Projekt nr POIG.01.01.02-14-013/09

Alina Wróblewska

Polish Dependency Parser Trained on an Automatically Induced Dependency Bank

INSTYTUT PODSTAW INFORMATYKI
POLSKIEJ AKADEMII NAUK

Alina Wróblewska

Polish Dependency Parser Trained on an Automatically Induced Dependency Bank

ROZPRAWA DOKTORSKA

INSTITUTE OF COMPUTER SCIENCE
POLISH ACADEMY OF SCIENCES

Alina Wróblewska

Polish Dependency Parser Trained on an Automatically Induced Dependency Bank

PhD Dissertation

Supervisor
dr hab. Adam Przepiórkowski, prof. IPI PAN

Warsaw 2014

The research presented in this dissertation was supported by grant no POIG.01.01.02-14-013/09 from Innovative Economy Operational Programme co-financed by the European Union (European Regional Development Fund).

Cover design by Joanna Tokarczyk

Contents

Streszczenie

W ostatnich latach coraz większą wagę przywiązuje się do parsowania zależnościowego, czyli do automatycznej analizy składniowo-semantycznej zdań. Dzieje się tak dlatego, że parsowanie wydobywa strukturę predykatywno-argumentową zdania, której można użyć do udoskonalenia systemów dialogowych, tłumaczenia maszynowego, czy ekstrakcji informacji. Większość współczesnych systemów parsowania zależnościowego opiera się na metodach statystycznych. Na podstawie danych treningowych parsery uczą się, jak należy analizować zdania w języku naturalnym i generować odpowiednie struktury zależnościowe dla tych zdań. Jak dotychczas najlepsze wyniki osiągają parsery trenowane za pomocą metod z nadzorem. Parsery zależnościowe trenowane na poprawnie zaanotowanych danych są bardzo skuteczne, nawet w odniesieniu do języków ze swobodnym szykiem zdania, takich jak czeski czy bułgarski.

Niemniej jednak metody z nadzorem wymagają dużej liczby poprawnie zaanotowanych struktur zależnościowych, które powstają w wyniku bardzo czasochłonnego i kosztownego procesu anotacji ręcznej. Dla wielu języków nadal nie istnieją żadne banki struktur zależnościowych, dlatego poszukuje się alternatywnych metod trenowania parserów albo pozyskiwania danych treningowych. Ponieważ uczenie bez nadzoru często nie jest najlepszym rozwiązaniem głównie za sprawą małej efektywności oraz bardzo dużej złożoności obliczeniowej, w niniejszej rozprawie doktorskiej rozpatrujemy alternatywne metody pozyskiwania wysokiej jakości struktur zależnościowych oraz szukamy odpowiedzi na następujące pytania badawcze:

1. Czy jest możliwe automatyczne (lub półautomatyczne) pozyskiwanie drzew zależnościowych?

2. Czy można przy pomocy metod z nadzorem wytrenować parser zależnościowy na automatycznie lub półautomatycznie pozyskanych danych?

Dysertacja rozpoczyna się teoretycznym opisem głównych założeń gramatyki zależnościowej oraz parsowania zależnościowego (rozdział drugi). W rozdziale trzecim został przedstawiony schemat anotacji zależnościowej zdań w języku polskim. Schemat ten definiuje zbiór reguł, przy pomocy których można wyznaczyć relacje dominacji (albo zależności) pomiędzy tokenami w zdaniu. Schemat jest dostosowany do specyfiki języka polskiego i bierze pod uwagę główne zjawiska lingwistyczne opisane w literaturze i występujące w losowo wybranych zdaniach. Schemat wyróżnia 28 typów relacji zależnościowych podzielonych na trzy grupy: relacje zawierające podrzędniki pełniące funkcje argumentów, relacje zawierające podrzędniki niepełniące funkcji argumentów oraz relacje przeznaczone do anotowania konstrukcji z koordynacją. Zgodnie z tym schematem anotujemy automatycznie wygenerowane struktury zależnościowe zdań w języku polskim.

W dysertacji zostały zaprezentowane dwie metody automatycznego pozyskiwania struktur zależnościowych. Pierwsza metoda wykorzystuje ideę konwersji drzew składnikowych do postaci drzew zależnościowych (rozdział czwarty). Wykorzystanie metody konwersji jest możliwe, ponieważ dla języka polskiego istnieje bank struktur składnikowych. W związku z tym, że relacje zależnościowe można stosunkowo łatwo wywieść ze struktur składnikowych z wyróżnionymi elementami głównymi, nacisk jest położony przede wszystkim na dostosowanie przekonwertowanych struktur do schematu anotacji drzew zależnościowych oraz na przypisanie etykiet do krawędzi w przekonwertowanych drzewach. W celu dostosowania struktur do schematu anotacji opracowano zbiór reguł modyfikujących relacje pomiędzy tokenami w konstrukcjach strony biernej oraz w konstrukcjach zawierających frazy nieciągłe, zdania podrzędne, frazy z korelatem, czy spójniki inkorporacyjne. Ponieważ współczesne systemy parsowania zależnościowego są dostosowane do uczenia modeli na drzewach, których krawędzie mają przypisane etykiety, istotne znaczenie miało opracowanie zbioru reguł etykietujących krawędzie w przekonwertowanych drzewach funkcjami gramatycznymi podrzędników danych relacji. Ostatecznym wynikiem procesu konwersji jest bank 8227 drzew zależnościowych z etykietami przypisanymi do krawędzi. W celu oceny jakości pozyskanych drzew zależnościowych wykorzystano zewnętrzną metodę ewaluacji (ang. 'extrinsic evaluation'). Metoda ta polega na wytrenowaniu parsera zależnościowego na przekonwertowanych drzewach, a następnie na ocenie wpływu danych treningowych na jakość parsowania. Zgodnie z wynikami, w stosunkowo prostych polskich zdaniach nawet 92,7% tokenów może mieć przypisany poprawny nadrzędnik, a 87,2% tokenów może mieć przypisany poprawny nadrzędnik oraz poprawną funkcję gramatyczną etykietującą relację. W przypadku bardziej skomplikowanych i rozbudowanych zdań wyniki te są zdecydowanie niższe – 76,6% tokenów ma przypisany poprawny nadrzędnik, a 70,1% tokenów ma przypisany poprawny nadrzędnik oraz etykietę relacji.

Drugi sposób automatycznego pozyskiwania drzew zależnościowych jest oparty na nowatorskiej metodzie rzutowania ważonego (rozdział piąty). Główna idea metody rzutowania informacji lingwistycznych polega na odwzorowaniu anotacji lingwistycznych w

zdaniach z części korpusu równoległego w jednym języku na odpowiednie zdania z części korpusu w drugim języku. Informacje lingwistyczne są rzutowane z wykorzystaniem automatycznie wygenerowanych przyporządkowań słownych (ang. 'word alignment'). W przedstawionej w rozprawie i opartej na idei rzutowania procedurze pozyskiwania struktur zależnościowych dla zdań w języku polskim można wyróżnić dwa główne kroki: rzutowanie ważone angielskich relacji zależnościowych na zdanie polskie oraz indukcję ważoną drzew zależnościowych na podstawie zbioru rzutowanych krawędzi. Angielskie relacje zależnościowe są rzutowane na odpowiednie zdania polskie poprzez rozbudowany zbiór przyporządkowań słownych z przypisanymi wagami. W wyniku tego zdaniom polskim zostają przypisane grafy skierowane z wagami na krawędziach. Wagi krawędzi są szacowane na podstawie wag przyporządkowań słownych wykorzystanych w rzutowaniu. Indukcja ważona polega na szukaniu – w grafach skierowanych zawierających rzutowane krawędzie ze zoptymalizowanymi wagami – maksymalnych drzew rozpinających, które spełniają kryteria poprawnego drzewa zależnościowego. Do optymalizacji wag wykorzystano rozkład prawdopodobieństwa krawędzi w k najlepszych drzewach rozpinających znalezionych w rzutowanym grafie skierowanym. Rozkład prawdopodobieństwa krawędzi można obliczyć za pomocą zmodyfikowanej wersji algorytmu EM. Nowatorstwo przedstawionej metody polega na włączeniu czynnika ważenia do procesów rzutowania relacji oraz indukcji struktur zależnościowych. W rzutowanych grafach skierowanych ze zoptymalizowanymi wagami na krawędziach znaleziono prawie 4 miliony maksymalnych drzew rozpinających spełniających kryteria poprawnego drzewa zależnościowego. Następnie krawędziom w drzewach pozyskanych z wykorzystaniem metody rzutowania ważonego zostają przypisane etykiety. Parser wytrenowany na takich drzewach znajduje poprawne nadrzędniki dla 74,6% tokenów, a poprawne nadrzędniki wraz z poprawną etykietą relacji dla 69,4% tokenów. W następstwie zastosowania dodatkowych reguł korygujących oraz filtrujących w odniesieniu do wyindukowanych drzew, parser wytrenowany na ulepszonym zbiorze drzew przypisuje poprawne nadrzędniki do 86,0% tokenów, a poprawne nadrzędniki i poprawne etykiety relacji do 80,5% tokenów. Mimo że te wyniki są istotnie niższe niż wyniki osiągnięte przez parser trenowany na przekonwertowanych drzewach, ewaluacja w oparciu o dłuższe i bardziej skomplikowane struktury pokazuje, że parser trenowany na drzewach pozyskanych metodą ważonej indukcji działa nieznacznie lepiej niż parser trenowany na przekonwertowanych drzewach.

Na podstawie wyników przeprowadzonych eksperymentów możemy udzielić pozytywnych odpowiedzi na zadane pytania badawcze, ponieważ udało nam się pozyskać struktury zależnościowe w sposób automatyczny, wykorzystując metody oparte na konwersji i indukcji, a także wytrenować parser na automatycznie wygenerowanych strukturach zależnościowych.

Acknowledgements

First and foremost, I would like to thank my scientific adviser, Adam Przepiórkowski, for his support, ideas that greatly contributed to this dissertation, time and words of encouragement towards my work.

The value of working within the Linguistic Engineering Group at the Institute of Computer Science of the Polish Academy of Sciences cannot be underestimated. I would like to thank all my colleagues from the NLP team, especially Marcin Woliński, Michał Lenart, Kasia Krasnowska and Łukasz Dębowski.

My first steps towards academia were taken under the guidance of Anette Frank who supervised my Master's thesis at the Heidelberg University. She sparked my interest in natural language processing and without her this dissertation would likely never have been written.

As funding is necessary when one wants to do research, I am grateful for the opportunity of conducting my research within the Innovative Economy Operational Programme co-financed by the European Union.

Above all I want to thank my family, my parents and sisters who always show unfailing belief in me, my husband Radek for staying by my side all the time and our daughter Waleria for her patience with me and my long working hours. It would not have been possible to face and complete such a challenge without them.

Chapter 1

Introduction

Dependency parsing has become important for various language processing tasks in recent years. The predicate-argument structure transparently encoded in dependency-based syntactic representations may support machine translation, question answering, information extraction, etc. Many contemporary dependency parsing systems are based on statistical methods. Using training data, parsers learn how to analyse sentences and predict dependency structures that are appropriate for these sentences.

Different statistical methods have been applied to data-driven dependency parsing. However, the best results so far are given by supervised methods. Supervised dependency parsers trained on correctly annotated data may have high parsing performance even for languages with relatively free word order, such as Czech or Bulgarian.

Nevertheless, supervised methods require manually annotated training data. The creation of such data is a very time-consuming and expensive process. Therefore, there is still a lot of languages without any manually annotated data and alternative methods of parser training or data gathering are needed. Since unsupervised training – with its low performance and high complexity – is often an infeasible solution, we study alternative methods of gathering training data. In this dissertation we address two research questions:

1. Is it possible to gather dependency trees automatically (or with a minimal human involvement)?
2. Is it possible to train a good quality supervised dependency parser on automatically or semi-automatically induced training data?

Due to the increasing interest in data-driven parsing, several shared tasks on multilingual dependency parsing were organised at the Conference on Computational Natural Language Learning (Buchholz and Marsi, 2006; Nivre et al., 2007). Different languages were

represented in these tasks, including some Slavic languages such as Slovene, Bulgarian and Czech. Polish was not represented in any of these tasks, probably due to the lack of dependency-annotated training data for this language. Furthermore, dependency parsing is hardly represented in the Polish NLP community. We are aware of neither experiments with data-driven Polish dependency parsing nor existence of any publicly available Polish dependency parser. The only Polish dependency parser was developed by Tomasz Obrębski within his doctoral research (Obrębski, 2002, 2003). However, this rule-based parser founded on the syntactic description of Polish by Saloni and Świdziński (1989) was only tested against a small artificial test set and no wide-coverage grammar seems to accompany the work. Among many aspects of Obrębski's work, a particularly interesting element is a definition of Polish relation types.

The shared tasks mentioned above prompted the creation of many high quality dependency parsing systems. Even if there are some sophisticated systems that could be used to train dependency parsers for Polish, the lack of high quality training data is a major bottleneck in the development of such parsers. To overcome this problem we address various methods of data gathering in this dissertation.

Two ways of inducing dependency structures automatically are investigated here: constituency-to-dependency conversion and cross-lingual projection of dependency information. The conversion method has been successfully applied for languages such as English, German or Bulgarian. This method presupposes that a constituency treebank for a particular language is available. Since there is a publicly available Polish constituency treebank, the conversion technique may be adapted to Polish.

The second method builds on the assumption that a linguistic analysis of a sentence largely carries over to its translation in an aligned parallel corpus. Projected annotations can then be used to train natural language processing tools for the target language. The cross-lingual projection method has been successfully applied to various levels of linguistic analysis and corresponding natural language processing tasks, such as part-of-speech tagging or semantic role labelling, as well as dependency annotation and parser induction.

1.1 Contributions

The doctoral research outlined in this dissertation contributes to the development of various aspects related to dependency parsing. First, a dependency annotation schema is designed to cover primary syntactic phenomena in Polish. Second, adaptation of a constituency-to-dependency conversion method contributes to the construction of a Polish dependency treebank annotated in accordance with the dependency annotation schema. Third, we propose a weighted induction method designed to acquire dependency

structures for Polish and possibly for other resource-poor languages. The weighted induction method is the principle scientific result of our doctoral research. Fourth, we conduct some experiments that consist in training and evaluation of dependency parsers on induced treebanks. In these experiments, we use publicly available dependency parsing systems. Fifth, we make results of our research, i.e., induced dependency treebanks and trained parsing models, publicly available in order to contribute to the further development of Polish dependency parsing and to serve as the basis for other NLP tasks.

1.2 Organisation of the Dissertation

This dissertation is organised into six chapters. **Chapter 2** gives some basic ideas of dependency theory in general (Section 2.1 *Pillars of the Dependency Theory*) and in Poland (Section 2.2 *Dependency in Poland*). Furthermore, it introduces some basic notions related to dependency parsing. Since the concept of a dependency structure originating from dependency frameworks plays a key part in this thesis, it is described in detail in Section 2.3 *Dependency Structure*. Section 2.4 *Data-driven Dependency Parsing* outlines supervised statistical machine learning methods for prediction of dependency structures.

Chapter 3 proposes a dependency annotation schema designed for the purpose of annotation of Polish sentences. Section 3.1 *Foundations* describes basic principles and properties of the annotation schema. Individual dependency relation types are presented in Section 3.2 *Polish Dependency Relation Types*. Section 3.3 *Syntactic Annotation Schemata: Related Work* provides an overview of the existing syntactic annotation schemata for various languages.

Chapter 4 describes adaptation of the constituency-to-dependency conversion method for acquiring a treebank of valid Polish dependency structures. Section 4.1 *Składnica – Polish Constituency Treebank* introduces the Polish constituency treebank *Składnica*, which is the source of constituency-to-dependency converted trees. The procedure of converting constituent trees into labelled dependency structures is described in the following sections. Section 4.2 *Conversion Procedure* describes main aspects of the conversion procedure and the technique of annotating relations with grammatical functions. Section 4.3 *Head Selection* provides an overview of some head selection heuristics. Section 4.4 *Rearrangement of Dependency Structures* presents some reorganisations of converted dependency structures. Empirical experiments that consist in training Polish dependency parsers on converted dependency structures and a detailed evaluation of the trained parsers are reviewed in Sections 4.5 *Experimental Setup* and 4.6 *Experiments and Results*. This chapter ends with an overview of related constituency-to-dependency approaches (Section 4.7 *Constituency-to-Dependency Conversion: Related Work*) and some conclusions (Section 4.8 *Partial Conclusions*).

