



Daniel Anthofer, BSc

A Neural Network for Open Information Extraction from German Text

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Mark Kröll

Institute for Interactive Systems and Data Science
Head: Univ.-Prof. Dr. Stefanie Lindstaedt

Assessor: Dipl.-Ing. Dr.techn. Roman Kern

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

Abstract

Systems that extract information from natural language texts usually need to consider language-dependent aspects like vocabulary and grammar. Compared to the development of individual systems for different languages, development of multilingual information extraction (IE) systems has the potential to reduce cost and effort. One path towards IE from different languages is to port an IE system from one language to another. PropsDE is an open IE (OIE) system that has been ported from the English system PropS to the German language. There are only few OIE methods for German available. Our goal is to develop a neural network that mimics the rules of an existing rule-based OIE system. For that, we need to learn about OIE from German text. By performing an analysis and a comparison of the rule-based systems PropS and PropsDE, we can observe a step towards multilinguality, and we learn about German OIE. Then we present a deep-learning based OIE system for German, which mimics the behaviour of PropsDE. The precision in directly imitating PropsDE is 28.1%. Our model produces many extractions that appear promising, but are not fully correct.

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Goal	3
2	Background	4
2.1	Language Agnostic Information Extraction	4
2.1.1	Multilingual Information Extraction	5
2.1.2	Cross-lingual Information Extraction	6
2.1.3	Truly Language Agnostic Information Extraction	6
2.2	Predefined Schema Extraction	8
2.3	Open Information Extraction Methods	10
2.3.1	Preprocessing	13
2.3.2	Extraction	20
3	Analysis of PropS and PropsDE	25
3.1	The Rules of PropS	26
3.1.1	Dependencies	27
3.1.2	Linguistic Features	27
3.1.3	Graph Conversions	30
3.1.4	Extractions	52
3.2	Differences between PropS and PropsDE	53
3.2.1	Dependencies	54
3.2.2	Linguistic Features	55
3.2.3	Graph Conversions	56
3.2.4	Extractions	57
4	Method	59
4.1	Word Embeddings	60
4.2	Artificial Neurons	62

Contents

4.3	Architecture	63
4.4	Training	66
4.5	Evaluation	67
5	Results	69
5.1	Experiments	69
5.2	Evaluation	70
6	Discussion	72
6.1	Limitations	72
6.2	Problems and Mistakes	73
6.3	Lessons Learned	74
6.4	Future Work	80
7	Conclusion	82
	Bibliography	84

List of Figures

2.1	Example ontology with biographical information (Embley et al., 2014).	10
2.2	Example dependency parse with universal dependencies. . .	17
3.1	Example dependency parse with Stanford dependencies. . . .	28

List of Tables

2.1	A selection of Open Information Extraction methods.	12
2.2	Part-of-speech tags example.	13
2.3	Selection of Part-of-speech taggers.	14
2.4	Selection of Universal Dependencies (Nivre et al., 2016) with short descriptions.	16
3.1	The rules for linguistic feature extraction in PropS.	29
3.2	Differences between PropS and PropsDE during linguistic feature extraction.	56
3.3	Differences between PropS and PropsDE during graph conversions.	57
5.1	Training experiments with Sequence-to-sequence models. . .	70
5.2	Detailed arguments evaluation.	71

1 Introduction

Structured information is information that is stored in a database and ready for immediate computational processing. Semi-structured information means that, as the name suggests, there is a partial structure to this information, for example lists or HTML tags. Fully unstructured information, on the other hand, is information without any structure. Unstructured information can, for example, be in form of images, audio data, or natural language text.

To a certain extent, information is available in structured form. For example the World Factbook¹, contains structured information. DBPedia² is an effort to present the contents of Wikipedia³ in a structured database. Wikipedia itself also contains some structured information within the information boxes. A lot of information is, however, only available in semi-structured or unstructured form. To gain structured information from semi- or unstructured information, Information Extraction (IE) methods can be applied.

Information and results in publications from scientific research would be one example of valuable information presented in natural language text. As there are roughly 2.5 million scientific papers published each year⁴, it is impossible for a human to read and apply all the potentially useful information. With IE methods, we can feed the knowledge bases of expert systems with the information from unstructured sources.

Expert systems might then be able to assist humans by finding appropriate information and resources. An example for an early medical expert system is MYCIN (Buchanan and Shortliffe, 1984). This system can assist doctors in

¹<https://www.cia.gov/library/publications/the-world-factbook/> (11.11.2017)

²<http://wiki.dbpedia.org/> (11.11.2017)

³<https://www.wikipedia.org> (11.11.2017)

⁴http://www.stm-assoc.org/2015_02_20_STM_Report_2015.pdf (11.11.2017)

1 Introduction

identifying bacteria that infect patients, and it can provide suggestions for antibiotics to apply for treatment. In a similar way, expert systems can be designed and established for different fields.

In order to automatically extract information from natural language text, the language itself needs to be taken into account. Language features, such as the vocabulary and various aspects of grammar, commonly are only valid for a single language. Therefore, IE methods usually are developed for specific languages.

1.1 Motivation

The fact that IE methods focus on specific languages is a sign of language dependence. We want to learn about possibilities for language independence in IE methods. With language independence, the results of further development in IE tasks and methods within one language can be adopted to other languages more easily than without language independence. This is an important type of scalability (Bender, 2011).

Language independence can reduce cost and effort for the development of IE methods for separate languages. With a certain level of language independence, adapting an IE method to a new language is easier than developing a new IE method from scratch. Technologies and products that rely on IE also benefit from this advantage. The effort to port products to new languages, for example with only few speakers, is lower, thus, it is more likely that products are deployed in new languages.

We focus on Open Information Extraction (OIE). In contrast to fixed schema IE, OIE extracts any piece of information available in natural language text. With OIE, general and domain-independent IE can be performed. This is an advantage in use cases which have no facts of interest specified.

One possible first step towards language independence is the translation of a method from one language to another. PropsDE (Falke et al., 2016) is a German port of the English OIE system PropS (Stanovsky et al., 2016). Currently there are only a few OIE methods for the German language. We

decided to develop a neural network for OIE from German text, that mimics the ruleset of the existing OIE System PropsDE.

1.2 Goal

In this master's thesis, we are interested in information extraction methods for multiple languages. In this context, we look into language independence and multilingual methods (see Section 2.1), and we focus on OIE (see Section 2.3). Alongside multiple languages, we take a special interest in our mother tongue, the German language.

As the two German OIE systems PropsDE and GerIE (Bassa, 2016) are rule-based, we want to approach this task differently: We develop a learning based method. For that, we first need to understand how PropsDE extracts information from German text. We analyze the rules of PropS and PropsDE, and their differences (see Chapter 3). This way, we can learn about OIE from German text, and we can observe a step towards language independence.

Then we develop a neural network for OIE from German text (see Chapter 4), which imitates the ruleset of the existing OIE System PropsDE. Thus, we can apply PropsDE to generate a reasonable amount of training data. The result of our development is a bidirectional sequence-to-sequence model (see Section 4.3) with gated recurrent units (see Section 4.2). The code can be found on GitHub¹.

We evaluate the neural network in Chapter 5. The evaluation shows that our neural network achieves a word precision of 28.1 % compared to PropsDE. Then we discuss the analysis of PropS and PropsDE, and the results of the evaluation of our neural network (Chapter 6).

¹<https://github.com/danielanthofer/nnoiegt> (13.11.2017)

2 Background

Before we can create an Information Extraction (IE) system for German text, we need to gain a general understanding of IE and the methods involved. In this chapter we learn about IE.

The investigations for this master's thesis led us to the impression, that the English language is the target of a large portion of IE systems. Since we focus on the German language in our practical part, we are interested in how different languages are represented in the field. In this context, we decided to take special interest in IE systems which support multiple languages. Thus, in Section 2.1, we cover language independence in IE. We look at the general idea of language-independent IE and at different types.

As we want to focus on Open Information Extraction (OIE), we cover OIE in more detail. From our perspective, we divide IE methods into two major types: OIE and IE with predefined schemata. In Section 2.2, we get to know different types of predefined schema IE. Finally, we learn about OIE in Section 2.3. We dissect the general OIE process into two parts, the preprocessing step (Section 2.3.1) and the extraction step (Section 2.3.2). Within these steps, we consider available types and methods.

2.1 Language Agnostic Information Extraction

As already stated in the motivation of this master's thesis (Section 1.1), one goal of IE research is to become language agnostic. De Wilde, 2016, categorizes language agnostic IE into three different degrees:

1. *Multilingual* IE is applicable to multiple languages. Multilingual IE methods, such as the multilingual OIE method from Gamallo and

2 Background

Garcia, 2015, extract information from sources of a specific set of languages. The extraction is not translated and stays within its language of origin.

2. *Cross-lingual IE*, on the other hand, does not only support the extraction of information from sources of a set of languages, but it also supports machine-translation into another supported language. Cross-lingual IE methods, such as the ontology-based IE method from Falaise et al., 2010, can resort to an interlingua.
3. *Truly language agnostic IE* methods ideally are not limited to a specific set of languages. Instead, they support any human language, and they perform equally well on any language.

2.1.1 Multilingual Information Extraction

Multilingual IE is the first step towards truly language agnostic IE, and Multilingual IE publications contain an interesting subset: publications, which port an existing method from its original language (usually English) to a different language. This appears to be a down-to-earth, and therefore promising, approach towards general multilinguality.

One example for this is PropsDE (Falke et al., 2016). It is a port of the OIE method PropS (Stanovsky et al., 2016) from English to German. This method is, however, not actually multilingual, as the ported variant, PropsDE, does not support the original language, English, any more. These two methods are subject to the analysis part of this thesis, and we describe them in more detail in the analysis chapter (Chapter 3).

Another example is Verga et al., 2015. They do not port an OIE system, but they propose a method which transfers an open information extractor to a new language.

A different approach is taken by Faruqui and Kumar, 2015. The goal of their method is to take a sentence from any language and return a relation tuple in that language. To achieve that, their method uses Google Translate¹ to translate a sentence into English. Then the open IE system OLLIE

¹<https://cloud.google.com/translate/> (11.11.2017)

2 Background

(Mausam et al., 2012) is applied on the English sentence to extract English relation tuples. Each tuple is then translated into the source language using cross-lingual projection (cross-lingual projection is a method for machine translation and not to be mixed up with cross-lingual IE). Therefore, the method supports any language that is supported by Google Translate.

2.1.2 Cross-lingual Information Extraction

Embley et al., 2014 (and Embley et al., 2011) propose a cross-language IE method. Their method is not capable of OIE, but instead it is based on predefined ontologies for specific domains. It supports different localizations with, for example, different languages or different measuring units. An existing setup of the system can be extended with additional languages. It is not limited to a specific set or number of languages, but for each language an extractor and a mapping is required. This mapping maps to a language-agnostic ontology. The language-agnostic ontology is the basic idea of this method. It describes the concepts and facts of the domain. The extraction itself only populates the localized ontology. Afterwards, queries are processed in a cross-lingual manner through the mapping to the language-agnostic ontology.

A cross-lingual OIE system is proposed by Zhang, Duh, and Van Durme, 2017. Their neural-network-based method directly maps a sentence from the source language into a predicate-argument structure in the target language. During this process, the neural network implicitly learns translations. Zhang, Duh, and Van Durme, 2017, ran experiments with their method on Chinese as source language and English as target language. We take a more detailed look at the method's architecture in the section about training-based extraction in OIE systems (Section 2.3.2).

2.1.3 Truly Language Agnostic Information Extraction

In order to be truly language agnostic, a method either has to consider language features of every existing language, or not rely on language features at all. This is not only valid for the source but also for the extraction.

2 Background

Truly language agnostic IE methods not only have to be able to extract from any language, but they also must be able to present the extraction in any language.

There are methods that do not rely on features of languages. Instead, they take different aspects of texts into account.

The goal of Mirończuk et al., 2013, is to extract named entities of predefined classes from semi-structured (XML or HTML) documents. In their method, they use an initial seed of named entities of the desired class to find patterns for additional entities. After applying these patterns, the new entities are added to the seed, and again patterns are generated. This process is applied iteratively. The patterns themselves are generated from characters that surround seed words within the documents. If two seed words appear in each other's neighbourhood, then a regular expression is constructed from the XML or HTML code in between. These regular expressions form the patterns which are applied to find further named entities. Since these patterns are constructed from XML or HTML code, this method relies on the document structure, but it does not require language-specific features.

Heist and Paulheim, 2016, focus on Wikipedia¹ abstracts in their publication. They trained classifiers to extract a predefined piece of information from Wikipedia abstracts of different languages. As exemplary information, they extracted the genre of music bands. A Random Forest classifier yielded the best results. They expect this information to be at similar locations within abstracts of different bands in different languages. Therefore, in their publication, instead of language features, the structure of the Wikipedia abstracts is most relevant to the extraction.

Even though the two methods presented here perform equally well on any language, they do not transfer this information between languages. Thus, they do not include the cross-lingual property. Because we consider that each degree of language agnostic IE includes its predecessor, the cross-lingual property is a requirement for truly language agnostic IE. From this perspective, the presented methods are not truly language agnostic. Instead, they are similar to multilingual methods, but they are not limited

¹<https://www.wikipedia.org/> (11.11.2017)

2 Background

to a specific set of languages. They are language-independent and possibly omnilingual, but not truly language agnostic.

A truly language agnostic IE method must be able to extract information from any language, and it must be able to represent each extraction in any language. Such a method does not exist.

2.2 Predefined Schema Extraction

In this thesis, we focus on OIE, but in this background chapter, we also consider other types of IE for completeness. Extracting fixed schema relations is especially interesting for data-rich and narrow-scope domains (Embley et al., 2014). IE with predefined schemata can be split into four types (Piskorski and Yangarber, 2013):

- *Named entity recognition* identifies and classifies named entities, such as organizations, persons, locations, numerical expressions, etc.
- *Co-reference resolution* is the task of identifying multiple mentions of the same entity in the text. It includes resolving whether an entity is referred to by different names and abbreviations.
- *Relationship extraction* detects relationships between entities in the text, with predefined entities and/or relations.
- *Event extraction* identifies events and information about them, like, for example, when, where, why, and with whom an event takes place.

Facts retrieved by IE methods are presented in a structured form. How each representation is organized largely depends on the method. In general, these structured representations consist of entities and relationships. In predefined schema extraction, several methods define the schema as an ontology. An ontology is an abstract representation of a domain. It defines relations between concepts, which are interesting for that specific domain.

In an early work, Andersen et al., 1992, propose a system called *Jasper*, which extracts fixed schema relations from press releases of organizations. They use a pattern matcher to populate predefined extraction frames with the specific goal to collect financial news. A part of their extraction representation looks like this:

2 Background

```
{{ EARNINGS
net-income-group: <net-income-group-object>
current-quarter-net: <net-income-object>
prior-quarter-net <net-income-object>
current-ytd-net: <net-income-object>
...
prior-ytd-net: <net-income-object>
...
}}
```

Embley et al., [2014](#), and Suchanek, Sozio, and Weikum, [2009](#), propose methods to populate predefined ontologies. These two publications focus on different aspects of IE: Embley et al., [2014](#), present a system for cross-language IE, whereas Suchanek, Sozio, and Weikum, [2009](#), focus on checking the plausibility of hypotheses and avoiding inconsistencies within the ontology. Nevertheless, these two methods have in common that the user defines an ontology, which can then be populated. Figure [2.1](#) shows how such an ontology might look like.

2 Background

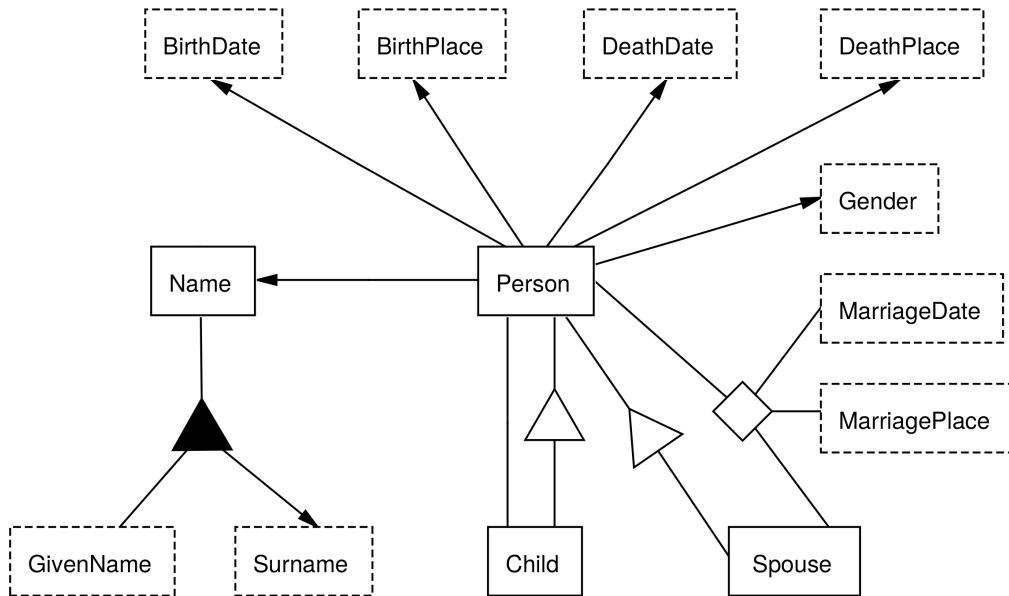


Figure 2.1: Example ontology with biographical information (Embley et al., 2014).

A subtype of relationship extraction is *relation classification*. The schema is simpler than an ontology, and there are many publications that focus on this task. The predefinition consists of a set of entities and a set of relations. The goal of the classifier is to identify a predefined relation between pairs of predefined entities from a given sentence. Example publications for this type of IE are Zhang and Wang, 2015, Xu et al., 2015, Zeng et al., 2015, Liu et al., 2015, and Xiao and Liu, 2016. All of these examples use neural networks.

2.3 Open Information Extraction Methods

OIE aims to find structured information independent from domains or any predefinitions. Instead of being predefined, relations and arguments are extracted from the unstructured text.

2 Background

Many OIE methods, such as TextRunner (Banko et al., 2007) and PropS (Stanovsky et al., 2016), extract relations tuples. That means, the extraction is of the form *relation(first entity, second entity)* (Stanovsky et al., 2016), sometimes also displayed as information triple (*argument₁, relation, argument₂*) (Zhila and Gelbukh, 2014).

Example 1. From the sentence *Amazon, the retail giant, sells products*, PropS extracts the facts *sell(Amazon, products)* and *sell(retail giant, products)* (Stanovsky et al., 2016).

Additionally, some OIE methods, for example ArgOE (Gamallo and Garcia, 2015) and ClausIE (Del Corro and Gemulla, 2013), extract facts, which consist of binary relations with additional information like time, location or other circumstances.

Example 2. From the sentence *In 1921, Albert Einstein has won the Nobel Prize*, ClausIE extracts the fact (*Albert Einstein, has won, the Nobel Prize, in 1921*) (Del Corro and Gemulla, 2013).

In this section we want to learn, how OIE methods work. Therefore, we take a look at the different steps that are necessary to get an extraction from an input sentence. In this context we especially care about multilinguality and how it is achieved.

Even though the different OIE methods differ in many details, there are processing steps and general concepts, which OIE methods have in common. As it is usually not possible to extract a fact directly from a sentence, OIE methods employ some kind of preprocessing. In this section, we have divided the process of OIE in two steps: the preprocessing step, and the extraction step.

There are two widespread types of preprocessing for OIE: Part-of-speech (POS) tagging and dependency parsing. Usually OIE methods use one of these two. There are other means for preprocessing available, and we look at a few other preprocessing methods briefly, but we focus on POS tagging and dependency parsing. We cover preprocessing steps in Section 2.3.1.

