposition (note that since many prepositions are phrases, the drop-down list must include “%20” for any spaces);
- the document corpus (**doc.corpus**) will limit the search to one of the three corpora (CPA, OEC, or FrameNet) (an equivalent limitation can be achieved using the filename (**file.filename**));
- the sense label (**s.sense_label**) can restrict the concordance to just those sentences with a specific sense tag (note that if this restriction is used, it should be with a specified preposition, since numbering is generally the same for each preposition; also, all sense labels include parentheses and these need to be preceded by backslashes);
- the sentence class (**s.class**) can examine all sentences that have been tagged as being in one of 12 classes (two other classes, **pv**, for phrasal verbs, and **x**, for infelicitous sentences, pulled out of random sample of the British National Corpus can be examined to identify uses of a word or phrase misidentified as a preposition use);

- the sentence subclass (**s.subc**) can examine sentences that have been tagged as being in one of 67 subclasses (each subclass falls under one main class, so these can be used for more fine-grained analyses); and
- the supersense tags (**compl.sst** and **gov.sst**) can be used to examine the occurrence of the various WordNet lexicographer file names within the corpora (values of the supersenses are generally examined after search for complements or governors).

The remaining text types will not be used in searches. The sense description (**s.sense_desc**) is a link to the PDEP entry for a sense and will usually be specified as part of the search results. The sense instance number (**s.inst**) was used in creating the vertical files and not used afterwards. The file information (**file.id**, **file.parent_folder**, and **file.url**) is used for Sketch Engine purposes.

When the useful text types above are employed in generating concordances, the specifications are incorporated into the CQL search term (accessible using the “i” information icon from the concordance screen). The query can be clicked to bring it up in the query form (Figure 1) in the CQL field. The specification can then be used as a template in which other structures and attributes can be used to quickly generate other similar concordances.

| Concordance description | | |
|---------------------------------|--|------|
| Corpus: PDEP Preposition Corpus | | |
| Operation | Parameters | Hits |
| Query | word <compl> [] </compl> within <s subc = "Species" /> | 2693 |

Figure 6. Concordance Description

3.2. Word List

The **Word list** option in SkE provides a form with several options. The first option, **Search attribute**, allows the examination of the corpus words, lemmas, tags, heads, and deprels. The word and lemma options essentially duplicate frequency lists that might be found with any corpus, e.g., showing prepositions and articles as the most frequent items. The head option is not informative, since these are only the token numbers of the tokens that are the heads of the current token. The tag and the deprel options may be slightly more informative, but they only provide an overview of the entire corpus.

3.3. Word Sketch

The **Word sketch** option in SkE provides a form to look at the behavior of a lemma, via grammatical relations, i.e., a one-page summary of the lemma’s grammatical and collocational behavior. This will not produce results for phrasal prepositions. For single-word prepositions, a word sketch will show the principal grammatical relations and the lemmas in these relations, ranked by the logDice score. At present, the word sketches for prepositions consist only of the dependency relations.

The advanced options for a word sketch allow selection of which grammatical relations will be shown. In general, the most relevant grammatical relation for prepositions is *pcomp*, i.e., the preposition complement, which will show the most frequent lemmas sorted by the logDice score. This score shows how much these lemmas stand out within the full corpus as related to the preposition. When all

grammatical relations are specified, some further insights into the collocational behavior are displayed, as in Figure 7.

