## Introduction:

* The purpose of sentence simplification is to take complex or lengthy sentences and restructure them in a way that makes them easier to be converted to USR while preserving the original meaning.
* Sentences are broken down based on a list of connectives.

## Installation:

* To setup a new project -

Visit <https://bitbucket.org/iscnlp/workspace/repositories/> to install the following in mentioned order -

* + - Tokenizer

- git clone https://irshadbhat@bitbucket.org/iscnlp/tokenizer.git

- cd tokenizer

- sudo python setup.py install

- cd ..

* + - Pos-tagger

- git clone https://irshadbhat@bitbucket.org/iscnlp/pos-tagger.git

- cd pos-tagger

- sudo python setup.py install

- pip install -r requirements.txt

- cd ..

* + - Parser

- git clone https://irshadbhat@bitbucket.org/iscnlp/parser.git

- cd parser

- sudo python setup.py install

- pip install -r requirements.txt

- cd ..

* clone the module available on Github: <https://github.com/FatemaBawahir/USR_Generation/tree/master/sentence_simplification>

## Input Format:

* The input file is plain text.
* It can contain one or more sentences for simplification.
* The input format for each sentence is -

Sentence\_id <any number of spaces> Sentence <followed by newline>

* Example input file -

Geo\_ncert\_6stnd\_2ch\_0023 इस प्रकार, विषुवत् वृत्त पृथ्वी पर एक काल्पनिक वृत्त बनाती है एवं यह पृथ्वी पर विभिन्न स्थानों की स्थिति बताने का सबसे महत्त्वपूर्ण संदर्भ बिंदु है।

Geo\_ncert\_6stnd\_2ch\_0028 इस प्रकार 90 अंश उत्तरी अक्षांश उत्तर ध्रुव को दर्शाता है तथा 90 अंश दक्षिणी अक्षांश दक्षिण ध्रुव को।

Geo\_ncert\_6stnd\_2ch\_0049 ये अर्धवृत्त हैं तथा उनके बीच की दूरी ध्रुवों की तरफ बढ़ने पर घटती जाती है एवं ध्रुवों पर शून्य हो जाती है, जहाँ सभी देशांतरीय याम्योत्तर आपस में मिलती हैं।

## Output Format:

* The output file is plain text.
* The output format for each sentence is -
  + Sentence\_id with letter to show subparts <space> Sentence <space> None/ Manual Evaluation
  + None tag represents that the sentence is ready to be sent for further processing
  + Manual Evaluation tag represents that the sentence had a connective but is still not broken. This is to inform the annotators if some manual action needs to be taken.
* For each input broken into subparts, letters a, b, c etc. are appended to the sentence\_id. If the sentence is not broken then nothing is appended.
* Example output file -

Geo\_ncert\_6stnd\_2ch\_0023a इस प्रकार, विषुवत् वृत्त पृथ्वी पर एक काल्पनिक वृत्त बनाती है । None

Geo\_ncert\_6stnd\_2ch\_0023b एवं यह पृथ्वी पर विभिन्न स्थानों की स्थिति बताने का सबसे महत्त्वपूर्ण संदर्भ बिंदु है । None

Geo\_ncert\_6stnd\_2ch\_0028 इस प्रकार 90 अंश उत्तरी अक्षांश उत्तर ध्रुव को दर्शाता है तथा 90 अंश दक्षिणी अक्षांश दक्षिण ध्रुव को । Manual evaluation

Geo\_ncert\_6stnd\_2ch\_0049 ये अर्धवृत्त हैं तथा उनके बीच की दूरी ध्रुवों की तरफ बढने पर घटती जाती है एवं ध्रुवों पर शून्य हो जाती है, जहाँ सभी देशांतरीय याम्योत्तर आपस में मिलती हैं । Manual evaluation