In contrast to the preprocessing, which is usually performed by external tools, the extraction step is the distinctive component of OIE systems. This

2 Background

Reference	Name	POS	DP	RB	TB	Language
Banko et al., 2007	TextRunner	✓	–	–	✓	English
Akbik and Bross, 2009	Wanderlust	–	–	–	✓	English
Wu and Weld, 2010	WOE ^{pos}	✓	–	–	✓	English
Wu and Weld, 2010	WOE ^{parse}	–	✓	–	✓	English
Fader, Soderland, and Etzioni, 2011	ReVerb	✓	–	✓	–	English
Mausam et al., 2012	OLLIE	✓	✓	–	✓	English
Bast and Hausmann, 2013	CSD-IE	–	–	✓	–	English
Del Corro and Gemulla, 2013	ClausIE	–	✓	✓	–	English
Zhila and Gelbukh, 2014	ExtrHech	✓	–	✓	–	Spanish
Wang, Li, and Huang, 2014	SCOERE	✓	✓	–	✓	Chinese
Faruqui and Kumar, 2015	–	–	✓	–	✓	multilingual
Gamallo and Garcia, 2015	ArgOE	–	✓	✓	–	multilingual
Verga et al., 2015	–	–	–	–	✓	multilingual
Stanovsky et al., 2016	PropS	✓	✓	✓	–	English
Falke et al., 2016	PropsDE	✓	✓	✓	–	German
Bassa, 2016	GerIE	✓	✓	✓	–	German
Zhang, Duh, and Van Durme, 2017	–	–	✓	–	✓	multilingual

Table 2.1: A selection of Open Information Extraction methods. The table shows for each method whether it uses Part-of-speech (POS) tagging or dependency parsing (DP) for preprocessing, and the type of extraction method: rule-based (RB) or training-based (TB).

step can be categorized into two types: rule-based extraction and training-based extraction. These extraction methods take the result from the preprocessing, e.g. POS tags, as input, and deliver a fact. We continue with extraction steps in detail in Section 2.3.2.

Table 2.1 shows a selection of OIE methods.

2.3.1 Preprocessing

Part-of-speech Tagging

POS tagging is the process of assigning a tag to each word in a sentence. This tag represents the grammatical function of that word within the sentence.

Each language has its own grammar and, therefore, its own POS tags. In English, for example, the Penn Treebank POS tags¹ are commonly used, and for the German language there is the STTS tag set². For simplicity reasons, and because we are interested in IE from multiple languages, we look at universal POS tags, which aim to cover multiple languages. Petrov, Das, and McDonald, 2011, propose such a universal tag set for many languages, and also a mapping from various languages towards this universal tag set. The example in Table 2.2 shows how tags of a sentence can look like.

Sentence	Universal Tag	English Tag	Description
The	DET	DT	determiner
oboist	NOUN	NN	noun, singular
Heinz	NOUN	NNP	proper noun, singular
Holliger	NOUN	NNP	
has	VERB	VBZ	verb, present, 3rd person singular
taken	VERB	VBN	verb, past participle
a	DET	DT	
hard	ADJ	JJ	adjective
line	NOUN	NN	
about	ADP	IN	preposition
the	DET	DT	
problems	NOUN	NNS	noun, plural

Table 2.2: An example for part-of-speech tags of the sentence *The oboist Heinz Holliger has taken a hard line about the problems* (Petrov, Das, and McDonald, 2011).

¹English POS tags: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html (11.11.2017)

²German POS tags: <http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/TagSets/stts-table.html> (11.11.2017)

2 Background

There are several tools available, which support POS tagging. Since we are mainly interested in the multilingual use case, Table 2.3 shows a selection of taggers which support multiple languages. Two of these taggers are especially interesting in the multilingual context: Morfette and the unsupervised POS tagging with bilingual graph-based projection.

Name	Languages	Reference
Mate Tools	English, German, Chinese, French, Spanish	Bohnet and Nivre, 2012
Morfette	Romanian, Spanish, Polish	Chrupala, Dinu, and Genabith, 2008
Stanford POS Tagger	English, German, French, Spanish, Chinese, Arabic	Toutanova et al., 2003
TreeTagger	English, German, French, Spanish, Chinese, Czech, Polish, Romanian, and many others	Schmid, 1995
Unsupervised POS Tagging with Bilingual Graph-based Projections	English, German, Spanish, Swedish, and many others	Das and Petrov, 2011
DepPattern	English, Spanish, Galician, French, Portuguese	Gamallo and González, 2012

Table 2.3: Selection of Part-of-speech taggers which support multiple languages. The languages column contains a subset of the tested languages.

Morfette (Chrupala, Dinu, and Genabith, 2008) is able to do POS tagging in any language after training with annotated datasets. Therefore, this tool might be interesting for OIE methods which should support multiple languages. If, however, there is no annotated dataset available yet for a language of interest, then this dataset has to be created before Morfette can be trained. This is usually done manually. Manually annotating sentences with POS tags takes a large effort.

Das and Petrov, 2011, address this issue with unsupervised POS tagging with bilingual graph-based projection. In their approach, text from a target language is translated into a resource-rich language (like English). The translation gets tagged, and then the POS tags of the translation are projected

2 Background

onto the original text in the target language. With these tags, a new POS tagger for the target language is trained. In order to avoid being language specific, Das and Petrov, 2011, use the universal POS tagset from Petrov, Das, and McDonald, 2011.

OIE methods that use POS tags in their preprocessing step are displayed in Table 2.1.

Dependency Parsing

Dependency parsing builds a dependency tree of a sentence. In contrast to POS tags, which represent the grammatical role of a word within a sentence, dependency parses display the relationships between words and expressions.

There are several different formats for storing dependency parses. One example is the CoNLL-X 2006 format (Buchholz and Marsi, 2006). In this format, each token (usually a single word) has its own row within a table. This row contains, among other data, the id of the token and the id of its parent in the dependency tree. The dependency parser DepPattern (Gamallo and González, 2012) uses this format, and the dependency parser from Mate Tools (Bohnet and Nivre, 2012) supports a 2009 variant of the CoNLL format. The Stanford dependency parser (Chen and Manning, 2014) has its own format

The Stanford dependency parser also supports, apart from language specific dependencies, Universal Dependencies (Nivre et al., 2016). Similar to POS tags, the types of dependencies within sentences vary between different languages. Therefore, Nivre et al., 2016, propose Universal Dependencies (UD). The goal of UD is to facilitate consistent annotation of dependencies across languages. Since this complies with our interest in multilingual IE, we use UD as an example of how dependency parses may look like.

As already mentioned, the Stanford dependency parser is capable of issuing UD. For an example of how a dependency parse can look like, we parse the sentence *Who's the funky-looking donkey over there?* (Lee et al., 2013) with the Stanford dependency parser. Each word of the sentence becomes a node in the dependency tree, and each node receives a node ID. Edges are displayed

2 Background

with an edge label, a start node, and an end node. The following listing shows the direct the output of the parser:

```
root(ROOT-0, Who-1)
cop(Who-1, 's-2)
det(donkey-5, the-3)
amod(donkey-5, funky-looking-4)
nsubj(Who-1, donkey-5)
case(there-7, over-6)
nmod(donkey-5, there-7)
punct(Who-1, ?-8)
```

Table 2.4 shows the meaning of these dependencies¹.

amod	adjectival phrase that modifies the meaning of the noun
case	marker for the case of the clause or noun
cop	copula in case of a nonverbal predicate
det	determiner of an expression
nmod	(nominal modifier) nominal attribute or genitive complement
nsubj	(nominal subject) syntactic subject of a clause
punct	punctuation
root	root of the sentence

Table 2.4: Selection of Universal Dependencies (Nivre et al., 2016) with short descriptions.

In order to make the dependencies more obvious, we visualize the dependency graph in Figure 2.2.

Just like for POS tagging, there are several tools for dependency parsing. Since there are many parsers available, we focus on those which support multiple languages:

- One that we already mentioned is the *Stanford dependency parser* (Chen and Manning, 2014). It is part of the preprocessing step in many OIE systems, for example in SCOERE (Wang, Li, and Huang, 2014),

¹A detailed description of each type of dependency with examples can be found at <http://universaldependencies.org/u/dep/all.html> (11.11.2017).

2 Background

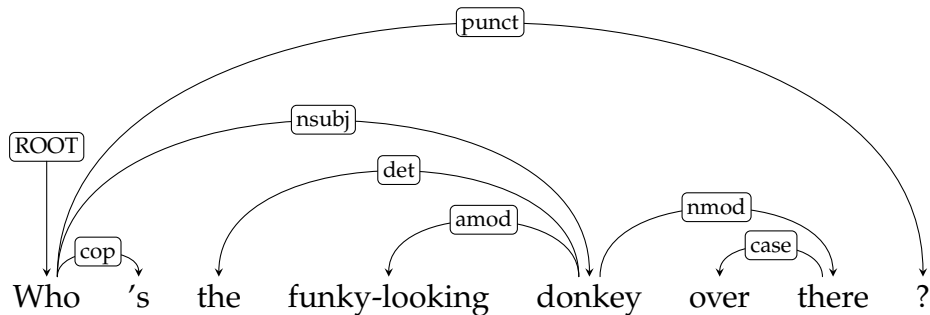


Figure 2.2: Example dependency parse with universal dependencies (Nivre et al., 2016) from the sentence *Who's the funky-looking donkey over there?* (Lee et al., 2013).

ClausIE (Del Corro and Gemulla, 2013), and WOE^{parse} (Wu and Weld, 2010). Similar to the Stanford POS tagger, the Stanford dependency parser supports several languages, e.g. English, German, Spanish and Chinese.

- Another set of tools, which is capable of POS tagging as well as dependency parsing is *Mate Tools* (Bohnet and Nivre, 2012). *Mate Tools* are used by the OIE system PropsDE (Falke et al., 2016), and they also support multiple languages, namely English, German, Spanish, French and Chinese.
- Furthermore, there is the *MaltParser* (Nivre et al., 2007), which is used by OLLIE (Mausam et al., 2012). *MaltParser* is capable of parsing English, French and Swedish texts.
- *DepPattern* (Gamallo and González, 2012) is a POS tagger and dependency parser for English, Spanish, Galician, French, and Portuguese. It is used in the preprocessing step of the multilingual OIE system ArgOE (Gamallo and Garcia, 2015).

Chunking

An example for a different preprocessing step can be observed in ReVerb (Fader, Soderland, and Etzioni, 2011), which is a OIE system for English. It uses POS tagging combined with *chunking* for preprocessing. For that, it

2 Background

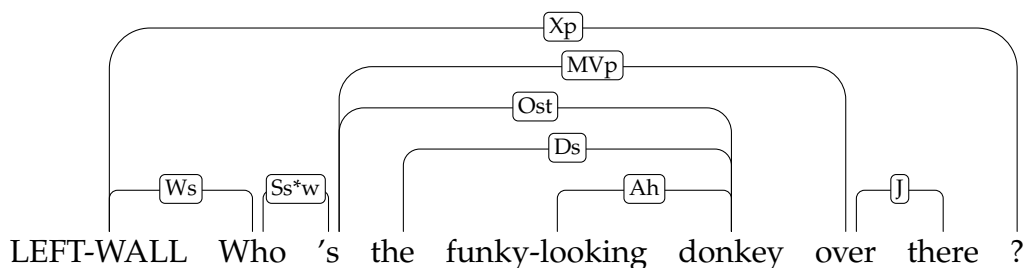
includes the Apache OpenNLP¹ library. A chunker groups the words of a sentence which syntactically belong together. Such a chunk can for example be a group of nouns and a determiner. "An information extraction system" would be an example of a chunk within this sentence.

Similar to ReVerb, TextRunner (Banko et al., 2007), also uses a combination of POS tags and chunking. TextRunner is another OIE system for English. In contrast to many other OIE systems, TextRunner contains its own implementation of a POS tagger and chunker.

Link Grammar

A link grammar (Sleator and Temperley, 1995) defines how words in a sentence are connected. The outcome is a graph that looks similar to a dependency tree at a first glance, but it is not equal. In a link grammar, links may not cross each other, any word of the sentence must be connected, and the linking requirements have to be fulfilled. Linking requirements are defined for a language. They express, which links between words in a sentence are allowed, required, and optional. An OIE system that uses link grammar is Wanderlust (Akbik and Bross, 2009).

Example 3. This example (generated with MorphAdorner¹) illustrates, how a sentence parsed with link grammar looks like. The expressions in the edge labels are the rules of the link grammar, that match for the respective link. We do not explain these rules, as they are complicated and not important for our work.



¹<https://opennlp.apache.org/> (11.11.2017)

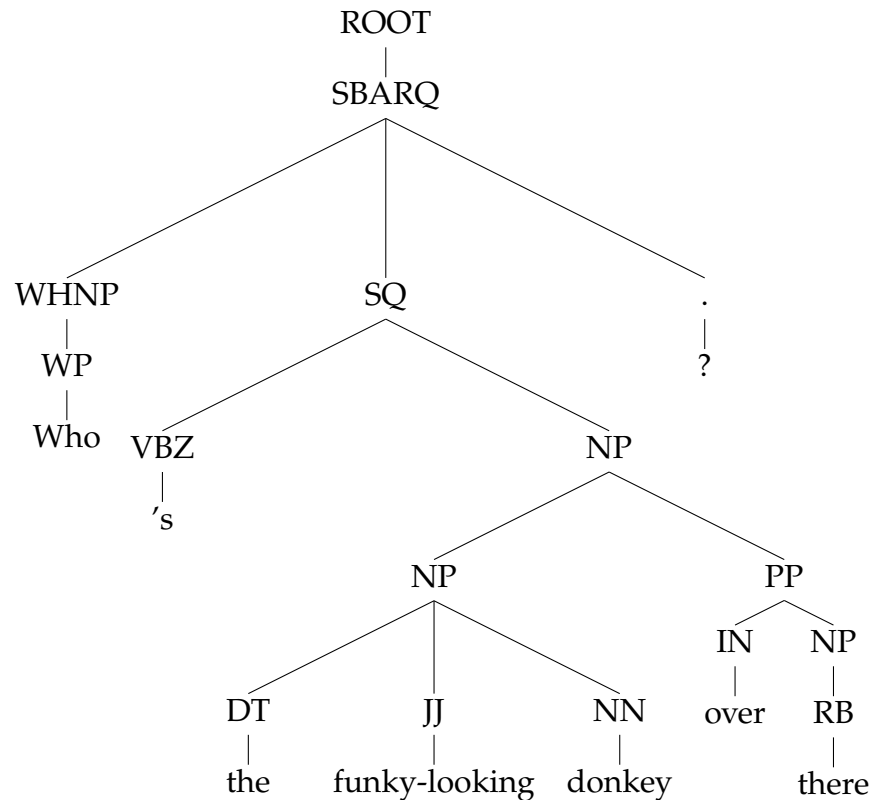
¹<http://morphadorner.northwestern.edu/morphadorner/parser/example/>
(11.11.2017)

2 Background

Constituency Parsing

A constituency tree (Frege, 1879) displays the constituency relations of an expression. In general, constituency trees can be any kind of parse tree, including that of mathematical formulas or program code. In case of natural languages, a constituency tree displays the grammatical hierarchy of tokens. Constituency relations are used by the OIE systems CSD-IE (Bast and Hausmann, 2013) and SCOERE (Wang, Li, and Huang, 2014, in addition to POS tags and dependency trees).

Example 4. Here is a constituency tree as it is produced by the Stanford Parser (Chen and Manning, 2014).



These constituents are labeled with Penn Treebank tags¹. The tags of our example are in the following table:

¹<http://web.mit.edu/6.863/www/PennTreebankTags.html> (11.11.2017)

2 Background

SBARQ	direct question introduced by <i>wh</i> -word
WHNP	<i>wh</i> -noun phrase
WP	<i>wh</i> -pronoun
SQ	main clause of <i>wh</i> -question
.	punctuation
VBZ	verb singular present 3rd person
NP	noun phrase
PP	prepositional phrase
DT	determiner
JJ	adjective
NN	noun singular
IN	preposition
RB	adverb

2.3.2 Extraction

As already mentioned, the preprocessing in OIE systems is usually performed by external tools. The extractor, on the other hand, is the distinctive element of an OIE system. This extractor can either be a set of rules or a learned model, e.g. a neural network or Naive Bayes.

Rule-based Extraction

GerIE (Bassa, 2016) is a rule-based OIE system for German. It expects a dependency parsed and POS tagged sentence as input. The ruleset consists of four layers:

1. The first layer contains rules for filtering "bad" sentences, tagging negation words (necessary due to the large variety of negation words in German), tagging quotes, and tagging nouns with a specific meaning, e.g. a number or a measurement unit.
2. The rules of the second layer already identify facts by extracting subtrees from the dependency parse graph for the relation and for the arguments.
3. The postprocessing layer applies some specific rules for processing conjunctions and relative pronouns.
4. Finally, the bottom layer of rules generates the actual output.

Another rule-based OIE system for German is *PropsDE* (Falke et al., 2016). Similar to *GerIE*, it applies rules on dependency parses. It takes German

2 Background

sentences as input and returns relation tuples. Mate Tools (Bohnet and Nivre, 2012), in combination with JoBimText (Biemann and Riedl, 2013), is used for dependency parsing. PropsDE is a port from the English OIE system PropS (Stanovsky et al., 2016). Therefore, the main effort was to port the rules for the German language. Due to our interest in multilingual OIE, we are especially interested in the porting-process that Falke et al., 2016, performed, and we look at that in more detail in Chapter 3.

Whereas GerIE and PropsDE are monolingual, *ArgOE* (Gamallo and Garcia, 2015) is a multilingual rule-based OIE system. It supports English, Spanish, Portuguese, French, and Galician. *ArgOE* takes dependency parses from DepPattern (Gamallo and González, 2012), and it performs the extraction in two steps: first it detects argument structures, and then it generates information triples.

1. To detect the argument structure, the verbs for each sentence are identified from the dependency parse. Then, for each verb, all dependents, which can be part of the verb's argument structure, are selected. These argument structures can contain subject, direct object, attribute, and complements. Five types of argument structures were defined, and within a sentence it is possible to find several different argument structures.
2. After the detection of the argument structure, a set of rules is applied on it. These rules transform each argument structure into triples.

The language specific elements of *ArgOE* are located within the dependency parser DepPattern. The rules themselves are language-independent. They do, however, not extract facts from certain dependency parse structures which produce correct facts in one language and invalid facts in one of the other languages. Thus, some facts are knowingly omitted.

The rule-based OIE systems we looked at until now in this section all apply dependency parsing for preprocessing. In contrast to that, *ExtrHech* (Zhila and Gelbukh, 2014) uses POS tags. *ExtrHech* is an OIE system for the Spanish language. The rule-based extraction consists of three steps, which are applied to single sentences:

1. Find a relation phrase within the sentence which contains a verb.
2. Find a noun phrase in front of the relation phrase.

2 Background

3. Find a noun phrase behind the relation phrase.

ReVerb (Fader, Soderland, and Etzioni, 2011) is another rule-based OIE system which does not use dependency parses. Instead, ReVerb uses a combination of POS tagging and chunking (see Section 2.3.1). Fader, Soderland, and Etzioni, 2011, focus on a specific problem that can occur in OIE systems: they want their system to explicitly not extract incoherent and uninformative facts. In order to achieve that, they introduce constraining rules for POS tag patterns, which prevent certain extractions.