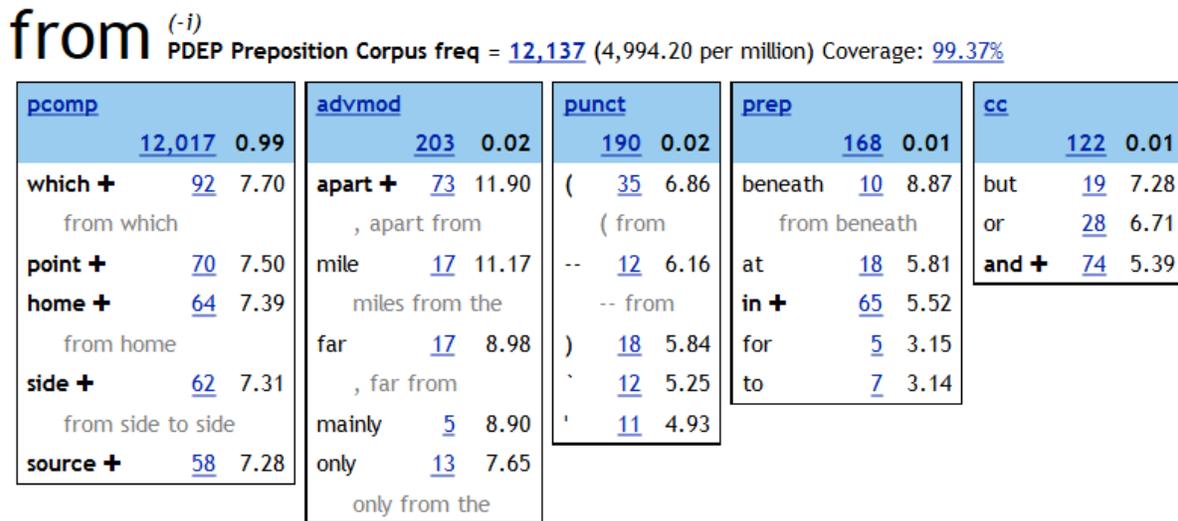


Figure 7. Word Sketch for *from*

At the top of each box, the name of the grammatical relation is given; the specification in the options has specified that only frequencies above a certain threshold will be displayed. Clicking any of the frequencies will displays a concordance for that item. The plus sign suggests that a particular collocate is relatively more frequent; a phrase below a collocation indicates a multi-word sketch can be displayed. The grayed represents the longest commonest match for the collocational pair.

We are currently investigating the possibility of including grammatical relations for the supersense tags of the complements and the governors.

3.4. Thesaurus

The **Thesaurus** option in SkE provides a form that requires a preposition lemma. For the preposition corpus, the results generate a list of similar prepositions, as shown in Figure 8. This list is ordered by an association score. The list also shows the frequency of each preposition in the corpus. Next to the list is a word cloud, with the most similar prepositions in the largest size. Each preposition may be selected (either from the list or the cloud) to show the [sketch difference](#) from the seed lemma. The form has a set of advanced options: (1) a maximum number of items, (2) a minimum score, (3) whether to show the head word in the cloud, (4) whether to cluster items, and (5) the minimum similarity between cluster items (a higher number produces smaller groups of words which are closer in meaning).

across (*preposition*)
 PDEP Preposition Corpus freq = 1,220 (502.01 per million)

| Lemma | Score | Freq |
|-------------------------|-------|------------------------|
| over | 0.284 | 3,336 |
| along | 0.280 | 1,467 |
| around | 0.280 | 1,404 |
| onto | 0.273 | 591 |
| into | 0.233 | 4,680 |
| through | 0.232 | 2,782 |
| off | 0.232 | 532 |
| up | 0.211 | 1,276 |
| down | 0.209 | 1,155 |
| round | 0.199 | 509 |
| within | 0.185 | 1,561 |
| beneath | 0.183 | 573 |
| on | 0.182 | 18,026 |
| outside | 0.181 | 759 |
| towards | 0.170 | 1,296 |
| near | 0.162 | 979 |
| from | 0.152 | 12,137 |
| above | 0.152 | 1,195 |



Figure 8. Thesaurus Results for Preposition **across**

The thesaurus finds words that tend to occur in similar contexts as the target word. In general, the similarity score between two words w_1 and w_2 is computed by first determining all the overlaps where the two words share a collocation in the same grammatical relation where an association score is greater than 0. The set of all word sketch triples (*headword, relation, collocation*) is used to find applicable contexts ($ctx(w_1)$), i.e., the set of (*relation, collocation*) in the word sketch triples set. The logDice is used as an association score. A distance score is then computed.