Chapter 5 proposes a different way of obtaining dependency structures based on a novel weighted induction method. The procedure of inducing dependency structures consists of two successive steps which are described in the following sections. Section 5.1 *Weighted Projection* presents the procedure of projecting English dependency relations to corresponding Polish sentences via an extended set of weighted alignment links. Section 5.2 *Weighted Induction*, in turn, describes the method of extracting unlabelled dependency structures from projected directed graphs. Section 5.3 *Rule-based Adaptation of Polish Dependency Structures* gives an overview of rule-based heuristics designed to label and correct induced dependency structures. The empirical part of this chapter (Sections 5.4 *Experimental Setup* and 5.5 *Experiments and Results*) outlines some experiments conducted to train dependency parsers on induced dependency structures. Since our method is set within the mainstream of the cross-lingual information projection study, Section 5.6 *Annotation Projection: Related Work* provides an overview of the related annotation projection approaches. The chapter ends with some conclusions (Section 5.7 *Partial Conclusions*).

Chapter 6 concludes the dissertation. It summarises (Section 6.1 *Summary*) and compares two ways of acquiring dependency structures (Section 6.2 *Comparison of Conversion-based and Projection-based Approaches*).

Chapter 2

Preliminaries

2.1 Pillars of the Dependency Theory

The tradition of the modern dependency grammar theory[1] derives from the pioneering work *Éléments de syntaxe structurale* by Lucien Tesnière (1959), which is the first comprehensive work on the dependency formalism. The central notions of Tesnière's theory of structural syntax are *connexion* and *valency* (cf. Ágel and Fischer, 2010). Connexions, which are nowadays called dependencies or dependency relations, are, next to *jonctions* and *translations*, basic relations of the structural syntax. They connect two word forms co-occurring in a sentence and represent government (or dependency) relations in a dependency tree (Tesnière's *stemma*) of this sentence. One of the two connected word forms is called *regent* (also *governor*, Tesniére's *terme supérieur*) and the other one is called *dependent* (Tesniére's *terme inférieur*). All regents and dependents are attributed *grammatical functions*.

The valency theory as defined by Tesnière assumes that the centre of a sentence is constituted by a verb (principle of *verb centrality*).[2] The verb requires some complements (Tesnière's *actants*) and may admit some adjuncts (Tesnière's *circumstantials*). However, a central verb (or a verbal nucleus) and its circumstantial may only build a dependency relation. A verbal nucleus and its actant, in turn, build both a dependency relation and a valency relation since the actant is anchored in the verb's meaning. Actants are distinguished on the basis of their semantic relations with the verbal nucleus.

[1] The dependency grammar has a very long tradition which may even go back to Pānini, an Indian grammarian living in the 4th or 5th century BC, who defined a formal grammar of Sanskrit. A theory of grammar was first formalised for the Arabic language by Kitāb al-Uṣūl of Ibn al-Sarrāǧ. This theory builds on dependency-based notions of a syntactic head and its dependent which are similar to the main concepts of the modern dependency grammar. The first European medieval grammarians who used the notion of dependency were Martin of Dacia living in 13th century and Thomas von Erfurt living in 14th century (cf. Kruijff, 2002).

[2] The principle of *verb centrality* assumes that the structure of a sentence unfolds from the verb and grammatical functions of the verb dependents are imposed by the verb.

The notions of *connexion* and *valency* have been adapted by theoreticians of the dependency grammar. Tesnière's successors developed his concept into different theoretical frameworks, e.g., *Meaning-Text Theory* by Igor Mel'čuk (Mel'čuk, 1988), *Functional Generative Description* defined by Prague School theoreticians (Sgall et al., 1986), *Word Grammar* by Richard Hudson (Hudson, 1990) or *Dependency Unification Grammar* (Hellwig, 1986, 2003). All of these frameworks are based on the common idea that the description of structures commences from individual word forms which have the potential for connecting and valency. A definition of a syntactic structure (after Polguére and Mel'čuk, 2009, p. xv) is formulated as follows:

Definition 2.1. The syntactic structure of a sentence is a set of lexical units of this sentence linked together by syntactic relations.

This definition indicates that a dependency structure of a sentence is understood as a set of lexical items making up the sentence. These lexical items are related among themselves by syntactic dependencies. While the dependency structure represents a syntactic description of a sentence, it is also semantically motivated. Since dependency relations may be justified by valency relations (similarly as in Tesnière, 1959), the dependency structure also encodes the semantic predicate-argument structure. Hence, the dependency grammar is a general syntactic theory with an integrated valency component that determines both the number and the kind of complement slots of a verb, a noun or an adjective (cf. Ágel and Fischer, 2010).

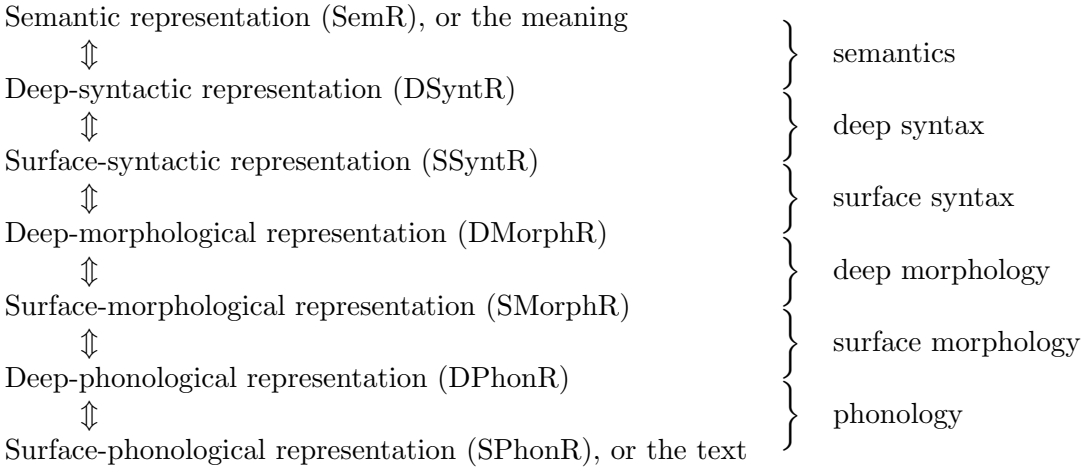


FIGURE 2.1: Levels of the utterance representation and modules of *Meaning-Text Theory* (taken from Kahane, 2003).

Moreover, some dependency frameworks (e.g., *Meaning-Text Theory* and *Functional Generative Description*[3]) argue that the true dependency analysis consists of multiple

[3]In terms of *Functional Generative Description*, there are five levels of the utterance representation: phonetic, morphonological, morphological, surface-syntactic and tectogrammatic.

dependency representations such as morphological, syntactic, or semantic. For example, *Meaning-Text Theory* assumes that syntactic and morphological representations are intermediate levels between the semantic level (corresponding to *Meaning*, i.e., semantic representation) and the phonetic level (corresponding to *Text*, i.e., phonetic representation). In terms of *Meaning-Text Theory*, a natural language may be understood as many-to-many correspondence between a set of possible meanings and a set of possible texts. The meaning-text correspondence, in turn, is defined with the *Meaning-Text Model* which consists of a finite set of rules. The *Meaning-Text Model* is divided into six modules ensuring correspondence between adjacent representation levels as depicted in Figure 2.1 taken from Kahane (2003). Hence, the morphological representation and the syntactic representation are intermediate levels between the semantic representation and the phonetic representation of an utterance.

In addition to 'pure' dependency formalisms, most of post-generative formalisms such as *Lexical Functional Grammar* (LFG, Bresnan, 2001; Dalrymple, 2001; Falk, 2001), *Head-driven Phrase Structure Grammar* (HPSG, Pollard and Sag, 1994) or *Tree-Adjoining Grammar* (TAG, Joshi and Schabes, 1997; Abeillé and Rambow, 2000) also emphasise dependency relations in their syntactic representations. However, these representations are substantially different from conventional dependency structures.

2.2 Dependency in Poland

Dependency theories which were developed after Tesnière gained recognition and followers primarily in Central and Eastern Europe, also in Poland. One of the first dependency descriptions of Polish was presented by Zenon Klemensiewicz (1968). Klemensiewicz distinguished between the main relation (Pol. 'związek główny') and the secondary relation (Pol. 'związek poboczny'). The main relation consists of the syntactically independent subject governing the sentence predicate.[4] All other relations of a sentence are secondary (cf. Klemensiewicz, 1968, p. 30). A secondary relation directly or indirectly depends on the governor (subject) or the dependent (predicate) of the main relation. The governor of the secondary relation is called a basis (Pol. 'podstawa'), while the dependent is called an attribute (Pol. 'określnik'). Klemensiewicz's grammar does not assume that a dependency analysis consists of multiple levels of linguistic description, but it merges various levels of language description.

A general overview of dependency types possible in Polish is given in an article by Marek Świdziński (1989). For word forms representing particular part of speech classes (e.g., verb, quasi-verb or noun), Świdziński defines parts of speech of their possible dependents and grammatical functions of these dependents (e.g., SUBJ, OBJ, PREC

[4] It is worth noting that Tesniére was exactly against such binary division of a sentence structure into a subject and a predicate. As an alternative, he introduced an approach based on the principle of *verb centrality* described in the previous section.

or PRED). He distinguishes 11 complement types and 8 adjunct types. Each possible Polish dependency type is illustrated by an exemplary sentence.

A dependency analysis of Polish numeral constructions is presented by Magdalena Derwojedowa (2011). The dependency description of numeral constructions is based on theoretical works by Igor Mel'čuk (e.g., Mel'čuk, 1988; Mel'čuk and Pertsov, 1987) and on the dissertation by Tomasz Obrębski (2002). Apart from an extensive description of dependency relations in numeral constructions, a general overview of Polish dependency types is also presented (Derwojedowa, 2011, Appendix A, pp. 175ff.).

Even if there are some theoretical works on the dependency description of Polish, there are hardly any approaches dealing with the issue of Polish dependency parsing. Nevertheless, the Polish NLP community is aware of the usefulness of syntactic parsing, as evidenced by numerous studies. The first formal description of Polish – *Metamorphosis Grammar* – was conceptualised and implemented by Stanisław Szpakowicz (1978, 1986). This formalism was further developed by Szpakowicz and Świdziński (1990, also Świdziński 1992) in the 1980s. This formal definition of Polish was used as a basis for implementing parsers described in Bień (1997), Bień et al. (2001) and Bień (2007), the *Świgra* parser (Woliński, 2004, 2005a,b) and a parser incorporated in the system POLINT (Vetulani, 2004). Apart from parsers employing Świdziński's formal description, there is also a wide-coverage constituent-based parser by Filip Graliński (2002, 2007) which is integrated with the machine translation system POLENG (Jassem, 2006). Deep syntactic parsers of Polish also employ other grammar formalisms, e.g., *Head-driven Phrase Structure Grammar* (Przepiórkowski et al., 2002) or *Lexical Functional Grammar* (Patejuk and Przepiórkowski, 2012). Apart from the above approaches involving deep syntactic parsing, there is also another trend in grammar-based Polish parsing – surface parsing. A surface parser identifies chunks in a sentence without setting relations between them, e.g., the *Spejd* parser (Przepiórkowski, 2008) or the *Puddle* parser (Graliński et al., 2012).

The only example of research on Polish dependency parsing is the pioneering PhD dissertation by Tomasz Obrębski (2002). In his work, Obrębski presents a grammar-based method of dependency parsing largely inspired by *Meaning-Text Theory*. The method is computationally efficient, but it is intended to analyse only a set of selected Polish sentences.[5] Based on the description of Polish morphosyntactic phenomena given by Saloni and Świdziński (1989), Obrębski proposes a dependency grammar formalism which is adjusted to the specificity of Polish. The formalism combines a language of the syntactic description with a mathematical model. The language of the syntactic description consists of elements describing word forms (e.g., part of speech tags, morphological features), agreement types, dependency types, dependency rules, separators, etc. The mathematical model, in turn, defines three representations of an utterance (a morphological structure, a punctuation structure, and a syntactic structure) and

[5]The parser is tested against 170 sentences from the test corpus described in Marciniak et al. (2000).

some axioms of the dependency syntax. The syntactic structure is represented as a dependency tree with vertices corresponding to tokens (vertices may be assigned flags encoding their syntactic attributes) and arcs which are equivalent to dependency relations with assigned dependency labels. The proposed grammar formalism defines a set of dependency relation types and a set of syntactic rules that determine possible or obligatory dependency relations between pairs of words. The parser based on this grammar takes possibly ambiguous morphological analyses of a sentence as input and outputs a dependency graph including alternative dependency analyses of this sentence.

2.3 Dependency Structure

Generally, a dependency structure consists of lexical items that occur in a sentence and are linked by relations called dependencies (see Figure 2.2).[6] A dependency relation is a special kind of a binary unilateral relation which is antisymmetric, antireflexive, antitransitive and labelled (cf. Mel'čuk, 1988, pp. 21ff.). One of the related lexical units is regarded as a *governor* (head, regent), while the other one is its *dependent*.

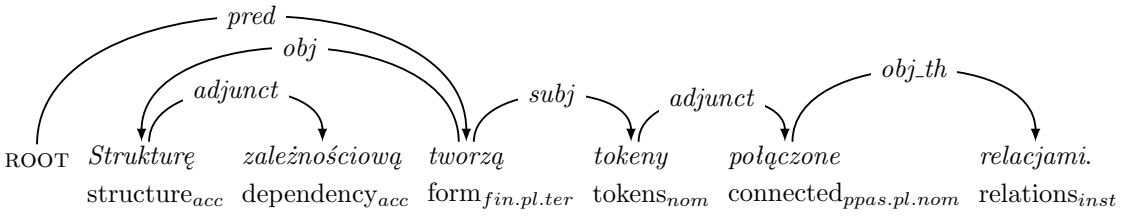


FIGURE 2.2: A dependency structure of the sentence *Strukturę zależnościową tworzą tokeny połączone relacjami.* (Eng. 'Tokens connected with relations build a dependency structure.').

The term *governor* traditionally refers to a word that selects grammatical features of another word (*governee*), e.g., a preposition determines the grammatical case of a complement noun phrase, or a noun determines the case and the gender of a modifying adjective. Since the description of syntactic relations established between pairs of words in a sentence is in the scope of the dependency grammar, the dependency theory adopts the term *governor*, but treats it broader than in other traditional syntactic theories. In the dependency grammar, the term *governor* is related to all dominant words that open slots for other words (dependents) with complementing or modifying meaning. However, the surface word form of a dependent does not have to be determined by the dominant word. For example, in the case of a relative clause modifying a noun phrase, the grammatical form of a relative pronoun introducing the relative clause is

[6] All tokens of a sentence, including punctuation marks, correspond to nodes of a dependency tree. However, punctuation marks, especially full stops and commas, are not displayed in most of graphic representations of dependency trees in this dissertation in order to preserve as much transparency as possible.

determined by the modified noun phrase (see *kobieta, która śpiewa* vs. *kobieta, *który śpiewa*, Eng., 'a woman who$_{sg.nom.f}$ sings' vs. 'a woman *who$_{sg.nom.m}$ sings'). Despite this, the relative pronoun depends on the predicate of the relative clause in the dependency representation and not on the modified noun phrase (see Figure 2.3).

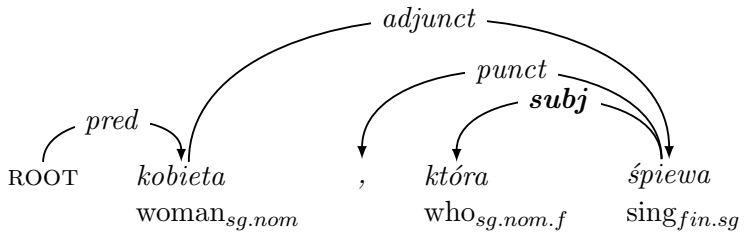


FIGURE 2.3: A dependency structure of the noun phrase *kobieta, która śpiewa* (Eng. 'a woman who sings').

The term *governor* may be used synonymously with the term *head* in dependency approaches. However, the term *head* may additionally refer to the element of a phrase determining morphosyntactic properties of the entire phrase in phrase structure approaches. From this perspective, the phrase structure head directly governs the entire phrase. In order to avoid misunderstandings in the current dissertation, we do not use these two terms interchangeably. Instead, the term *head* is used with regard to the central element of a phrase structure. The term *governor*, in turn, refers to the element dominating a dependency relation.

In a dependency structure, each lexical item depends on exactly one governing item, except for the top element of the dependency structure which is independent. A depending item may be characterised by some properties, e.g., its morphological form either agrees with the form of the governor (agreement) or is specified by the governor (government), its linear position is specified in relation to the governor position. A governor not only selects its dependents but also determines their obligatory or optional status, and their grammatical functions. A dependent, in turn, may semantically specify the governor.