Training-based Extraction

One possibility for training-based extraction is the use of a *Naive Bayes classifier*. A Naive Bayes classifier is part of the first OIE system. In their publication, Banko et al., 2007, introduce the concept of OIE by proposing the OIE system TextRunner. TextRunner uses POS tagging and chunking for preprocessing (see Section 2.3.1). In the extraction step, it generates extraction candidates for each sentence. These extraction candidates consist of chunks as arguments and parts of the text between chunks as relations. TextRunner’s Naive Bayes classifier is trained to identify correct extractions from these candidates. This classifier takes a feature vector as input, which contains POS tags and some other features, like the number of words in the relation. WOE^{pos} and WOE^{parse} (Wu and Weld, 2010) also employ a Naive Bayes classifier. They inherited the classifier from TextRunner and use it in the same manner.

Another possibility for a training-based extraction is the use of *neural networks*. Zhang, Duh, and Van Durme, 2017, propose an interesting cross-lingual OIE system that uses neural networks (see Section 2.1). Their OIE system extracts and implicitly translates facts from Chinese sentences to English extractions. In this case, the extractions look different compared to other OIE systems: they extract PredPatt (White et al., 2016) predicate-argument structures. Those structures are trees, similar to universal dependencies (Section 2.3.1), but they only distinguish between predicates and arguments. The architecture of this neural network is a Sequence-to-sequence (Seq2Seq) model (see Section 4.3) with long short-term memory units (see Section 4.2). In order to train the network, Zhang, Duh, and

2 Background

Van Durme, 2017, generate predicate-argument structures with PredPatt for English sentences, for which direct Chinese translations are available. A training sample consists of the Chinese sentence as input and the predicate-argument structure as output. This publication inspired us to implement a similar architecture in the practical part of this thesis (see Chapter 4).

A different approach on training-based extraction is taken by OLLIE (Mausam et al., 2012). OLLIE is an English OIE system that uses the Stanford dependency parser (Chen and Manning, 2014) to retrieve the dependency parse of a sentence. For the extraction, it holds a list of *open pattern templates*. An open pattern template is a mapping between a dependency parse tree and an extraction, and it consists of dependency parse pattern templates with corresponding extraction templates. These templates have been learned from training data. A training sample consists of a sentence and a (correct) extraction. In order to get a parse pattern template and the extraction template from a training sample, the words in the extraction were correlated with the dependency parse of the sentence. The parse pattern template is the dependency parse tree structure without the end nodes (words), and the extraction template contains the roles that the relation and the arguments have within the tree structure. Since this does not always yield correct results, additional constraints and refinements were necessary (which we do not cover here). For the extraction, OLLIE matches the dependency parse with the parse pattern template and fills the extraction template.

The training-based methods we looked at until now all use supervised learning. In their publication, Wang, Li, and Huang, 2014, show an approach with *semi-supervised* learning. They propose SCOERE, an approach for OIE in the Chinese language. Similar to OLLIE, SCOERE holds a list of learned open pattern templates to extract facts from sentences. For preprocessing, SCOERE uses the Stanford dependency parser (Chen and Manning, 2014) to get the dependency tree and the constituency tree, and POS tags are extracted from the sentence with FudanNLP¹. In the supervised part of their method, they employ another type of machine learning method, a conditional random field (CRF). They train the CRF with manually annotated corpora. Afterwards, in the unsupervised part, they generate annotations for new sentences with this CRF. These new annotated sentences are then used as

¹<https://github.com/FudanNLP/fnlp> (11.11.2017)

2 Background

training data to learn the open pattern templates. With this semi-supervised approach, the amount of training data for the open pattern templates can be increased significantly.

3 Analysis of PropS and PropsDE

PropsDE (Falke et al., 2016, see Section 2.3.2) is a rule-based Open Information Extraction (OIE) (see Section 2.3) system for the German language, that applies its rules on dependency parses. It has been developed by porting the English OIE system PropS (Stanovsky et al., 2016) to German. In the practical part of this thesis, we want to extract facts from German text by imitating PropsDE. Therefore, we want to know how PropsDE works. By analyzing and comparing PropS and PropsDE, we can learn about OIE from German text, and we can observe the translation of a system from one language to another.

In this chapter, we look in detail at the rules of PropS and PropsDE, and at the differences. We start this chapter by learning about the rules in general in Section 3.1. In this section, we are not interested in the rules from a linguistic point of view. Instead, we want to know about the layout of the rules and how they are applied, and we want to know about hierarchies, groups and types within the ruleset. Since PropS and PropsDE are structured similarly, we only look at PropS here. A high-level perspective over the rules is already given in the publication of Stanovsky et al., 2016. In this chapter, we want to learn about the rules from a low-level point of view. Therefore, we learn about the rules directly from the program code and we describe them the way they are implemented. Since we focus on the code structure, we explicitly mark actual variables with this typewriter font.

In Section 3.2, we inspect the differences in the rulesets of PropS and PropsDE. We reflect upon the analysis of the ruleset and the differences between the two systems in the discussion in Section 6.3

3.1 The Rules of PropS

From our point of view, we divide the general process of PropS into four major steps:

1. Parsing dependencies and loading the dependency graph (including Part-of-speech (POS) tags) into a graph data structure
2. Finding certain linguistic features at graph nodes, e.g. lemma and tense
3. Converting the graph structure so that the nodes and edges represent the final extractions
4. Generating final extractions from the converted graph

Whereas the first step is actually performed by an external tool, the other steps form the ruleset of PropS. We look at the rules in detail as far as necessary in order to understand how PropS works, thus, we omit some of the details that we do not consider important for a general understanding.

Each step makes use of the results of the previous steps. After POS tagging and dependency parsing, the second step finds linguistic features by considering specific aspects of the dependency parse and the POS tags. In the third step, the graph structure of the dependency parse is converted to a different structure using some of the previously found linguistic features. From the converted graph, the fourth step extracts the final facts. This step relies on the conversions from the third step.

As we have seen in Section 2.3 of the previous chapter, extractions from OIE systems are often represented as n-ary relations. This is also the case for PropS, and they are extracted in the final step. In order to do that, a distinction between relation and argument is required within the graph. PropS achieves this by marking the relation node as *top* node in the conversion step.

In this section, dependencies and POS tags (see Section 2.3.1) tags occur rather frequently. In order to make it more understandable for the reader, there are listings of dependencies and POS tags together with a brief description of their meaning. The reference for these descriptions can be found at https://nlp.stanford.edu/software/dependencies_manual.pdf (11.11.2017) for dependencies and <https://www.ling.upenn.edu/courses/>

3 Analysis of PropS and PropsDE

[Fall_2003/ling001/penn_treebank_pos.html](https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html) (11.11.2017) for POS tags. These listings look like this example:

advcl	adverbial clause modifier
mark	marker word introducing advcl

Within these listings, dependencies are written in small letters, and POS tags are written in capital letters.

3.1.1 Dependencies

PropS uses the Stanford dependencies, which are similar to Universal Dependencies (Nivre et al., 2016, see Section 2.3.1) but not the same. The Stanford dependencies are described in detail in the Stanford dependencies manual¹. PropS includes the Berkeley Parser (Petrov and Klein, 2007) for POS tagging, and the Stanford Dependency Parser (Chen and Manning, 2014) for dependency parsing. If we parse the example sentence from the previous chapter with Stanford dependencies, we receive the dependency parse graph in Figure 3.1. The POS tags are in accordance with the Penn Treebank Project's POS tags².

3.1.2 Linguistic Features

In order to find linguistic features, such as passive voice, a set of rules is applied on each node of the dependency graph. In this step, the graph is not altered, and the features are stored in a member variable of the node.

Most of these rules appear quite similar to each other. Since the rules do not depend on each other, the order of the rules is not important. The common ground is that most of them apply string-matching on the nodes, neighbours and edges. In order to do that, many rules employ the generalized pattern

¹https://nlp.stanford.edu/software/dependencies_manual.pdf (11.11.2017)

²https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html (11.11.2017)

3 Analysis of PropS and PropsDE

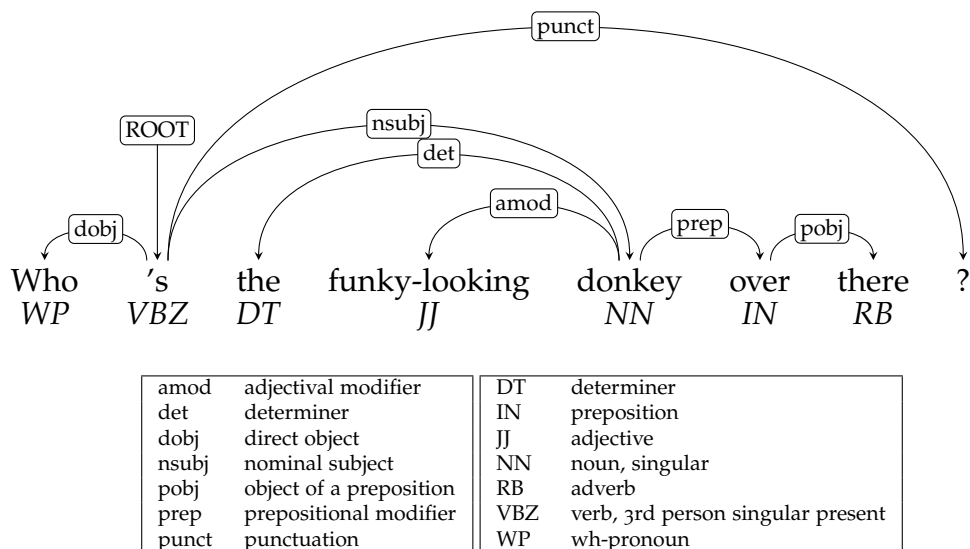
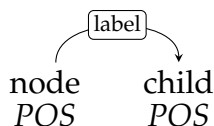


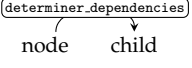
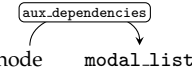
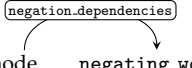
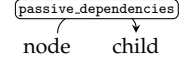
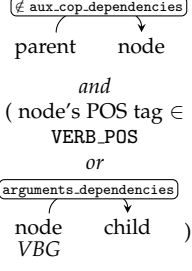
Figure 3.1: Example dependency parse with Stanford dependencies from the sentence *Who's the funky-looking donkey over there?* (Lee et al., 2013), including POS tags.



and test whether the POS tags, label, and the word of node and child node match a certain string or are within a certain list of strings. The rules for the linguistic feature extraction are listed in Table 3.1. The code for these extractions is located in *dependency_tree/tree.py*.

As we can see in Table 3.1, there is another set of rules for finding the tense feature. We do not look into these rules, because Falke et al., 2016, did not port them. Instead, they include an external library in order to determine the tense.

3 Analysis of PropS and PropsDE

Feature	Pattern	Feature Value	Note
Determiner		words of determiner child nodes	determiner_dependencies= {"det"}
Definite	$determiner \in$ definite_determiners or $POS \in$ determined_labels	definite_label or indefinite_label	definite_determiners= {"the", "this", "that", "these", "those", "another"} determined_labels= {"NNP"} definite_label="definite" indefinite_label="indefinite" executes the determiner-rule to get the <i>determiner</i>
Lemma	-	lemma	uses WordNet lemmatizer of Python nltk (http://www.nltk.org/11.11.2017)
Modal		list of modal auxiliary verbs	aux_dependencies= {"aux", "auxpass"} modal_list= {"may", "might", "must", "shall", "should"}
Negation		true (if there is a negation) or false	negation_dependencies= {"neg"} negating_words= {"not", "no"}
Passive Voice		true or false	passive_dependencies= {"auxpass"}
Tense	-	TENSE_PAST, TENSE_PRESENT, or TENSE_FUTURE	TENSE_PAST= "past" TENSE_PRESENT= "present" TENSE_FUTURE= "future" applies set of tense rules
Predicate		node object (or nothing)	aux_cop_dependencies= {"aux", "auxpass", "cop"} VERB_POS= {"VB", "VBD", "VBP", "VBZ", "VBN"} arguments_dependencies= {"arg", "agent"}

VB	verb base form	aux	auxiliary
VBD	verb past tense	auxpass	passive auxiliary
VBG	verb gerund/present participle	cop	copula
VBN	verb past participle	arg	argument
VBP	verb singular present non 3rd person	agent	complement of passive verb
VBZ	verb singular present 3rd person	neg	negation modifier
		det	determiner

Table 3.1: The rules for linguistic feature extraction in PropS.

3.1.3 Graph Conversions

After features are extracted and added to the respective nodes in a member variable, another set of rules is applied on the graph structure. These rules modify the graph, i.e. they delete edges and nodes and introduce new edges and nodes. After these rules have been applied, the graph represents the final extractions.

Whereas the feature extraction rules are rather similar to each other, the rules for graph conversions are more distinct. The rules rely on the dependency graph and on the extracted features. Also, some of the rules only work due to prior modifications. Thus, the order of the rules is important.

The rules are called in function *convert* in the file *graph_representation/convert.py*, and they are implemented in *graph_representation/graph_wrapper.py*.

At the beginning, the rules modify the graph so that unnecessary information is removed. That means that certain edges and nodes are deleted, and some nodes are merged together into a single node. Then the dependency graph is fixed in case the preprocessing did not provide the dependency graph completely the way PropS' rules expect it. After that, different aspects of the sentences are tackled, namely passives, existentials, conditionals, properties, adjectival complements, possessives, relative clauses, verbal modifiers, and conjunctions. Finally, certain edge labels are normalized.

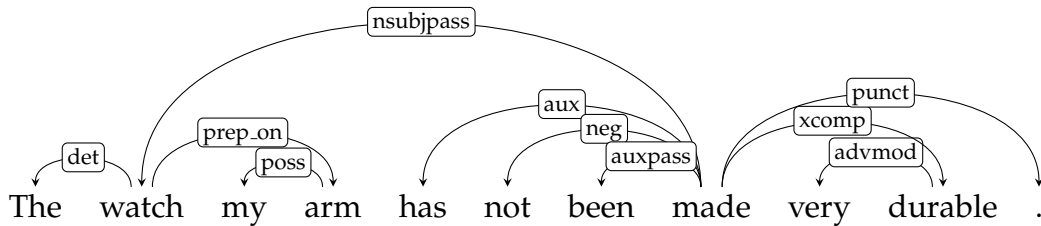
Remove Auxiliary Words

At this point, the information that auxiliary words contain, has already been stored as linguistic features in the previous set of rules. Therefore, these parts of the graph became unnecessary. Function *remove_aux* deletes all edges and end nodes, where the edge label $\in \text{ignore_labels} = \{\text{"det"}, \text{"neg"}, \text{"aux"}, \text{"auxpass"}, \text{"punct"}\}$.

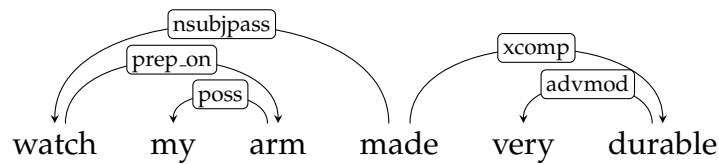
det	determiner
neg	negation modifier
aux	auxiliary
auxpass	passive auxiliary
punct	punctuation

3 Analysis of PropS and PropsDE

Example 5. In this example, we parse the sentence *The watch on my arm has not been made very durable* with the dependency parser. This results in the following graph:



After applying the rule for removing auxiliary words, the graph looks like this:



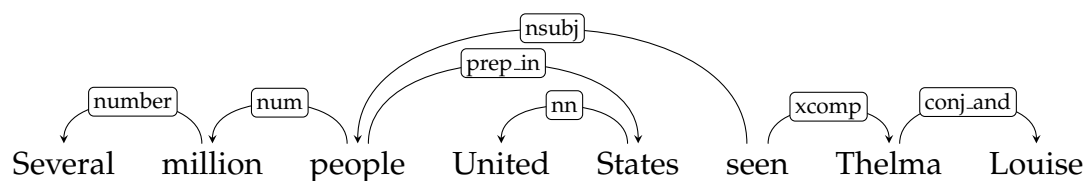
Merge Nodes

In order to further process entities which consist of several words, function *merge* iteratively combines two nodes to a single node, where the label of the connecting edge \in `join_labels` = {"mwe", "nn", "num", "number", "possessive", "prt", "predet", "npadvmod"}, or where there is a conjunction with the symbol '&'.

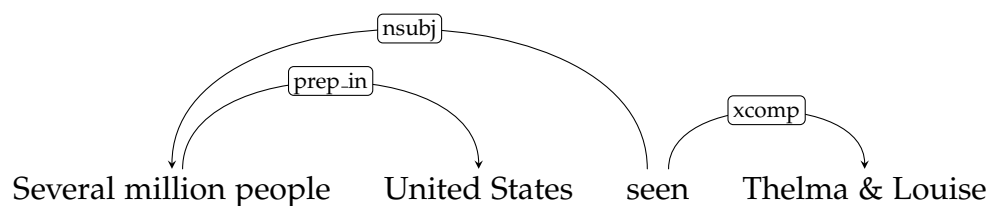
mwe	multi-word expression
nn	noun compound modifier
npadvmod	noun phrase as adverbial modifier
num	numeric modifier
possessive	possessive modifier
predet	predeterminer
prt	phrasal verb particle

3 Analysis of PropS and PropsDE

Example 6. When we parse the sentence *Several million people in the United States have seen Thelma & Louise* and apply the rule to remove auxiliary words on the dependency graph, we get the following graph:



After applying the current rule to merge nodes, the graph looks like this:



Fix Graph

There seem to be some issues where under certain circumstances the dependency graph does not have the structure which PropS expects. Function *fix* addresses these issues and removes certain nodes and edges and adds some edges. For example, this rule renames all edge labels "agent" to "prep_by".

Passives

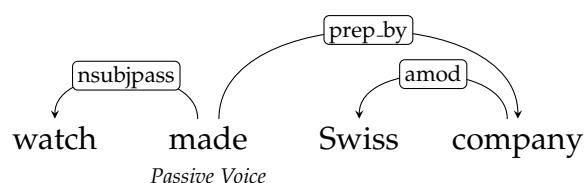
One explicit goal of PropS is to represent extractions in a uniform manner (Stanovsky et al., 2016). That includes that equal facts are represented in an equal way, independently from whether they are expressed in active voice or passive voice. This rule processes the passive voice feature, which has

3 Analysis of PropS and PropsDE

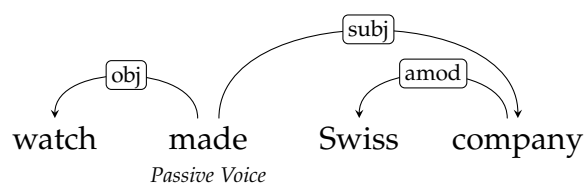
been extracted in the first set of rules for linguistic feature extraction. It is implemented in function *do_passives*. For each node, where passive voice is set as a feature, all outgoing edges with edge label \in `subject_dependencies` = {"subj", "nsubj", "nsubjpass", "csubj", "csubjpass", "xsubj"} are changed to edge label "obj". All outgoing edges with edge label "prep.by" are changed to edge label "subj".

nsubj	nominal subject
nsubjpass	passive nominal subject
csubj	clausal subject
csubjpass	passive clausal subject
xsubj	controlling subject

Example 7. For this rule, we look at the example sentence *The watch was made by a Swiss company*. After applying the preceding rules, we get this graph:



Then we apply the current rule which results in this graph:



Existentials

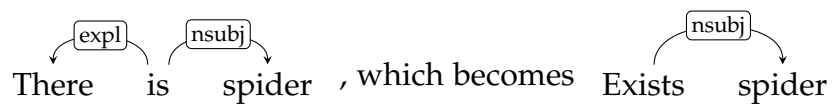
When a text states the existence of an entity, then the extracted fact should do so as well. The rule to process existentials is implemented in function

3 Analysis of PropS and PropsDE

do_existentials. In this rule, the source node of each edge with edge label "expl" is renamed to EXISTENSIAL = "Exists". The target node of this edge, which always contains an existential "there", and the edge, is deleted.

expl expletive

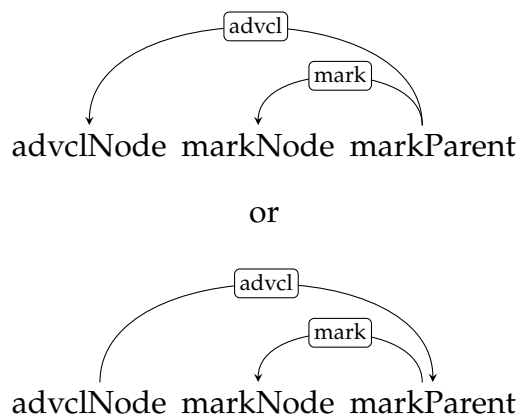
Example 8. The example sentence *There is a spider* results, after applying the prior rules, in the graph



with the current rule.

