1 Named Entity Recognition (NER)

A named entity refers to a sequence of words that correspond to a specific entity in the real world.[1]
Goal: Find (1) and Classify (2) entities in the text. Consider this example:[2]

1. The body of Julián Carillo, a member of the Alianza Sierra Madre organisation, was found with multiple bullet wounds in the mountains of Chihuahua state on the evening of 24 October. He was killed a few weeks after his community of Coloradas de la Virgen registered opposition to a mining concession that they say was located in their territory without their permission, according to Amnesty International.

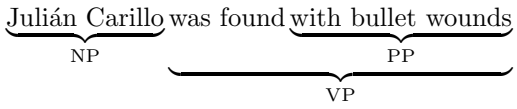
2. The body of ⟨*Person*⟩ Julián Carillo ⟨/*Person*⟩, a member of the ⟨*Organization*⟩ Alianza Sierra Madre ⟨/*Organization*⟩ organisation, was found with multiple bullet wounds in the mountains of ⟨*Location*⟩ Chihuahua state ⟨/*Location*⟩ on the evening of ⟨*Date*⟩ 24 October ⟨/*Date*⟩. ⟨*Person*⟩ He ⟨/*Person*⟩ was killed a few weeks after his community of ⟨*Location*⟩ Coloradas de la Virgen ⟨/*Location*⟩ registered opposition to a mining concession that they say was located in their territory without their permission, according to ⟨*Organization*⟩ Amnesty International ⟨/*Organization*⟩.[3]

Question: How can we find and classify entities?
Answer: Broadly, existing models can be summarized under the umbrella of two techniques, **Rule-Based Methods** and **Sequence Models**.

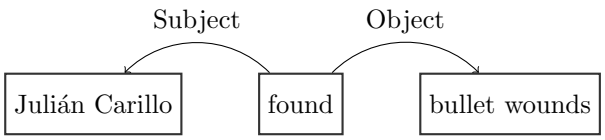
1.1 Rule-Based Methods

First each token is converted into a set of features. A non-exhaustive list of **features** (associated with individual tokens) is given below:

- *Surface value:* The token itself constitutes the most basic feature.[4]
- *Orthography:* Characteristics such as capitalization, punctuation, or spelling.
- *Parts-of-speech (POS)* tags: Each token is assigned to part of a speech such as a noun, verb, adjective, adverb, pronoun, conjunction, preposition, and article (many redefined categorizations of these basic types).
- *Parsing:* A parser extracts a hierarchical structure from each sentence in the form of a *parse tree*, in which the lower-level subtrees group the POS tags into syntactically coherent phrases like noun (NP), verb (VP) or preposition (PP) *phrases*. E.g.:



- *Dependency graphs/paths*: A dependency graph captures the dependencies between tokens.[5] For instance, the subject and object of a verb depend on it:



- *N-Grams:* An n-gram is a probability distribution over a sequence of n tokens that learns the statistical likelihood of the nth token following a sequence of $n-1$ tokens. For example, a bigram ($n = 2$) would be {bullet wounds}.

[1] Originally an entity referred to a *name* such as a person, location, or organization, but entities commonly also comprise for instance dates.
[2] https://www.theguardian.com/environment/2018/oct/26/latest-land-defender-cements-mexicos-deadly-reputation
[3] Entities often comprise multiple tokens (= "words") → mark *starting point*, *ending point* (here "/"), and the *class/label* of the entity.
[4] For details on Tokenization and Pre-Processing see Appendix A.1.
[5] Dependency graphs can be constructed much more efficiently than parse-trees.