The distance between two prepositions is preserved, i.e., when changing the seed word back and forth. However, the position in the list will change. For example, for *at*, the closest word is *from* (with a score of 0.410), whereas with *from*, *at* is in the 9th position. (Since the matrix is symmetric, one could conceivably prepare a matrix with the prepositions as rows and columns. Then, one could highlight the cells that show the highest scores. I don't know how such a matrix could be generated, other than going through the prepositions one by one and saving the results.)

When the minimum similarity between cluster items is increased, fewer prepositions are included in the clusters and there seems to be a tendency toward greater similarity. In addition, it almost seems as if the clusters hold more generally, and not just in relation to the seed lemma. (But also, the clusters may not meet intuitive expectations (e.g., *under* doesn't show similarity with *underneath* or *below*).

3.5. Sketch Difference

The **Sketch diff** option in SkE provides a form that allows the comparison of two lemmas. (This option can also be accessed from [Thesaurus](#) results.) This option shows the differences in the word sketches between two lemmas. The basic form asks for the first and the second lemma. There are several advanced options.

The result assigns red and green colors to the two lemmas, as shown in Figure 9. The collocates in each color tend to combine with the lemma of the same color, with white collocates tending to combine with both lemmas. Bolder shades indicate stronger collocations. The top line shows the two lemmas and their respective frequencies within the corpus. The next line shows the two lemmas at the ends of a spectrum of color, with the color boldness corresponding to the collocation strength. The results are shown in boxes, each of which is labeled by the grammatical category (if not clear, looking at examples will help identify the category). For the preposition corpus, the grammatical categories are the dependency relations (deprels). The main results will generally be for the *pcomp* relation.