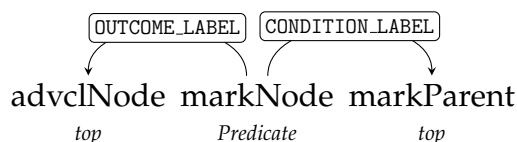
Conditionals

Another rule of PropS processes the parts of a sentence, where conditions are expressed. This rule is implemented in function *do_conditionals*. In this rule, the pattern



3 Analysis of PropS and PropsDE

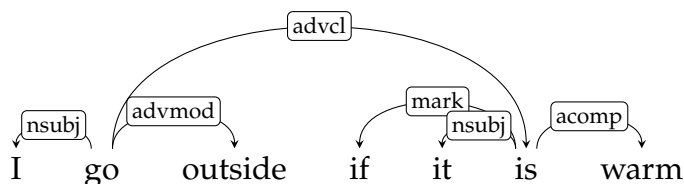
is matched, where the text of markNode $\in \{“if”, “while”, “because”, “although”, “as”, “once”\}$. (The order of the nodes within the sentence is not important.) These two edges are then deleted and replaced by two other edges:



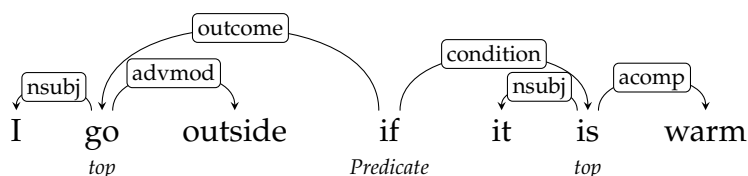
where CONDITION_LABEL = “condition” and OUTCOME_LABEL = “outcome”. This rule also sets the markNode as a predicate, which is important for later rules.

advcl	adverbial clause modifier
mark	marker word introducing advcl

Example 9. Just like in the examples of the other rules, we look at a sentence which has been processed by the preceding rules. The sentence *I will go outside if it is warm* results in the graph



The current rule converts this graph to

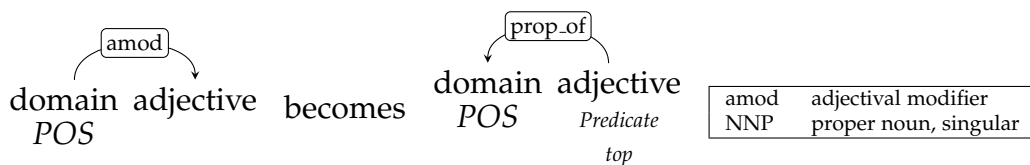


3 Analysis of PropS and PropsDE

Properties

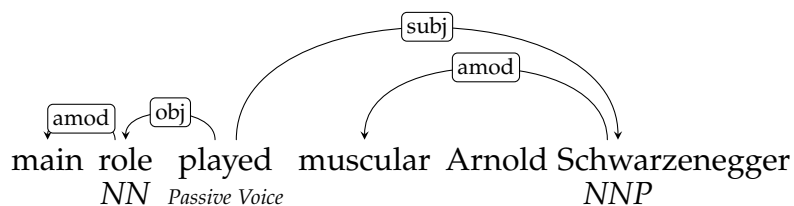
When a sentence expresses a property of an entity, the extraction should represent this property as well. PropS distinguishes between two types of properties: the "prop_of" property, which states an attribute of an entity, and the "SameAs" property, which states the equality of entities. This rule processes these properties. It is more complex compared to the preceding rules. It consists of three parts. The first part handles adjectives, the second part handles copula verbs, and the third part processes appositions. It is implemented in function *do_prop*.

The **first part** of this rule, which processes adjectives, identifies the following pattern with $POS \in \text{determine_labels} = \{\text{"NNP"}\}$. The edge is then replaced with a different edge, so that



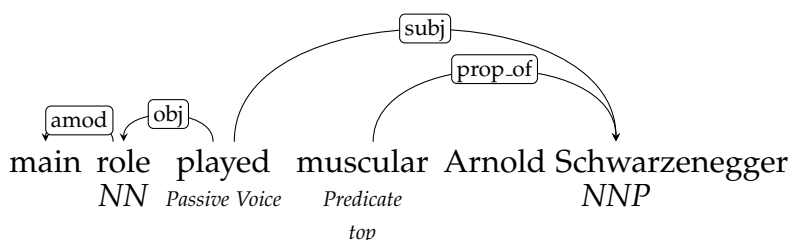
and the adjective node is marked as a predicate in the graph data structure, which is important for later rules.

Example 10. Let us take the sentence *The main role is played by muscular Arnold Schwarzenegger* as an example. The preceding rules produce the graph



The current rule modifies this graph:

3 Analysis of PropS and PropsDE

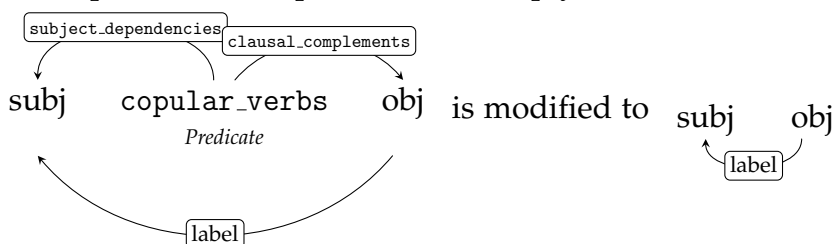


The **second part** of the properties rule processes copula verbs. In this part, we can distinguish four different cases. The following enumeration shows these cases and their results. The variables in these cases are `copular_verbs` = {"be", "am", "is", "are", "being", "was", "were", "been", "'s", "'re", "become", "became", "becomes"}, which is matched with text of nodes, `subject_dependencies` = {"subj", "nsubj", "nsubjpass", "csubj", "csubjpass", "xsubj"}, which is matched with edge labels, and `clausal_complements` = {"acomp", "xcomp", "comp", "ccomp"}, which is also matched with edge labels.

The patterns in the enumeration are ordered, and the order is important. The first pattern that matches is applied, even if another pattern would match as well. A dashed line means that the respective elements are optional.

nsubj	nominal subject	xsubj	controlling subject
nsubjpass	passive nominal subject	acomp	adjectival complement
csubj	clausal subject	xcomp	open clausal complement
csubjpass	passive clausal subject	ccomp	clausal complement

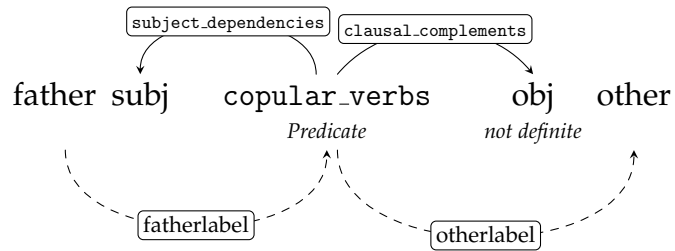
1. In this pattern, the copula node is simply removed:



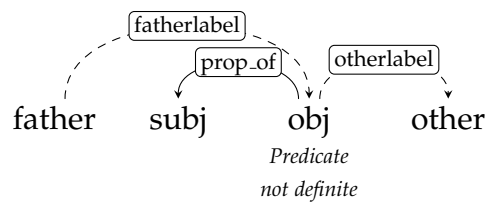
The `obj` node receives features of the copula node, e.g. tense, but it does not become predicate if it has not been predicate before.

2. The pattern

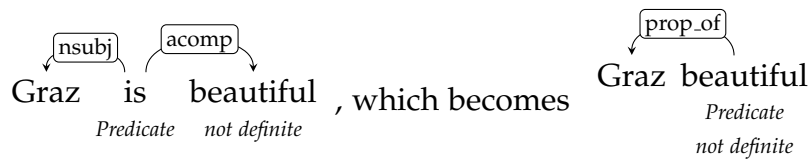
3 Analysis of PropS and PropsDE



is modified to

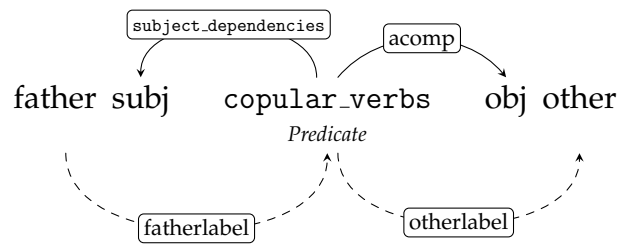


Example 11. From the sentence *Graz is beautiful*, we get the dependency graph



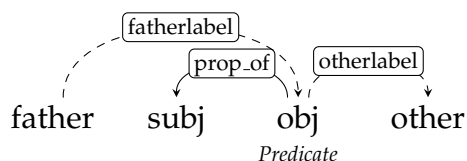
with the application of the current rule.

3. The pattern

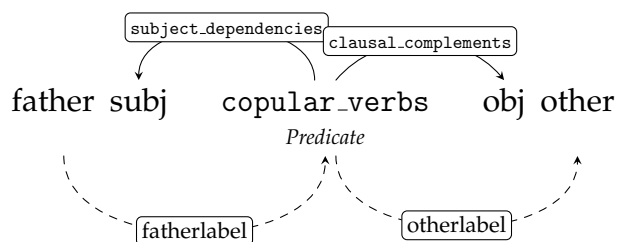


is modified to

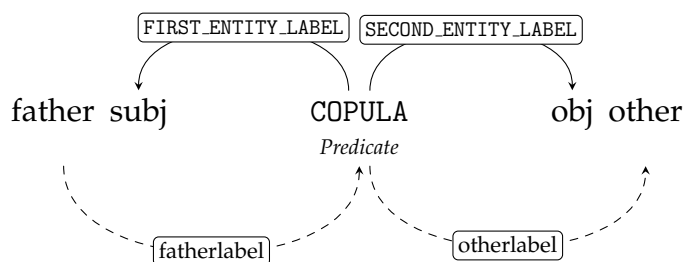
3 Analysis of PropS and PropsDE



4. The pattern

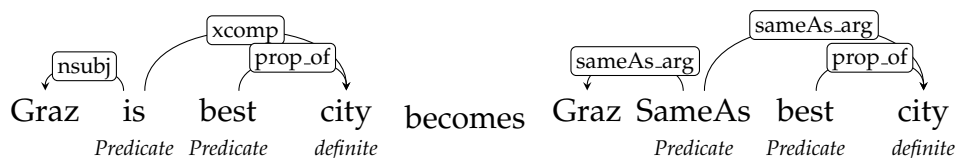


is modified to



with COPULA = "SameAs", FIRST_ENTITY_LABEL = "sameAs_arg", and SECOND_ENTITY_LABEL = "sameAs_arg". The COPULA node receives all the features of the previous copular_verbs node, e.g. tense.

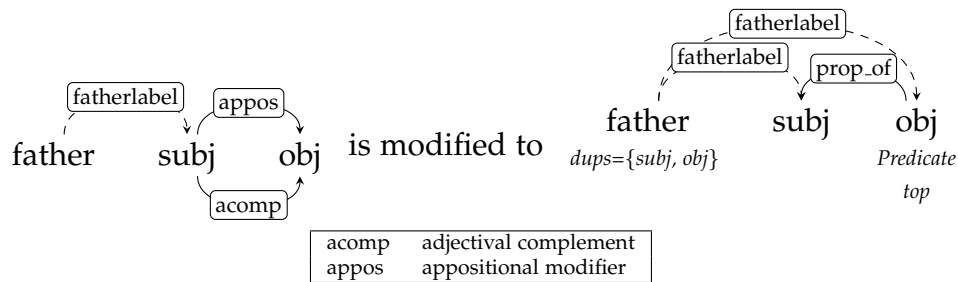
Example 12. Let us apply the preceding rules on the sentence *Graz is the best city*. After that, we apply the current rule. Thus,



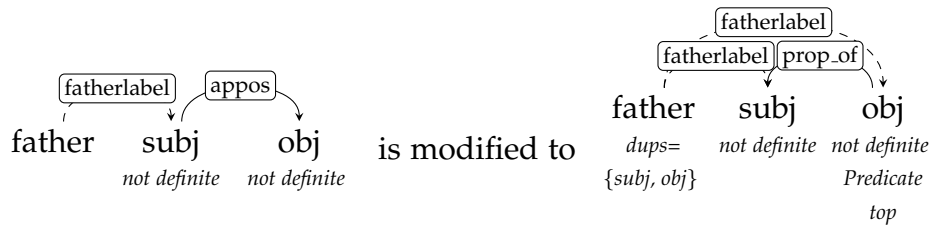
3 Analysis of PropS and PropsDE

The **third part** of this rule handles appositions. In this part, the feature *dups* is set. It means that from a contextual point of view, the nodes stored within this feature have equal meaning and are duplicates. This feature is important for the final extraction. Similar to the second part, it can be described with pattern matching and modification of three patterns.

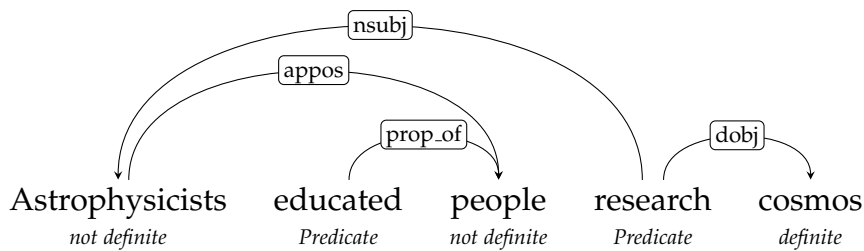
1. The pattern



2. The pattern

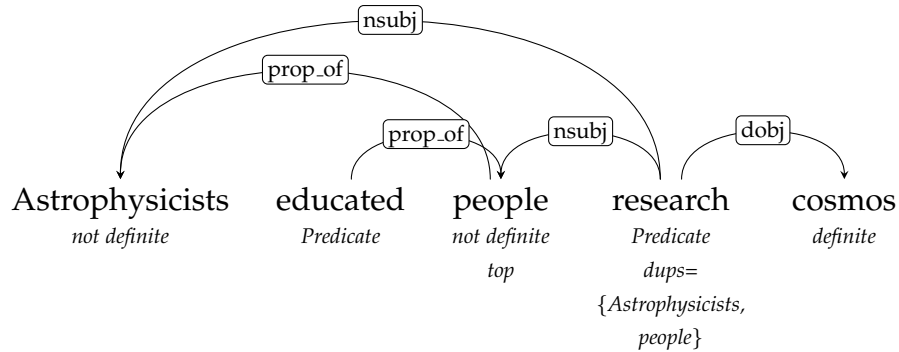


Example 13. The sentence *Astrophysicists, educated people, research the cosmos* results, after application of the preceding rules, in the graph

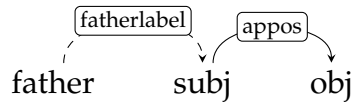


When we apply the current rule, we get

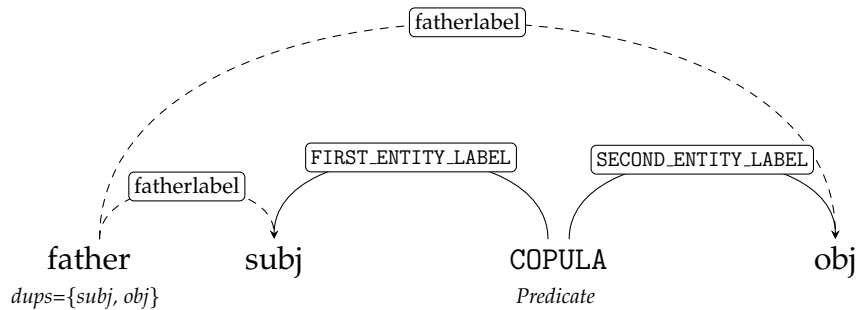
3 Analysis of PropS and PropsDE



3. The pattern



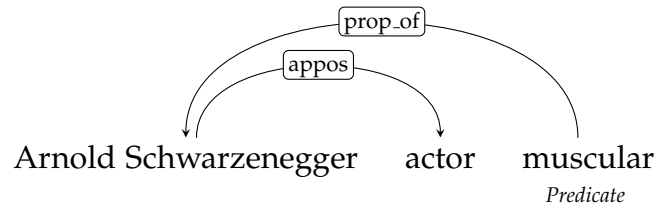
is modified to



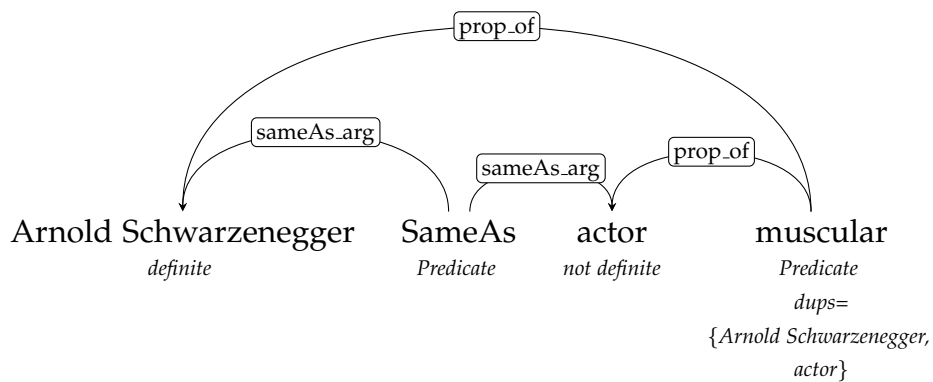
with COPULA = "SameAs", FIRST_ENTITY_LABEL = "sameAs_arg", and SECOND_ENTITY_LABEL = "sameAs_arg".

Example 14. Let us look at the sentence *Arnold Schwarzenegger, an actor, is muscular*. The preceding rules leave us with the graph

3 Analysis of PropS and PropsDE

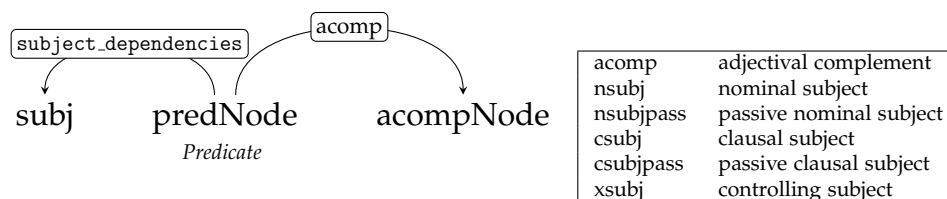


The current rule modifies this graph to



Adjectival Complements

While the previous rule handles adjectival complements together with copular verbs, this rule covers general occurrences of adjectival complements. The rule is implemented in function *do_acomp*. It matches the pattern

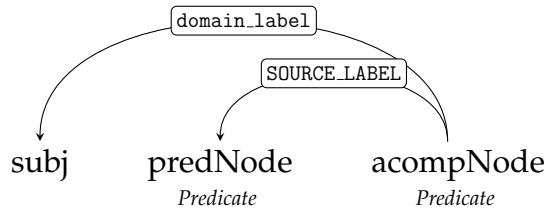


with `subject_dependencies = {"subj", "nsubj", "nsubjpass", "csubj", "csubjpass", "xsubj"}`.