The concept of a dependency structure has been adapted for the purpose of dependency parsing. The dependency structure may be defined as a labelled directed tree – the so-called dependency tree. Based on Kübler et al. (2009), Polguére and Mel'čuk (2009) and graph theory (e.g., Diestel, 2000; Newman, 2010), we define the dependency tree as follows:

Definition 2.2. A well-formed dependency tree $T = (V, A)$ of a sentence S consists of a set of vertices $V = \{v_0, v_1, ..., v_n\}$, where v_0 is the ROOT of T and for $i \in \{1, ..., n\}$, the vertex v_i corresponds to the ith token in the sentence $S = t_1, ..., t_n$, and a set of directed edges (arcs) $A \subseteq \{(v_i, v_j, l) | v_i, v_j \in V, l \in L\}$, for $L = \{l_1, ..., l_m\}$ being a finite set of possible grammatical functions with which arcs are labelled. The dependency tree T has the following properties:

- if $(v_i, v_j, l) \in A$ then $(v_i, v_j, l') \notin A$, for $v_i, v_j \in V$, $l, l' \in L$ and $l' \neq l$,
- if $(v_i, v_j, l) \in A$ then $(v_k, v_j, l') \notin A$, for $v_i, v_j, v_k \in V$, $v_i \neq v_k$ and $l, l' \in L$,
- $(v_i, v_0, l) \notin A$, for any $v_i \in V$ and $l \in L$,
- if $(v_0, v_i, l) \in A$ then $(v_0, v_j, l') \notin A$, for $v_i, v_j \in V$, $v_i \neq v_j$ and $l, l' \in L$,
- T contains a path from v_0 to v_i, for any $v_i \in V$.

The following terms and notions used in the above definition need to be clarified. The set V corresponds to lexical items (tokens) of the sentence $S = t_1, ..., t_n$ (i.e., the dependency tree spans over all tokens of the sentence S) which are extended with an additional artificial ROOT node not corresponding to any token of this sentence. All nodes in the dependency tree are assigned a unique index. The artificial ROOT node is always assigned the index 0 (i.e., v_0). All other nodes are assigned an index corresponding to the position of the token in the sentence. Nodes in the set V are annotated with word forms and possibly with morphosyntactic information (e.g., lemmata, part of speech tags or morphological features).

Arcs in the set A represent binary relations between governors and their dependents. The dependency tree contains only directed edges (arcs) representing asymmetric dependency relations between two tokens, one of which dominates the other one and allows for the appearance of the dependent token in the sentence. In order to reflect the hierarchy in the sentence, any arc (v_i, v_j, l), for $v_i, v_j \in V$ and $l \in L$, is directed from the governor v_i to the dependent node v_j. Arcs are explicitly labelled with dependency types from the set L (e.g., *subj*, *obj*) which are equivalent to grammatical functions borne by dependents and determined by governors. Therefore, we use the term *dependency label* interchangeably with the terms *grammatical function*, *syntactic function*, *dependency type* and *dependency relation type* in this dissertation. We are aware of the slight differences between these terms, but they seem to refer to the same entity, namely to the label of a dependency relation.

Multiple arcs between two lexical nodes are not possible in the dependency tree. If there is an arc $(v_i, v_j, l) \in A$ then no other arc that connects the same lexical nodes and is labelled with another grammatical function is part of the dependency tree, i.e., $(v_i, v_j, l') \notin A$, for $v_i, v_j \in V$, $l, l' \in l$ and $l \neq l'$. Furthermore, as each token may be governed by only one other token, any lexical node has at most one incoming arc, i.e., if $(v_i, v_j, l) \in A$ then $(v_k, v_j, l') \notin A$, for $v_i, v_j, v_k \in V$, $v_i \neq v_k$ and $l, l' \in L$.

The dependency tree originates from the ROOT node v_0, which simplifies both the computational implementation and the formal definition of the dependency structure. The ROOT node does not have any predecessor, i.e., $(v_i, v_0, l) \notin A$, for $v_i \in V$ and $l \in L$, and it has only one successor, i.e., if $(v_0, v_i, l) \in A$ then $(v_0, v_j, l') \notin A$, for $v_i, v_j \in V$, $v_i \neq v_j$ and $l, l' \in L$. The ROOT node directly or indirectly dominates all other nodes in

this tree following the direction of arcs, i.e., all nodes are directed away from the ROOT node. Since T contains a path from v_0 to v_i, for any $v_i \in V$, each v_i, for $i \neq 0$, has exactly one governor.

The dependency tree is therefore a *directed graph* (or *digraph*), as defined by Diestel (2000, p. 25),[7] which is connected, single-headed, rooted and acyclic, and spans over all lexical items of a sentence. Dependency parsing may be understood as a process of automatic allocation of a dependency tree to an input sentence.

2.4 Data-driven Dependency Parsing

The main idea of dependency parsing is to annotate an input sentence with an output dependency tree. Dependency parsing has recently become increasingly popular. There are various reasons for the increase in popularity of dependency parsing, especially data-driven dependency parsing. First, several shared tasks on multilingual dependency parsing have been organised in recent years. For example, such shared tasks were hosted at the Conference on Computational Natural Language Learning (CoNLL) in 2006 (Buchholz and Marsi, 2006), in 2007 (Nivre et al., 2007) and in 2009 (Hajič et al., 2009). These shared tasks resulted in the development of multiple dependency parsing systems that achieve the state-of-the-art parsing performance. An important advantage of parsing systems participating in these shared tasks is their multilingual dimension, i.e., they may be employed for training dependency parsing models for every language with an existing dependency treebank. These shared tasks also offered access to training data in a wide variety of languages. Second, efficient dependency-based parsing algorithms have been designed such that they can annotate sentences even in linear time (Eisner, 1996; Yamada and Matsumoto, 2003; Nivre, 2008). Third, dependency structures transparently encoding predicate-argument structures are probably better suited for further semantic processing than constituent trees. Consequently, dependency parsing may support different sophisticated language processing tasks such as machine translation, question answering or information extraction.

There are two main dependency parsing approaches: grammar-based and data-driven. Grammar-based dependency parsing relies on a formal grammar. A distinction is made between dependency parsing based on *context-free dependency grammars* (e.g., *Link Grammar*, cf. Sleator and Temperley 1993, *Bilexical Grammar*, cf. Eisner 1996, 2000) and

[7] Diestel (2000, p. 25) gives the following definition of a directed graph:

A *directed graph* (or *digraph*) is a pair (V, E) of disjoint sets (of *vertices* and *edges*) together with two maps init: $E \to V$ and ter: $E \to V$ assigning to every edge e an *initial vertex* $init(e)$ and a *terminal vertex* $ter(e)$. The edge e is said to be directed from $init(e)$ to $ter(e)$.

constraint-based dependency grammars (e.g., *Weighted Constraint Dependency Grammar*, cf. Harper and Helzerman 1995 and Menzel and Schröder 1998, *Probabilistic Constraint Dependency Grammar*, cf. Wang and Harper 2004, *Functional Dependency Grammar*, cf. Tapanainen and Järvinen 1997, Järvinen and Tapanainen 1998 and Obrębski 2002, *Topological Dependency Grammar*, cf. Duchier and Debusmann 2001, or *Extensible Dependency Grammar*, cf. Debusmann et al. 2004). Since the discussion of grammar-based dependency parsing is beyond the scope of this thesis, interested readers are referred to the referenced works or Kübler et al. (2009) for further information.

Data-driven dependency parsing relies on a parsing model trained on data using machine learning techniques. There are two problems to solve in the data-driven dependency parsing. The first one is a learning problem that aims to induce a parsing model given a training set of sentences (possibly annotated with dependency trees). The second one is a parsing problem that aims to predict an optimal dependency tree for an input sentence given the learnt parsing model.

We begin with the learning problem. A dependency parsing model is defined by Kübler et al. (2009, p. 18) as follows:

Definition 2.3. A *dependency parsing model* consists of a set of constraints Γ that define the space of permissible dependency structures for a given sentence, a set of parameters $\boldsymbol{\lambda}$ (possibly null), and a fixed parsing algorithm h. A model is denoted by $\boldsymbol{M} = (\Gamma, \boldsymbol{\lambda}, h)$.

Constraints Γ defining the space of permissible structures for a given sentence are specified by an underlying formalism used by the parsing system. They limit the space of dependency graphs by forcing the dependency model to produce dependency graphs which are well-formed in terms of these constraints (e.g., well-formed dependency trees). Γ may be represented as a set of simple constraints that restrict the space of dependency graphs to dependency trees, or as a set of more complex constraints, e.g., grammars that further limit the space of dependency graphs to particular dependency trees. Values of parameters in the set $\boldsymbol{\lambda}$ are learnt from training data. The parsing algorithm h, in turn, is a fixed function that searches over well-formed dependency graphs and returns a single tree for an input sentence.

The idea of data-driven dependency parsing is to train a parsing model that will predict a correct dependency tree for an input sentence. Parameters of dependency parsing models may be learnt either from annotated data (a training set of dependency structures) using supervised machine learning techniques or from large unannotated text corpora using unsupervised learning techniques.

Since the manual annotation of training data is a very time-consuming and expensive process, unsupervised techniques of training dependency models on unannotated textual data have been proposed. Some unsupervised dependency parsing approaches (e.g., Klein

and Manning, 2004; Spitkovsky et al., 2010) are based on different versions of the EM algorithm (Dempster et al., 1977). There are also approaches applying the Bayesian framework (e.g., Finkel et al., 2007), linear-interpolation smoothing (e.g., Headden et al., 2009), posterior regularisation (e.g., Gillenwater et al., 2010), unambiguity regularisation (e.g., Tu and Honavar, 2012), etc. Despite all efforts devoted to research on unsupervised dependency parsing, its performance is far behind the performance of supervised dependency parsers. Moreover, almost all unsupervised dependency parsing approaches currently focus on unlabelled dependency structures which may be insufficient for many sophisticated language processing applications. According to Tu (2012), unsupervised dependency parsing is restricted to unlabelled dependency structures because of very high complexity of training and parsing procedures if labels are added. Addition of dependency labels may enormously increase the number of parameters and the number of possible parses of a sentence.

Supervised methods are very well-established in data-driven dependency parsing and they give the best results so far. Supervised dependency parsers trained on correctly annotated data can achieve high parsing performance, even for relatively non-configurational Slavic languages which are characterised by multiple discontinuous constructions and a relatively free word order. For example, *MaltParser* trained on *Prague Dependency Treebank*[8] (Nivre et al., 2007) achieved 78.4% LAS[9] and 84.8% UAS[10] in the shared task at CoNLL 2006 and 77.2% LAS and 82.3% UAS in the shared task at CoNLL 2007. Further experiments show that the Czech *MaltParser* is outperformed by other dependency parsers, e.g., the Czech *MST* parser with 80.2% LAS (Buchholz and Marsi, 2006) and the *Mate* dependency parser with 80.96% LAS (Bohnet, 2010). The Russian *MaltParser* (Nivre et al., 2008), in turn, trained on the large dependency treebank SynTagRus[11] (Boguslavsky et al., 2002) achieved 82.2% LAS and 89.1% UAS.

The dependency parsing systems mentioned above, i.e., *MaltParser* (Nivre et al., 2006a), the *MST* parser (McDonald et al., 2005a) and the *Mate* parser (Bohnet, 2010), are among the best state-of-the-art dependency parsers. They are widely used for various languages and different language processing tasks. They represent two main data-driven dependency parsing approaches: transition-based (*MaltParser*) and graph-based (*MST* and *Mate*).

[8] The parser was trained on 72,700 sentences (1,249,000 tokens in total and 17.2 tokens per sentence on average) in CoNLL 2006 and on 25,400 sentences (450,000 tokens, 17 tokens per sentence on average) in CoNLL 2007.

[9] Labelled attachment score (LAS) – the percentage of tokens that are assigned the correct head and the correct dependency type.

[10] Unlabelled attachment score (UAS) – the percentage of tokens that are assigned the correct head.

[11] SynTagRus dependency treebank (Boguslavsky et al., 2002) contains over 32,000 sentences (460,000 tokens) taken from various genres.

2.4.1 Transition-based Dependency Parsing

A transition-based dependency parser is a state machine for mapping a sentence to its dependency tree. It finds a sequence of valid transitions between complex configurations in a way resembling a shift-reduce procedure of context-free grammars. The transition system consists of a set of configurations (states) and transitions (actions) between configurations.

Configurations may be understood as partial analyses of an input sentence. Following Kübler et al. (2009), a configuration for a sequence of tokens plus a ROOT node $S = v_0, v_1, ..., v_n$ may be defined as a triple $c = (\sigma, \beta, A)$, where σ is a stack of partially processed words, β is a buffer of remaining input words, and A is a set of dependency arcs of the form (v_i, v_j, l). The set A constitutes a partially built dependency tree. Parsing starts with an initial configuration in which the first token (conventionally the ROOT node v_0) is on the stack σ, all other tokens are in the buffer β and there are no arcs in the set A, i.e., $([v_0]_\sigma, [v_1, ..., v_n]_\beta, \varnothing)$. The parsing of an input sentence terminates in a configuration in which the buffer β is empty, i.e., $(\sigma, []_\beta, A)$. The terminal configuration defines the valid dependency tree for an input sentence.

Transitions correspond to the steps in the process of deriving a dependency tree. They are equivalent to parsing actions that add an arc to the dependency tree, or modify the stack or the buffer. The basic transitions in the shift-reduced parsing are LEFT-ARC$_l$ $(\sigma|v_i, v_j|\beta, A) \Rightarrow (\sigma, v_j|\beta, A \cup \{v_j, v_i, l\})$[12] and RIGHT-ARC$_l$ $(\sigma|v_i, v_j|\beta, A) \Rightarrow (\sigma, v_i|\beta, A \cup \{v_i, v_j, l\})$[13] that add arcs to the set A, and SHIFT $(\sigma, v_i|\beta, A) \Rightarrow (\sigma|v_i, \beta, A)$ which pushes the first token in the buffer on the top of the stack.

In the learning phase, the parsing system induces a model for predicting next transitions given an input and previously constructed arcs (the transition history). For the purpose of transition-based training, training data which is usually arranged in pairs of sentences and their dependency trees has to be transformed into sequences of parser configurations and transitions used to derive the training trees. The model is parametrised over these training transitions.

Parsing of an input sentence consists in the construction of an optimal transition sequence given the induced parsing model. Since more than one transition may be valid for a non-terminal configuration, the transition system is non-deterministic. In order to parse sentences deterministically, the transition system requires an *oracle*. The oracle is a function that determines an optimal transition from the current non-terminal configuration to the next configuration. The oracle is approximated by a classifier trained on

[12]The LEFT-ARC$_l$ transition adds an arc between a token v_j, which is the first token in the buffer, and its dependent v_i, which is on the top of the stack, to the set A. Meanwhile, v_i pops the stack.

[13]The RIGHT-ARC$_l$ transition adds an arc between a token v_i, which is on the top of the stack, and its dependent v_j, which is the first token in the buffer, to the set A. Furthermore, v_i pops the stack and replaces v_j at the head position in the buffer.

treebank data using various learning methods (e.g., support vector machines, memory-based learning or maximum entropy modelling).

The classifier predicts an oracle transition for any configuration represented as a feature vector. Typical features of the feature representation are defined in terms of lexical items (e.g., lexical item of the current node, lexical item of its governing node, lexical items of neighbouring, child or siblings nodes) and attributes of lexical items (lemmata, parts of speech, morphosyntactic features, dependency types or distance between target tokens).

A remarkable feature of transition-based dependency parsers is their relatively low complexity. Transition-based dependency parsers have a linear or quadratic parsing time complexity. One of the first transition-based parsers (Yamada and Matsumoto, 2003), which uses support vector machines to predict transitions, outputs unlabelled dependency trees for input sentences. Other data-driven transition-based parsers, which have been developed later, output labelled dependency structures, e.g., *MaltParser* (Nivre and Nilsson, 2005; Nivre et al., 2006a), and systems by Attardi (2006), Attardi and Ciaramita (2007), Duan et al. (2007), Johansson and Nugues (2007), and Titov and Henderson (2007).

2.4.2 Graph-based Dependency Parsing

A graph-based dependency parser predicts a dependency tree as a correct analysis of an input sentence. The parsing procedure starts with defining a space of well-formed candidate dependency trees for an input sentence and scoring them given an induced parsing model. Then, the highest scoring dependency tree is selected as a correct analysis of the input sentence.

A score of a dependency tree is obtained as a function of scores of its dependency subgraphs.[14] The majority of graph-based dependency parsing models assume that a tree score is the sum of subgraph scores (factor parameters). However, models may also define a tree score as the product of subgraph parameters. Scores of dependency subgraphs, in turn, are estimated based on a pretrained parsing model. In arc-factored models, the score of an arc $\lambda_{(v_i,v_j,l)}$ is estimated as the dot product of a high dimensional feature representation $\mathbf{f}$ of the arc and a learnt weight vector $\mathbf{w}$, i.e., $\lambda_{(v_i,v_j,l)} = \mathbf{w} \cdot \mathbf{f}(v_i, v_j, l)$. The tree score, in turn, is estimated as the sum of individual arc scores (see Equation (2.1)).

$$score(T = (V, A)) = \sum_{(v_i,v_j,l)\in A} \lambda_{(v_i,v_j,l)} = \sum_{(v_i,v_j,l)\in A} \mathbf{w} \cdot \mathbf{f}(v_i, v_j, l) \tag{2.1}$$

[14] Dependency subgraphs are also called factors. In the first-order factorisation, a dependency subgraph (factor) is associated with a single arc (i.e., arc-factored models).

