Stanford CoreNLP OpenIE

[What is OpenIE? 1](#_Toc78553835)

[System requirements 2](#_Toc78553836)

[Java 2](#_Toc78553837)

[Input 2](#_Toc78553838)

[Passive sentences 2](#_Toc78553839)

[Adjectival Clause (acl) 3](#_Toc78553840)

[Adverbial Clause (advcl) 4](#_Toc78553841)

[Verb Semantics 4](#_Toc78553842)

[Verb and Preposition 4](#_Toc78553843)

[Verb, Preposition, and Object 5](#_Toc78553844)

[Future Developments 6](#_Toc78553845)

[Output 6](#_Toc78553846)

[References 7](#_Toc78553847)

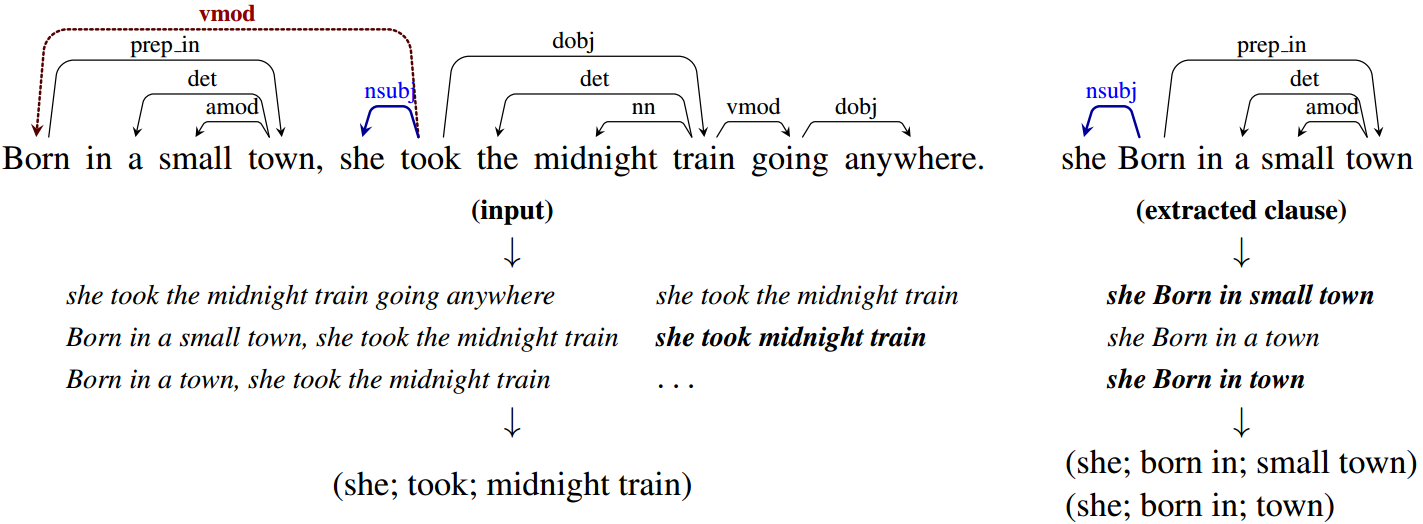
# What is OpenIE?

From the Stanford CoreNLP website (<https://nlp.stanford.edu/software/openie.html>) we read:

Open information extraction (open IE) refers to the **extraction of relation tuples**, typically binary relations, from plain text, such as (Mark Zuckerberg; founded; Facebook). The central difference from other information extraction is that the schema for these relations does not need to be specified in advance; typically the relation name is just the text linking two arguments. For example, Barack Obama was born in Hawaii would create a triple (Barack Obama; was born in; Hawaii), corresponding to the open domain relation was-born-in (Barack-Obama, Hawaii). This software is a Java implementation of an open IE system described in the paper:

Gabor Angeli, Melvin Johnson Premkumar, and Christopher D. Manning. Leveraging Linguistic Structure for Open Domain Information Extraction. In *Proceedings of the Association of Computational Linguistics (ACL)*, 2015.

The system first splits each sentence into a set of entailed clauses. Each clause is then maximally shortened, producing a set of entailed shorter sentence fragments. These fragments are then segmented into OpenIE triples, and output by the system. An illustration of the process is given for an example sentence below:



## System requirements

OpenIE requires Java 8+ to be installed, and generally requires around 50MB of memory in addition to the memory used by the part of speech tagger and dependency parser (and optional named entity recognizer). Stanford CoreNLP recommends running java with around 1gb of memory (2gb if using NER) to be safe (i.e., java -mx1g).

### Java

**You will need to download and install Java. See the TIPS file TIPS\_NLP\_Java download install run.pdf**

# Input

The Stanford CoreNLP OpenIE takes in input a single text file or a set of text files in a directory.