## Connectives List:

* SIMPLE\_CONNECTIVES = ['और', 'एवं' ,'इसलिए', 'क्योंकि', 'जबकि' ,'तथा', 'ताकि', 'मगर', 'लेकिन', 'किंतु', 'परंतु', 'फिर', 'या', 'तथापि','नहीं तो', 'व', 'चूंकि', 'चूँकि', 'वरना','अन्यथा', 'बशर्तें', 'हालाँकि', 'इसीलिये', 'इसीलिए' , 'इसलिए', 'अथवा', 'अतः', 'अर्थात्', 'जब', 'तो']
* COMPLEX\_CONNECTIVES =

{ 'चूँकि' : ['अतः'],

'जब' : ['तब', 'तो'],

'अगर' : ['तो', 'तब'],

'यदि' : ['तो'],

'यद्यपि' : ['फिर भी'], }

## Major modules in sentence simplification:

1. validate\_sentence(sentence) :
   1. This function ensures that input sentence is-
   2. Not empty
   3. Is not only numeric values
2. sanitize\_input(sentence):
   1. This function ensures that input sentence is-
   2. converted to WX convention
   3. then each word is cleaned to remove dZ, jZ, DZ
   4. convert each word back to hindi
   5. replace full stop if any with a space and a poornaviram at the end of sentence
3. breakAllPairedConnective(sentence, allPairedConnectiveList, manual\_evaluation):
   1. This is a recursive function which calls breakPairConnective(sentence, manual\_evaluation) to break the input sentence based on the list of paired connectives.

It runs until all the sub parts of the input sentence cannot be further broken down by the list of paired connectives.

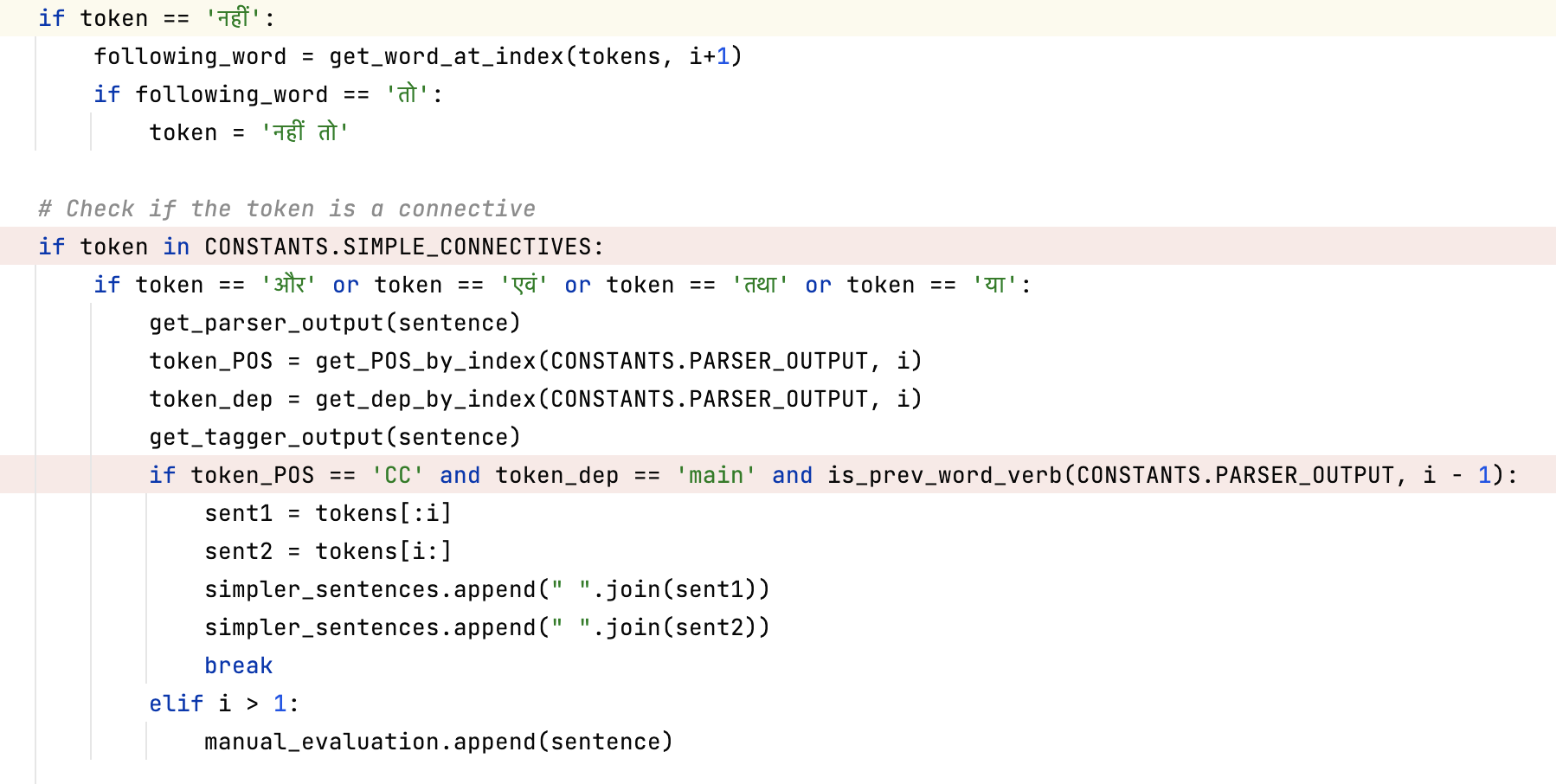
* 1. allPairedConnectiveList - contains the sentences after being processed upon by breakAllPairedConnective.
  2. Manual\_evaluation - contains the sentences which had a paired connective but could not be broken due to some unfulfilled condition.