The idea behind graph-based dependency parsing is to define an algorithm that finds a tree for which factor parameters sum to a maximum value. In arc-factored models, the problem of selecting the highest scoring tree is defined with the function h (see Equation (2.2) after Kübler et al., 2009, p. 43). In Equation (2.2) S is a sentence, Γ is a set of constraints that define the space of permissible dependency trees for the sentence S, $\boldsymbol{\lambda}$ is a set of possible parameters $\lambda_{(v_i,v_j,l)}$, $T = (V, A)$ is a dependency tree from the set of all well-formed (compliant to Γ) candidate trees $\mathcal{G}_{\Gamma,S}$ of S, and $\lambda_{(v_i,v_j,l)} \in \boldsymbol{\lambda}$ is a real numbered parameter of the arc $(v_i, v_j, l) \in A$.

$$h(S, \Gamma, \boldsymbol{\lambda}) = \underset{T=(V,A)\in\mathcal{G}_{\Gamma,S}}{\operatorname{argmax}} \sum_{(v_i,v_j,l)\in A} \lambda_{(v_i,v_j,l)} = \underset{T=(V,A)\in\mathcal{G}_{\Gamma,S}}{\operatorname{argmax}} \sum_{(v_i,v_j,l)\in A} \mathbf{w} \cdot \mathbf{f}(v_i, v_j, l) \quad (2.2)$$

Graph-based dependency parsing approaches are based on the idea that a spanning tree of a digraph corresponds to a dependency tree. Hence, solving the arc-factored dependency parsing problem corresponds to finding the maximum spanning tree in a set of well-formed spanning trees of S. Standard algorithms aim to find maximum spanning trees in digraphs, e.g., the MST algorithm by McDonald et al. (2005a) built on the Minimum Spanning Tree Algorithm provided independently by Chu and Liu (1965) and Edmonds (1967).

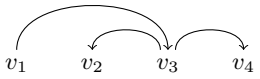
Algorithm 2.1 (based on Kübler et al., 2009, p. 47)[15] lists the pseudocode for the MST algorithm that finds possibly non-projective[16] maximum spanning trees in directed graphs. The MST algorithm iteratively searches for a spanning tree $T = (V, A)$ which maximises the value $\sum_{(v_i,v_j)\in A} \lambda_{(v_i,v_j)}$, i.e., the sum of arc parameters $\lambda_{(v_i,v_j)}$. The algorithm starts with selecting an incoming edge with the highest score from the digraph G for each

[15] The MST algorithm is adapted to labelled dependency parsing. Hence, each arc has the form of a triple (v_i, v_j, l), where v_i, $v_j \in V$ and $l \in L$. However, Kübler et al. (2009) present the version of the algorithm with the reduced representation of arcs without the relation label, i.e., (v_i, v_j). We keep this convention in our presentation too.

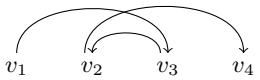
[16] *Projectivity* may be defined as follows (based on Kübler et al., 2009):

1. An arc $(v_i, v_j, l) \in A$ in a dependency tree $T = (V, A)$ is **projective** iff there exists the transitive closure of a relation $v_i \rightarrow v_k$ (i.e., $v_i \overset{*}{\rightarrow} v_k$) for all words v_k intervening between v_i and v_j (i.e., for all $i < k < j$ when $i < j$ or $j < k < i$ when $j < i$).
2. A dependency tree $T = (V, A)$ is projective iff all arcs in A are projective.

This definition implies that the projective dependency structure is a directed tree of a sentence constrained on the linear word order, i.e., dependency arcs in this tree do not cross with respect to the word order of the sentence, e.g.:



A **non-projective** dependency tree, in turn, models syntactic constructions such as topicalisation, wh-movement, discontinuous noun phrases, or other constructions resulting in crossing arcs, e.g.:



Algorithm 2.1 MST algorithm (based on Kübler et al., 2009, p. 47)

h$(S, \Gamma, \boldsymbol{\lambda})$ *# non-projective parsing*
Sentence $S = v_1, ..., v_n$
Set of constraints Γ
Arc parameters $\lambda_{(v_i,v_j)} \in \boldsymbol{\lambda}$

$G_S := (V_S, A_S)$, where $V_S = \{v_0, v_1, ..., v_n\}$ with v_0 being the ROOT node and
$A_S = \{(v_i, v_j)|$ for all $v_i, v_j \in V_S\}$
Return Chu-Liu-Edmonds$(G_S, \boldsymbol{\lambda})$

Chu-Liu-Edmonds$(G, \boldsymbol{\lambda})$
Graph $G = (V, A)$
Arc parameters $\lambda_{(v_i,v_j)} \in \boldsymbol{\lambda}$

$A' := \{(v_i, v_j)|v_j \in V, v_i = \text{argmax}_{v_i}\lambda_{(v_i,v_j)}\}$
$T := (V, A')$
If T has no cycles, then it is the MST
Return T
Otherwise, find any subgraph $C := (V_C, A_C)$ corresponding to a cycle in T
$< G_C, v_c, ep >$:= contract$(G, C, \boldsymbol{\lambda})$
$T := (V, A)$ = Chu-Liu-Edmonds$(G_C, \boldsymbol{\lambda})$
For the arc $(v_i, v_c) \in A$, where $ep(v_i, v_c) = v_j$,
identify an arc $(v_k, v_j) \in A_C$ for some v_k
Find all arcs $(v_c, v_l) \in A$
$A := A \cup \{(ep(v_c, v_l), v_l)|$ for all $(v_c, v_l) \in A\} \cup A_C \cup \{(v_i, v_j)\} - \{(v_k, v_j)\}$
Return T

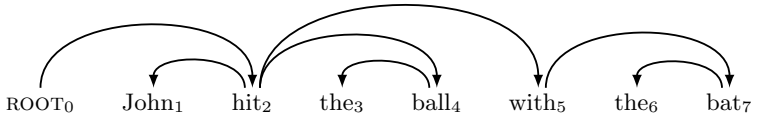
contract$(G, C, \boldsymbol{\lambda})$
Graph $G = (V, A)$
Subgraph $C = (V_C, A_C)$ of T forming a cycle
Arc parameters $\lambda_{(v_i,v_j)} \in \boldsymbol{\lambda}$

Let G_C be the subgraph of G excluding nodes in C
Add a node v_c representing the cycle to G_C
For $v_j \in V \setminus V_C : \exists_{v_i \in V_C}(v_i, v_j) \in A$
Add arc (v_c, v_j) to G_C with $ep(v_c, v_j) := \text{argmax}_{v_i \in V_C}\lambda_{(v_i,v_j)}$, for the function ep
keeping track of arcs coming in and out of v_c
$v_i := ep(v_c, v_j)$
$\lambda_{(v_c,v_j)} := \lambda_{(v_i,v_j)}$
For $v_i \in V \setminus V_C : \exists_{v_j \in V_C}(v_i, v_j) \in A$
Add arc (v_i, v_c) to G_C with $ep(v_i, v_c) := \text{argmax}_{v_j \in V_C}\left[\lambda_{(v_i,v_j)} - \lambda_{(a(v_j),v_j)}\right]$
$v_j := ep(v_i, v_c)$
$\lambda_{(v_i,v_c)} := \left[\lambda_{(v_i,v_j)} - \lambda_{(a(v_j),v_j)} + \text{score}(C)\right]$, where $a(v)$ is the predecessor of v
in C and $\text{score}(C) = \sum_{v \in V_C} \lambda_{(a(v),v)}$
Return $< G_C, v_c, ep >$

node except the ROOT node. The selected directed edges build a candidate graph T with the highest total score of arc weights. Then, the candidate graph T is searched for cycles. If there is no cycle in T, it is regarded as the maximum spanning tree and is returned. If there is a cycle in the candidate graph T, this cycle is contracted into a new node and arc parameters are recalculated as presented in the **contract** function in Algorithm 2.1. The cycle nodes are contracted into a new node v_c, i.e., a new node representing the contraction of nodes in the cycle. Original input arcs (except for arcs building the cycle) are split into three groups: arcs coming out of the cycle, arcs coming into the cycle and arcs without the cycle nodes. Arcs without the cycle nodes are directly added to the contracted graph G_C. Scores of the arcs going into and out of the cycle are recalculated and the best scored arcs are added to the contracted graph. The parameter $\lambda_{(v_c,v_j)}$ for an arc coming out of the contracted node v_c is equal to the score of the arc. The parameter $\lambda_{(v_i,v_c)}$ for an arc going into the contracted node v_c is equal to the highest score of an arc coming into the cycle and breaking it. The Chu-Liu-Edmonds algorithm is then recursively called until a tree is found in the contracted graph. Finally, arcs of the spanning tree selected from the contracted graph are merged with cycle arcs recovered using the function *ep*. This function keeps track of the arcs coming in and out of v_c, and is used for the graph reconstruction. The cycle arc (v_k, v_j) is not part of the output tree, since this arc has to be removed in order to break the cycle. The highest scoring tree output by the Chu-Liu-Edmonds algorithm is considered a valid dependency tree.

The above paragraphs indicate that factors in arc-factored models correspond to single dependency arcs of a given dependency tree. Hence, arc-factored models represent first-order graph-based parsing models. In the dth-order factorisation, the score of a tree is factored over d arcs, i.e., the current arc and $d-1$ arcs from the neighbourhood of the current arc. First-order parsing algorithms have a quadratic complexity. In second-order parsing algorithms (e.g., McDonald and Pereira, 2006), factors may be represented as pairs of arcs sharing the same parent node. Dependents of these arcs should be located on the same side of the governor.[17] This algorithm, designed for projective trees, has the parsing time complexity of $O(n^3)$,[18] whereas a non-projective version of this parsing algorithm is intractable. The higher-order algorithm by Carreras (2007) further expands the scope of arcs within subgraphs by arcs between a dependent and its child which is

[17] In the bottom tree (taken from McDonald and Pereira, 2006), the second-order scoring function considers two arcs *hit*→*with* and *hit*→*ball* instead of the main arc *hit*→*with* only. These two arcs share the same governor and their dependents are located on the same side of the governor *hit*.



[18] This projective parsing algorithm can be extended to dth-order model with a complexity of $O(n^{d+1})$, for $d > 1$.

located between the dependent and its governor,[19] or arcs pointing to grandchildren-nodes. This parsing algorithm has the time complexity of $O(n^4)$.

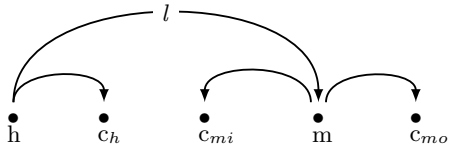
As we have already stated, the dependency parsing problem consists in finding a parse tree for an input sentence that maximises the scoring function. The learning problem, in turn, consists in determining the scoring function so that the accuracy of a parse tree predicted by the parser is as high as possible. In the learning phase, parsing models are trained using algorithms that optimise model parameters in order to maximise the difference in score between the correct dependency tree and all other dependency trees built for the training sentence.

Graph-based dependency parsing models are parametrised over dependency subgraphs (factors). In the simplest but commonly used version of graph-based models – an arc-factored model – arc parameters are estimated based on a vector of predefined features over the arc and a corresponding feature weight vector. The following features may be considered by a feature representation: lexical items (dependent, governor), attributes of related items (e.g., part of speech tags, lemmata, morphological features, distance between related tokens, arc direction or morphosyntactic attributes of surrounding lexical items) and grammatical functions which arcs are labelled with. These categorical features are converted into binary features. For a categorical feature with k possible values, the arc feature function includes k different features with values of 0 or 1 indicating the absence or the presence of the feature-value pair.

In order to learn parameters from training data (sentences with corresponding dependency trees), various learning algorithms may be employed, e.g., inference-based learning algorithms, probabilistic log-linear models, large-margin models or generative bigram models. Assuming that arc parameters are linear classifiers in arc-factored models, the inference problem may be defined as the function h in Equation (2.2).

The most common inference-based learning algorithm is the *perceptron* algorithm (see Algorithm 2.2, after Kübler et al., 2009, p. 57) which induces model parameters $\boldsymbol{\lambda}$ from training data $\mathcal{D}$ ($\boldsymbol{\lambda}$ contains the predefined feature vector $\mathbf{f}$ and the learnt weight vector $\mathbf{w}$). The general procedure of training dependency parsing models is as follows. For each training sentence, the algorithm predicts the highest scored dependency tree T. If there are no errors in the parser output T relative to the training tree T_d, the parser moves on to the next sentence. If there are some discrepancies, the parser compares

[19] In the bottom tree diagram (taken from Carreras, 2007), $h \rightarrow m$ is the main dependency labelled with l and $h \rightarrow c_h$ is the second-order arc. The higher-order factorisation would additionally consider $c_{mi} \leftarrow m$, since c_{mi} is a child of m located between h and m.



the predicted tree with the gold standard tree and updates the weight vector w taking the difference between trees. Feature weights increase or decrease depending on their correctness status, i.e., their presence in the correct solution or in the possibly incorrect solution. In the training procedure, data are iteratively reparsed in order to adjust the feature model so that it produces trees that match the gold standard trees (training data). The model parametrised over dependency subgraphs should score correct trees higher than incorrect ones.

Algorithm 2.2 Perceptron learning algorithm used in graph-based dependency parsing (after Kübler et al., 2009, p. 57).

Perceptron($\mathcal{D}$)
Training data $\mathcal{D} = \{(S_d, T_d)\}_{d=1}^{|\mathcal{D}|}$, where $T_d = (V_d, A_d)$
$\mathbf{w} := \mathbf{0}$

For $n : 1..N$, where N is a number of training iterations
  For $d : 1..|\mathcal{D}|$
    $T := h(S_d, \Gamma, \boldsymbol{\lambda}) = \text{argmax}_{T=(V',A')\in\mathcal{G}_{S_d}} \sum_{(v_i,v_j,l)\in A'} \mathbf{w} \cdot \mathbf{f}(v_i, v_j, l)$
    If $T \neq T_d$
      $\mathbf{w} := \mathbf{w} + \sum_{(v_i,v_j,l)\in A_d} \mathbf{f}(v_i, v_j, l) - \sum_{(v_i,v_j,l)\in A'} \mathbf{f}(v_i, v_j, l)$
Return $\mathbf{w}$

Arc parameters can then be calculated: $\lambda(v_i, v_j, l) := \mathbf{w} \cdot \mathbf{f}(v_i, v_j, l)$

One of the first graph-based models was defined by Eisner (1996). The work on graph-based models was continued by McDonald et al. (2006), Carreras (2007), Koo et al. (2007), Nakagawa (2007), Smith and Smith (2007), etc. There are also some widely used graph-based dependency parsing systems, e.g., the *MST* parser (McDonald et al., 2005a, 2006; McDonald and Pereira, 2006) and the *Mate* parser (Bohnet, 2010).

Chapter 3

Polish Dependency Annotation Schema

In this chapter we define a dependency annotation schema adjusted to the linguistic characteristics of Polish. The schema is not equivalent to any comprehensive Polish dependency grammar description and it is not defined with the aim of underlying any grammar-based Polish dependency parser. It is rather a set of annotation constraints imposed on domination (or dependency) relations in Polish sentences.[1] However, as one solution that could meet all problematic cases could not always be found, the schema constitutes a kind of compromise and it will be improved and expanded if it is necessary.

The chapter starts with some background information underlying the design of the Polish dependency annotation schema (see Section 3.1 *Foundations*). In the main part of the chapter, we characterise dependency relation types which are used to annotate Polish sentences (see Section 3.2 *Polish Dependency Relation Types*). Finally, an overview of existing annotation schemata which influenced the definition of the Polish dependency annotation schema is given in Section 3.3 *Syntactic Annotation Schemata: Related Work*.

3.1 Foundations

The annotation schema is designed to cover primary linguistic phenomena existing in common Polish sentences and described in the linguistic literature. The proposed schema is primarily based on theoretical works by Mel'čuk (Mel'čuk, 1988; Polguére and Mel'čuk, 2009) and Świdziński (1989). It also applies the annotation proposals by Obrębski (2002) and by the authors of the *Prague Dependency Treebank* (PDT, Hajič, 1998; Böhmová

[1] Note that coreference relations are not treated as dependencies in the following description.

et al., 2003).[2] The Polish dependency annotation schema bears a debt to *Lexical Functional Grammar* (Bresnan, 2001; Dalrymple, 2001; Falk, 2001), especially to the set of grammatical functions defined in this formalism. For a theoretical explanation of some Polish phenomena, we refer to Saloni and Świdziński (2011) and Saloni (2010).