## Passive sentences

The SVO script converts passive sentences to active ones. In the dependency parser ((<https://nlp.stanford.edu/software/nndep.html>) of Stanford CoreNLP, the object in passive sentence would be tagged as “﻿nsubj:pass,” and the POS tag (<https://stanfordnlp.github.io/CoreNLP/pos.html>) of the verb would usually be “VBN” (although sometimes the Stanford CoreNLP incorrectly tags a passive tense as an adjective). When the passive subject (the agent) of the sentence is not available, the script will insert the default unknown subject as **Someone?**

**Negation**

The meaning of a sentence captured by the simple SVO structure changes drastically in the presence of negation (e.g., “I will give you a present” vs. “I will not give you a present”). The NLP Suite SVO algorithm captures a variety of negation forms (“no,” “not,” “n’t,” “seldom,” “never,” “hardly,” “neither,” “nor”).

By recursion with deep-first search through a token’s adverbial modifier (dep: “advmod”), determiner (dep: “det”), and coordinating conjunction (dep: “cc” and “cc:preconj”) (https://universaldependencies.org/docs/u/dep/), the negation would be detected if the searched token is in a list of words with meaning of negation.

**Predicate Nominative**

From <https://www.thesaurus.com/e/grammar/predicate-nominative-vs-predicate-adjectives/> we read: “In general, a predicate completes a sentence by providing information about what the subject is or does. The subject of a sentence is who or what is doing the action. The predicate explains the action. There’s often a linking verb (like is or became) in between the two. A predicate nominative is a noun that completes the linking verb in a sentence. Predicate adjectives complete the linking verb by describing the subject of a sentence.”

“For example, ‘Ben is a fireman’ can read ‘Ben equals a fireman’ without changing the point. In this example, fireman is the predicate nominative.”

“For example, in ‘Jack is handsome,’ Jack is the subject, and handsome is the predicate adjective.”

In other words, when the lemma (<https://stanfordnlp.github.io/CoreNLP/lemma.html>) of a verb is “be”, it is usually not followed by an object but by a predicate nominative. In the sentence “He is a traitor”, “traitor” is a predicate nominative. In a subject-verb-predicate collocation, the object is the syntactic head, which governs the subject (dep: “nsubj”) and the link verb (dep: “cop” or “aux”. While searching for the verb among the tokens governed by the object, the dep of “cop” is searched first, and if that dep is missing, “aux” is then searched).

**@Claudio This would allow to correctly extract an SVO such as @ example that would otherwise be missed or misclassified? Claude, let’s try to be specific.**

**Basically, we need a to parse a sentence via OpenIE, show what SVO looks like WITHOUT your handling of the json file and show the new SVO after our json handling. This would make very clear the approach you are taking.**

## Adjectival Clause (acl)

From <https://universaldependencies.org/docs/u/dep/acl.html> we read: “[The adverbial clause] acl stands for finite and non-finite clauses that modify a nominal. The acl relation contrasts with the advcl relation, which is used for adverbial clauses that modify a predicate. The head of the acl relation is the noun that is modified, and the dependent is the head of the clause that modifies the noun. A verb as a clausal modifier may not be the syntactic head of the subject as usual, so their subjects need extracting differently. In this sentence “Their hair was the same length, their lipstick the same color, their bodies curving in the same way beneath their same uniform.” (*Ghostwritten*, David Mitchell, 1999), “bodies” is the subject of “curving”, and “curving” is also in a clause that modifies “bodies”. In the output of dependency parser, “bodies” is the syntactic head of “curving”, which is its clausal modifier (dep: “acl”). Nouns that governs a verb with the dep of “acl” is recognized as its subject. When a noun governs a clausal modifier verb, that noun will substitute the that verb’s default subject (“Someone?”) in the output.

**@Claudio This would allow to correctly extract an SVO such as @ example that would otherwise be missed or misclassified? Claude, let’s try to be specific.**

**Basically, we need a to parse a sentence via OpenIE, show what SVO looks like WITHOUT your handling of the json file and show the new SVO after our json handling. This would make very clear the approach you are taking.**

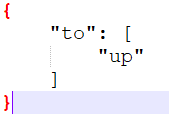
## Adverbial Clause (advcl)

From <https://universaldependencies.org/docs/u/dep/advcl.html>, we read: “An adverbial clause modifier is a clause which modifies a verb or other predicate (adjective, etc.), as a modifier not as a core complement. This includes things such as a temporal clause, consequence, conditional clause, purpose clause, etc. The dependent must be clausal (or else it is an advmod) and the dependent is the main predicate of the clause.”