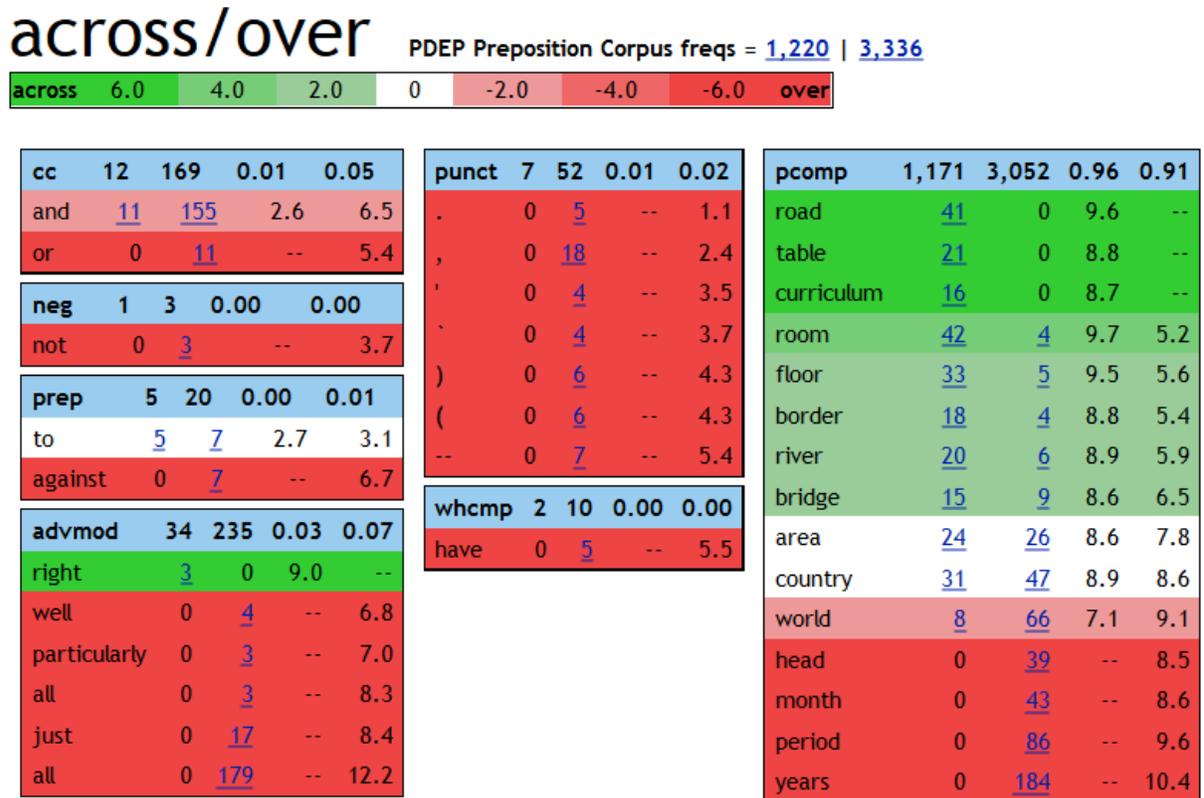


Figure 9. Sketch Difference between *across* and *over*

The boxes show the results in the gradations of color, from green to red (listed as +6.0 to -6.0, with no explanation). Each box has a line showing the relation, the total frequency of this relation and two scores. Within each box, the lemma of the relation is given on each line, along with the frequency of this deprel for each preposition (clickable so that these instances can be examined, with the lemma highlighted), and two numbers (the logDice scores for each preposition). The frequencies for the individual lemmas do not add to the total for the deprel; the other items do not attain the minimum frequency or the maximum number of items that can appear in a block, as specified in advanced options.

With the advanced options, one can specify that the differences be shown in a single box for each relation, or in common/exclusive blocks. With a single box, one can see a gradation from one word to the other, with common lemmas in white in between. This perhaps allows a little bit of judgment as to the overlap between the two lemmas.

4. Use of Sketch Engine for PDEP Database

The primary purpose of uploading the PDEP corpora to the Sketch Engine is to provide a systematic basis for completing fields in the patterns describing each sense of a preposition. Specifically, this includes a syntactic and a semantic characterization of the complement and the governor. In the initial creation of the TPP database, the complement and the governor were described using general lexicographic principles. In addition, TPP also identified a syntactic position (e.g., noun postmodifier, adjunct, verb complement, or adjective complement). These can be made more precise by examining the sketch engine data.

Another major use afforded by the sketch engine data is the ability to examine the PDEP classes and subclasses. [Srikumar and Roth](#) (2013) showed the utility of examining preposition behavior across prepositions. [Litkowski](#) (2017) performed a series of feature analyses across prepositions using Kullback-Leibler and Jensen-Shannon divergences, based on the PDEP classes. With the addition of PDEP subclasses, it may be possible to judge the consistency and differences among subclasses compared to their classes. The TPP data also identified substitutable prepositions, viewed as somewhat narrower than classes or subclasses.

We describe procedures for carrying out these analyses. In formulating the queries for these procedures, it is useful to keep in mind that they can be performed over the entire corpus (remembering that it is not representative) or a specific subcorpus (CPA, OEC, or FrameNet). When these analyses are performed, it may be useful to save the results.

4.1. Examining the Complement or Governor of a Preposition Sense

To examine the complement, it is necessary to use the CQL specification `<compl> [] </compl>` or `<gov> [] </gov>`. To narrow the concordance to instances of a specific preposition and sense, they must be specified by **within** clauses. These values can also be indicated by using the [text type](#) fields of a search query. (After making this specification, the full CQL query can be viewed, as shown in Figure 6, making it easy to vary the detailed specifications of the preposition or the sense, i.e., using the query as a template.)

The syntactic form of the complement or governor can be seen using the **Frequency** option for the **Node tags**. The frequency list will show each tag and will not aggregate, e.g., all noun or verb tags. The **Node forms** option can also be selected to see whether there are lexical sets that seem to predominate; the possibility of a lexical set can be examined more accurately by selecting the **lempos** of the node in the frequency analysis.