1. breakPairConnective(sentence, manual\_evaluation):
   1. The sentence is broken down into tokens.
   2. We iterate on the list of tokens. If we find a token which exists as a key in the paired connective list, we check the presence of its counterpart (value) in the sentence.
   3. We find the index of its counterpart in the sentence, and check if the term just before the counterpart is a verb (VM or VAUX as tagged by pos-tagger)
      1. If yes, we break the sentence on the counterpart, such that it is included in the second half. We drop the token identified as the key from the first half.
      2. If no,then we tag the sentence for manual evaluation.
2. breakAllSimpleConnective(sentence, allSimpleConnectiveList, manual\_evaluation):
   1. This is a recursive function which calls breakSimpleConnective(sentence, manual\_evaluation) to break the input sentence based on the list of simple connectives.
   2. It runs until all the sub parts of the input sentence cannot be further broken down by the list of simple connectives.
   3. allSimpleConnectiveList - contains the sentences after being processed upon by breakSimpleConnective. This is the resultant list after all possible breakdown.
   4. Manual\_evaluation - contains the sentences which had a simple connective but could not be broken due to some unfulfilled condition.
3. breakSimpleConnective(sentence, manual\_evaluation):
   1. The sentence is broken down into tokens.
   2. We iterate on the list of tokens. If we find a token which exists in the list of simple connectives, we call the pos-tagger to find if the previous term is verb(VM or VAUX as tagged by pos-tagger)
      1. If yes, we break the sentence on the connective, such that it is included in the second half.
      2. If no,then we tag the sentence for manual evaluation.
   3. If no connective was found, we keep iterating over the list of tokens.

## Exceptional flow:

For simple connectives, some of them are handled in a different way -

1. 'नहीं तो':
   1. If नहीं exists in the sentence, we check if it is followed by तो. If yes then check if the previous term of नहीं is a verb and then break the sentence accordingly.
2. 'और', 'एवं', 'तथा', 'या':
   1. If any of the above connectives is found, we call the parser output and check if its POS\_tag is CC and its dependency is main.
      1. If yes then we break the sentence otherwise tag it for manual evaluation.



## Performance Considerations:

Some known limitations are -

1. For all the subparts of the sentence we call the pos tagger.
2. For large inputs, the script takes time to generate output.

## Scope of improvement:

1. Optimise the pos tagger calls to reduce the overall time taken by the program for execution.