3 Analysis of PropS and PropsDE

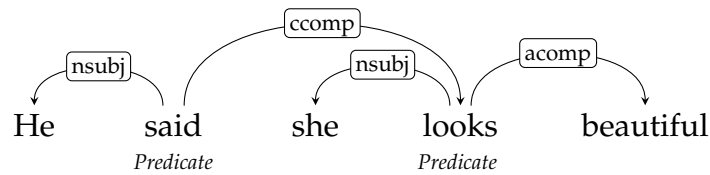
There are two cases:

1. The text in `predNode` \in `modalVerbs` = {"look", "appear", "begin", "come", "fail", "happen", "include", "prove", "remains", "said", "seem", "stand", "tend", "turn out"}. Then the graph is modified to

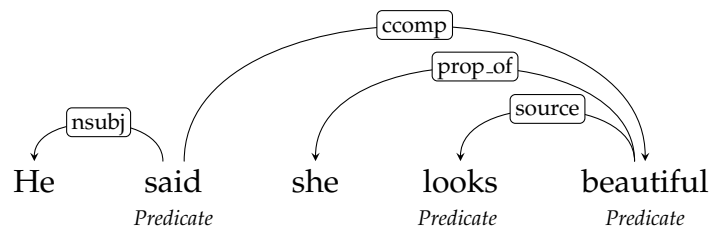


with `domain_label` = "prop_of" and `SOURCE_LABEL` = "source". All incoming edges to `predNode` become incoming edges to `acompNode`.

Example 15. In the example sentence *He said that she looks beautiful*, the graph



becomes



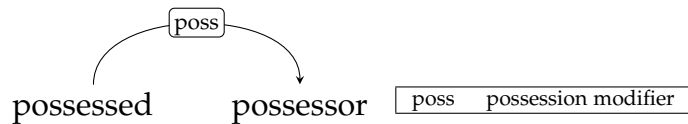
2. The text in `predNode` \notin `modalVerbs`. Then the `predNode` and the `acompNode` are treated like a multi-word expression. They are processed the same way as explicit multi-word expressions in the merge

3 Analysis of PropS and PropsDE

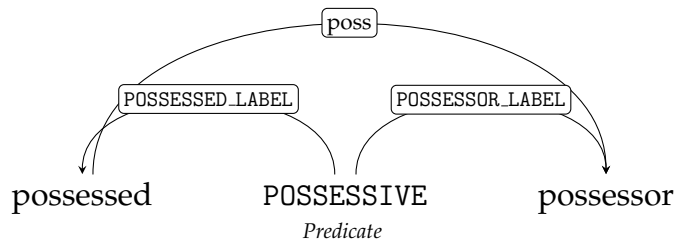
rule. Both nodes are combined to a single predicate node with the incoming and outgoing edges of both nodes as well as the features of both nodes.

Possessives

The rule that processes possessives is short and simple as dependencies already represent possession relationships. It is implemented in function *do_poss*.

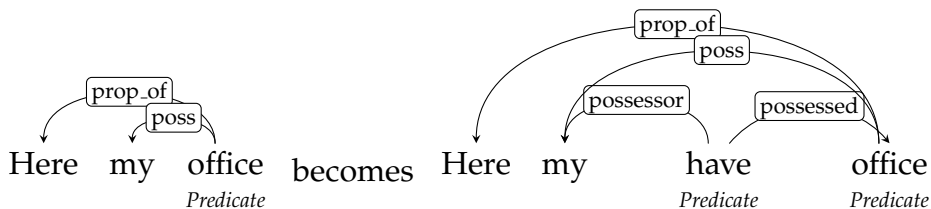


is modified to



with POSSESSIVE = "have", POSSESSED_LABEL = "possessed", and POSSESSOR_LABEL = "possessor".

Example 16. In the example sentence *Here is my office*



Relative Clauses and Verbal Modifiers

This rule is another short and simple one. It is implemented in function *do_vmod_relclause*. In this rule, each edge with edge label "rcmod" or "vmod", where the start node's POS tag \in `determined_labels = {"NNP"}`, is removed. If there is not already an edge in the opposite direction, then such an edge is added, with edge label `ARG_LABEL = "arg"`. In case of "rcmod", the end node of the "rcmod" edge is marked as *top*.

rcmod	relative clause modifier
vmod	reduced non-finite verbal modifier
NNP	proper noun, singular

Conjunctions

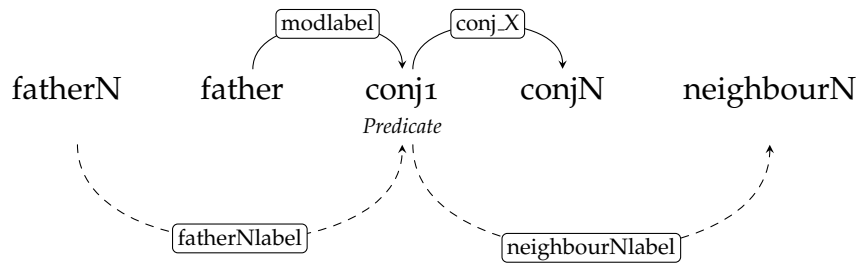
The conjunction rule is another complex rule. It handles conjunctions like "and" and "but". We again look at this rule from a perspective of pattern matching. This way, we can distinguish three patterns, and the first pattern that matches is applied. The common element of the patterns are dependencies with "conj_" edge labels. These can for example be "conj.and" and "conj.but", depending on the conjunction word. Within the following patterns, we simply use an X instead of conjunction words. A dashed line means that the respective elements in the pattern are optional. Each pattern has two distinct characteristics.

1. In the first pattern, the two distinct characteristics are: The element that is connected with a conjunction to others is a predicate, and there is a father node with an edge to the predicate with edge label \in `argument_dependencies \cup clausal_complements`.
`argument_dependencies` is a union of `subject_dependencies` (see earlier rules) and several other variables.
`argument_dependencies = {"arg", "agent", "obj", "dobj", "iobj", "pobj", "subj", "nsubj", "nsubjpass", "csubj", "csubjpass", "xsubj", "prop_of", "sameAs_arg", "possessed", "possessor", "outcome", "reason", "condition", "event"}`
`clausal_complements = {"acomp", "xcomp", "comp", "ccomp"}`

3 Analysis of PropS and PropsDE

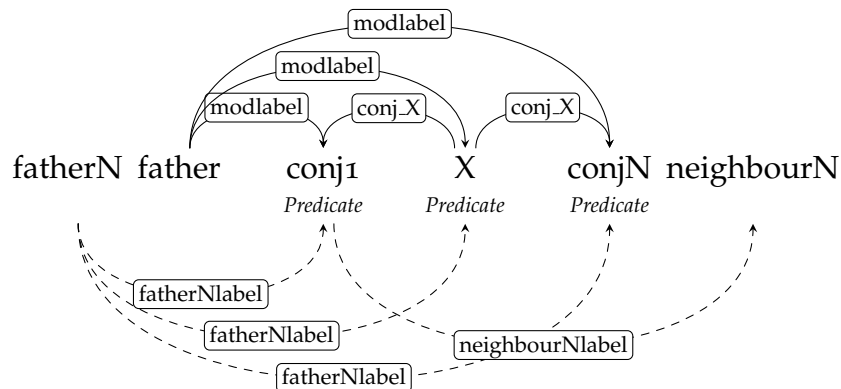
arg	argument	csubj	clausal subject
agent	complement of passive verb	csubjpass	passive clausal subject
dobj	direct object	xsubj	controlling subject
iobj	indirect object	acomp	adjectival complement
pobj	object of a preposition	xcomp	open clausal complement
nsubj	nominal subject	ccomp	clausal complement
nsubjpass	passive nominal subject	conj	conjunct

The pattern itself looks like this



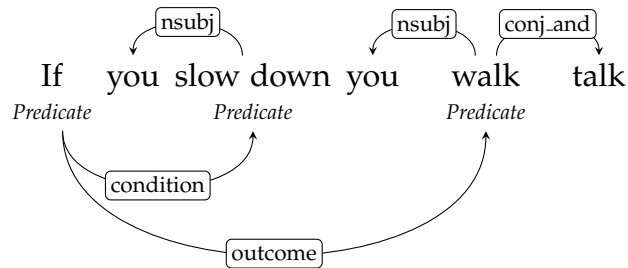
with $\text{modlabel} \in \text{argument_dependencies} \cup \text{clausal_complements}$. In this pattern, there can be an arbitrary number of additional father nodes and neighbour nodes, shown with fatherN and neighbourN. In a similar manner, there has to be at least one conjN node with a conj_X edge, and there can be arbitrary many additional ones, for which the same pattern applies.

The graph is then modified to

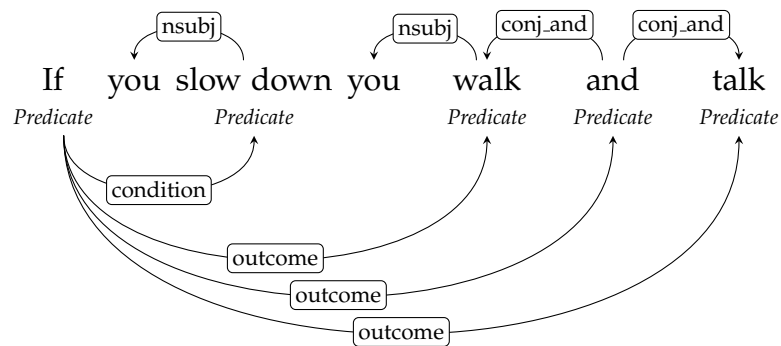


Example 17. For the sentence *If you slow down, you can walk and talk*, we receive the following graph from the preceding rules:

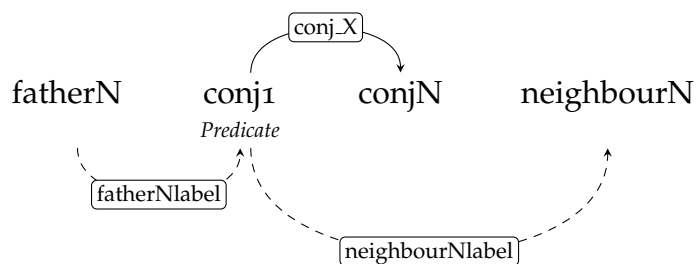
3 Analysis of PropS and PropsDE



This matches the current pattern and is therefore transformed to

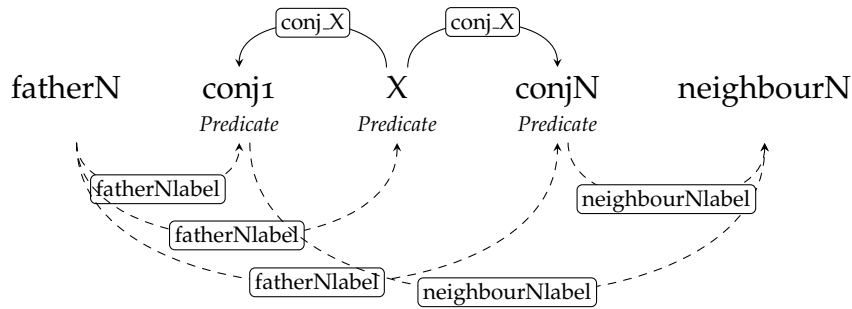


- In the second pattern, the two distinct characteristics are: The edge labels of edges from father nodes to the node of interest \notin `argument_dependencies` \cup `clausal_complements`, and the node of interest is again a predicate. The pattern is



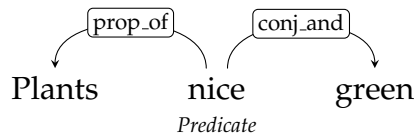
with `fatherNlabel` \notin `argument_dependencies` \cup `clausal_complements`. This pattern is modified to

3 Analysis of PropS and PropsDE

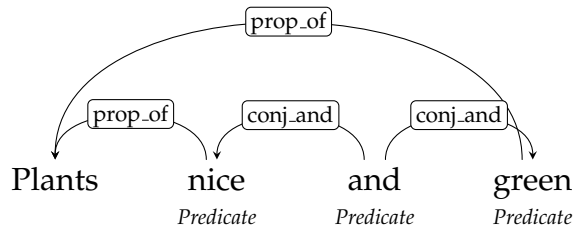


Only in this pattern, edges to neighbours are copied to conjN nodes.

Example 18. Preceding rules produce the following graph from the example sentence *Plants are nice and green*:

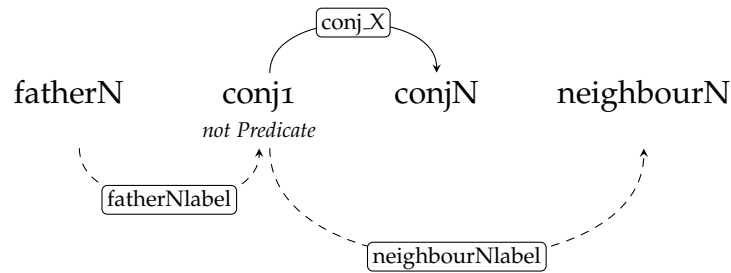


The current rule transforms this graph to

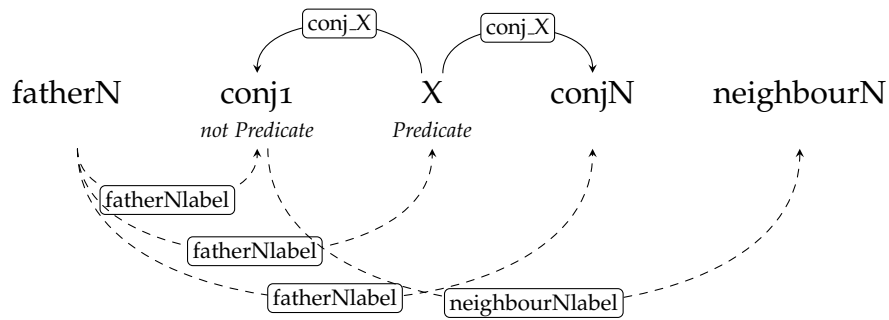


- In the third pattern, the two distinct characteristics are: The node of interest is not a predicate, and the edge labels of edges from father nodes to the node of interest \notin $\text{argument_dependencies} \cup \text{clausal_complements}$.
The pattern is

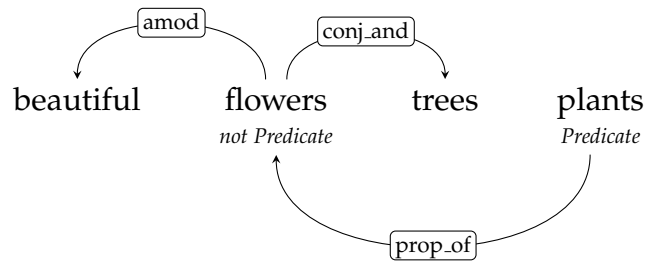
3 Analysis of PropS and PropsDE



with $\text{fatherNlabel} \notin \text{argument_dependencies} \cup \text{clausal_complements}$. This pattern is modified to

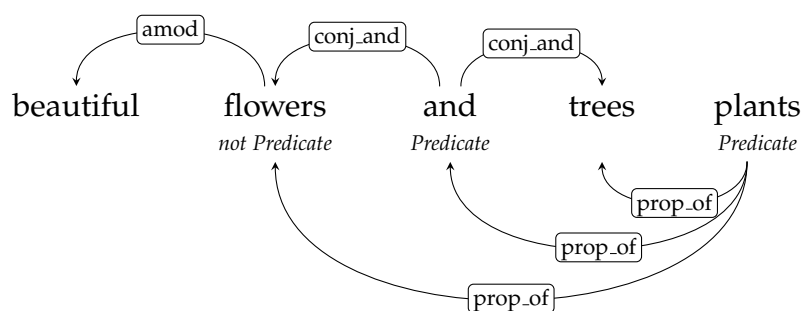


Example 19. From the sentence *Beautiful flowers and trees are plants* we receive from the preceding rules the graph



which is then modified by the current rule to

3 Analysis of PropS and PropsDE



Normalize Labels of Remaining Edges

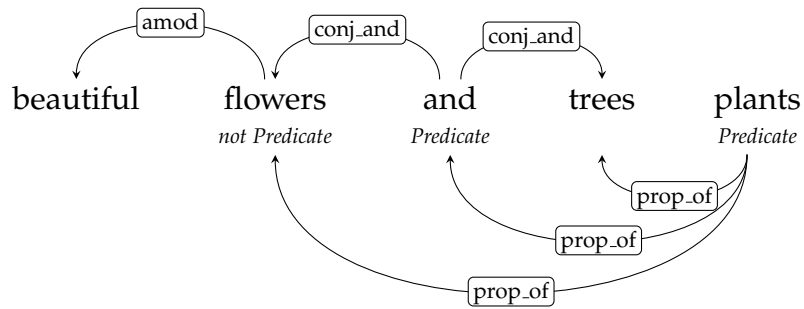
When the program flow arrives at this rule, most of the edges that come from the dependency parser have been processed. For the remaining edges from the dependency graph, the edge labels are normalized. The following table shows how function *normalize_labels* does that: (The table is implemented in a Python dictionary, and we have noticed that "acomp" occurs twice. Since the content of Python dictionaries is in arbitrary order, we do not know whether "acomp" is normalized to "source" or to "comp". It is possible that this case never occurs, since "acomp" is processed in multiple preceding rules.)

Normalized	Original
subj	xsubj, nsubj, nsubjpass, csubj, csubjpass, possessor
comp	xcomp, ccomp, acomp
source	acomp
mod	amod, advcl, rmod, advmod, quantmod, vmod
dobj	possessed

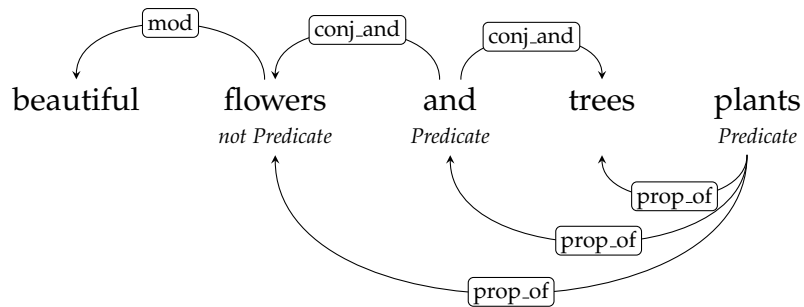
Example 20. For this example, we continue with the sentence from the previous example, *Beautiful flowers and trees are plants.*

With the application of the current rule,

3 Analysis of PropS and PropsDE



is modified to



Set Top Nodes

In function *calcTopNodes*, some additional nodes are set as *top*. This feature is used in the extraction part of PropS. *top* marks nodes, which become relations in contrast to arguments.

