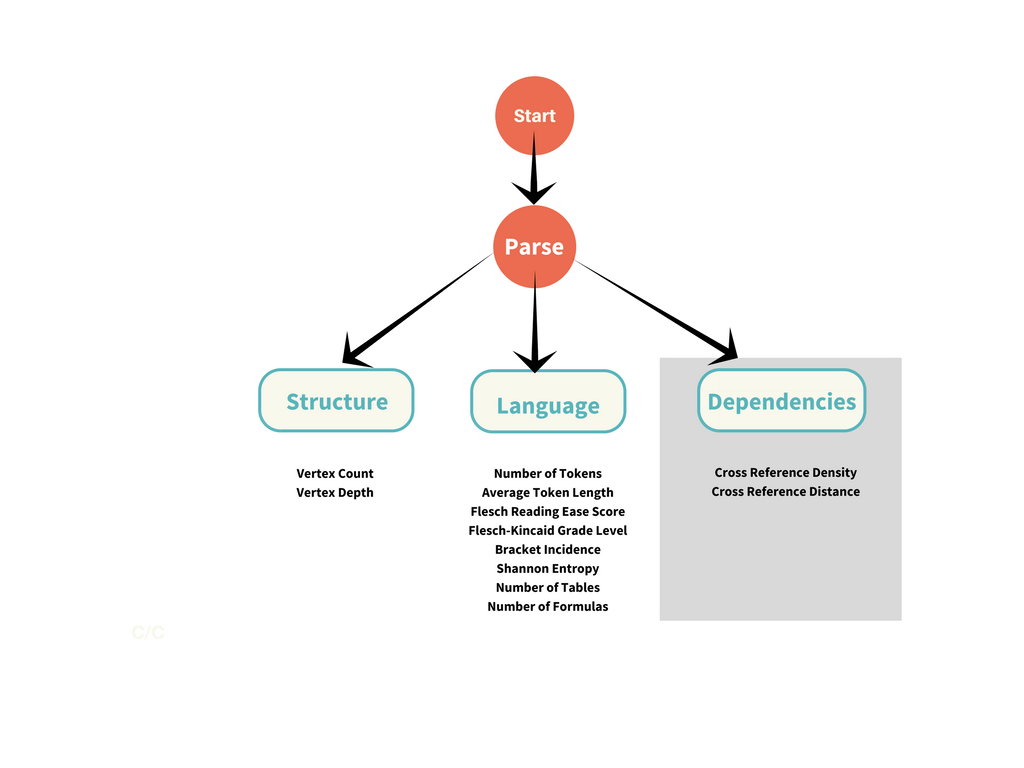
**CFR Complexity Analysis Technical Design**

Several approaches exist to effectively analyze the complexity of content in the financial regulations -- Titles 12, 17, and 31 of the CFR. The three major factors used for analyzing complexity are structure, language, and dependencies. The below flow diagram shows the process followed.



The U.S. Government Publishing Office provides a compressed XML file of federal regulations by year (i.e. 2017, 2016, 2015, etc.). XML, or Extensible Markup Language, stores files in a format that is readable by both humans and machines. The relevant parts of the XML are organized in the following hierarchy:

<PART>

<EAR>

<SECTION>

<SECTNO>

<SUBJECT>

<P>

<P>

* The first step is to parse the Title 17 XML file for a year (eg: 2017). The Python 3.6.2 has pre-defined functions, XMLBuffer and XML etree, which can be used to parse the compressed XML files.
* The custom parseBuffer method defined in the program finds and iterates through <PART> tags. Then, the outputSections method iterates through <SECTION> tags to find the remaining information (section number <SECTNO>, subject <SUBJECT>, and paragraph text <P>).
* To find the title-# format in the file name, the pre-defined Python split method was used. Using this title-# format, only the relevant titles of ‘title-12’, title-17’, and ‘title-31’ will be matched and analyzed.
* Non-ASCII characters were stripped to facilitate the display of section numbers. The parsed files are then used for estimating these structural complexity measures.

**Vertex count**

Vertex count is considered to be the number of elements under each section. For instance, consider the structure under Section 3.42 of Title 17(Termination):

(a)

(1)

(2)

(3)

(4)

(5)

(6)

(7)

(8)

(i)

(ii)

(b)

Section 3.42 has 12 vertices. By this measure, a section with three vertices will be simpler than Section 3.42. One with 20 vertices will be more complex, according to this structural measure. Regarding the XML format of the files, <P> tags generally represent the division of paragraphs. In this case, each vertex is separated by a <P> tag. While iterating through sections, count the number of <P> tags using the pre-defined findall method.

**Table Number:**

Table number is the number of tables under each section. For instance, Title 12 Part 3, Subpart B, Section 3.11 (a)(3)(iv) contains a table. Tables are identified in the XML file by the tag <GPOTABLE>. Similar to vertex count, while iterating through sections, count the number of <GPOTABLE> tags using the findall method.

**Token Number:**

Token number represents the number of words in each section, excluding stopwords (a, the, etc.). To calculate this measure of complexity, the following methodology was used:

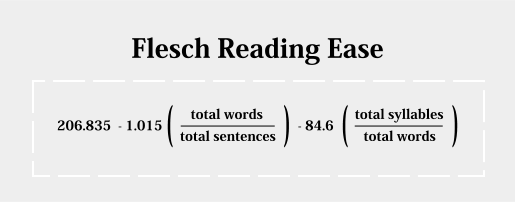
First, download the Natural Language Toolkit (NLTK), and import the stopwords list from the nltk.corpus library. Then, use regular expressions to remove the stopwords from the corpus under each section. <E> tags, which italicize text, prevented the <P> tag text from being received correctly. Therefore, the <E> tags were removed. Lastly, use the pre-defined split method to place each token into a 1-dimensional array. To find the number of tokens, the length of the array was calculated.