The semantic characteristics of the complement or governor can be seen using the **Frequency** option and specifying the **compl.sst** or **gov.sst**. This will list the WordNet lexicographer file names in descending frequency. When viewing these results, it should be kept in mind that many lexical items are not assigned a supersense tag. This may be particularly the case when the complement is a personal pronoun or the governor is a copular verb. When these results are saved, the number of instances without a supersense tag will also be listed.

4.2. Class and Subclass Analyses

PDEP has assigned [classes and subclasses](#) to each sense. The sketch engine provides an opportunity to question these assignments and to examine the consistency of the classes and subclasses. In general, these analyses involve a [text type](#) restriction in the <s> structure to specific classes or subclasses. In CQL, such a restriction is specified as **class = “value”** or **subc = “value”**. The 12 class values are available as checkboxes under the text types; the 67 subclass values are available as a drop-down list. Values can be combined in a specification, by enclosing optional values in parentheses separated by “|”, e.g., (**subc = “Above”** | **subc = “Below”**).

There are several dimensions of properties that can be examined under the class analyses. Of primary interest are the complement and the governor properties. To access these, either <compl> [] </compl> or <gov> [] </gov> should be used as the principal specification in the CQL statement. Once a concordance has been generated, the properties can be examined in detail using the frequency and collocation tools.

The **tag** and **deprel** frequencies provide an initial overall picture. For the complements, we should expect that noun tags and **pcomp** deprels (preposition complements) will predominate. For some prepositions, wh-clauses or gerundial forms will show heightened frequencies. For the governors, we can expect verb forms and **ROOT** deprels, although in many cases the preposition phrase will be governed by a **pcomp**, suggesting that it is modifying a noun.

The supersense tags may provide the best characterization of the complements and the governors. When examining a class or subclass, it may be useful to first bring up a frequency analysis by the document preposition. Then, the sketch engine provides the ability to generate a concordance for each preposition and then to generate frequency analyses for each in turn, enabling a side-by-side comparison of the frequencies. These results can also be saved and use for more detailed divergence analyses.

5. General Observations About Preposition Behavior Using Sketch Engine

In addition to the more specific results described above, examination of the corpora in the Sketch Engine provides some further insights into preposition behavior.

5.1. Collocates of Prepositions

The CQL search <prep> [] {1,4} </prep> generates a concordance of all the prepositions. When examining the collocation candidates using **deprels** within five tokens to the left and to the right, **ROOT** has the highest score, following by **subj**, **pcomp**, **det**, **punct**, and **amod**. This can be interpreted as saying that a prepositional phrase usually modifies the root of the sentence or its subject; it generally consists of a preposition complement modified by a determiner or an adjective; it frequently ends a sentence.

5.2. Infelicities in Preposition Tagging

The sketch engine can be used to identify errors in recognizing prepositions using the Tratz parser. When the TPP corpora were prepared, the target preposition was specified using its character position within the sentence, prior to parsing. In general, we would expect the parser to yield a **prep** deprel and an IN tag for the preposition. However, this is not the case. To investigate such infelicities, we use the CQL query <prep> “...” </prep>, with the preposition specified by one or more quoted lemmas (for phrasal prepositions). To examine the deprel, we use the multilevel frequency distribution option, specifying the deprel attribute for the node position. For the tags, we use the node tags frequency option.

For the most part, single-word prepositions follow the expected deprel and tag patterns. For these prepositions, the tag IN may be consistent for all instances, but the deprels will show several values. With the sketch engine, we can use the frequency lists to examine the deprels or tags that have given rise to unexpected values.

Several single-word prepositions, such as “about”, are frequently confused in parsing as adverbs or particles. For phrasal prepositions, such as “apart from” or “because of”, the parsing goal is to label them with the deprel “prep combo”. The sketch engine frequency analyses permit us to see where this goal has not been achieved.