Subsequently a set of "if then" rules $\mathcal{R} = \{R_1 ... R_m\}$ are mined from the data that take the following (general) form:

$$\text{Contextual Pattern} \quad \Rightarrow \quad \text{Action}$$

The LHS of the rule (the *antecedent*) is a combination of conditions (*conjuncts*) corresponding to features associated with a sequence of tokens. A rule is triggered if a sequence of tokens in the text matches the logical condition in the antecedent.[6] For instance a rule can take the form:

$$(\text{Dictionary-Class} = \textit{Titles}, \text{Orthography} = \textit{FirstCap}) \quad \Rightarrow \quad \text{Person}$$

Automated training algorithms for rule-based systems learn from labeled training data by iteratively adding rules that have good precision and coverage with respect to the tagged entities. E.g.:

$\mathcal{R} = \{\}$
repeat;
    Select a tagged entity E in the training data that is uncovered;
    Create a rule R that covers E;
    $\mathcal{R} = \mathcal{R} \cup \{R\}$
until no more uncovered entities;

Existing algorithms for rule-generating methods are either *top-down* or *bottom-up*.[7]

1.2 Sequence Models

Goal: Label **each** token for its entity class or "other (O)" by making use of extracted features and observed data. Consider the following example, where we want to find the label for Carillo at position 0 ("decision point") of the sequence:

-4	-3	-2	-1	0	+1	+2	+3	+4	+5	+6	+7	...
O	O	O	Person	???	O	O	O	O	ORG	ORG	ORG	...
The	body	of	Julián	Carillo	a	member	of	the	Alianza	Sierra	Madre	...

The features considered for labeling could be for example:

Current Token x_0	Carillo
Previous Token x_{-1}	Julián
Next Token x_{+1}	a
Previous label y_{-1}	Person
Previous labels $y_{-1} - y_{-2}$	Person-O
POS (x_0)	Noun
POS (x_{-1})	Noun
FirstCap (x_0)	True
...	....

Two common sequence models are the *Maximum Entropy Markov Model* and the *Conditional Random Fields*. While the former classifier makes a single decision at a time on evidence from observations and **previous** labels, the latter considers **both**, the labels occurring before and after the decision point.[8]

[6] In context of training instances, it is said that such a rule *covers* the training instance.
[7] See Appendix A.2 for details on the usually applied *bottom-up* rule generation.
[8] For more details on *Maximum Entropy Markov Models* see Appendix A.3.

2 Relationship Extraction

In our example we are interested in "Mining-Assassinations" and would like to extract the following relation:

Mining-Assassinations

Target	Julián Carillo
Location	Chihuahua state
Date	24 October 2018[9]
Company	NONE

The former is a fairly complex relation, so for the sake of simplicity, we will focus on the relation:

Target-Location(Julián Carillo, Chihuahua state)

In general, relation extractors can be grouped into four categories:

1. Hand-written patterns
2. Supervised machine learning
3. Semi-supervised machine learning
4. Unsupervised machine learning

In the following we will focus on supervised and semi-supervised machine learning, as both are particularly suitable for our purposes.

2.1 Supervised Relation Extraction

The steps in classifying relations can be summarized as follows:

1. Preliminary step: Choose set of relations to extract and the set of relevant named entities
2. Find and Label Data
   (a) Choose representative corpus
   (b) Label named entities
   (c) Find all pairs of named entities (usually in same sentence)
   (d) Build a boolian classifier that makes a "Yes/No" decision if the two entities are related[10]
   (e) If "Yes", (hand-) label the relation
3. Train a classifier on the training set

Let's look at a specific example, where we want to extract the relation between Julián Carillo and Chihuahua state from the sentence:

The body of $\underbrace{\text{Julián Carillo}}_{\text{Entity 1}}$ was found with multiple bullet wounds in the mountains of $\underbrace{\text{Chihuahua state}}_{\text{Entity 2}}$.

First, we extract features. In particular, features extracted from within the entity are referred to as *entity features*, whereas features surrounding or between the argument entities are called *contextual features*. A potential list of extracted features is presented in the table on the next page.[11]
After extracting all features, one can choose any classifier (MaxEnt, Naive Bayes, SVM,...) that will subsequently be trained on the training set, (tuned on the dev set) and tested on the test set. In particular, the supervised relation extraction classifier is commonly **evaluated** based on the following measures:

$$\text{Precision (P)} = \frac{\#\text{ of correctly extracted relations}}{\text{Total }\#\text{ of extracted relations}}$$

$$\text{Recall (R)} = \frac{\#\text{ of correctly extracted relations}}{\text{Total }\#\text{ of gold relations}}$$

$$\text{F}_1 = \frac{2 \times P \times R}{P + R}$$

[9] Important: the text itself does not contain the year of the assassination, but it rather has to be inferred from the date of the article.
[10] The Steps (c)-(d) are incorporated instead of purely hand-labeling for efficiency gains.
[11] For a *Python* implementation and details of feature extraction see Section 3 and Appendix A.4.