**Average Token Length:**

Average token length is the average word length of all the words in a section. If the average word length of a section increases over time, this may be an indication that words are getting longer and more complex. To find this value, iterate through the array created to find the number of tokens. Add the length of each token and divide by the total number of tokens to find the average token length.

**Flesch Reading Ease Score:**

The Flesch Reading Ease Test provides a score that shows the readability of the text. A higher score indicates that the text is simple to read and fairly straightforward, while a lower score indicates a difficult corpus. The Python TextStat package, which has pre-written methods that calculate certain language metrics, was used. The download and installation instructions can be found at <https://pypi.python.org/pypi/textstat/0.1.4>. The flesch\_reading\_ease method was used in this case, where the text in the <P> tags was passed through as the parameter. This method follows the defined formula:



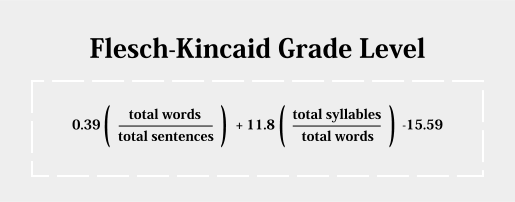
The data for this measure shows some values over 100 other values below 0. Further research indicates that these values are extremes, but still valid.

1. Table from Wikipedia:

|  |  |  |
| --- | --- | --- |
| **Score** | **School level** | **Notes** |
| 100.00-90.00 | 5th grade | Very easy to read. Easily understood by an average 11-year-old student. |
| 90.0–80.0 | 6th grade | Easy to read. Conversational English for consumers. |
| 80.0–70.0 | 7th grade | Fairly easy to read. |
| 70.0–60.0 | 8th & 9th grade | Plain English. Easily understood by 13- to 15-year-old students. |
| 60.0–50.0 | 10th to 12th grade | Fairly difficult to read. |
| 50.0–30.0 | College | Difficult to read. |
| 30.0–0.0 | College graduate | Very difficult to read. Best understood by university graduates. |

**Flesch-Kincaid Grade Level:**

The flesch\_kincaid\_grade method from the Python TextStat package was used to calculate the Flesch Kincaid Grade Level. The text in the <P> tags is passed through as the parameter. This method follows the defined formula:



**Bracket Incidence:**

Bracket incidence refers to the total count of all brackets, including the following: ({[]}). The pre-defined count method, which can count the occurrence of a given character, was used to find this value. An unusual result shows odd numbers of bracket incidence, meaning opening and closing brackets are not balanced. This imbalance is mostly occurring when bracket incidence is higher than 50.

**Conditional Words:**

Conditional phrases usually detail a hypothetical situation and their consequences. These phrases introduce another level of complexity, as they place restrictions or define other factors. The conditional words used for this study are the following:

“If, Except, But, Provided, When, Where, Unless, Whenever, Notwithstanding, In no event, In the event, In case, On the condition, Wherever.”

The occurrence of these phrases in the CFR was found using the pre-defined findall method and regular expressions. A regular expression is a sequence of characters that outlines a search pattern. In this case, the regular expression was (?=("+'|'.join(conditionalPhrases)+r")).

**Token Entropy:**

The entropy of a piece of the text is a measure of token diversity. The idea is that the more diverse the language in a piece of the text, the more difficult it will usually be to understand. It is a statistical measure of the uncertainty of a signal or a message. To calculate this value for each section, the method calculateEntropy was defined, with support from [Ero Carrera’s blog](http://blog.dkbza.org/2007/05/scanning-data-for-entropy-anomalies.html). This code is used to calculate the byte entropy, or entropy by letter, for a given text. The byte entropy values range from 0.0 to 8.0, and a value closer to 8.0 indicates higher entropy. This procedure may not be realistic, as entropy by word or sentence may give a better representation of complexity.

**Vertex Depth:**

Vertex depth is the average depth of the text. For instance, consider the structure under Section 3.42 of Title 17(Termination):

(a)  1

(1)  2

(2)  2

(3)  2

(4)  2

(5)  2

(6)  2

(7)  2

(8)  2

(i)  3

(ii)  3

(b)  1

In this case, parts (a) and (b) have a depth of 1, (1), (2), (3), etc. have a depth of 2, and (i), (ii) have a depth of 3. So, of the total 12 vertices, there are 2 vertices with depth of 2, 8 vertices with depth of 2, and 2 vertices of depth 3. The average vertex depth of the section computes to (2x2 + 8x2 + 2x3)/(12).

The regular expression pattern, ('\(([^)]+)'), was used to find the part designation inside the parentheses. Then, this designation was classified as a lowercase letter, integer, Roman numeral, or uppercase letter. The occurrence of each was found and multiplied by the corresponding depth. This total was divided by the total number of vertices. If the section had 1 vertex with no part designation, it was still said to have an average vertex depth of 1.

**Output:**

For the analysis of trends, the calculated values for each section were averaged to produce a value for the year. For example, consider the measure of average word length.

In 2017, if Section 1.1 had an average word length of 5.5 and Section 1.2 had an average word length of 6.5. The overall average word length for both sections in 2017 would be (5.5 + 6.5)/2 = 6. These averages were compiled to produce the following spreadsheet with average measures for each year.