The Polish dependency annotation schema aims to annotate Polish sentences with accurate and high-coverage labelled dependency trees that provide a transparent and articulated account of the predicate-argument structure. The schema primarily covers syntactic phenomena, but also phenomena which have a morphological background (e.g., a relation between two tokens *zrobili-* and *-śmy* which constitute the word *zrobiliśmy*, Eng. 'do'$_{praet.pl.pri}$), or a semantic or idiomatic background (e.g., relations between elements of multiword expressions). Even if dependency trees represent syntactic-like analyses of sentences, they may contain semantically oriented relations in which a content word dominates a function word (e.g., a main verb dominates an auxiliary verb or the sentence predicate of a subordinate clause dominates a complementiser). The semantically oriented schema enables annotation of dependencies that may be useful in the future semantic processing.

The schema is designed for annotating naturally occurring sentences with dependency structures. Acquired dependency structures may then be employed to train dependency parsers using publicly available dependency parsing systems. Hence, the schema constraints should comply with the formal constraints imposed by these parsing systems, e.g., in terms of data encoding. Constraints of the parsing systems imply that dependency relations represented as directed edges in a tree connect lexical nodes which are equivalent to tokens in a sentence. Additional nodes corresponding to empty tokens (e.g., in ellipsis or for pro-drop pronouns) are not possible.

Dependency trees are encoded in the column-based CoNLL format (Buchholz and Marsi, 2006) which allows to include only a limited amount of information. We use eight out of ten columns of the CoNLL format to encode dependency trees:[3] `ID` (an integer token identifier), `FORM` (a token – a word form or a punctuation symbol), `LEMMA` (the lemma of the token), `CPOSTAG` (the coarse-grained part of speech tag of the token), `POSTAG`

[2] An informative overview of Polish dependency types is presented in Derwojedowa (2011, Appendix A, pp. 175ff.). Many of these dependency types correspond to dependency relations distinguished in our annotation schema. However, we do not adapt Derwojedowa's convention of labelling dependency relations, because it is not sufficiently homogeneous for our purposes. For example, the dependency relation between a preposition and its nominal dependent is labelled with *prep* (prepositional dependency, Pol. *zależność przyimkowa*) derived from the governing preposition, while the dependency relation between a noun functioning as the subject and its governing verb is labelled with *subj* (subjective dependency, Pol. *zależność podmiotowa*) derived from the function of the dependent. Labels of dependency relations distinguished in our annotation schema are uniformly derived from grammatical functions fulfilled by dependents.

[3] The last two columns of the CoNLL format contain information about projective versions of encoded trees: an `ID` of the projective head of the token (`PHEAD`) and a label of the dependency relation governed by the `PHEAD` (`PDEPREL`). A dependency structure resulting from the `PHEAD` column is guaranteed to be projective. However, as Polish allows non-projective dependency structures that do not have any correct projective alternatives, these two columns are not filled with data.

(the fine-grained part of speech tag of the token), `FEATS` (the set of morphosyntactic features of the token separated by vertical bars), `HEAD` (the `ID` of the governor of the token) and `DEPREL` (the label of the dependency relation governed by the `HEAD`). If a value is not available (e.g., noninflected words do not have morphological features), an underscore is used as a default value.

The dependency schema assumes that sentences (also clauses or phrases) are annotated with well-formed dependency trees as described in Section 2.3 *Dependency Structure* and labelled with grammatical functions presented in the following section 3.2.

3.2 Polish Dependency Relation Types

In order to cover various language-specific syntactic phenomena and to annotate sentences with the correct dependency structures, precise definitions of dependency relation types are crucial. A list of valid Polish dependency relations is established based on linguistic literature, syntactic descriptions of Polish and annotation solutions proposed for other languages. The distinguished dependency relations are assigned grammatical functions fulfilled by dependents of these relations. In this dissertation, we identify dependency relations with grammatical functions assigned to them. Dependency relations are distinguishable by the features of related tokens. While defining individual dependency relation types, we take into account morphosyntactic, topological, thematic, and alternation criteria.

In highly inflected languages, morphological marking has an impact on the identification of grammatical functions. Morphological features of a depending lexical item usually either agree with features of the governor or are required by the governor. However, dependency relations may not be solely identified by means of morphology. For example, grammatical functions of accusative nouns *tydzień* (Eng. 'a week') functioning as the adjunct of duration in the sentence *Pił tydzień.* (Eng. 'He drank for a week.') and *wodę* (Eng. 'water') functioning as the object in *Pił wodę.* (Eng. 'He drank water.') may not be distinguished at the morphological level. Topological clues may also be useful to identify grammatical functions. The linear position of a dependent may result from the governor position, e.g., the complement of a preposition usually follows this preposition. Thematic and argument-structure alternation criteria, which support morphological and topological criteria, contribute to the definition of some grammatical functions, especially those with the argument status. A thematic role (e.g., Agent, Patient, Theme, Beneficiary, Recipient or Instrument) constitutes a semantically oriented indication for defining a grammatical function, assuming that syntactic and semantic levels of the relation correspond with each other. Thematic roles are particularly useful for distinguishing between direct objects and thematically restricted objects.

As mentioned above, dependency relations are identified with their labels equivalent to grammatical functions borne by dependents. Some grammatical functions (e.g., subject, object, adjunct) are commonly shared by various languages and various annotation schemata. However, even if a lot of dependency theories rely on the notion of a syntactic function, they avoid defining precisely individual grammatical properties of common grammatical functions. This is due to the fact that bearing grammatical functions by different lexical items (or phrases) is a language-specific property. Hence, an inventory of distinct grammatical functions associated with dependency relations should be defined individually for each language or at least group of languages.

In the further course of this chapter we present a repertoire of dependency relations identified by their labels. Our review of dependency relation types starts with the presentation of relations between governors and their arguments. It is followed by the description of non-argument relations. This overview closes with the treatment of relations in coordinating constructions. Dependency relation types are listed alphabetically within the distinguished subgroups.

3.2.1 Arguments

The group of arguments consists of subjects subcategorised by predicates and different complements required by predicates, prepositions, adjectives, numerals, etc. Conventionally, each argument type can occur only once as a dependent of a governor. However, it is possible that a predicate subcategorises more then one thematically restricted object. In this case, all objects must be associated with different semantic roles.

***comp* (complement)**
The syntactically obligatory *comp* argument may be required by differently realised governors and may have diverse surface realisations – adjectival, adverbial, nominal or prepositional phrases.

- An **adjectival complement** may be governed by a verb (see Figure 3.1) or a preposition (see Figure 3.2).

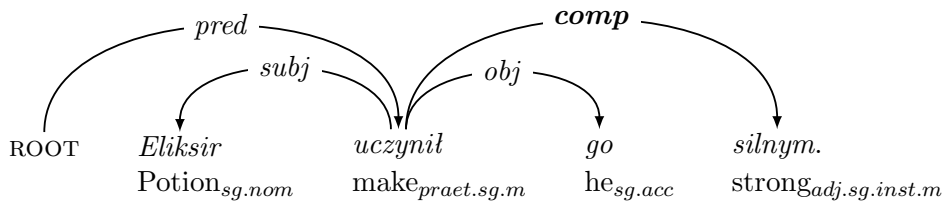


FIGURE 3.1: A dependency structure of the sentence *Eliksir uczynił go silnym.* (Eng. 'The potion made him strong.').

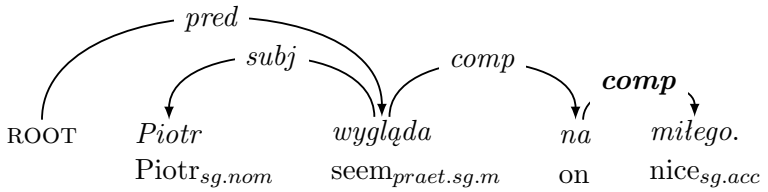


FIGURE 3.2: A dependency structure of the sentence *Piotr wygląda na miłego.* (Eng. 'Piotr seems to be nice.').

- An **adverbial complement** may be governed by a verb (see Figure 3.3) or a preposition (see Figure 3.4).

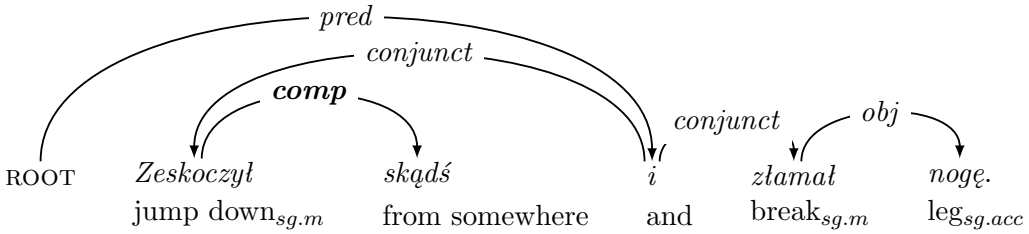


FIGURE 3.3: A dependency structure of the sentence *Zeskoczył skądś i złamał nogę.* (Eng. 'He jumped down from somewhere and broke his leg.').

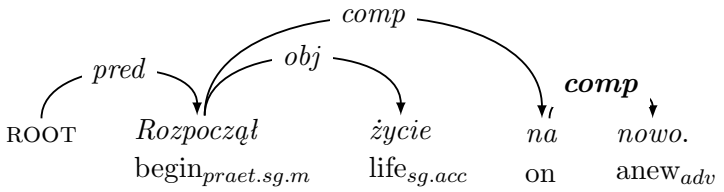


FIGURE 3.4: A dependency structure of the sentence *Rozpoczął życie na nowo.* (Eng. 'He began a new life.').

- A **nominal complement**, in turn, may be governed by an adjective (see *przeciwny **globalizacji*** in Figure 3.5), a preposition (see *Od **lat*** in Figure 3.5) or a numeral. Numerals are annotated as governors of depending noun phrases, even if this results in a non-projective dependency tree (see Figure 3.6). Saloni and Świdziński (2011) argue for treating numerals as heads in Polish numeral phrases since it is the numeral that is governed by the verb form and not the noun. The case of the depending noun phrase, in turn, either agrees with the case of the governing numeral (dative, instrumental or locative) or is determined as genitive if the numeral is marked for nominative, accusative, vocative or genitive. On the other hand, the depending noun phrase imposes the gender on the governing numeral. Due to this fact, the morphological dependency relation between the two elements

of a numeral phrase (numeral and noun phrase) is bilateral (Mel'čuk, 1988, p. 109). Nevertheless, we carry on the Polish tradition of analysing numeral expressions and annotate numerals as governors of depending noun phrases.

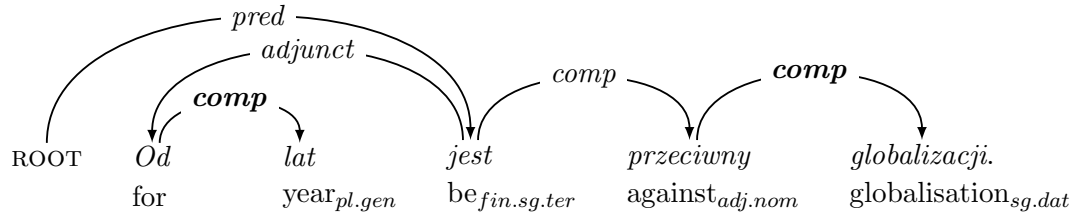


FIGURE 3.5: A dependency structure of the sentence *Od lat jest przeciwny globalizacji.* (Eng. 'He has been against the globalisation for years.').

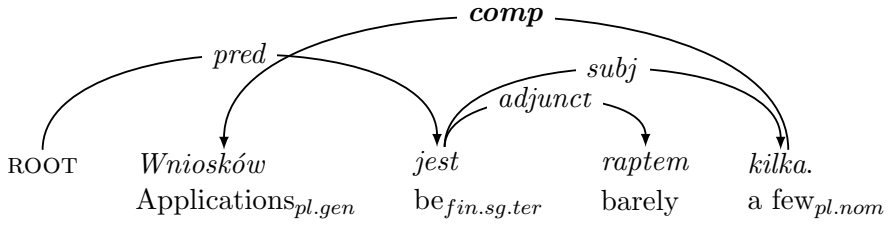


FIGURE 3.6: A non-projective dependency structure of the sentence *Wniosków jest raptem kilka.* (Eng. 'There are barely a few applications.').

- An obligatory **prepositional complement** may be governed by a verb form (see *Czekając* ***na*** in Figure 3.7) or an adjective (see *zdolny* ***do*** in Figure 3.7). Similarly as in many other languages, semantically and idiosyncratically marked prepositional complements may be distinguished in Polish. In the case of semantically marked prepositional complements, the preposition expresses its thematic role, e.g., *mieszkać w Paryżu/na statku/pod mostem* (Eng. 'to live in Paris/on a ship/under a bridge'). In the case of idiosyncratically marked prepositional complements, the form of a preposition is lexically specified by the governing predicate. The prepositional complement is required to bear a particular form unrelated to the semantic role of the argument, e.g., *czekać na pociąg/na odpowiedź* (Eng. 'to wait for a train/an answer'). The current annotation schema does not distinguish between these two types of prepositional complements since it is not essential for annotating dependency structures.

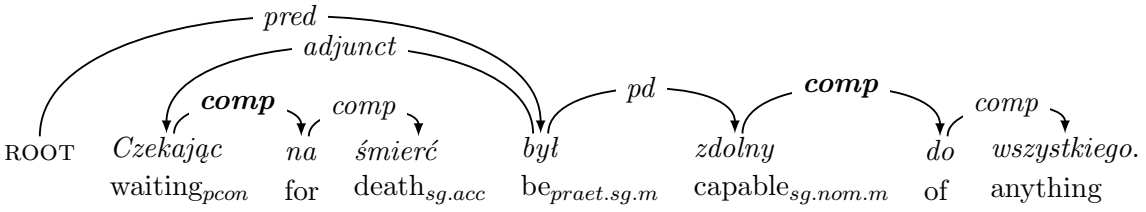


FIGURE 3.7: A dependency structure of the sentence *Czekając na śmierć był zdolny do wszystkiego.* (Eng. 'Waiting for death, he was capable of anything.').

***comp_ag* (agentive complement in passive)**

The *comp_ag* complement is a demoted subject which is realised as a *przez*-prepositional phrase in passive sentences. The *comp_ag* argument is governed by a passive adjective participle (see Figure 3.8).

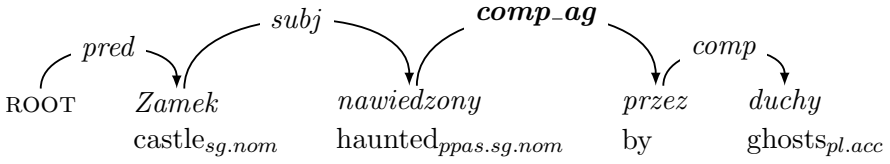


FIGURE 3.8: A dependency structure of the noun phrase *Zamek nawiedzony przez duchy* (Eng. 'A haunted castle').

***comp_fin* (clausal complement)**

The *comp_fin* function is borne by a closed complement clause[4] (declarative, interrogative, or exclamatory) with an internal subject that may be realised as a pro-drop pronoun. The *comp_fin* argument may be governed by a predicate of a superordinate clause (see Figure 3.9), a subordinating conjunction (see Figure 3.10) or a noun (see Figure 3.11). A possible interrogative word (pronoun, conjunction, particle) at the beginning of an interrogative clause bears a grammatical function determined by the predicate of this clause (e.g., the interrogative particle *co* in Figure 3.11 is the subject of the predicate *działo*).

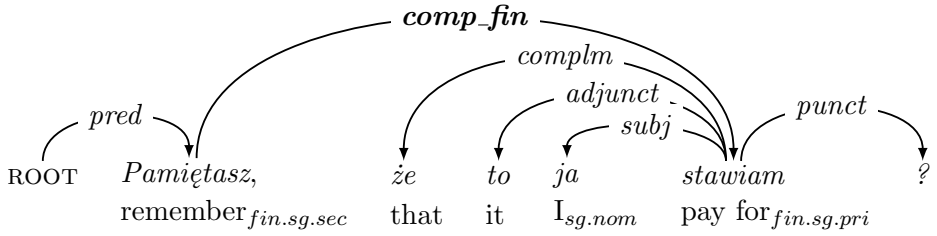


FIGURE 3.9: A dependency structure of the sentence *Pamiętasz, że to ja stawiam?* (Eng. 'Do you remember that this is on me?').

[4] A closed complement clause is a subordinate clause with an internal subject.

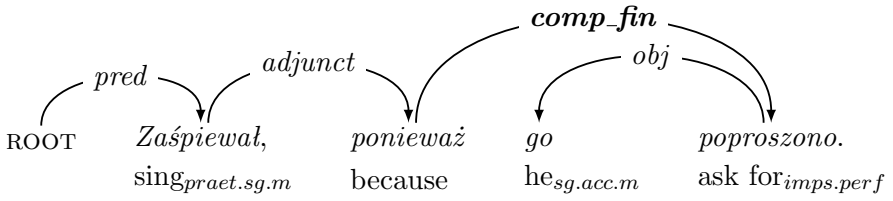


FIGURE 3.10: A dependency structure of the sentence *Zaśpiewał, ponieważ go poproszono.* (Eng. 'He sang because he was asked to.').