There are two cases, where nodes become top nodes:

- Nodes which do not have parent nodes
- Nodes, which have an outgoing edge with edge label SOURCE_LABEL = "source" to a top node

3.1.4 Extractions

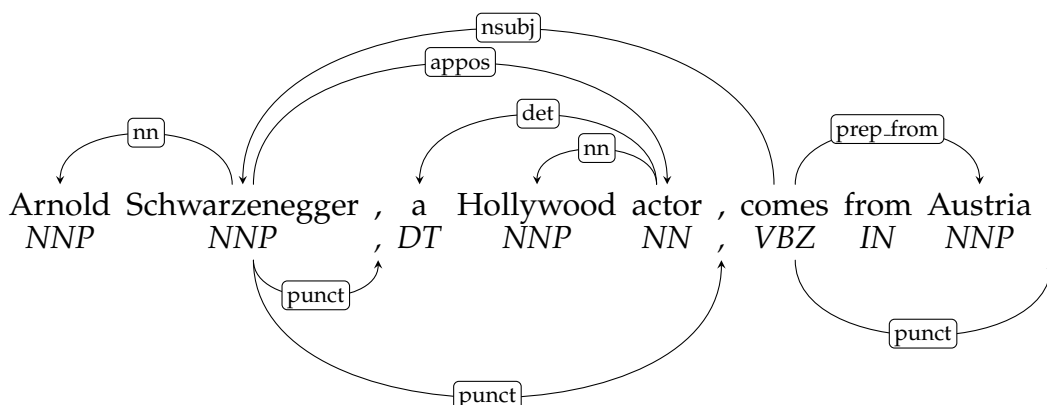
Finally, the actual facts are generated in function *getPropositions* in file *graph_representation/graph_wrapper.py*. This function builds extractions by identifying the relations and the arguments in the converted graph. The nodes which represent the relations have already been marked in the previous step. They are marked with the feature *top*. The rules, which set this feature are the rules for *Conditionals*, *Relative Clauses*, *Properties*, and *Set Top Nodes*.

A relation and its arguments are strings. Each argument is built from the subgraph of a child of the relation. In order to build the argument, the surface forms of the respective nodes are concatenated. The relation also is the surface form of the node.

Apart from *top*, there is another feature that is important for the extractions. The feature *dups* is set in the *Properties* rule in case an apposition occurs. It means that two subtrees are semantically equivalent.

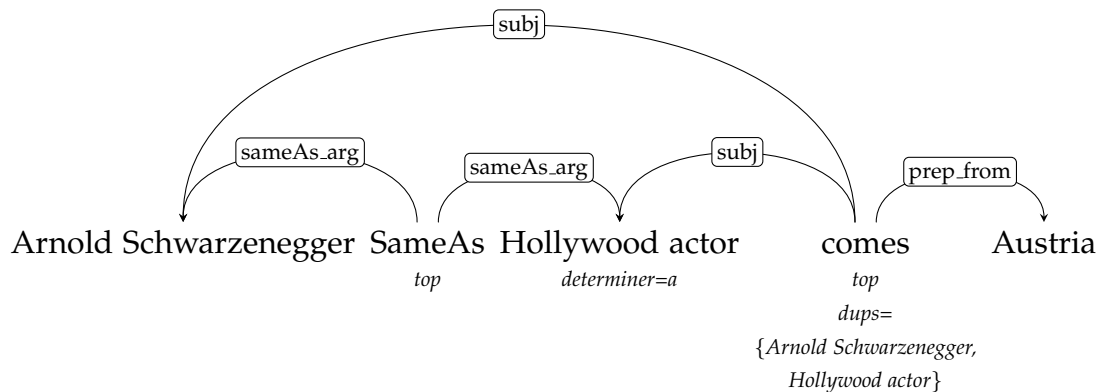
If a *top* node has *dups* nodes, then the extraction is generated multiple times, with one of the *dups* nodes as argument at each instance.

Example 21. In this example, we process the sentence *Arnold Schwarzenegger, a Hollywood actor, comes from Austria.* POS tagging and dependency parsing generates the following graph as input for the ruleset:



3 Analysis of PropS and PropsDE

After extracting lexical features, applying the graph conversion rules results in the graph



The final extractions from this sentence are

- *comes:(subj:Arnold Schwarzenegger , prep_from:Austria)*
- *comes:(subj:a Hollywood actor , prep_from:Austria)*
- *SameAs:(sameAs_arg:Arnold Schwarzenegger , sameAs_arg:a Hollywood actor)*

3.2 Differences between PropS and PropsDE

In this section, we look at the differences between PropS (Stanovsky et al., 2016) and PropsDE (Falke et al., 2016). We are only interested in differences that concern the parsed language and the ruleset. Apart from that, Falke et al., 2016, also made a few other changes to the system, which we do not look into.

As with the previous section about the ruleset of PropS (Section 3.1), we mainly look at the layout of the rules and the structural differences instead of the linguistic implications. We classify these differences in small and large

3 Analysis of PropS and PropsDE

differences. This classification is performed subjectively from our own point of view.

Within the differences between PropS and PropsDE, we can observe two general types.

- One general type of difference is at the edge labels and POS tags. Whereas PropS uses Penn Treebank style POS tags and Stanford dependencies for English, PropsDE uses Tiger Treebank style¹ POS tags and dependencies. These POS tags and dependencies are specified for the German language. This difference appears in every rule, and especially in the variables that we saw in the previous section.
- The other general type of difference is that the structure of the dependency tree is different. Therefore, the elements within the graph that are identified in order to extract linguistic features, and the elements that are identified and then modified, are also different.

Section 3.1 shows how facts are extracted in PropS. There, we already gained a general understanding of how the rules work. Therefore, a detailed comparison between each single element of the rules of PropS and PropsDE does not help to improve our general understanding. Instead, we want to have an overview over the position, the extent, and the manner of the difference.

In the following, we take a look at the differences in the dependency graph (Section 3.2.1). Then we continue with the differences in the extraction of linguistic features (Section 3.2.2). After looking at the differences in the graph conversions (Section 3.2.3), we also want to know how Falke et al., 2016, changed the final fact extraction.

3.2.1 Dependencies

PropsDE uses different POS tags and dependencies, namely Tiger Treebank style POS tags and dependencies, instead of Stanford dependencies and Penn Treebank style POS tags. Mate parser (Bohnet and Nivre, 2012) is used

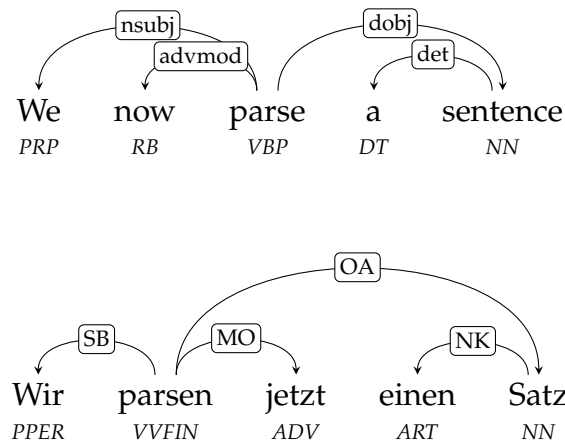
¹http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/TIGERCorpus/annotation/tiger_introduction.pdf (11.11.2017)

3 Analysis of PropS and PropsDE

to find those POS tags and dependencies, and JoBimText (Biemann and Riedl, 2013) is used for collapsing dependencies (dependency collapsing is an additional step of dependency parsing, and other parsers do that implicitly).

The dependency parsing in PropsDE, as well as in PropS, results in a tree structure which contains the dependencies between words of a sentence. The difference between PropS and PropsDE is the structure itself, as it is based on a different language, and the edge labels.

Example 22. These are the dependency trees of the same sentence in English and in German. The example illustrates the different POS tags, dependency labels, and a slightly different tree structure.



3.2.2 Linguistic Features

As immediate part of the ruleset, the extraction of linguistic features exhibits interesting differences between PropS and PropsDE. Table 3.2, displays an overview over the differences with the most important aspects. The rules for extraction of linguistic features in PropS are shown in Table 3.1 in the previous section.

3 Analysis of PropS and PropsDE

Feature	L/S	Feature Value	Difference in PropsDE
Determiner	S	words of determiner child nodes	additional test for POS tag \in determiner_pos
Definite	S	definite_label or indefinite_label	additional case where node has child with specific edge label
Lemma	L	lemma	lemma found by Mate parser
Modal	L	list of modal auxiliary verbs	single modal verb in parent node instead of multiple modal verbs in child nodes
Negation	S	true (if negation) or false	no difference except in negation_dependencies
Passive Voice	S	true or false	different pattern
Tense	L	"past", "present", or "future"	external Python module <i>pattern.de</i> (www.clips.ua.ac.be/pages/pattern-de 11.11.2017)
Predicate	S	node object (or nothing)	different pattern
Subjunctive	L	subjunctive_label (or nothing)	only in PropsDE

Table 3.2: Differences between PropS and PropsDE during linguistic feature extraction. Column *L/S* shows, whether there is a large (L) or a small (S) difference.

3.2.3 Graph Conversions

After linguistic features have been extracted, PropsDE, as well as PropS, converts the dependency graph to a graph with semantic connections between nodes. In this section, we look at the differences between the graph conversions of PropS and PropsDE.

PropsDE has four additional rule functions, that do not have immediate counterparts in PropS. These four functions are implemented in the same file as the other rules (*dependency_tree/graph_wrapper.py*). They are *remove_aux_mod_de*, *do_conj_propagation*, *do_relc_de*, and *do_comp_de*.

In principle, the graph conversions of PropS and PropsDE are similar. In

3 Analysis of PropS and PropsDE

each rule, elements within the graph are identified and modified. The structure of the final graph is also similar between PropS and PropsDE. The differences within a single rule mainly concern the elements which are identified in order to modify the graph. Table 3.3 shows the main differences in graph conversions between PropS and PropsDE.

Rule	L/S	Function	Difference in PropsDE
Remove Auxiliary Words	L	<i>remove_aux</i> <i>remove_aux_mod.de</i>	pattern in <i>remove_aux</i> completely different <i>remove_aux_mod.de</i> only in PropsDE nodes to remove may have children to keep
Merge Nodes	L	<i>merge</i> <i>do_comp.de</i>	accept indefinitely long chains of nodes different pattern in <i>merge</i> <i>do_comp.de</i> only in PropsDE <i>do_comp.de</i> merges indirectly connected nodes
Fix Graph	L	<i>fix</i>	relabel collapses from JoBimText additional collapses
Passives	L	<i>do_passives</i>	different pattern for subj node text manipulation in subj
Existentials	L	<i>do_existentials</i>	different pattern additional existential from grammatical object
Conditionals	L	<i>do_conditionals</i>	different pattern accept indefinitely long chain of nodes no <i>top</i> node
Properties	S	<i>do_prop</i>	slightly different patterns
Adjectival Complements	S	<i>do_acomp</i>	slightly different pattern
Possessives	S	<i>do_poss</i>	slightly different pattern
Relative Clauses	L	<i>do_vmod_relclause</i> <i>do_relc.de</i>	<i>do_vmod_relclause</i> only in PropS <i>do_relc.de</i> only in PropsDE completely different consider German relative pronoun set <i>top</i> node in <i>do_relc.de</i>
Conjunctions	L	<i>do_conj</i> <i>do_conj_propagation</i>	<i>do_conj_propagation</i> only in PropsDE modify edges to Stanford dependencies style <i>do_conj</i> almost identical
Normalize Labels of Edges	S	<i>normalize_labels</i>	different labels to normalize rename unhandled labels to generic "dep"
Set Top Nodes	-	<i>calcTopNodes</i>	identical

Table 3.3: Main differences between PropS and PropsDE during graph conversions. Column L/S shows, whether there is a large(L) or a small(S) difference.

3.2.4 Extractions

After the graph has been converted so that it contains semantic connections between the nodes, the final facts are extracted. We have seen how the extraction works in PropS in Section 3.1.4. In general, the extraction in

3 Analysis of PropS and PropsDE

PropsDE works the same way. Whereas the *dups* feature is set the same way in both OIE systems, the *top* nodes, which become relations in the extractions, are set at slightly different locations. The *top* nodes are set at the following locations:

Rule	PropS	PropsDE
Conditionals	<i>do_conditionals</i>	–
Relative Clauses	<i>do_vmod_relclause</i>	<i>do_relc_de</i>
Properties	<i>do_prop</i>	<i>do_prop</i>
Set Top Nodes	<i>calcTopNodes</i>	<i>calcTopNodes</i>

The final extractions are set in function *getProposition*. In this function, there are some minor differences compared to PropS. Two examples are, that argument subtrees, which start with an edge labeled "dep", are omitted, and if a relation is negated, the word "nicht" is added in front of the relation text. Apart from small differences, the final extractions are assembled the same way in PropsDE as in PropS.

4 Method

In the practical part of this thesis, we construct and train a neural network for Open Information Extraction (OIE) (see Section 2.3). For the implementation, we use the machine-learning framework TensorFlow¹ in version 1.3.

We perform these major steps:

1. **Experiments:** We train and test several models to compare a few configurations.
2. **Design:** With knowledge gained from the experiments, we design our model.
3. **Training:** We train the selected model with a large dataset.
4. **Evaluation:** Finally, we evaluate the model.

In this chapter we cover the methods that we apply in these steps. Since the focus of this thesis is Information Extraction (IE), we only apply the methods in this chapter as means to do IE. (We do not extensively study these methods.) Therefore, we only go into detail as far as it is required to build our open information extractor.

In order to feed words and sentences into a neural network, we encode them into word embeddings, which we describe in Section 4.1. We take a look at types of neurons in Section 4.2, and at neural network architectures in Section 4.3. We train the model with training data that we automatically generated with PropsDE (see Section 2.3.2). In Section 4.4, we explain how we train the model. After training the model, we evaluate our information extractor. Section 4.5 contains our plans for the evaluation. We reveal the detailed results of the training and evaluation of the model in the next chapter (Chapter 5).

¹<https://www.tensorflow.org/> (11.11.2017)

4.1 Word Embeddings

To extract facts from sentences, a neural network first needs to receive those sentences in a way that it can process them. A way for that is to replace words with word embeddings (Bengio et al., 2006). Embeddings are vector representations of words, and we can set these vectors as input to our neural network. They are stored as a lookup table and trained together with the neural network.

In order to speed up training and get better results, we initialize the embeddings with GloVe (Pennington, Socher, and Manning, 2014) embeddings. GloVe is a tool that learns embeddings from a dataset from word-word co-occurrence statistics. To generate initialization values for the embeddings of regular German words, we use GloVe on 300k German sentences (Remus, Quasthoff, and Heyer, 2010).

We feed the sentences into the neural network in a format similar to CoNLL-X (Buchholz and Marsi, 2006). In this format, the dependency graph is displayed in a table, where each word is given together with its Part-of-speech (POS) tag and dependency to its parent in the graph. Therefore, in addition to embeddings for regular German words, we generated joint GloVe embeddings for POS tags and dependency labels (see Section 2.3.1) from our training data.

For the embeddings of regular words, we have a few practical issues to address:

1. If a certain word appears only once within the whole training dataset, it is possible, that the neural network does not fully learn how to process this word. Therefore, we want the neural network to see each word multiple times. We limit the vocabulary to the most common words. This is not required for POS tags and dependency labels, since there is only a small and specific number of them.
2. This measure leads us to the problem that there appear words in the training dataset (as well as in other sentences) which the neural network cannot learn as it has no vector representation of them. To cope with this problem, we introduce 15 placeholders. These replace the unknown word with a placeholder-word, and the embedding of

4 Method

that placeholder-word is fed into the neural network instead. If the number of available placeholders is exceeded, we replace any other unknown word with an *unknown* token. The embedding vectors of the placeholders and the *unknown* token are initialized randomly with a low standard deviation around the average embedding.

3. Apart from that, there is a set of special tokens that appear only at the output. These tokens are argument types, that PropsDE emits, a special relation ("SameAs", see also Section 3.1.3), separators for the extraction structures, an *empty* token, and an *end-of-sentence* token. We also initialize the embeddings for these tokens (except for *empty* and *end-of-sentence*) randomly, although we expect to never see them in the input.
4. Finally, we also have cases, where there are embeddings for all the words at the input, but not for the relation word at the output. This case occurs due to a specific part of the German language. It is possible, that the unknown word at the output is a compound word of two words that appear at the input. One solution to that problem would be to add this word to the embeddings, but we have already mentioned the reason for limiting the embeddings to a certain set of words. Since this word only appears at the output, it also does not make much sense to use a placeholder-word here. Instead, we decided that the two words have to appear separately at the output. This is not correct, but the meaning of the relation should be undeniably clear.

Setup

For regular words, we have embedding vectors with 128 dimensions, and the vocabulary consists of 11399 words. For POS tags and dependency labels, the embedding vectors consist of 64 dimensions, and the vocabulary contains 256 words. We decided to use medium sized vectors, because from the extractor's perspective, the actual meaning of the words is not important. The main task, the extractor has to learn, is to correctly assign words from the input to specific locations at the output. Except for specific ones, all the output words are also in the set of input words. Therefore, we expect there to be enough information in vectors of this size plus the POS tags to

perform the task successfully.

4.2 Artificial Neurons

Feedforward Neural Network

Our model includes a feedforward neural network. In a feedforward neural network (Rosenblatt, 1958), information flows only in one direction, from one layer to the next. A single neuron in this network is defined by its weights. An incoming vector \vec{x} is multiplied by the weight vector of the neuron $\vec{w}_{layer_i, neuron_j}$ and a bias $b_{layer_i, neuron_j}$ may be added. Before the result is passed on to the next layer (or the output), an activation function f is applied. A complete layer within a neural network therefore calculates

$$f(\vec{x}W_{layer_i} + \vec{b}_{layer_i})$$

In our model, we actually do not use an activation function, because our feedforward layer produces the output for a multiclass classification (each class represents a word index). Instead, we simply select the class with the highest value, and we can skip the additional computation.

Long Short-term Memory

Long short-term memory units (LSTM, Hochreiter and Schmidhuber, 1997) are a type of recurrent units. Recurrent neurons are a class of artificial neurons, that are capable of memory. Compared to feedforward neurons, the structure of LSTMs is rather complex. Since we use TensorFlow's LSTM implementation, we do not need to consider its details. Thus, we describe the LSTM only briefly.

LSTMs consist of an input gate, a forget gate, an output gate, 8 weight matrices, 4 bias vectors, and an internal state vector. Each gate represents an activation function. Multiple operations are performed on the input and the internal state in order to determine the output and the next internal state.

4 Method

We decided to try LSTMs because they have proven themselves useful in natural language processing applications, e.g. for summarizing text (Lopyrev, 2015). We set *tanh* as activation function for the LSTMs in our model. During training experiments (see Section 5.1), we discovered that gated recurrent units outperform LSTMs for our purpose.

Gated Recurrent Units

Similar to LSTMs, gated recurrent units (GRU) are a type of recurrent neurons. They were proposed by Cho, Merrienboer, Bahdanau, et al., 2014, for neural machine translation. Since their initial proposal is in the field of natural language processing, we want to see how well they perform for our task. Our experiments (see Section 5.1) show, that GRUs yield better results than LSTMs. Therefore, they are a central element of our final model.

GRUs are related to LSTMs, as they have a comparable structure. They consist of an internal state vector, weight matrices, and gates. But in contrast to LSTMs, GRUs only have two gates, 6 weight matrices and 3 bias vectors. For the same vector size, a GRU requires less memory and a smaller amount of operations than an LSTM.

We use TensorFlow's implementation of GRUs. In our model, we use the rectifier (ReLU, Hahnloser et al., 2000) as activation function:

$$f(x) = \max(0, x)$$

4.3 Architecture

Input and Output

We already mentioned, that we use embedding vectors to feed words into the neural network. Since we want to imitate PropsDE, we want to feed complete dependency graphs into the neural network. In order to achieve that, we use a graph format that is inspired by the CoNLL format (Buchholz

4 Method

and Marsi, 2006). We represent each node with a 400-dimensional node vector that contains

1. embedded node ID (8d)
2. embedded head node ID (8d)
3. embedded surface form (128d)
4. embedded lemma (128d)
5. embedded POS tag (64d)
6. embedded dependency label (64d)

The node ID and the head node ID are replaced with a position embedding vector of size 8. The embeddings of node ID and head node ID come from two separate embedding matrices, which were randomly initialized.