Similarly, if a verb is an adverbial clause modifier (dep: “advcl:xxx”) which modifies a verb, it may also not govern its subject. For example, in the sentence “she would have to take refuge in my arms to escape her terror.” (*Strange Pilgrims*, Gabriel Garcia Marquez, 1992), “escape” (dep: “advcl:to”) modifies “take”, and the subject of “take” is “she”. In this sentence, “she” is also the subject of “escape”, but neither is the syntactic head of the other. In this case, the subject of the verb which is the syntactic head of the adverbial clause modifier will be the default subject (unless the adverbial clause modifier is the syntactic head of its own subject). If the POS tag (<https://stanfordnlp.github.io/CoreNLP/pos.html>) of that verb is “VBN”, that default subject will be its object.

**@ Claudio, does this go here? Is it this file?** advmod\_obl\_json.txt

**Can it be expanded?**



**@Claudio This would allow to correctly extract an SVO such as @ example that would otherwise be missed or misclassified? Claude, let’s try to be specific.**

**Basically, we need a to parse a sentence via OpenIE, show what SVO looks like WITHOUT your handling of the json file and show the new SVO after our json handling. This would make very clear the approach you are taking.**

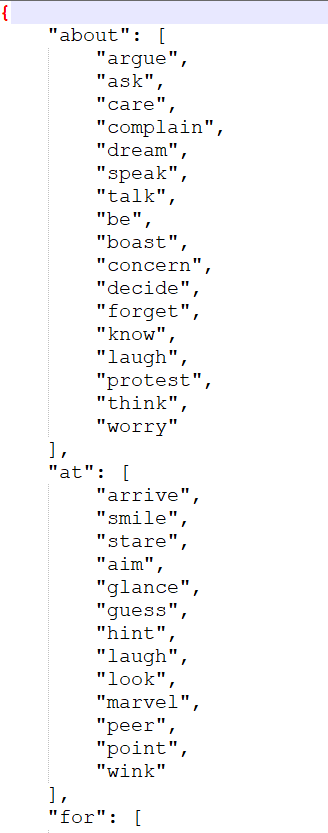
## Verb Semantics

Verbs can change their meaning depending upon their wider collocations or combinations of words, namely, prepositions they are linked to (e.g., “sit up” and “sit down”) and combinations of verb+noun+preposition (e.g., “take care of” which has a different meaning from simply “take”).

### Verb and Preposition

In a verb-preposition-object collocation (for example: “he puts on the cover”), the object’s dep is “obl:preposition” (in “he puts on the cover,” the dep of “cover” would be “obl:on”). When multiple tokens governed by the verb have the dep of “obl:preposition”, we use the txt file (**verb\_prep\_json.txt**) in the “OpenIE” folder in the lib path, based on common verb-preposition collocations (<https://7esl.com/verb-preposition-combinations/>; <https://eslgold.com/grammar/verb_preposition_collocations/>), to extract the correct object: if the verb’s lemma is in the keys of the json and the preposition precedes one “obl” token is in that key’s value----a list of prepositions, that token would be selected as the object. That txt file of the json can be manually edited if there’s any new verb-preposition combination.

**verb\_prep\_json.txt**

****

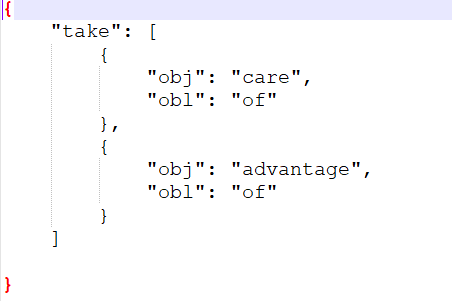
**@Claudio This would allow to correctly extract an SVO such as @ example that would otherwise be missed or misclassified? Claude, let’s try to be specific.**

**Basically, we need a to parse a sentence via OpenIE, show what SVO looks like WITHOUT your handling of the json file and show the new SVO after our json handling. This would make very clear the approach you are taking.**

### Verb, Preposition, and Object

A verb may be part of idiomatic expressions that affect its meaning (e.g., the verb “take,” as in “He took a book,” will have a completely different meaning in such idiomatic use “take care of”, e.g., “He offered to take care of the check at the restaurant”). For a phrase that plays the role of a single verb (such as “be in charge of”, “be responsible for”, “take care of”), in the output of token-based dependency parser it will not be one idiomatic ensemble, and in OpenIE they needs extracting as one verb. Some of these phrases are in regular dependency patterns: in “take care of” and “take advantage of”, “care” / “advantage” has the dep of “obj” and is governed by “take”, but the dep of the real object should be “obl:of”. A json was constructed to help extract these phrases (the txt file “**v\_obj\_obl\_json.txt**” in the “OpenIE” folder in the lib path). That json’s keys are verbs (for example: “take”) and each value is a list of dictionaries with two values: the first key is “obj”, and the second key is “obl” (for example: {“obl”: “care”, “obl”: “of”}). If one verb’s lemma is in that json’s keys, its object matches the first value in a dictionary in that key’s value, that verb governs a token whose dep is “obl:preposition”, and preposition matches that dictionary’s second value, then that verb, it’s object, and the preposition in the dep “obl:preposition” will be extract as one verb, while the token whose dep is “obl:preposition” will be its object. That txt file of the json can be manually edited when there’s need to add another phrase that plays the role of a single verb and has a similar dependency pattern: verb + object + preposition, and the real object of this verb phrase is governed by the verb with the dep “obl:preposition”.