Entity-Based Features	
Entity_1 type	Person (PERS)
Entity_1 headword	Carillo
Entity_2 type	Location (LOC)
Entity_2 headword	state
Concatenated types	PERSLOC
Bigrams in Entity_1 & Entity_2	{Julián Carillo, Chihuahua state}
Context (Word-based) Features	
Between entity bag of words	{was, found, with, multiple, bullet, wounds, in, the, mountains, of }
Words(s) before Entity_1	{the, body, of}
Word(s) after Entity_1	{was, found}
Words(s) before Entity_2	in
Trigger words	{bullet, kill, shoot, assassinate...}
Gazeteer	{Country name list, Admin region name list, lakes, ...}
Syntactic Features	
Basic syntactic chunk path	NP $\rightarrow$ PP $\rightarrow$ VP $\rightarrow$ PP $\rightarrow$ PP $\rightarrow$ PP
(Typed-) Dependency Path	Julián Carillo $\leftarrow_{comp}$ body $\leftarrow_{subj}$ found $\rightarrow_{comp}$ bullet wounds

Summary: While one can achieve high accuracies with *supervised relation extraction* if the hand-labeled training data is large enough and the test sample similar to the training data, the labeling of the training set is often too time consuming $\Rightarrow$ resort to *semi-supervised relation extraction.*

2.2 Semi-Supervised Relation Extraction

For the remainder of this section, let's assume we do not have any labeled training and test set.
First, consider the case of **Relation Bootstrapping**. Assume that we have have a few seed tuples with information on the names of land and environmental defenders as well as on the countries they were assassinated in:[12]

⟨Edmilson Alves da Silva, Brazil⟩
⟨Cecilia Coicue, Colombia⟩
⟨Alejandro Nolasco Orta, Mexico⟩
⟨Sikhosiphi "Bazooka" Rhadebe, South Africa⟩
⟨Roger Gower, Tanzania⟩

The iterative algorithm of Relation Bootstrapping can be summarized as follows:

Iterate:

1. Find sentences containing these pairs. For instance grep google for the environments of the seed tuples.
2. Look at the context between or around the pair and generalize the context to create patterns/features. For instance:
   Cecilia Coicue was assassinated in the capital of Columbia

   X was assassinated in the capital of Y

   Cecilia Coicue was shot the near his home in Bogota, Columbia

   X was shot the near his home in Y
3. Use these patterns/features to grep for new tuples

[12] For further details see "Defenders of the Earth Report: https://www.globalwitness.org/en/campaigns/environmental-activists/defenders-earth/

Second, let's turn to **Distant Supervision**, which combines bootstrapping with supervised learning. In specific, instead of using just a small number of seeds, we use a large database with a large number of seed examples and create a lot of features from these examples. Furthermore, instead of iterating, we take the extracted features and build a supervised classifier. The distant supervision paradigm can be summarized as follows:

- Like *supervised* classification:
  - Learn a classifier with large number of features
  - that is supervised by detailed hand-created knowledge
  - but does not require iteratively expanding patterns (c. Bootstrapping)
- Like *unsupervised* classification:
  - Use very large amounts of unlabeled data
  - Not sensitive to genre issues in training corpus, as it is the case for supervised classifiers

For instance, the algorithm of distantly supervised learning of relation extraction could take the form of:

1. For each relation	**Target-Location**
2. For each tuple in big database	⟨Edmilson Alves da Silva, Brazil⟩ ⟨Cecilia Coicue, Colombia⟩ ...
3. Find sentences in large corpus with both entities	Cecilia Coicue was assassinated in the capital of Columbia Edmilson da Silva was murdered just near his home in Rio de Janeiro ...
4. Extract frequent features/patterns (f) (POS, entities, surface values, dependency paths, ...)	PERS was assassinated in the capital of LOC PERS was murdered just near his home in LOC ...
5. Train supervised classifier using thousands of features[13]	$\mathrm{P}(\text{Target-Location}\|f_1, f_2, f_3, ..., f_{30000})$