For the output, we expect one-hot encoded token indices. To achieve that, we add a single feedforward layer at the output. From this layer's result vector, the index with the largest value becomes the output token index.

Sequence-To-Sequence Model

The publication of Zhang, Duh, and Van Durme, 2017, inspired us to use a Sequence-to-sequence (Seq2Seq) model in the practical part of this master's thesis. Seq2Seq models (Cho, Merrienboer, Gulcehre, et al., 2014) implement the general idea that the neural network first receives a sequence of inputs, and then emits a sequence of results. Due to this nature, Seq2Seq models consist of recurrent neural networks. In our case, we used LSTMs and GRUs for our Seq2Seq implementation.

Seq2Seq models consist of two neural networks, an encoder and a decoder, with the same size. The encoder receives a sequence of inputs, and during training it learns to keep important information within the internal states. Therefore, after the encoder has seen the whole input sequence, the final internal state is a representation of the whole input sequence. After the encoder has finished, the internal state of the decoder is initialized with the final state of the encoder. In the simple case of Seq2Seq, which we use for our model, the encoder's outputs are discarded. The decoder's outputs, on

4 Method

the other hand, are fed back into the decoder's input as embeddings, which improves the decoder's performance compared to zero input.

To our encoder, we feed a sequence of single node vectors, and we expect a sequence of single tokens from the decoder.

Bidirectional Recurrent Neural Network

Bidirectional recurrent neural networks have been introduced by Schuster and Paliwal, 1997. They support the neural network in finding links between pieces of information at different locations in an input sequence. This way, they can improve the neural network's result.

In bidirectional recurrent neural networks, the input sequence is provided in two directions, forward and backward. The two passes, forward pass and backward pass, can be performed with the same neurons (same weights and activation). The outputs and the states of the two passes are not connected, and the internal states are set to an initial state for each pass. After the two passes, the internal states and/or the output states of the two passes can be concatenated and fed into an output layer.

We use the bidirectional architecture in our model. The encoder of our Seq2Seq model becomes a bidirectional recurrent neural network (with GRUs). It receives the node vectors forward and backward. After the two passes, the encoder's internal states are concatenated and set as the initial internal state of the decoder. A few test runs showed us that distinct forward and backward networks performed slightly better compared to a single network for both passes. Thus, in our model there are two distinct recurrent neural networks for the forward pass and the backward pass. Our encoder neural networks consist of 3 layers with 512 GRUs. The decoder is a single neural network of 3 layers with 1024 GRUs.

4.4 Training

In order to train our model, we need data. We apply Mate Tools (Bohnet and Nivre, 2012) and JobImText (Biemann and Riedl, 2013) to generate the dependency graphs for the input part, and PropsDE to generate the output part of our dataset. The sentences come from Remus, Quasthoff, and Heyer, 2010. For experiments, we use $\sim 10k$ sentences, and for the final training, we use $\sim 54k$ sentences. We divide the dataset into training data, validation data for the stopping criterion, and test data for the evaluation.

We apply a few modifications on the dataset (see Sections 4.1 and 4.3). These modifications address embedding issues and the way we feed the dependency graph into our models. Apart from that, we also modify the separators and other tokens in the extractions in order to simplify the distinction between special symbols for the extraction format and regular punctuation.

Our loss function is cross entropy. For discrete values, cross entropy is

$$H(p, q) = - \sum_x p(x) \log q(x)$$

We minimize that with the Adam optimizer (Kingma and Ba, 2014) with a minibatch size of 64. In order to speed up the final training, we use sampled softmax loss (Jean et al., 2014) for that.

To prevent the neural network from overfitting, we apply two regularization techniques, dropout (Hinton et al., 2012) and simple validation-based early stopping. During the initial experiments we set the dropout rate to 0.5. For the final training, we start with dropout rate 0.5 in the first epoch, and lower it towards zero over a few epochs. For the validation-based early stopping, we randomly select 640 data samples as validation set from the generated dataset. In order to find a good validation minimum and not stop too early in a local minimum, we allow the validation error to be higher than the minimum for 5 consecutive epochs. When these epochs are overstepped, we stop and accept the weights at the validation minimum.

We set the target result from the training data as the decoder’s input for training. For the final training, we additionally train with a random mix of

4 Method

the decoder's output and the ground truth as the decoder's input in order to increase robustness (Bengio et al., 2015).

4.5 Evaluation

For the evaluation, we use two datasets: a test dataset with 640 samples with a ground truth produced by PropsDE, and an evaluation set of 300 sentences, which has been used to evaluate PropsDE (Falke et al., 2016). The test dataset is a random mix of sentences from Wikipedia and news texts. It is randomly selected out of the training dataset and removed from the training data, thus, the model has never seen it before. The 300 sentences from the evaluation set are in equal partitions from the web, Wikipedia, and from the news.

From the extractions of the test dataset, we calculate the word precision, which is

$$\frac{\text{number of correct words}}{\text{total number of words}}.$$

This gives us an idea of how well the model is able to directly imitate PropsDE.

From the test dataset, we also calculate a soft precision. We view the outputs of our neural network and of PropsDE as bags of words, and we count how many words from the neural network's prediction output also appear in the target output from PropsDE. To get the soft precision value, we calculate

$$\frac{\text{number of positives}}{\text{total number of words}}.$$

This precision value gives us a hint about the portion of information that our neural network is able to remember from the input.

With the evaluation set, we let the model produce extractions, and we manually label those extractions as correct and incorrect. We also take a look at how well the model performs at predicate extraction. To do that, we calculate the precision of the relations without their arguments from

4 Method

the evaluation set. In the same manner, we evaluate the different types of arguments, such as subject and object, and the syntax. The syntax precision shows how well the neural network is able to produce correct relation-argument structures, independently from the actual content.

Since a sentence can result in multiple extractions, an issue during this evaluation is to correctly separate different extractions in a single sequence. We simply split the sequence at the symbol that represents the end of an extraction ");". This also means, that, if the model fails to correctly set this symbol either by not setting it, or by setting it within an extraction, it influences the total number of extractions, and therefore the precision value. Nevertheless, we split the sequences in order to properly represent multiple extractions. In some cases, the neural network might produce a sequence of separator symbols without words, e.g. ":: ::););)". After the end of an extraction, we consider these sequences to be random output and discard them.

5 Results

5.1 Experiments

Fix-Input-Sequence-Output Architecture

We experimented with several variants, where we set the whole dependency graph as fixed input to the neural network. A few LSTM layers should then read from the fixed input and yield extractions in an output sequence. Variants considered different numbers of layers and different layer sizes. In all of the variants we tried with this architecture, the loss function did not converge to an acceptable level. Therefore, we discarded these models.

Sequence-to-sequence Models

In order to design a promising model, we further tested several configurations with Sequence-to-sequence (Seq2Seq) models. Here, we did not yet employ bidirectional recurrent neural networks. The results of these experiments can be seen in Table 5.1.

Upon these results, we decided to use three layers of GRUs for our model, with 512 units as bidirectional encoder, and 1024 units as decoder. We initially trained it with ~18k training samples. With the test dataset of 640 sentences, this model produces an output word precision of 24.4 %. The manual evaluation with 300 sentences shows, that the model achieves a predicate precision of 20.0%. From those 300 sentences, only one extraction is completely correct. In the final training, we train the model with ~54k training samples.

5 Results

Model	Validation Loss	Test Loss	Training Time
GRU, 2 × 512	0.933	1.097	28 h
GRU, 3 × 512	0.913	1.074	32 h
GRU, 3 × 1024	0.873	1.038	73 h
LSTM, 2 × 512	0.982	1.154	35 h
LSTM, 3 × 512	1.001	1.175	80 h
LSTM, 3 × 1024	1.013	1.186	70 h (manual stop)

Table 5.1: Training experiments with Sequence-to-sequence models. Each configuration employs either GRU or LSTM cells, and consists of two or three layers with 512 or 1024 units. The same validation and test dataset have been applied to each configuration. Stopping criterion: validation error remains higher than its minimum for 5 consecutive epochs. Loss function: cross entropy. During these experiments, the decoder exclusively received ground truth values as input.

5.2 Evaluation

In this section, we evaluate our final model (see Section 4.5).

We directly compare our model with PropsDE extractions with a test dataset. This results in a per-word precision of 28.1% (26.5% with the validation set). The soft precision is 51.5%. That means that 51.5% of the output words from our model are also found in the output words of PropsDE, independently from their location in the sequence.

We also manually evaluate the extractions with a new set of sentences. It is the same set of sentences, that has been used to evaluate PropsDE. From the evaluation set, 4 completely correct extractions are generated by our model. From 300 sentences, three are too long for our model. In total, the model generates 327 extractions from the 297 sentences, which means an average of 1.1 extractions per sentence.

The vocabulary for our neural network contains $\sim 11k$ words. In order to cope with the fact, that many words do not appear in this vocabulary, we replace them with placeholder words. 27.1% of the input words (surface form and lemma) are replaced with placeholders.

For a more detailed impression of the model’s performance, we evaluate the precision of multiple aspects of the output. Predicates are correctly identified

5 Results

Argument	Occurrences	Precision
subj	219	25.6%
mod	74	20.3%
sameAs_arg	5	20.0%
obj	24	16.7%
iobj	6	16.7%
dobj	151	13.9%
prop_of	29	10.3%
prep_	105	8.6%
conj_	4	0.0%
poss	1	0.0%
condition	3	0.0%
outcome	7	0.0%

Table 5.2: Detailed arguments evaluation. Number of occurrences and precision values of different types of arguments.

in 38.8% of the extractions, and 34.6% of the extractions have correct syntax (i.e. correct relation-argument structure). Apart from that, we also evaluate the various types of arguments. The evaluation of the arguments shows, that their performance is unevenly distributed. The results are displayed in Table 5.2. In the next chapter we discuss possible reasons for these results (see Section 6.3).

6 Discussion

During the analysis of PropS (Stanovsky et al., 2016) and PropsDE (Falke et al., 2016) (see Chapter 3), we learned how a rule-based Open Information Extraction (OIE) system can be ported from English to German. Then we developed a neural network, that mimics the OIE ruleset from PropsDE. This neural network consists of a bidirectional sequence-to-sequence model, which receives nodes from a dependency graph as input, and produces PropsDE extractions (see Chapter 4).

In this chapter, we reflect over the analysis, the implementation, and the results of this thesis. At first, we look at the immediate limitations of the design in our practical part (Section 6.1). Then we describe a few problems that we encountered during the implementation phase (Section 6.2). In Section 6.3, we discuss what we ultimately learned with this master’s thesis, from the analysis as well as from the practical part. Finally, in Section 6.4, we give some ideas for possible future work based on this thesis.

6.1 Limitations

Due to the fact that generating training data is slow, we limit the training data to $\sim 54k$ samples. Because of this limitation, we have to limit certain aspects of our design.

- Sentence length: The amount of training data requires us to limit the sentence length. This is necessary because we input each word together with a position embedding. The model learns the embedding of each position. For rarely occurring (i.e. high) word positions, there are only few training samples. Thus, the learned embeddings for these

6 Discussion

word positions might not be optimal. In order to reduce badly trained position embeddings, we limit the input sequence length to 50 words.

- Vocabulary size: Similar to the sentence length, the size of the training dataset also requires us to limit the number of known words (~11k words). Since the neural network has to learn an embedding for each word, we decided to only include the most frequent words. Other words are replaced by placeholder words, because, similar to position embeddings, we expect the performance for rarely occurring word embeddings to be low. As a result of this limitation, 27.1% of input words are replaced with placeholder words (see Section 5.2).
- Placeholder words: The number of placeholder words is limited to 15.
- Compound words: The relation word can be a compound word of two words from the input (see Section 4.1). In this case, we split the concatenation into two separate words. This is not correct, but we decided to do that anyways to facilitate the learning task. If we would keep the original variant of the relation word, the neural network needs to learn which concatenations of words lead to which results. This requires learning capacities from the neural network, which we prefer to keep for our main task.

6.2 Problems and Mistakes

During the implementation and training of the neural network, we made mistakes and encountered issues that we had to find and solve. We describe two problems that we found educational as they had a large impact on the model.

- Output sequences often have different lengths. Therefore, each batch is made as long as the longest output sequence of this batch, and shorter sentences are padded with an *empty* token. Our mistake here was that we included the padding in the calculation of the word precision without realizing that this would distort the result. We thought that it was necessary for the neural network to learn that when the output sequence is done, it should not emit any more words, and we also wanted to reflect that in the evaluation. It turned out that

6 Discussion

the neural network quickly learns to emit padding values after an *end-of-sentence* symbol, but it is much more difficult for it to learn to emit the correct words in front of the *end-of-sentence*. If we include padding into the calculation, the value looks much better than it would without padding, but it does not accurately represent the actual performance. We address this issue by excluding the emitted values, that should be padding, from this calculation.

- We also had a seemingly small programming mistake. The result of it was, that the frequently occurring German word "der" was not part of the embedding vocabulary. Therefore, "der" was always replaced with a placeholder word. Placeholder words should only replace words that rarely appear in text, and "der" was strongly overrepresented within replaced words. In this situation, the neural network learned to output placeholder words too often and incorrectly.

6.3 Lessons Learned

Analysis of PropS and PropsDE

Falke et al., 2016, show us how to port an English OIE system to German. In our analysis (Chapter 3) we see that it is necessary to attend to the details of the languages in order to correctly adapt Part-of-speech tags, dependency labels, and dependency tree structures for the new language. Even though the two languages English and German are similar to a certain extent, a direct port with simply replacing tags, labels and structures is not possible. In the analysis, we see that in multiple cases, the rules for the two languages need to be completely different. We can assume that this would also be valid when porting PropS to another language.

The ruleset of the two OIE systems consists of three parts, that build upon each other:

1. Extraction of linguistic features
2. Conversion from the dependency tree to a graph that represents semantic aspects

6 Discussion

3. Generation of the final extractions

In the first two parts of the ruleset, there is a mix of large and small differences, and in the third part, there are only a few small differences. The differences in the ruleset represent the differences in the languages. In most of the cases, small differences in the ruleset originate from the different Part-of-speech (POS) tags, dependency labels, and dependency tree structures, and therefore from the small differences in grammar. Due to the fact that German and English are languages with many similarities, these small differences are numerous. Large differences in the ruleset originate from large differences in the languages. These large differences in the languages are concepts and lingual constructs that only exist in one language, but not in the other. One example for this is the possibility of indefinitely long chains of nodes in PropsDE: In the *Merge Nodes* rule, it represents the German *Funktionsverbgefüge* (e.g. "Platz nehmen"). This is a construction, that does not exist in English.

For the development of our neural network, the most important observation is that the rules completely rely on the dependency tree and the POS tags. From this observation, we conclude that the dependency tree and the POS tags carry all the syntactic information that is required to retrieve semantic aspects. Our neural network has to learn to retrieve semantic aspects in order to extract facts. If our neural network also had to identify the syntactic elements, then it would explicitly require a partition of its capacity for this task. Therefore, we expect that feeding the dependency tree with POS tags to the neural network simplifies the learning task.

Neural Network for OIE from German Text

Zhang, Duh, and Van Durme, 2017, who use a bidirectional Sequence-to-sequence (Seq2Seq) model for OIE and implicit machine translation, achieve a BLEU (bilingual evaluation understudy, Papineni et al., 2002) of 18.94 % for extractions. BLEU is a score related to precision but modified for the use in bilingual settings. We can compare their performance with ours, but we have to keep the similarities and differences of their task and ours in mind. They use a similar architecture, and they train their model to do

6 Discussion

OIE by imitating an existing system. In contrast to our task, they include the implicit translation from Chinese sentences to English extractions. We expect the machine translation to increase the difficulty, and therefore decrease the performance of their task in comparison to ours. Taking that into consideration, we expect their result of 18.94% to be a lower bound for our result.

Our model achieves a word precision of 28.1% (see Section 5.2). This precision is in relation to PropsDE extractions. That means, our model is able to directly mimic PropsDE's output to 28.1%.

Apart from that, our model achieves a soft word precision of 51.5%. It is the portion of tokens that is shared between our model's output and PropsDE's output, without considering the location of the words. This is an interesting value, as it gives us an idea of our model's apprehension. We can interpret this value as the amount of information, the Seq2Seq model holds, after receiving the full input sequence. Following this interpretation, our model is just large enough to grasp half of the information that it receives at the input.

Our model produces less extractions than PropsDE. From the 300 sentences of the evaluation dataset, our model generated 327 extractions, whereas PropsDE generated 487 extractions from the same dataset. With an average of 1.1 extractions per sentence, our model is below PropsDE's 1.6 extractions per sentence.

From the manual evaluation with the evaluation dataset, we get a glimpse at how real-world application of our model would perform. Our model was able to generate 4 extractions, that we consider correct:

6 Discussion

1974, nach dem ersten verlorenen Duell um das
Ministerpräsidentenamt, holte ihn der Sieger Rabin als
Verteidigungsminister in sein Kabinett.

haben :(subj :: er , dobj :: Ministerpräsidentenamt);

Jean-Jacques Rousseau hieß der Mann.

heißen :(subj :: der Mann , mod :: <unknown> Rousseau);

Erst rund vier Stunden nach der Festnahme durfte die BI-Vorsitzende
telefonisch die Betreuung ihrer Kinder organisieren und über das
"Legal Team" der Demo-Organisatoren eine Rechtsanwältin
verständigen.

haben :(subj :: sie , dobj :: BI-Vorsitzende);

Traditionelle Lernmedien stoßen hier schnell auf ihre Grenzen.

haben :(subj :: sie , dobj :: Lernmedien);

The fact that our model only produces a few correct extractions in our evaluation leads us to the conclusion, that it is not yet suitable for application environments.

In the following table, we can observe how our model attempts and fails to assemble three example extractions:

6 Discussion

Wegen dieses ehrenamtlichen Engagements wurde er 2006 von
Kinderlachen mit dem Kind-Award ausgezeichnet.

Our model	Kind-Award :(subj :: von Kinderlachen , obj :: er er prep_ prep_ :: dieser <unknown> , , prep_ mit dem dem Kinderlachen , , mod :: ::);
-----------	---

PropsDE	auszeichnen:(subj:von Kinderlachen , obj:er , prep_wegen:dieses ehrenamtlichen Engagements , prep_mit:dem Kind-Award , mod:2006)
---------	---

So stiegen die Fahrgastzahlen drastisch an.

Our model	steigen :(subj :: die Fahrgastzahlen , dobj :: Fahrgastzahl an. , mod :: So , mod :: ::);
-----------	---

PropsDE	ansteigen:(subj:die Fahrgastzahlen , mod:So , mod:drastisch)
---------	---

GASAG ist seit 1995 Hauptsponsor des Eishockey-Clubs Eisbären
Berlin im Profi- und Juniorenbereich.

Our model	SameAs :(prep_ seit :: 1996 , sameAs_arg :: <unknown> , sameAs_arg :: Vorsitzender des Eishockey-Clubs Eishockey-Club seit Eisbären und und und im im im <placeholder7>);
-----------	---

PropsDE	SameAs:(prep_seit:1995 , sameAs_arg:GASAG , sameAs_arg:Hauptsponsor des Eishockey-Clubs Eisbaeren Berlin im Profi- und Juniorenbereich)
---------	--

Within these extractions, the extractor has difficulties with the correct order of words as well as with the full extent of information contained in the sentence. The model needs to at least partially learn to apply German grammar, and fully remember the information contained within a sentence. These results show us that this task is too difficult to learn for our model.

We also made an interesting observation: After the initial training with ~18k training samples, our neural network was only able to produce a single correct extraction (see Section 5.1). The final model is, however, not able to produce this extraction any more:

6 Discussion

Aber während seines letzten Berliner Studienjahres ging eine grundlegende Veränderung mit ihm vor.