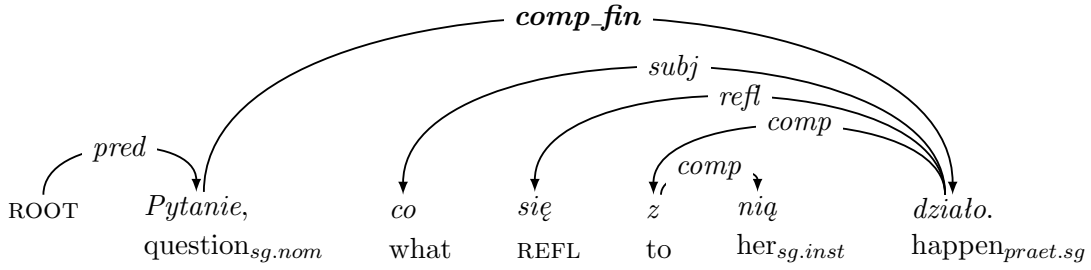


FIGURE 3.11: A dependency structure of the noun phrase *Pytanie, co się z nią działo.* (Eng. 'The question what happened to her.').

***comp_inf* (infinitival clausal complement)**

The *comp_inf* function fulfilled by an infinitival clausal complement (non-finite subordinate clause) may be governed by an adjective (see Figure 3.12), a noun (see Figure 3.13) or a verb. Some control verbs, e.g., *kazać* (Eng. 'to order, to tell'), *chcieć* (Eng. 'to want') (see Figure 3.14) or some quasi-verbs (see Figure 3.15) may require the *comp_inf* complement. Polish distinguishes between proper and quasi-verbs, e.g., *brak* (Eng. 'to miss, to fail, to lose'), *grzmi* (Eng. 'it's thundering'), *można* (Eng. 'it's allowed'), *należy* (Eng. 'it's necessary, should'), *szkoda* (Eng. 'it's pointless'), *trzeba* (Eng. 'it's necessary, should'), *warto* (Eng. 'it's worth'), *wiadomo* (Eng. 'it's known'), *wolno* (Eng. 'might, must'). Quasi-verbs do not inflect by number, person or gender, but they are marked for mood and tense, e.g., *będzie trzeba* (Eng. 'it will be necessary), *byłoby warto* (Eng. 'it would be worth').

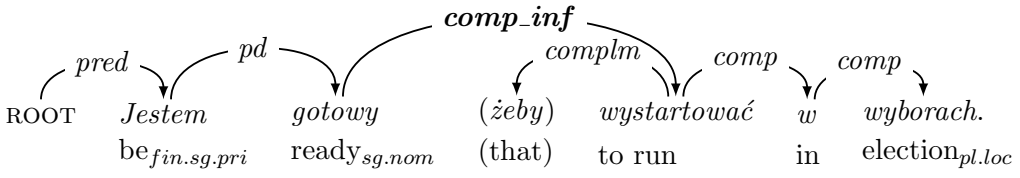


FIGURE 3.12: A dependency structure of the sentence *Jestem gotowy wystartować w wyborach.* (Eng. 'I'm ready to run in the election.').

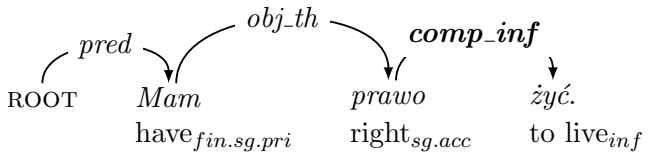


FIGURE 3.13: A dependency structure of the sentence *Mam prawo żyć.* (Eng. 'I have a right to live.').

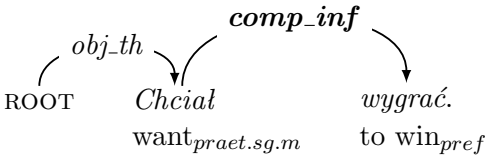


FIGURE 3.14: A dependency structure of the sentence *Chciał wygrać.* (Eng. 'He wanted to win.').

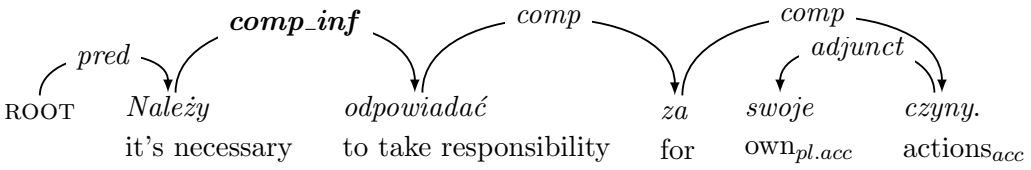


FIGURE 3.15: A dependency structure of the sentence *Należy odpowiadać za swoje czyny.* (Eng. 'One should take responsibility for one's actions.').

***obj* (direct object)**

The *obj* argument subcategorised by a sentence predicate may be realised as a noun/numeral phrase marked for accusative, genitive, instrumental or even dative (see Figure 3.16), or as a finite clause. The principal property of the *obj* argument is its ability to transform into the subject in passive constructions (see Figure 3.17). This feature distinguishes the *obj* argument from other arguments of a verb realised as nouns with the *obj_th* function or as closed complement clauses with the *comp_fin* function.

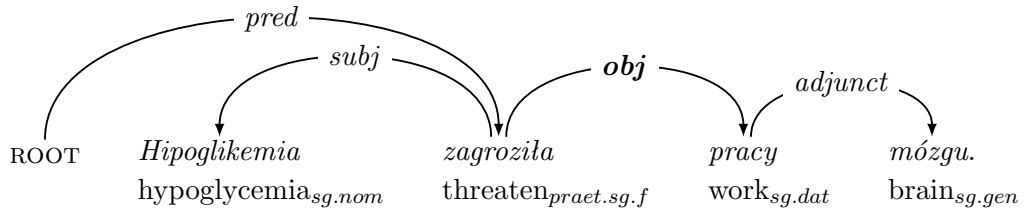


FIGURE 3.16: A dependency structure of the sentence *Hipoglikemia zagroziła pracy mózgu.* (Eng. 'Hypoglycemia threatened the brain function.').

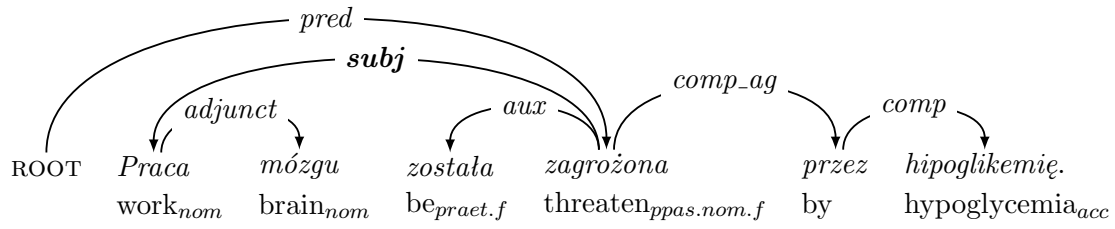


FIGURE 3.17: A dependency structure of the sentence *Praca mózgu została zagrożona przez hipoglikemię.* (Eng. 'The brain was threatened by hypoglycemia').

***obj_th* (thematically restricted object)**

A nominal *obj_th* argument is subcategorised by a sentence predicate. The *obj_th* argument may be marked for different cases, e.g., dative and instrumental in Figure 3.18, except for locative which is reserved for nominal complements of prepositions. The *obj_th* function is thematically restricted, i.e., it is restricted to a particular thematic or semantic role, as indicated by the *th* symbol. Thematically restricted objects bear various semantic roles in Polish, e.g., Location, Instrument, Goal, Recipient or Experiencer. For example, the token *Ani* in the sentence *Dał Ani kwiaty.* (Eng. 'He gave Ania some flowers.') fulfils the *obj_Recipient* function, which is a member of the group of thematically restricted *obj_th* functions with the semantic role *Recipient*.

Some verbs subcategorise multiple thematically restricted objects which differ in their semantic roles. Hence, a particular thematically restricted object (e.g., *obj_Recipient*) can occur only once as an argument of a predicate. In Figure 3.18, there are two nouns fulfilling the *obj_th* function. The first one *rodzicom* bears the semantic role of Recipient/Experiencer, and the second one *piątką* bears the Theme role. However, it should be noted that thematically restricted objects are distinguished by their thematic roles only theoretically. In practice, all relations in which dependents function as thematically restricted objects are labelled with the *obj_th* function. Therefore, two *obj_th* arguments of the predicate *pochwalił* are displayed in Figure 3.18.

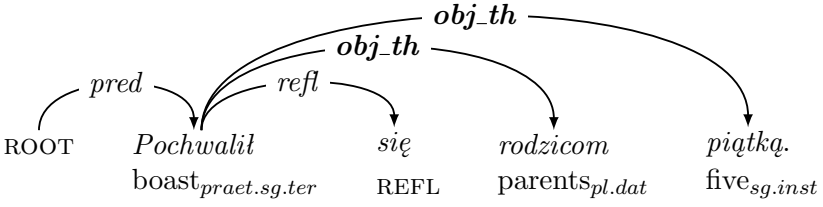


FIGURE 3.18: A dependency structure of the sentence *Pochwalił się rodzicom piątką.* (Eng. 'He boasted about the grade 5 to his parents.').

Morphological criteria are insufficient to distinguish thematically restricted objects *obj_th* from direct objects *obj*. Instead, an alternation criterion is applied – a thematically restricted object cannot be promoted to the subject during passivisation. Furthermore,

obj_th may be realised optionally in some cases, even if it is not required by the sentence predicate, e.g., *opowiadać* (*dzieciom*) *bajkę* (Eng. 'to tell (the children) a story').

***pd* (predicative complement)**

Any element (verbal or small clause, adjective phrase, noun phrase, etc.) in the predicative position in a sentence is annotated with the predicative complement function *pd*. The *pd* argument may be governed by a form of the copula verb *być* (Eng. 'to be') (see Figure 3.19) or copula-like verbs, e.g., *stać się* (Eng. 'to become').

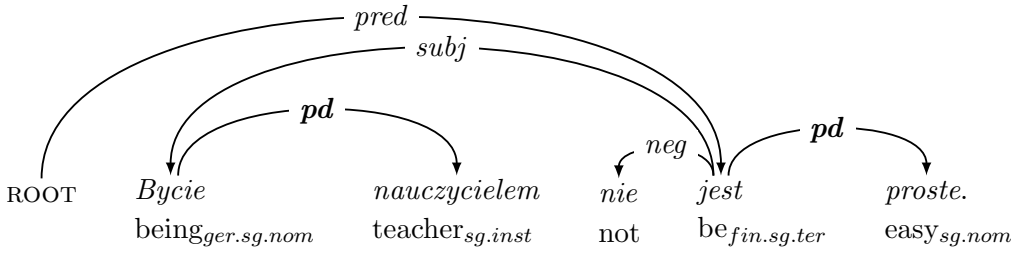


FIGURE 3.19: A dependency structure of the sentence *Bycie nauczycielem nie jest proste.* (Eng. 'It is not easy to be a teacher.').

***subj* (subject)**

The *subj* argument is subcategorised by the sentence predicate. If *subj* takes the form of a nominative phrase, it must morphologically agree with the predicate in person, number and gender. If it takes the form of a phrase marked for case other than nominative, the predicate is represented as a 3rd person singular verb form marked for the neuter gender. The *subj* function may also be fulfilled by a sentential clause or an infinitival phrase. Furthermore, the Polish subject can take the form of an elliptic pro-drop pronoun *pro*. The pronoun *pro* is not encoded in a dependency structure since the dependency structure consists of nodes that correspond to tokens of a sentence and additional nodes except for the ROOT node are not allowed. In contrast to other complements governed by a predicate, *subj* is responsible for binding anaphoric expressions (reflexive and reciprocal pronouns) and controls adverbial participles in Polish.

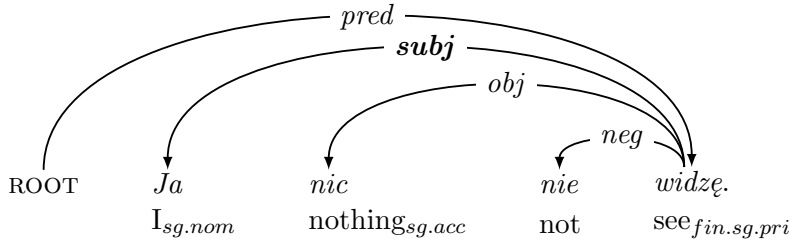


FIGURE 3.20: A dependency structure of the sentence *Ja nic nie widzę.* (Eng. 'I do not see anything.')

3.2.2 Syntactically Motivated Non-arguments

***abbrev_punct* (abbreviation marker)**
A full stop with the *abbrev_punct* function depends on the immediately preceding abbreviation (see Figure 3.21). However, if a sentence ends with an abbreviation marker, this marker is annotated with the *punct* function.

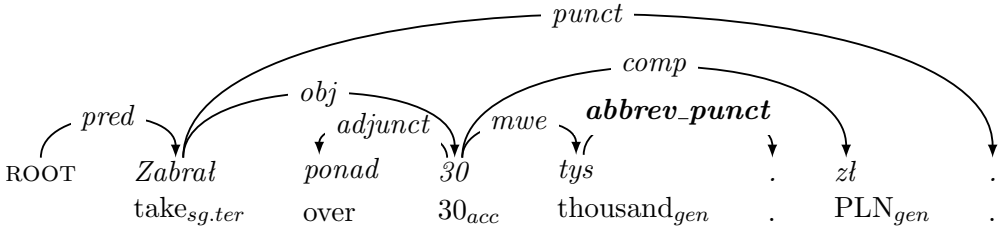


FIGURE 3.21: A dependency structure of the sentence *Zabrał ponad 30 tys. zł.* (Eng. 'He took more than PLN 30,000.')

adjunct
An adjunct is a non-subcategorised dependent with a modifying function. Multiple adjuncts may depend on the same governor. Adjuncts have various surface realisations:

- an adjective depending on a numeral or a noun (e.g., ***Radosne*** *przedszkolaki* in Figure 3.22),
- an adverb depending on an adjective, another adverb, a preposition or a verb form (e.g., ***głośno*** *śpiewając* in Figure 3.22),
- a past or present adverbial participle depending on a verb form (e.g., *szły* ***śpiewając*** in Figure 3.22),

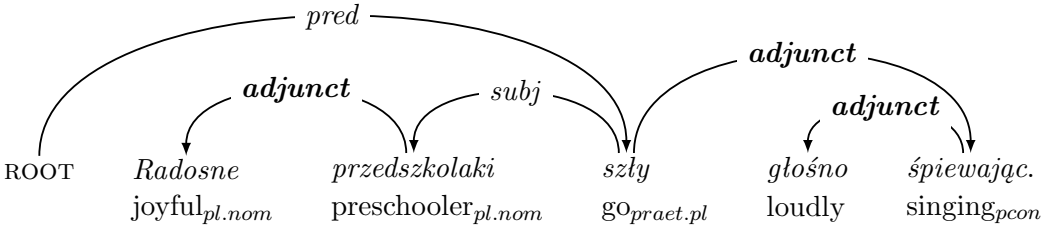


FIGURE 3.22: A dependency structure of the sentence *Radosne przedszkolaki szły głośno śpiewając.* (Eng. 'Some joyful preschoolers went singing loudly.').

- an attributive noun which is marked for genitive and depends on a noun (e.g., *dziele* ***sztuki*** in Figure 3.23),

- a noun with a temporal, locative, etc., meaning that depends on a verb (e.g., *Myślał* ***godzinami*** in Figure 3.23),
- an active or passive adjectival participle depending on a noun (e.g., ***znienawidzonym*** *dziele sztuki* in Figure 3.23),

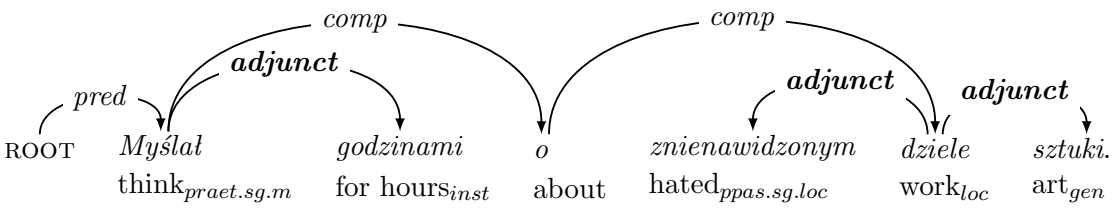


FIGURE 3.23: A dependency structure of the sentence *Myślał godzinami o znienawidzonym dziele sztuki.* (Eng. 'He thought about the hated artwork for hours.').