Important: In order to construct the classifier, we need **both** positive and negative training instances!
Finally, since there exists no gold-standard set of correct instances for the examples described above, we are not able to calculate Precision and Recall. How can we then evaluate the performance of our semi-supervised relation extractors?
We can calculate the approximate precision, i.e. we draw a random sample of relations from the output and check the precision manually:

$$\hat{\mathrm{P}} = \frac{\text{\# of correctly extracted relations in the random sample}}{\text{Total \# of extracted relations in the random sample}}$$

For instance, we might draw a random sample of 100 from the top 1000 highest probability ranked relations.
Note: In the examples outlined above, it is however not possible to evaluate Recall.

[13] I.e, calculate the probability of "Target-Loacation" relation at a particular datapoint conditional on all features $(f_1, f_2, f_3, ..., f_n)$ from a sentence.

3 Feature and Relation Extraction with Spacy

In this section we will look at how to extract features from text with the help of the *SpaCy* library in *Python* and write a pattern by hand to extract the **Target-Location** relation. First, we investigate the tokenization, POS tags, dependency paths and named entities for the simple example sentence:

Julián Carillo was shot in the mountains of Chihuahua state.

The code and output is presented in the following:

```
!pip install spacy              # install spacy
!python -m spacy.en.download    # download language model

# import all libraries that will be used
import spacy
import pandas as pd
from spacy import displacy
# instantiate a language model
nlp = spacy.load('en')           # or spacy.en.English()

# Define our example text from above
supervision_example = u"Julian Carillo was shot in the mountains of Chihuahua state."

#create a document
doc = nlp(supervision_example)

# For each token in our text doc we apply the following SpaCy methods
attrs = map(lambda token: {
                    "token":token
                  , "part of speech":token.pos_
                  , "Dependency" : token.dep_
                  , "Entity": token.ent_type_}
                  , doc)

# Convert the output to a DataFrame and inspect it
pd.DataFrame(list(attrs))
```

Dependency	Entity	part of speech	token
compound	PERSON	PROPN	Julián
nsubjpass	PERSON	PROPN	Carillo
auxpass		VERB	was
ROOT		VERB	shot
prep		ADP	in
det		DET	the
pobj		NOUN	mountains
prep		ADP	of
compound		PROPN	Chihuahua
pobj		NOUN	state
punct		PUNCT	.

While "Julián Carillo" was correctly identified as entity type "PERSON", "Chihuahua" was not identified as an entity and in particular as a state ("GPE").[14] This illustrates that even pre-tuned algorithms may lead to false predictions, but we can get rid of these hick-ups by training a model for our purposes.
In the next step, it might be useful to retokenize by recognizing so called *noun chunks*, i.e. token spans that are "logically connected" with each other. For example, instead of having two separate tokens "Chihuahua" and "state" or "Julián" and "Carillo", we would like to have one token "Chihuahua state" and "Julián Carillo":

```
spans = list(doc.ents) + list(doc.noun_chunks)
for span in spans:
    span.merge()     # method to merge tokens within span
```

[14] A detailed list of POS, entity and dependency label definitions in *SpaCy* is provided in Appendix A.4.

Further, we would like to redefine the entity type for "Chihuahua state", so that next time the model would recognize them correctly:

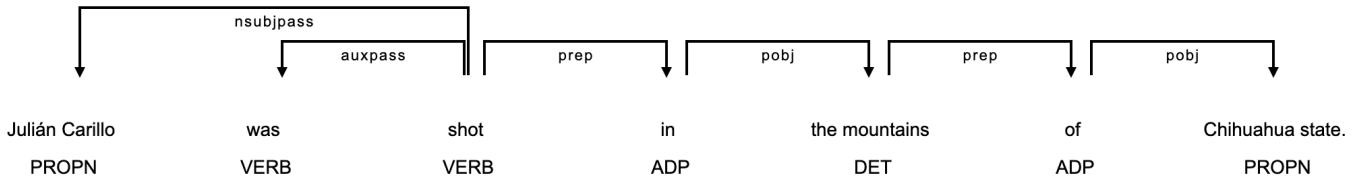
```
# redefine entity types for "Julian Carillo" and "Chihuahua state"
doc[6].ent_type_ = 'GPE'

attrs = map(lambda token: {
                      "token":token
                    , "part of speech":token.pos_
                    , "Dependency" : token.dep_
                    , "Entity": token.ent_type_}
                    , doc)
pd.DataFrame(list(attrs))
```