5.3. Infelicities in Tagging Preposition Contexts

The sketch engine also facilitates an examination of the contexts of the preposition use to identify problematic cases. This can be done using the collocation or the frequency options for a concordance.

The collocation option is best used in conjunction with `<prep> “...” </prep>` queries. For the most part, these collocations are best used to examine 1 to 3 positions to the right of the preposition. Examination of the left context is generally not as fruitful. In general, we would expect **pcomp** to be the highest ranked deprel (corresponding to the preposition complement), with higher logDice values for **det** (determiner), **amod** (adjective modifiers), and **poss** (for possessive modifiers). Other collocation candidates can be examined from the results, but these results are unlikely to illustrate problematic cases.

The frequency option is best used to examine the complements (with `<compl> [] </compl>`) and the governors (with `<gov> [] </gov>`), specifying particular prepositions, classes, or subclasses, as described above. In general, the focus of these frequency analyses is to identify predominant syntactic or supersense tags, as described earlier. In addition, we can examine the less frequent values. The tag frequencies may reveal parsing problems or may uncover less frequent, but important possible values for the complement or the governor. The supersense frequencies may identify problems with using the most frequent WordNet sense; this may suggest that some of the more frequent supersenses may have an even higher frequency.

References

- Massimiliano Ciaramita and Yasemin Altun. 2006. Broad-coverage sense disambiguation and information extraction with a supersense sequence tagger. *EMNLP-’06 Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia, ACL, 594-604.
- Ken Litkowski. 2013. *The Preposition Project Corpora*. Technical Report 13-01. Damascus, MD: CL Research.
- Ken Litkowski. 2014. Pattern Dictionary of English Prepositions. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Baltimore, Maryland, ACL, 1274-83.
- Ken Litkowski. 2017. Pattern Dictionary of English Prepositions. In M. Diab, A. Villavicencio, M. Apidianaki, V. Kordoni, A. Korhonen, P. Nakov, and M. Stevenson, editors *Essays in Lexical Semantics and Computational Lexicography – In Honor of Adam Kilgarriff*. Springer Series Text, Speech, and Language Technology. Springer.
- Ken Litkowski and Orin Hargraves. 2005. The preposition project. *ACL-SIGSEM Workshop on “The Linguistic Dimensions of Prepositions and Their Use in Computational Linguistic Formalisms and Applications”*, pages 171–179.
- Diana McCarthy, Adam Kilgarriff, Milos Jakubicek, and Siva Reddy. 2015. Semantic Word Sketches. *Corpus Linguistics 2015*. Lancaster University.

- Vivek Srikumar and Dan Roth. 2013. Modeling Semantic Relations Expressed by Prepositions. *Transactions of the Association for Computational Linguistics*, 1.
- Angus Stevenson and Catherine Soanes (Eds.). 2003. *The Oxford Dictionary of English*. Oxford: Clarendon Press.
- Stephen Tratz. 2011. *Semantically-Enriched Parsing for Natural Language Understanding*. PhD Thesis, University of Southern California.
- Stephen Tratz and Eduard Hovy. 2011. A Fast, Accurate, Non-Projective, Semantically-Enriched Parser. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.

Appendix

Python Script to Create a Vertical File

This section describes the Python script (**download_parses.py**) used to create the vertical file **preposition.vert** and the file **preposition.conll** that were uploaded to the sketch engine. This script is accompanied by a **Makefile** that automates this creation, iterating over the files in the **/data** subdirectory (**(cpa|fn|oec).html**), each of which contains a list of ***.parse** files in the directory **/db/parses/(cpa|fn|oec)** at the PDEP web site. The Python script is called with one of the HTML files as argument. The output of each processing is appended to **preposition.vert**, at the completion of which, this file is used to create **preposition.conll**.