- a symbol (e.g., number, letter) depending on a noun (e.g., *punkt* ***5*** in Figure 3.24),
- a prepositional phrase depending on a noun, a verb, an adverb or a participle (e.g., *punkt* ***o zabytkach*** in Figure 3.24),
- the question particle (*czy*) depending on a verb (e.g., ***Czy*** *znasz* in Figure 3.24),

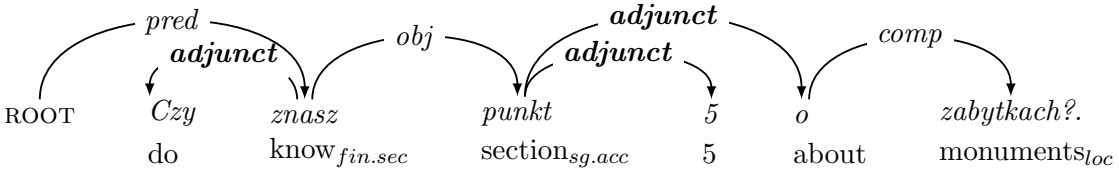


FIGURE 3.24: A dependency structure of the sentence *Czy znasz punkt 5 o zabytkach?* (Eng. 'Do you know the section 5 about monuments?').

- a conditional subordinate clause that depends on the sentence predicate of a superordinate clause (see Figure 3.25),

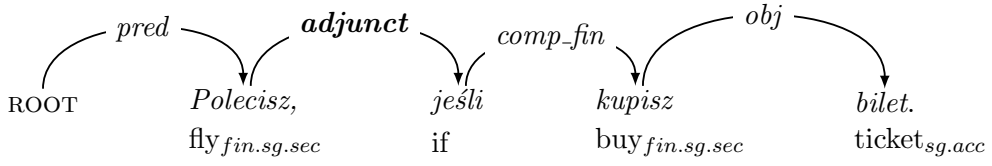


FIGURE 3.25: A dependency structure of the sentence *Polecisz, jeśli kupisz bilet.* (Eng. 'You will fly if you buy a ticket.').

- a subordinate clause with a governor represented by a verb form (see Figure 3.26), a numeral or a noun (see Figure 3.27),[5] etc.

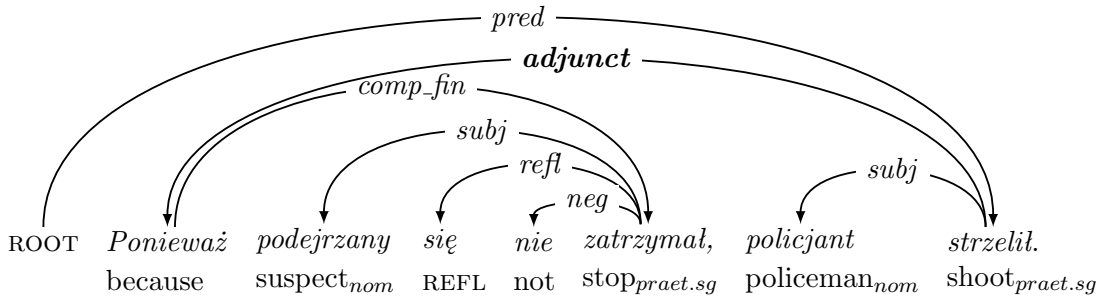


FIGURE 3.26: A dependency structure of the sentence *Ponieważ podejrzany się nie zatrzynał, policjant strzelił.* (Eng. 'Since the suspect didn't stop, the policeman shot.').

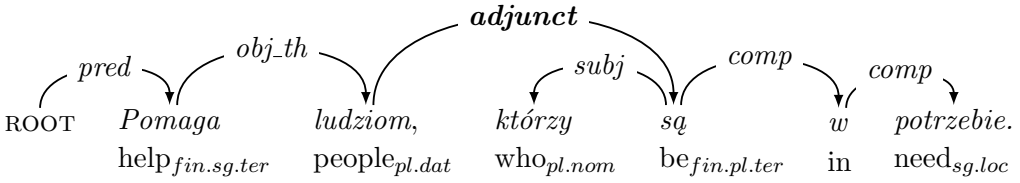


FIGURE 3.27: A dependency structure of the sentence *Pomaga ludziom, którzy są w potrzebie.* (Eng. 'He helps people who are in need.').

adjunct_qt (quotation adjunct)

Direct speech sentences are annotated as quotation adjuncts *adjunct_qt* (see Figure 3.28).

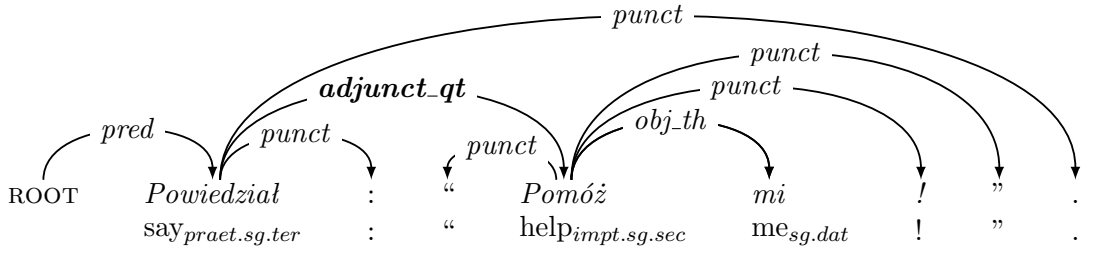


FIGURE 3.28: A dependency structure of the sentence *Powiedział: "Pomóż mi!"* (Eng. 'He said: "Help me!"').

[5]Relative clauses modifying noun phrases are analysed similarly in Obrębski (2002).

***app* (apposition)**

An apposition must refer to the same entity as its governor. In Polish, the *app* function is most commonly fulfilled by a noun phrase depending on the immediately preceding noun (see Figure 3.29) or as the second noun of a noun-noun compound that depends on the first one (e.g., *strażak-ratownik*, Eng. 'fireman-rescuer').

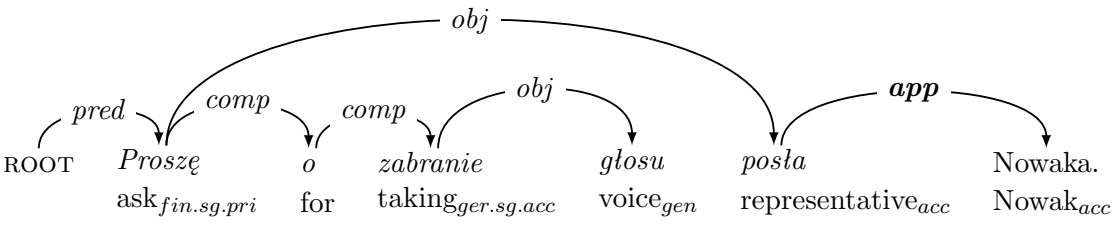


FIGURE 3.29: A dependency structure of the sentence *Proszę o zabranie głosu posła Nowaka.* (Eng. 'May I ask the representative Nowak for taking the floor.').

***complm* (complementiser)**

A complementiser, e.g., *że, iż* (Eng. 'that'), *żeby, aby, by* (Eng. 'so as to') fulfils the *complm* function. A complementiser introducing a complement clause depends on the predicate of this clause (see Figure 3.30). In some contexts, a complementiser may be realised optionally (see Figure 3.12 on p. 30).

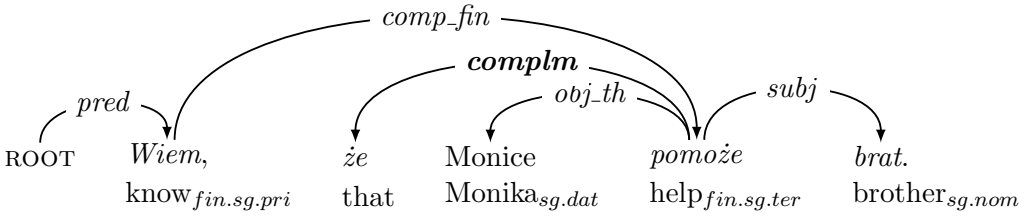


FIGURE 3.30: A dependency structure of the sentence *Wiem, że Monice pomoże brat.* (Eng. 'I know that the brother will help Monika.').

***imp* (imperative marker)**

The particles *niech*, *niechaj*, *niechże* or *niechajże* (Eng. 'let, may') fulfil the *imp* function. They depend either on a future verb form in the case of perfective verbs (e.g., *niech zaśpiewa*, Eng. 'let him sing$_{perf}$') and the verb *być* (Eng. 'to be'), or on a present verb form in the case of imperfective verbs (e.g., *niech śpiewa*, Eng. 'let him sing$_{imperf}$'). An example of the analytical imperative construction is given in Figure 3.31.

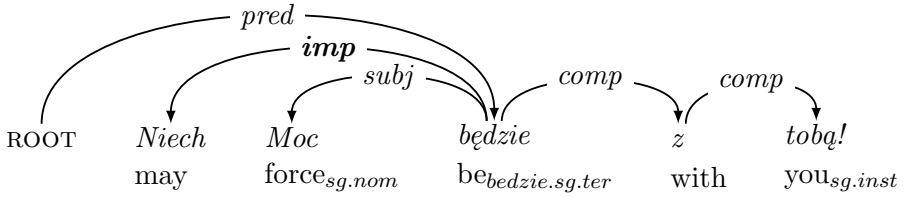


FIGURE 3.31: A dependency structure of the sentence *Niech Moc będzie z tobą!* (Eng. 'May the force be with you!').

***item* (enumeration marker)**

Symbols (e.g., numbers, letters, shapes or bars), which appear at the beginning of a sentence or a phrase, mark individual items on a list. Various enumeration markers depend on the predicate and are labelled with the *item* function (see Figure 3.32).

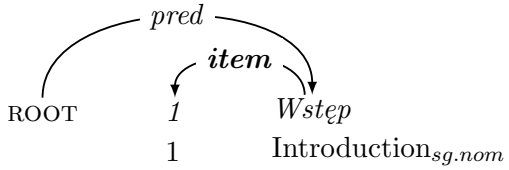


FIGURE 3.32: A dependency structure of the item of a content list *1 Wstęp* (Eng. '1 Introduction').

***pred* (sentence predicate or nominal predicate)**

The *pred* function is usually fulfilled by a sentence predicate realised as a finite verb, a *-no/-to*-impersonal, a quasi-verb, an infinitive, a passive adjective participle or an imperative. In the case of non-sentential segments, the *pred* function may be borne by the head noun of a noun phrase or almost every constituent in other phrase types. A token with the *pred* function always depends on the ROOT node (see Figure 3.33). A sentence predicate governs a restricted number of arguments and a theoretically unlimited number of adjuncts. If a nominal dependent of a sentence predicate is marked for nominative and fulfils the *subj* function, it agrees with the sentence predicate in person, number and gender.

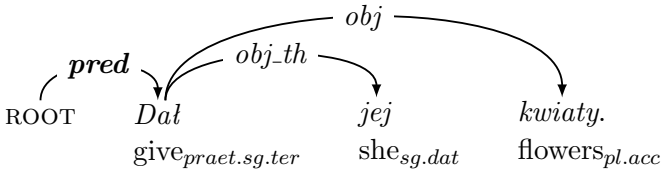


FIGURE 3.33: A dependency structure of the sentence *Dał jej kwiaty.* (Eng. 'He gave her some flowers.').

***punct* (punctuation mark)**

The *punct* type represented by a punctuation mark (e.g., *.,:;?!()"*-) depends on the element which is delimited by this punctuation mark (see Figures 3.21 on p. 34, 3.28 on p. 36, and 3.34).

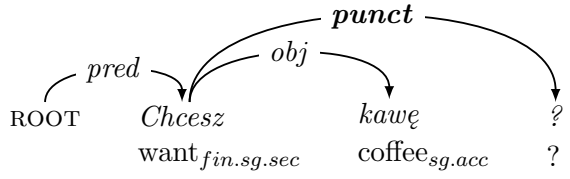


FIGURE 3.34: A dependency structure of the sentence *Chcesz kawę?* (Eng. 'Do you want a cup of coffee?').

***refl* (reflexive marker)**

The particle *się* fulfils the *refl* function in Polish. The reflexive marker depends on a verb with the reflexive meaning (see Figure 3.35) or with the reciprocal meaning (see Figure 3.36), on a verb in impersonal constructions (see Figure 3.37), etc.

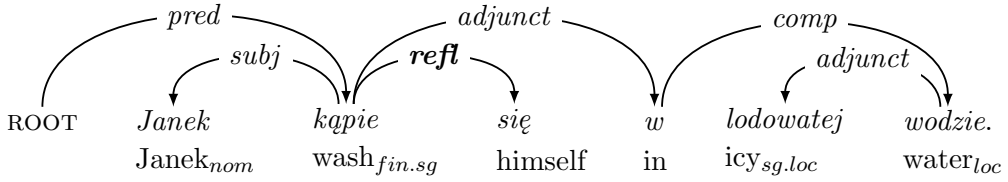


FIGURE 3.35: A dependency structure of the sentence *Janek kąpie się w lodowatej wodzie.* (Eng. 'Janek washes himself in the icy water.').

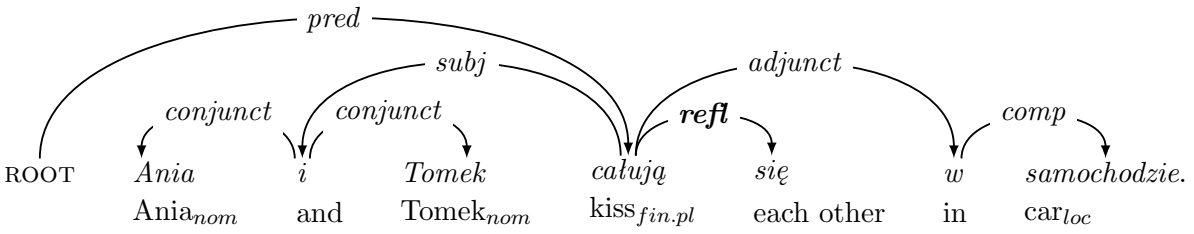


FIGURE 3.36: A dependency structure of the sentence *Ania i Tomek całują się w samochodzie.* (Eng. 'Ania and Tomek are kissing in a car.').

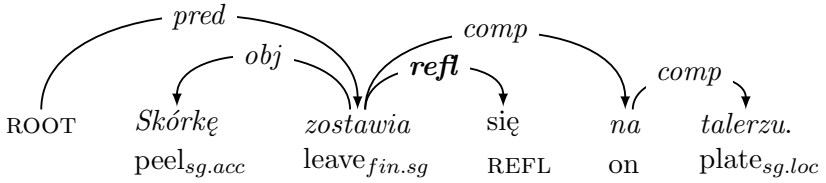


FIGURE 3.37: A dependency structure of the sentence *Skórkę zostawia się na talerzu.* (Eng. 'Peel should be left on the plate.')

3.2.3 Morphologically Motivated Non-arguments

***aglt* (mobile inflection)**

A mobile affix, the so-called mobile inflection, fulfils the *aglt* function. The mobile inflection is an agglutinate/clitic form marked for number and person. It is a characteristic feature of the mobile inflection that it may appear in different positions within a clause. The mobile inflection is predominantly appended to a verb (e.g., *Wygrali**śmy***., Eng. 'We won.'). It may also be appended to an adverb (e.g., *Długo**śmy** czekali.*, Eng. 'We have been waiting for a long time.'), a complementiser (e.g., *Chcę, żeby**śmy** wygrali.*, Eng. 'I want us to win.'), a pronoun (e.g., *My**śmy** to słyszeli.*, Eng. 'We heard it.'), etc., which

precede the verb that is grammatically complemented by the mobile inflection. The mobile inflection syntactically depends on a finite verb (see Figure 3.51 on p. 46) or a conditional clitic *by* appended to a verb (see Figure 3.38).

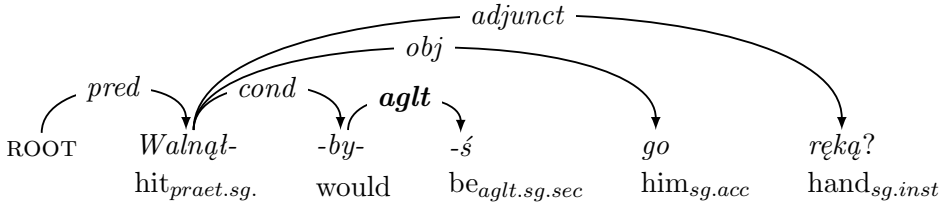


FIGURE 3.38: A dependency structure of the sentence *Walnąłbyś go ręką?* (Eng. 'Would you hit him with your hand?').

***aux* (auxiliary verb)**

The *aux* function is realised as a conjugated form of the auxiliary verbs *być* or *zostać* (Eng. 'to be', 'to become') which bear morphosyntactic features. As the annotation schema is semantically oriented, an auxiliary verb depends on the main verb form (participle, infinitive) in analytical future tense[6] constructions (see Figure 3.39), analytical past conditional[7] constructions (see Figure 3.40), analytical quasi-verb constructions (see Figure 3.41) or passive constructions (see Figure 3.42).