Dependency	Entity	Part of Speech	Token
nsubjpass	PERSON	PROPN	Julián Carillo
auxpass		VERB	was
ROOT		VERB	shot
prep		ADP	in
pobj		DET	the mountains
prep		ADP	of
pobj	GPE	PROPN	Chihuahua state
punct		PUNCT	.

For better a understanding of dependency paths and its interdependencies with POS tags, the dependency tree for our example text is depicted below:

```
# display dependency path as a figure
displacy.render(doc, style='dep', jupyter = True, options = {'compact': True})
```



Finally, we will use *SpaCy* to extract the **Target-Location** relation by applying a hand-written pattern. In this example, we require a trigger word such as "kill", "assassinate", or "shoot" and that a person was the direct object ("dobj") or passive subject ("nsubjpass") of the verb. Moreover, if there exists a location of any kind ("LOC", "GDP"), we require that the location is a proper noun ("PROPN").

```
def target_location_relation(doc):
    columns = ['Target', 'Location']
    df = pd.DataFrame(columns=columns)
    count = -1
    for token in doc:
        if token.lemma_ == 'shoot' or 'kill' or 'assassinate':
            for child in token.children:
                if child.dep_ == 'nsubjpass' or 'dobj':
                    if child.ent_type_=='PERSON':
                        df = df.append({'Target': child}, ignore_index=True)
                        count += 1
                if child.dep_ == "pobj":
                    if child.ent_type == "LOC" or "GDP":
                        if child.pos_ == "PROPN":
                            df.at[count, 'Location'] = child
    return(df)

target_location_relation(doc)
```

Target	Location
Julián Carillo	Chihuahua state

In reality, we would need to think of further and more specific restrictions to filter out the events of interest in a large text corpus (for instance millions of newspaper articles). However, it is near to impossible to think of all potential cases and exceptions. Hence, it is necessary to apply the machine learning techniques outlined in Section 2 to extract the appropriate patterns.

A Appendix

A.1 Tokenization and Pre-Processing

A *token* is a contiguous sequence of characters with a semantic meaning that allows for repetitions, whereby no additional processing, e.g. stemming (s. below), has been done.
Tokens can subsequently be transformed into *terms* (with specific frequencies). **Intuition:** Highly frequent tokens are (often) not discriminative → the following (common) "tools" are applied:

- *Stop-word removal* or *inverse document frequency normalization.*
- *Case folding*, i.e. process of converting to the true case (*"truecasing"*).
- *Hpyhens*, i.e. dictionaries of commonly adjacent words that (1) should be hyphenated and (2) should not be hyphenated.
- *Usage-based Consolidation*, i.e. common spelling differences such as "color" and "colour"
- *Stemming* process of consolidating related words with the same root. Common techniques are *Semmi-automatic look-up tables, lemmatization, (suffix stripping).*

A.2 Bottom-Up Rule Generation

Bottom-up rule generation methods start with very specific rules and then generalize it so as to allow the rule to cover more positive examples (higher precision on the training data, but do not generalize as well to the test data). The broad approach can be summarized as follows:

$\mathcal{R} = \{\}$
repeat;
    Select a tagged entity E in the training data that is uncovered;
    Create a rule R that covers E by:
        Starting with the most specific rule covering the entity
        and successively generalizing this specific rule;
    $\mathcal{R} = \mathcal{R} \cup \{R\}$
    Remove instances covered by R;
until no more uncovered entities;

For instance a series of generalizations may be:

$$\begin{aligned} (\text{Token} = \text{"Ms."}, \text{Token} = \text{"Smith"}) &\Rightarrow \text{Person} \\ (\text{Token} = \text{"Ms."}, \text{Orthography} = \textit{FirstCap}) &\Rightarrow \text{Person} \\ (\text{Dictionary-Class} = \textit{Titles}, \textit{FirstCap}) &\Rightarrow \text{Person} \end{aligned}$$