1. main

This function creates an argument parser to read the input file name (the HTML file). It defines the PDEP **domain** as <http://www.clres.com/db/parses/> and **corpus_name** as a result of matching **(.*)html** from the file name. We then iterate over each **line** in the HTML file, with a regular expression search that obtains the **prep_file_name** and the **prep** from any line containing **(.*)parse**. When we have a **prep_file_name**, we get the JSON file at **r** from <http://www.clres.com/db/getDeps.php> with **corpus_name** and **prep** as arguments; this file contains the complement and the governor locations and lengths for all instances of the preposition in the corpus. We set **remote_file** to **r.data** and **compl_gover** (complement and governor) to the result of **json.loads** of **remote_file**. We print the opening tag for this preposition, **<doc corpus="corpus_name" preposition="prep_file_name">**. The function **get_content** is then called with the url, **corpus_name**, **prep**, and **compl_gover** as arguments (1) to link the dependency file with the preposition sentence file, (2) to link these with the sentences in the dependency parses file, and (3) to create a new vertical file with tags (for the preposition, complement, and governor) surrounded by a sentence element with a sense label and a link to a sense description and with part-of-speech tags appended to the lemmas. Finally, we print the closing tag, **</doc>**.

1.1. **get_content (url, corpus_name, prep_file_name, compl_gover)**

This function has the arguments **url** (the url of CoNLL formatted verticals for the given corpus and preposition word, e.g., <http://www.clres.com/db/parses/fn/above.parse>), **corpus_name** (the name of the corpus), **prep_file_name** (name of file that stands for preposition we want to investigate), and **compl_gover** (a list of JSON objects that consists of the sense, the sentence number, the complement location and its length, and the governor location and its length).

The function first gets the vertical file at the **url** and sets **prep_sents** to the result of calling [get_sentences](#) to get all the sentences for the specified preposition in the specified corpus in CoNLL tokenized format. The function next iterates over the elements in **compl_gover**, using **j_obj** as the iterate, with a message identifying which record is being processed, where each iterate identifies a sense, an instance number, and the locations and lengths of the complement and the governor. The function next calls the PDEP script [http://www.clres.com/db/prepsents.php?source=FN&prep=about&sense=1\(1\)](http://www.clres.com/db/prepsents.php?source=FN&prep=about&sense=1(1)) to get all the sentences that have been tagged in the specified corpus for the preposition with the given sense. Next, we iterate over these sentences, searching for the one whose 'inst' element is equal to the 'inst' element of **j_obj**. Once this link is made, we next need to find this sentence in the vertical file. This is done by iterating over the

sentences in **prep_sents**, with the iterate **sent**, calling [sent_plain](#) with **sent** until its result matches the string formed by concatenating the word associated with the 'sentence' element of the matched instance.

When a match with the file in the vertical file is made, **matching_sent** is set to **sent** and the variables **sense**, **p_sent**, **prep_loc**, and **prep_len** are set to the 'sense', 'sentence', 'preploc' (preposition location), and 'prep' (preposition) elements of the sentence that was matched with the 'inst' element. Next, tags are added to **p_sent**, first to surround the preposition of interest with the **<prep>** tag, and then to try to do this as well for the governor (**<gov>**) and the complement (**<compl>**), into the variables **preposition_sent**, **governor_sent**, and **complement_sent**.

If **matching_sent**, **sense**, and **preposition_sent** have values, [create_new_vert](#) is called with **matching_sent**, **sense**, **prep_file_name**, and the collapsed sentences **preposition_sent**, **governor_sent**, and **complement_sent** (each of which will now have a tag) to print a vertical file entry for **matching_sent**, surround by a sentence tag with attributes for the sense and a link to the PDEP pattern for that sense. Otherwise, a message is written to stderr that the sentence wasn't found for this **j_obj**.