**v\_obj\_obl\_json.txt**

****

**@Claudio This would allow to correctly extract an SVO such as @ example that would otherwise be missed or misclassified? Claude, let’s try to be specific.**

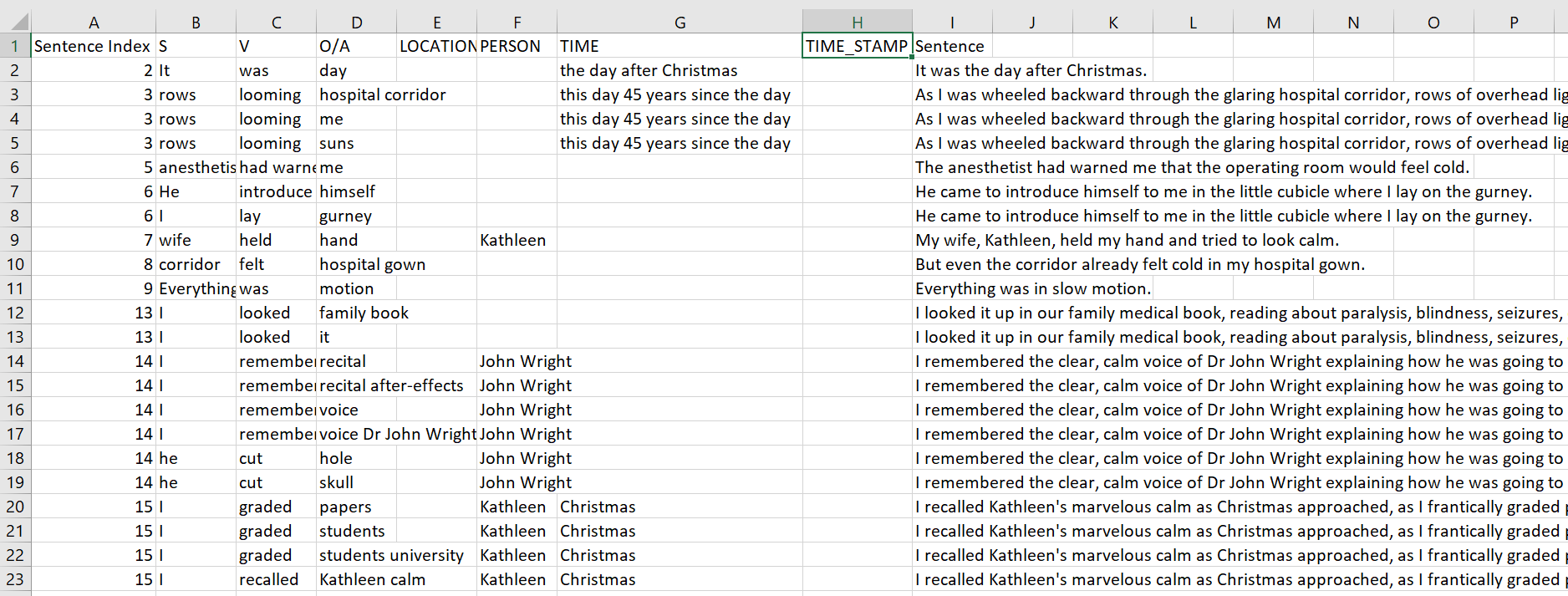
**Basically, we need a to parse a sentence via OpenIE, show what SVO looks like WITHOUT your handling of the json file and show the new SVO after our json handling. This would make very clear the approach you are taking.**

### Future Developments

For phrases with different or completely irregular dependency patterns, one potential method to extract these phrases is to collect enormous English sentences and ask volunteers to label the index of tokens that are subjects, verbs, or objects. If there are multiple tokens right next to each other are labeled as one verb, a graph of these tokens’ dependency relationships would be generated by breadth-first search, and specific json can be generated from these graphs.

# Output

The NLP Suite uses Stanford CoreNLP OpenIE in the SVO script. Contrary to the NLP Suite Stanford CoreNLP parsers, the OpenIE script does not produce in output a CoNLL table. Just a csv file with the SVO triplets and results of negation detection.



**@Claudio where is negation?**

# References

Klein**,** Dan and Christopher D. Manning. 2003. “Accurate Unlexicalized Parsing.” *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pp. 423-430.

TIPS\_NLP\_Java download install run.pdf

TIPS\_NLP\_Stanford CoreNLP parser.pdf

TIPS\_NLP\_Stanford CoNLL table.pdf