A.3 Maximum Entropy Markov Models

Maximum entropy Markov models *directly model* the probability of labeling based on the states ($\rightarrow$ *discriminative* model).
Let $\bar{x}_{i-q}^{i+q}$ denote the segment $(x_{i-q}...x_{i+q})$ of the sequence of tokens $\bar{x}$ from the $(1-q)$th position to the $(1+q)$th position. Similarly, let $(y_{i-p}...y_{i-1})$ be the segment of the sequence of labels $\bar{y}$. One can now extract features from the neighborhood of the tokens in the ith position and the history of labels ($\rightarrow$ labels up to the $(i-1)$th position are inferred but labels including and after position i are unkown).
E.g. $p = q = 1$ and the token x_i follows "Ms" at $x_i - 1 \rightarrow$ the binary feature $f_1(y_i, y_{i-1}, \bar{x}_{i-q}^{i+q})$ can be defined as:

$$f_1(y_i, y_{i-1}, \bar{x}_{i-q}^{i+q}) = \begin{cases} 1 & \text{if} \quad [y_{i-1} == \text{Person}] \quad \text{AND} \quad [x_{i-1} == \text{"Julián"}] \quad \text{AND} \quad [\text{FirstCap}(x_i) == True] \\ 0 & \text{otherwise} \end{cases}$$

Important: An implicit restriction is placed on the probabilistic modeling of y_i, i.e. it depends only on the labels $(y_{i-p}...y_{i-1})$ occurring before, but not after y_i. Note however that the tokens occurring before and after y_i can be used.

A.4 SpaCy Definitions

A detailed description of (universal) definitions applied in *SpaCy* can be found.[15]

Named Entities: The entity types differentiated in *SpaCy* are:

TYPE	DESCRIPTION
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first", "second", etc.
CARDINAL	Numerals that do not fall under another type.

Part-of-Speech: The universal (language independent) part-of-speech (POS) tags are listed below:

POS	DESCRIPTION	EXAMPLES
ADJ	adjective	big, old, green, incomprehensible, first
ADP	adposition	in, to, during
ADV	adverb	very, tomorrow, down, where, there
AUX	auxiliary	is, has (done), will (do), should (do)
CONJ	conjunction	and, or, but
CCONJ	coordinating conjunction	and, or, but
DET	determiner	a, an, the
INTJ	interjection	psst, ouch, bravo, hello
NOUN	noun	girl, cat, tree, air, beauty
NUM	numeral	1, 2017, one, seventy-seven, IV, MMXIV
PART	particle	's, not,
PRON	pronoun	I, you, he, she, myself, themselves, somebody
PROPN	proper noun	Mary, John, London, NATO, HBO
PUNCT	punctuation	., (,), ?
SCONJ	subordinating conjunction	if, while, that
SYM	symbol	$, %, §, ©, +, −, ×, ÷, =, :)
VERB	verb	run, runs, running, eat, ate, eating
X	other	sfpksdpsxmsa
SPACE	space	

[15] An exhaustive list of universal and additionally language-specific POS and dependency label tags in *SpaCy* can be found here: https://spacy.io/api/annotation

Dependency Labels: The universal dependency labels are defined in *spacy* as follows:

DEP	DESCRIPTION
acl	clausal modifier of noun (adjectival clause)
advcl	adverbial clause modifier
advmod	adverbial modifier
amod	adjectival modifier
appos	appositional modifier
aux	auxiliary
case	case marking
cc	coordinating conjunction
ccomp	clausal complement
clf	classifier
compound	compound
conj	conjunct
cop	copula
csubj	clausal subject
dep	unspecified dependency
det	determiner
discourse	discourse element
dislocated	dislocated elements
expl	expletive
fixed	fixed multiword expression
flat	flat multiword expression
goeswith	goes with
iobj	indirect object
list	list
mark	marker
nmod	nominal modifier
nsubj	nominal subject
nummod	numeric modifier
obj	object
obl	oblique nominal
orphan	orphan
parataxis	parataxis
punct	punctuation
reparandum	overridden disfluency
root	root
vocative	vocative
xcomp	open clausal complement