Old model	haben :(subj :: er , dobj :: <unknown> Studienjahr);
Final model	gehen :(subj :: eine Studienjahres Mannschaft mit fünf grundlegende vor. , prep_ während :: seiner <unknown> <unknown> , , prep_ :: :: ::);

It is especially notable that the simple structure of the correct extraction resembles the structure of those extractions, that our final model is able to build correctly. From this observation, we suspect that our neural network has already reached the limit of its capacities. The larger amount of training data might have changed the learning focus towards different lingual concepts. The limit of this neural network forces it to forget previously learned concepts in order to learn the concepts which are stronger represented in the larger dataset.

The unevenly distributed precision values of the different types of arguments (see Table 5.2 in Section 5.2) might fit this interpretation. The precision order displays the learning focus, but we do not know the reason for this result. We consider two possibilities:

- A larger amount of training data, e.g. 1 million training samples, would lead to a more even distribution of the precision values.
- A larger neural network is required in order to provide the learning capabilities for all types of arguments.

The second possibility would fit the interpretation of the forgotten lingual concept due to limited learning capacities.

For predicate extraction as well as for the extraction of full relation-argument tuples, there is a lot of room for improvement. Future work can be based upon our design. In the following section (Section 6.4), we contemplate about ways for improvement.

6.4 Future Work

PropS and PropsDE

The effort to port PropS to another language similar to English, like Dutch or Danish, might be similar to the effort for developing PropsDE. In contrast to that, porting it to a completely different language, e.g. an Asian language, might be more difficult. In order to possibly facilitate future porting processes, it would be interesting to develop an abstraction of the ruleset. This abstraction could consist of easy-to-use building blocks for rules with the goal to provide a programming library or framework for developing a rule-based OIE system for a new language.

Neural Network for OIE from German Text

In order to improve the neural network that we developed in our practical part, we have to consider the neural network itself as well as its training environment.

A first approach for improvements to the model could be to increase the amount of training data. We trained our model with a dataset with $\sim 54k$ samples. A larger amount of data could allow for a larger vocabulary and better performance with rare word embeddings and rare position embeddings.

The training data has been generated by a large and complex set of rules. During training, we expect the neural network to learn to accurately imitate the application of these rules. That includes a partial understanding of German grammar, and fully remembering the contents of the sentence. This might be beyond its capacity. A way to address this issue would be to train a much larger network with, for example, 6 layers of 2048 bidirectional encoder units and the respective 6 layers of 4096 decoder units. The task to imitate PropsDE could be kept, and, therefore, training data can easily be generated with PropsDE. A drawback to this approach is, however, that it would require a large amount of computational resources.

6 Discussion

Another approach would be to tweak the target task to not directly imitate PropsDE. PropsDE (and other OIE systems) do not emit absolutely correct extractions, and this approach could reduce learned errors. In order to compensate for incorrect extractions from individual OIE systems, it might be possible to include multiple OIE systems that support German, e.g. PropsDE and GerIE (Bassa, 2016). After their extractions are converted to a compatible form, a classifier could be trained (with manually annotated data) to distinguish between extractions with errors and extractions that humans consider correct. A pipeline of multiple OIE systems, converting tools, and the classifier could then be used to generate training data. With this training data, a training experiment could show, whether the bidirectional Seq2Seq architecture can learn to build correct extractions. The neural network would not have to learn to imitate PropsDE's ruleset, and it might be able to gain a more general understanding of the OIE problem.

7 Conclusion

Information Extraction (IE) is the task to extract structured information from semi- or unstructured information. Unstructured information includes natural language text, on which we focus in this master's thesis. IE from natural language text usually relies on language features, thus, IE methods usually are designed for specific languages.

We consider three degrees of language independence in IE: multilingual IE, cross-lingual IE, and truly language agnostic IE. Multilingual IE methods extract information from texts from a specific set of languages. Cross-lingual IE methods additionally deliver the extractions in languages different from the source language. The ideal in language independence is truly language agnostic IE, which supports and performs equally well on any human language. Truly language agnostic IE methods do not exist.

IE methods implement either fixed schema IE or Open Information Extraction (OIE). In fixed schema IE, domains, relations, and entities are at least partially predefined. In contrast to that, OIE extracts any relation and entity that is given by the text. OIE systems usually consist of two steps: a preprocessing step and an extraction step. Part-of-speech (POS) tagging and dependency parsing are the most common preprocessing methods among several possibilities. The extraction step is either rule-based or training based.

There are only a few OIE systems for German text available. Two rule-based OIE systems for German are PropsDE and GerIE. PropsDE is a German port of the English OIE system PropS. We introduce a neural network that mimics PropsDE.

In order to understand how the rule-based system PropsDE performs OIE from German text, we have analyzed PropS and PropsDE. Apart from

7 Conclusion

learning about OIE from German text, this analysis shows us how a system can be translated from one language to another. The translation of an IE system into a new language is a possible step towards multilinguality.

The ruleset consists of three major partitions: extraction of linguistic features, modifying the dependency graph, and building the final extractions. When we look at their differences, we find that there are many small, but also a few large differences. The majority of the small differences are in POS tags, dependency labels, and dependency graph structures. The large differences address lingual concepts that are only available in one of the languages.

After we analyzed PropS and PropsDE, we built a neural network that mimics PropsDE's behaviour. The bidirectional sequence-to-sequence model consists of gated recurrent units. Its encoder receives a sequence of nodes from the dependency graph, encoded in embeddings. The decoder's target output is a set of PropsDE extractions as a sequence of one-hot encoded word indices. We trained the model with training data that we generated with PropsDE.

In the evaluation, we showed that our model is able to directly mimic PropsDE to 28.1%. Our model produces many extractions that appear promising, but are not completely correct. In future work based on our design, it should be possible to increase the performance.

Bibliography

- Akbik, Alan and Jürgen Bross (2009). “Wanderlust: Extracting semantic relations from natural language text using dependency grammar patterns.” In: *Proceedings of the 2009 Semantic Search Workshop at the 18th International World Wide Web Conference*. SemSearch '09 (cit. on pp. 12, 18).
- Andersen, Peggy M., Philip J. Hayes, Alison K. Huettner, Linda M. Schmandt, Irene B. Nirenburg, and Steven P. Weinstein (1992). “Automatic Extraction of Facts from Press Releases to Generate News Stories.” In: *Proceedings of the Third Conference on Applied Natural Language Processing*. ANLC '92, pp. 170–177 (cit. on p. 8).
- Banko, Michele, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni (2007). “Open Information Extraction from the Web.” In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence*. IJCAI'07 (cit. on pp. 11, 12, 18, 22).
- Bassa, Akim (2016). *GerIE: Open Information Extraction for German Texts*. Graz University of Technology (cit. on pp. 3, 12, 20, 81).
- Bast, Hannah and Elmar Haussmann (2013). “Open Information Extraction via Contextual Sentence Decomposition.” In: *2013 IEEE Seventh International Conference on Semantic Computing*. ICSC '13 (cit. on pp. 12, 19).
- Bender, Emily M. (2011). “On Achieving and Evaluating Language - Independence in NLP.” In: *Linguistic Issues in Language Technology* (cit. on p. 2).
- Bengio, Samy, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer (2015). “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks.” In: *Computing Research Repository CoRR abs/ 1506.03099* (cit. on p. 67).

Bibliography

- Bengio, Yoshua, Holger Schwenk, Jean-Sébastien Senècal, Frédéric Morin, Jean-Luc Gauvain, and Lakhmi C. Jain (2006). "Neural Probabilistic Language Models." In: *Innovations in Machine Learning: Theory and Applications* (cit. on p. 60).
- Biemann, Chris and Martin Riedl (2013). "Text: now in 2D! A Framework for Lexical Expansion with Contextual Similarity." In: *Journal of Language Modelling* 1.1 (cit. on pp. 21, 55, 66).
- Bohnet, Bernd and Joakim Nivre (2012). "A Transition-Based System for Joint Part-of-Speech Tagging and Labeled Non-Projective Dependency Parsing." In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. EMNLP-CoNLL '12 (cit. on pp. 14, 15, 17, 21, 54, 66).
- Buchanan, Bruce G. and Edward H. Shortliffe (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project* (cit. on p. 1).
- Buchholz, Sabine and Erwin Marsi (2006). "CoNLL-X Shared Task on Multilingual Dependency Parsing." In: *Proceedings of the Tenth Conference on Computational Natural Language Learning*. CoNLL '06 (cit. on pp. 15, 60, 63).
- Chen, Danqi and Christopher D. Manning (2014). "A Fast and Accurate Dependency Parser using Neural Networks." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. EMNLP '14 (cit. on pp. 15, 16, 19, 23, 27).
- Cho, KyungHyun, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio (2014). "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches." In: *Computing Research Repository CoRR* abs/1409.1259 (cit. on p. 63).
- Cho, Kyunghyun, Bart van Merriënboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." In: *Computing Research Repository CoRR* abs/1406.1078 (cit. on p. 64).
- Chrupala, Grzegorz, Georgiana Dinu, and Josef van Genabith (2008). "Learning Morphology with Morfette." In: *Proceedings of Language Resources and Evaluation Conference*. LREC '08 (cit. on p. 14).
- Das, Dipanjan and Slav Petrov (2011). "Unsupervised Part-of-speech Tagging with Bilingual Graph-based Projections." In: *Proceedings of the 49th*

Bibliography

- Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. HLT '11 (cit. on pp. 14, 15).
- De Wilde, Max (2016). "From Information Extraction to Knowledge Discovery: Semantic Enrichment of Multilingual Content with Linked Open Data." PhD thesis. Free University of Brussels (cit. on p. 4).
- Del Corro, Luciano and Rainer Gemulla (2013). "ClausIE: Clause-based Open Information Extraction." In: *Proceedings of the 22Nd International Conference on World Wide Web*. WWW '13 (cit. on pp. 11, 12, 17).
- Embley, David W., Stephen W. Liddle, Deryle W. Lonsdale, Byung-Joo Shin, and Yuri Tijerino (2014). "Multilingual Extraction Ontologies." In: *Towards the Multilingual Semantic Web: Principles, Methods and Applications*. Ed. by Paul Buitelaar and Philipp Cimiano (cit. on pp. 6, 8–10).
- Embley, David W., Stephen W. Liddle, Deryle W. Lonsdale, and Yuri Tijerino (2011). "Multilingual Ontologies for Cross-Language Information Extraction and Semantic Search." In: *Conceptual Modeling – ER 2011* (cit. on p. 6).
- Fader, Anthony, Stephen Soderland, and Oren Etzioni (2011). "Identifying Relations for Open Information Extraction." In: *Proceedings of the Conference of Empirical Methods in Natural Language Processing*. EMNLP '11 (cit. on pp. 12, 17, 22).
- Falaise, Achille, David Rouquet, Didier Schwab, Hervé Blanchon, and Christian Boitet (2010). "Ontology Driven Content Extraction Using Interlingual Annotation of Texts in the OMNIA Project." In: *Proceedings of the 4th International Workshop on Cross Lingual Information Access* (cit. on p. 5).
- Falke, Tobias, Gabriel Stanovsky, Iryna Gurevych, and Ido Dagan (2016). "Porting an Open Information Extraction System from English to German." In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. EMNLP '16 (cit. on pp. 2, 5, 12, 17, 20, 21, 25, 28, 53, 54, 67, 72, 74).
- Faruqui, Manaal and Shankar Kumar (2015). "Multilingual Open Relation Extraction Using Cross-lingual Projection." In: *Computing Research Repository CoRR* abs/1503.06450 (cit. on pp. 5, 12).
- Frege, Gottlob (1879). *Begriffsschrift - Eine der arithmetischen nachgebildete Formelsprache des reinen Denkens* (cit. on p. 19).
- Gamallo, Pablo and Marcos Garcia (2015). "Multilingual Open Information Extraction." In: *Proceedings of the 17th Portuguese Conference on Artificial Intelligence*. EPIA '15 (cit. on pp. 4, 11, 12, 17, 21).

Bibliography

- Gamallo, Pablo and Isaac González (2012). "DepPattern: A Multilingual Dependency Parser." In: *Demo Session of the International Conference on Computational Processing of the Portuguese Language*. PROPOR 2012 (cit. on pp. 14, 15, 17, 21).
- Hahnloser, Richard H. R., Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung (2000). "Digital Selection and Analogue Amplification Coexist in a Cortex-inspired Silicon Circuit." In: *Nature* 405 (cit. on p. 63).
- Heist, Nicolas and Heiko Paulheim (2016). *Language-agnostic Relation Extraction from Wikipedia Abstracts for Knowledge Graph Extension* (cit. on p. 7).
- Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2012). "Improving Neural Networks by Preventing Co-adaptation of Feature Detectors." In: *Computing Research Repository CoRR abs/ 1207.0580* (cit. on p. 66).
- Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory." In: *Neural Computation* 9.8 (cit. on p. 62).
- Jean, Sébastien, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio (2014). "On Using Very Large Target Vocabulary for Neural Machine Translation." In: *Computing Research Repository CoRR abs/ 1412.2007* (cit. on p. 66).
- Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization." In: *Computing Research Repository CoRR abs/ 1412.6980* (cit. on p. 66).
- Lee, Jennifer, Hans Christian Andersen, Chris Buck, and Shane Morris (2013). *Frozen*. Movie. Directors: Buck, Chris and Lee, Jennifer. Walt Disney (cit. on pp. 15, 17, 28).
- Liu, Yang, Furu Wei, Sujian Li, Heng Ji, Ming Zhou, and Houfeng Wang (2015). "A Dependency-Based Neural Network for Relation Classification." In: *Computing Research Repository CoRR abs/ 1507.04646* (cit. on p. 10).
- Lopyrev, Konstantin (2015). "Generating News Headlines with Recurrent Neural Networks." In: *Computing Research Repository CoRR abs/ 1512.01712* (cit. on p. 63).
- Mausam, Michael Schmitz, Robert Bart, Stephen Soderland, and Oren Etzioni (2012). "Open Language Learning for Information Extraction." In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Nat-*

Bibliography

- ural Language Processing and Computational Natural Language Learning*. EMNLP-CoNLL '12 (cit. on pp. 6, 12, 17, 23).
- Mirończuk, M., D. Czerski, M. Sydow, and M. A. Kłopotek (2013). "Language-independent Information Extraction based on Formal Concept Analysis." In: *Second International Conference on Informatics Applications*. ICIA '13 (cit. on p. 7).
- Nivre, Joakim, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryiğit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi (2007). "MaltParser: A Language-independent System for Data-driven Dependency Parsing." In: *Natural Language Engineering* 13.2 (cit. on p. 17).
- Nivre, Joakim et al. (2016). "Universal Dependencies v1: A Multilingual Treebank Collection." In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation*. LREC '16 (cit. on pp. 15–17, 27).
- Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu (2002). "BLEU: A Method for Automatic Evaluation of Machine Translation." In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. ACL '02 (cit. on p. 75).
- Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "GloVe: Global Vectors for Word Representation." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. EMNLP '14 (cit. on p. 60).
- Petrov, Slav, Dipanjan Das, and Ryan T. McDonald (2011). "A Universal Part-of-Speech Tagset." In: *Computing Research Repository CoRR* abs/1104.2086 (cit. on pp. 13, 15).
- Petrov, Slav and Dan Klein (2007). "Improved Inference for Unlexicalized Parsing." In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. HLT-NAACL '07 (cit. on p. 27).
- Piskorski, Jakub and Roman Yangarber (2013). "Information Extraction: Past, Present and Future." In: *Multi-source, Multilingual Information Extraction and Summarization* (cit. on p. 8).
- Remus, Robert, Uwe Quasthoff, and Gerhard Heyer (2010). "SentiWS - a Publicly Available German-language Resource for Sentiment Analysis." In: *Proceedings of the 7th International Language Resources and Evaluation*. LREC '10 (cit. on pp. 60, 66).
- Rosenblatt, Frank (1958). *The perceptron: A probabilistic model for information storage and organization in the brain*. (Cit. on p. 62).

Bibliography

- Schmid, Helmut (1995). "Improvements in Part-of-Speech Tagging with an Application to German." In: *Proceedings of the ACL SIGDAT-Workshop* (cit. on p. 14).
- Schuster, M. and K. K. Paliwal (1997). "Bidirectional Recurrent Neural Networks." In: *IEEE Transactions on Signal Processing* 45.11 (cit. on p. 65).
- Sleator, Daniel Dominic and David Temperley (1995). "Parsing English with a Link Grammar." In: *Computing Research Repository CoRR abs/cmp-lg/9508004* (cit. on p. 18).
- Stanovsky, Gabriel, Jessica Fidler, Ido Dagan, and Yoav Goldberg (2016). "Getting More Out Of Syntax with PropS." In: *Computing Research Repository CoRR abs/1603.01648* (cit. on pp. 2, 5, 11, 12, 21, 25, 32, 53, 72).
- Suchanek, Fabian M., Mauro Sozio, and Gerhard Weikum (2009). "SOFIE: A Self-organizing Framework for Information Extraction." In: *Proceedings of the 18th International Conference on World Wide Web. WWW '09* (cit. on p. 9).
- Toutanova, Kristina, Dan Klein, Christopher D. Manning, and Yoram Singer (2003). "Feature-rich Part-of-speech Tagging with a Cyclic Dependency Network." In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1. NAACL '03* (cit. on p. 14).
- Verga, Patrick, David Belanger, Emma Strubell, Benjamin Roth, and Andrew McCallum (2015). "Multilingual Relation Extraction using Compositional Universal Schema." In: *Computing Research Repository CoRR abs/1511.06396* (cit. on pp. 5, 12).
- Wang, Mingyin, Lei Li, and Fang Huang (2014). "Semi-supervised Chinese Open Entity Relation Extraction." In: *2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems. CCIS '14* (cit. on pp. 12, 16, 19, 23).
- White, Aaron Steven, Drew Reisinger, Keisuke Sakaguchi, Tim Vieira, Sheng Zhang, Rachel Rudinger, Kyle Rawlins, and Benjamin Van Durme (2016). "Universal Decompositional Semantics on Universal Dependencies." In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. EMNLP '16* (cit. on p. 22).
- Wu, Fei and Daniel S. Weld (2010). "Open Information Extraction Using Wikipedia." In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. ACL '10* (cit. on pp. 12, 17, 22).

Bibliography

- Xiao, Minguang and Cong Liu (2016). "Semantic Relation Classification via Hierarchical Recurrent Neural Network with Attention." In: *Proceedings of the 26th International Conference on Computational Linguistics: Technical Papers*. COLING '16 (cit. on p. 10).
- Xu, Kun, Yansong Feng, Songfang Huang, and Dongyan Zhao (2015). "Semantic Relation Classification via Convolutional Neural Networks with Simple Negative Sampling." In: *Computing Research Repository CoRR abs/1506.07650* (cit. on p. 10).
- Zeng, Daojian, Kang Liu, Yubo Chen, and Jun Zhao (2015). "Distant Supervision for Relation Extraction via Piecewise Convolutional Neural Networks." In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. EMNLP '15 (cit. on p. 10).
- Zhang, Dongxu and Dong Wang (2015). "Relation Classification via Recurrent Neural Network." In: *Computing Research Repository CoRR abs/1508.01006* (cit. on p. 10).
- Zhang, Sheng, Kevin Duh, and Benjamin Van Durme (2017). "MT/IE: Cross-lingual Open Information Extraction with Neural Sequence-to-Sequence Models." In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. EACL '17 (cit. on pp. 6, 12, 22, 64, 75).
- Zhila, Alisa and Alexander Gelbukh (2014). "Open Information Extraction for Spanish Language based on Syntactic Constraints." In: *Proceedings of the Association for Computational Linguistics 2014 Student Research Workshop*. ACL '14 (cit. on pp. 11, 12, 21).