1.2. **create_new_vert (matching_sent, governor_sent, preposition_sent, complement_sent, sense, preposition)**

This function prints out a new vertical file for the argument **matching_sent**. The function first calls [include_component](#) to identify the locations for the **<gov>**, **<prep>**, and **<compl>** tags, i.e., the beginning and ending token numbers. The function prints the opening tag for the sentence, **<s sense_lab="sense" sense_desc=<http://clres.com/db/TPPpattern.php?prep=preposition&sense=sense>>**. The function then iterates over the tokens in **matching_sent** setting **i** to obtain the token. If **i** is the starting or ending position for the governor, preposition, or complement, the corresponding tag is printed as part of the vertical file. The function next prints a line for the token, with tab-separators, including the token id, the token itself, the lemma (lempos), the (part-of-speech) tag, the dependency id, and the dependency relation. After printing all tokens, a closing **<s>** tag is printed.

1.3. **include_component (tok_sent, plain_sent, structure)**

This function marks the beginning and ending locations for the **<gov>**, **<prep>**, and **<compl>** tags in the vertical file, i.e. the **structure** argument. If **plain_sent** is empty, returns -1, -1. Otherwise, enters a while loop over the tokens in **tok_sent**, setting **token** to each token in turn. If **token.word** begins **plain_sent**, resets **plain_sent** by this word, increments a token counter, and continues to the next word. If instead, **plain_sent** begins with **<structure>**, removes the tag from **plain_sent** and sets **start** to **token.id**. If **plain_sent** begins with **</structure>**, removes the end tag from **plain_sent** and sets **end** to **token.id**. Proceeds over all tokens of the sentence, possibly unnecessarily. After the loop, if 1000 tokens have been counter, prints an error message. If both **start** and **end** have been set, returns them. Otherwise, prints an error message.

1.4. **get_sentences (memory_file)**

This function builds an array at **result** by iterating over **memory_file** as long as a sentence **s** is returned, appending each sentence to **result**. This array is the set of all sentences for a particular corpus and preposition, with each tokenized in CoNLL format for printing to the vertical file.

1.5. **get_sentence (memory_file)**

This function gets a sentence from **memory_file**. Starts with an empty array at **sentence**. For each **line** in **memory_file** that is not empty, splits into **spl** based on tab separators. Sets **id**, **word**, **lemma**, **tag**, **depid**, and **dep_lable** to **spl[0]**, **spl[1]**, **spl[2]**, **spl[4]**, **spl[6]**, and **spl[7]**, respectively, with an error message if there is an index error. If successful, sets **token** to the result of creating a new [Token](#) with these variables as argument and appends this token to **sentence**. If a line is empty, the function yields **sentence** (i.e., returns this result to the calling function [get_sentences](#)) and empties it for the next sentence.

1.6. **sent_plain (sentence)**

This function builds an array of the **token.word** elements of **sentence**. It then returns a string concatenating the elements of the array into a string corresponding to the sentence without any whitespace. The output of this function is used in matching the sentence with an instance number in **get_content** and in identifying the location of the preposition, the complement, and the governor tags in **include_component**.

1.7. **Token (id, word, lemma, tag, depid, dep_lable)**

This class forms an instance of **Token** using its arguments to set the respective elements of the instance. The **id** member is set directly. All other members are set to the result of decoding the string according to *utf-8*. The **word** member may be modified if it is a double backquote, a double singlequote, or contains an ampersand. The same is done for the **lemma**. The **tag** member is set directly. The **depid** and **dep_lable** members are set directly.

The **lempos** member is set based on a call to the member function **make_lempos** with **lemma** and **tag** as arguments. If **tag** is in `u"CC"`, `u"IN"`, `u"JJ"`, `u"JJR"`, `u"JJS"`, `u"NN"`, `u"NNS"`, `u"NNP"`, `u"NNPS"`, `u"VB"`, `u"VBD"`, `u"VBG"`, `u"VBN"`, `u"VBP"`, `u"VBZ"`, the first character of the tag is appended to the lemma. If **tag** is one of `"RB"`, `"RBR"`, `"RBS"`, an **a** is appended. If **tag** is none of these, **x** is appended.