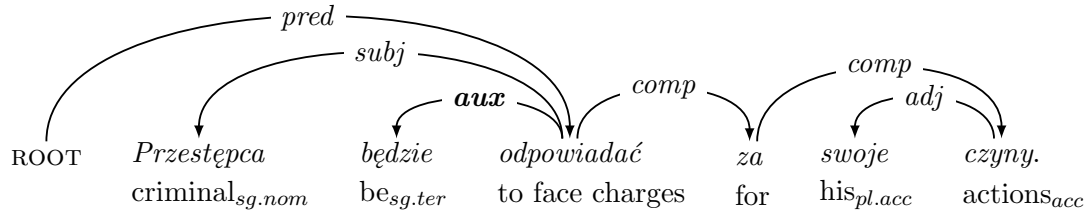


FIGURE 3.39: A dependency structure of the sentence *Przestępca będzie odpowiadać za swoje czyny.* (Eng. 'The criminal will face charges for his actions.').

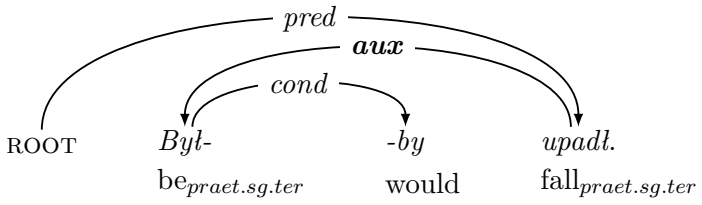


FIGURE 3.40: A dependency structure of the sentence *Byłby upadł.* (Eng. 'He would have fallen.').

[6]The future tense form of an imperfect verb is built of the conjugated future form of the auxiliary verb *być* (Eng. 'to be') combined either with a past participle (e.g., *Będzie śpiewał.*, Eng. 'He will sing.') or with an infinitive (e.g., *Będzie śpiewać.*, Eng. 'He will sing.') of the main verb.

[7]The past conditional verb form is built of the conditional form of the auxiliary verb *być* combined with the past participle of the main verb (e.g., *Byłabym upadła.*, Eng. 'I would have collapsed.').

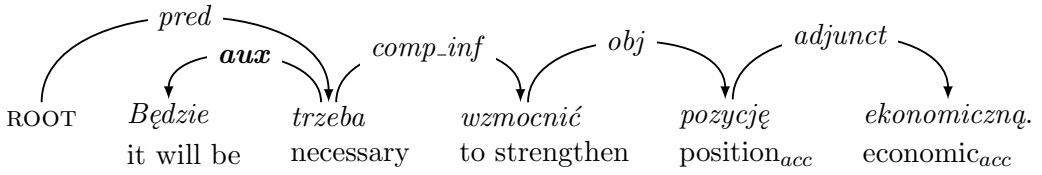


FIGURE 3.41: A dependency structure of the sentence *Trzeba będzie wzmocnić pozycję ekonomiczną.* (Eng. 'It is necessary to strengthen the economic position.').

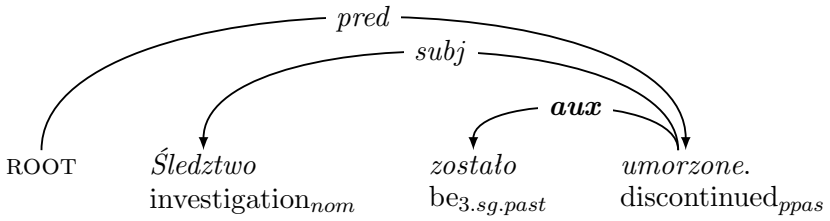


FIGURE 3.42: A dependency structure of the sentence *Śledztwo zostało umorzone.* (Eng. 'The investigation was discontinued.').

***cond* (conditional clitic)**

The particle *by* bearing the *cond* function underlines the conditional modality of a sentence. The conditional particle may depend on a verb form in the past tense and third person singular or plural (e.g., *śpiewałby*, Eng. 'he would sing', *zaśpiewaliby*, Eng. 'they would sing', *byłby śpiewał*, Eng. 'he would have sung'), on an impersonal verb form (e.g., *śpiewano by*, Eng. 'it would be sung') or on a quasi-verb (e.g., *można by śpiewać*, Eng. 'it could be sung'). The conditional particle may be appended to its governing verb form as an enclitic or may appear anywhere in a sentence, subject to various island constraints (see Figure 3.43). Regardless of its location, the conditional particle depends on a verb form.

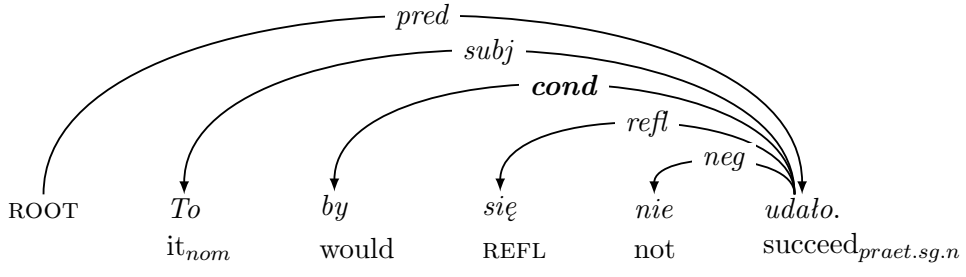


FIGURE 3.43: A dependency structure of the sentence *To by się nie udało* (Eng. 'It would not have succeeded.').

***neg* (negation marker)**
The negation marker *nie* (Eng. 'not') always fulfils the *neg* function. It is predominantly governed by a subsequent verb, but it may also be governed by other constituents (see ***nie*** *Ola* in Figure 3.44).

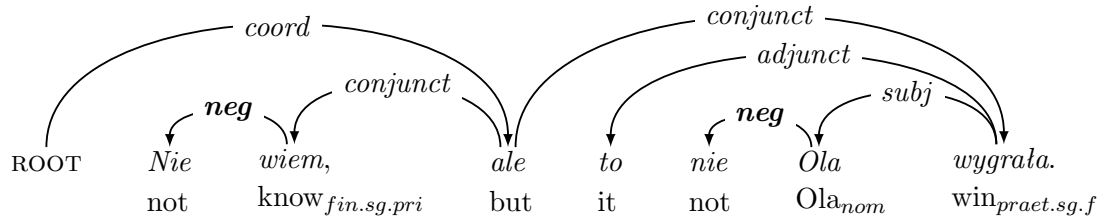


FIGURE 3.44: A dependency structure of the sentence *Nie wiem, ale to nie Ola wygrała.* (Eng. 'I don't know, but it is not Ola who won.').

3.2.4 Semantically Motivated Non-arguments

***mwe* (multiword expression)**
Relations between successive tokens of a multiword expression are annotated with the *mwe* function. The first token of a multiword expression governs the second token which is, in turn, the governor of the next token, etc. The following token combinations are annotated as multiword expressions:

- some prepositional-adjectival phrases, e.g., *po prostu* (Eng. 'simply'), *co gorsza* (Eng. 'what is worse'),
- some prepositional-adverbial phrases, e.g., *na zewnątrz* (Eng. 'outside'), *co najmniej* (Eng. 'at least'),
- some adverbial-prepositional phrases,[8] e.g., *wraz z* (Eng. 'along with'), *zgodnie z* (Eng. 'in accordance with') (see Figure 3.45),
- complex conjunctions, e.g., *jak i* (Eng. 'and'), *ale także* (Eng. 'but also'), *mimo że* (Eng. 'although'), *podczas gdy* (Eng. 'whereas'),
- adjective compounds, e.g., *biało-czerwona* (Eng. 'white-red'),
- compound numerals, e.g., *tysiąc dwieście trzydzieści cztery* (Eng. 'one thousand two hundred thirty-four'), etc.

[8]Several combinations of adverbs and prepositions are regarded by Milewska (2003) as complex prepositions (Pol. 'przyimki wtórne').

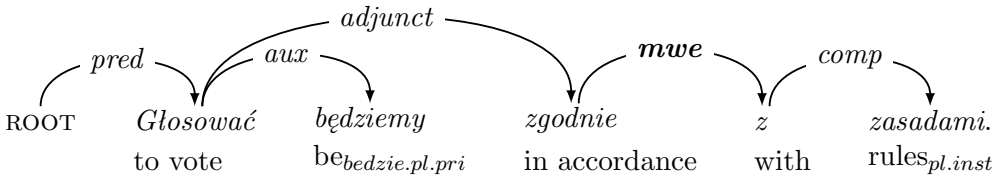


FIGURE 3.45: A dependency structure of the sentence *Głosować będziemy zgodnie z zasadami.* (Eng. 'We will vote in accordance with the rules.').

***ne* (named entity)**

Named entities are phrases that indicate expressions of time and names of persons, locations, and organisations. The limitation of named entities to these types is motivated by the hierarchy of named entity types defined for Polish (Savary et al., 2012). Since named entities may contain more than one token, i.e., they form a special kind of multiword expressions, relations between singular tokens of named entities are annotated with the *ne* function. The successive tokens of a named entity are typically annotated according to their linear order, e.g., the surname *Nowaka* depends on the forename *Piotra* in Figure 3.46.

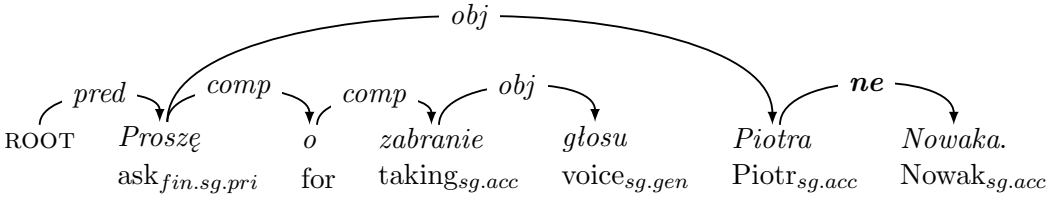


FIGURE 3.46: A dependency structure of the sentence *Proszę o zabranie głosu Piotra Nowaka.* (Eng. 'May I ask Piotr Nowak to take the floor.').

3.2.5 Functions Used in Coordination Structure

There are various ways of representing coordination in dependency structures. Popel et al. (2013) outline three main styles of annotating coordination structures in dependency trees – Stanford style, Mel'čuk style and Prague style (see Figure 3.47). All dependency treebanks adapt one of these styles or mix them to annotate coordinating constructions.

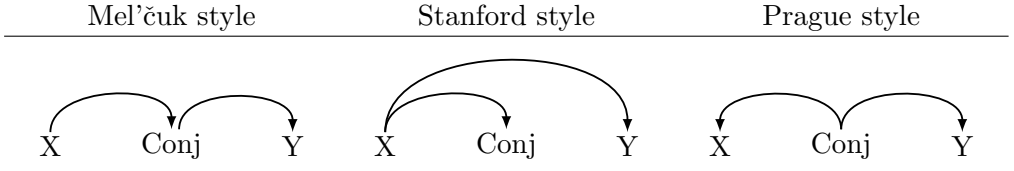


FIGURE 3.47: Annotation of coordinating constructions in various dependency annotation schemata (taken from Täckström, 2013, p. 122).

In the Stanford style (de Marneffe and Manning, 2008b,a), the leftmost conjunct is treated as the head of coordinating conjunctions and other conjuncts, i.e., the first conjunct governs remaining conjuncts and conjunctions. This annotation solution seems to facilitate the parsing procedure, but it is inappropriate for representing some Polish phenomena, e.g., argument sharing[9] or grammatical agreement.[10] In the Mel'čuk style (Mel'čuk, 1988), elements of coordinating constructions (conjuncts and conjunctions) are annotated linearly, i.e., the first coordinated conjunct is the governor of the entire coordinating construction. Even if this annotation proposal seems to be applicable for Polish, we apply the style of annotating coordinating constructions proposed by the designers of the *Prague Dependency Treebank* (Böhmová et al., 2003). In the Prague style, all conjuncts are annotated as dependents of the coordinating conjunction. Popel et al. (2013) provide a number of good arguments for using the Prague style to annotate coordinating constructions, e.g., a relatively simple annotation of modifiers or arguments shared by all coordinated conjuncts.

The current schema determines the rightmost conjunction (or a punctuation mark that substitutes a coordinating conjunction if the conjunction is absent) as the governor of coordinated conjuncts. Possible arguments or adjuncts shared by all conjuncts are annotated as dependents of the coordinating conjunction.

***conjunct* (coordinated conjunct)**

Conjuncts of the same syntactic category or various categories depend on a coordinating conjunction (see Figures 3.48 and 3.49) or a conjunction-like punctuation mark (see Figure 3.51) in dependency structures.

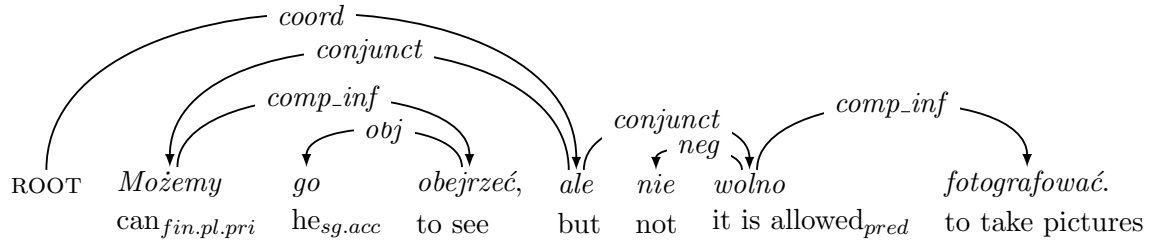


FIGURE 3.48: A dependency structure of the sentence *Możemy go obejrzeć, ale nie wolno fotografować.* (Eng. 'We can see it, but it is not allowed to take pictures.').

[9] In argument sharing, two or more coordinated verbs share the same argument. In the example tree in Figure 3.49, the coordinated sentence predicates *złapał* (Eng. 'grab'$_{praet.sg.m}$) and *ścisnął* (Eng. 'press'$_{praet.sg.m}$) require a subject argument and an object argument. In this sentence, only one token *Chłopak* (Eng. 'a boy') is the subject (Agens) that initiates the two actions – grabbing and pressing. Furthermore, one token *posążek* (Eng. 'a statuette') is the object (Patient) that is grabbed and pressed. Hence, both arguments should be shared by the coordinated sentence predicate.

[10] In an example of grammatical agreement, the attributive adjective *piękni* in the coordinating construction *piękni kobiety i mężczyźni* (Eng. 'beautiful$_{pl.m}$ women$_{pl.f}$ and men$_{pl.m}$') modifies the entire phrase *kobiety i mężczyźni*. It may not depend on the noun *kobiety* because of divergent grammatical gender features. Polish nouns typically agree with their adjectival modifiers in case, number and gender. In coordinating constructions, the masculine human gender (Pol. 'rodzaj męski osobowy') always overrides gender features of other conjuncts.

***coord* (coordinating conjunction)**

A conjunction coordinating two sentences (see Figure 3.48) depends on the ROOT node and the relation is labelled with the *coord* function. Relations between the conjunction and the predicates of coordinated sentences are labelled with the *conjunct* function. However, if two or more predicates sharing the same subject are coordinated on the sentential level, the relation between the ROOT node and the coordinating conjunction is labelled with the *pred* function. Furthermore, coordinated predicates are annotated as dependents of the conjunction labelled with the *conjunct* function and the shared subject (and possibly other shared arguments) is also annotated as the dependent of the coordinating conjunction (see Figure 3.49). The conjunction coordinating elements other than sentences/clauses, e.g., nouns or adjectives, is labelled with an appropriate syntactic function (see *comp* in Figure 3.52).

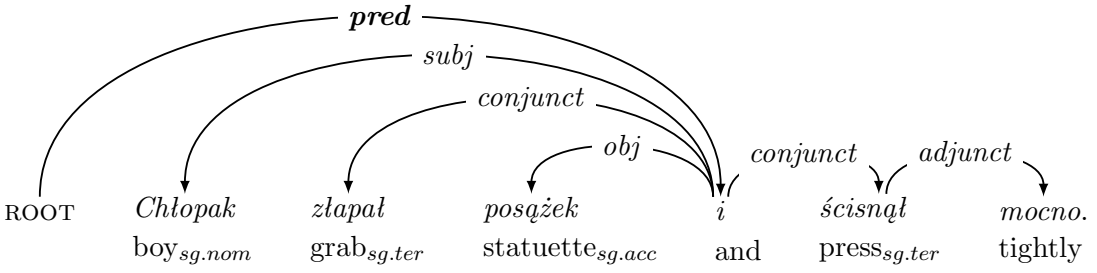


FIGURE 3.49: A dependency structure of the sentence *Chłopak złapał posążek i ścisnął mocno.* (Eng. 'The boy grabbed a statuette and pressed it tightly.').

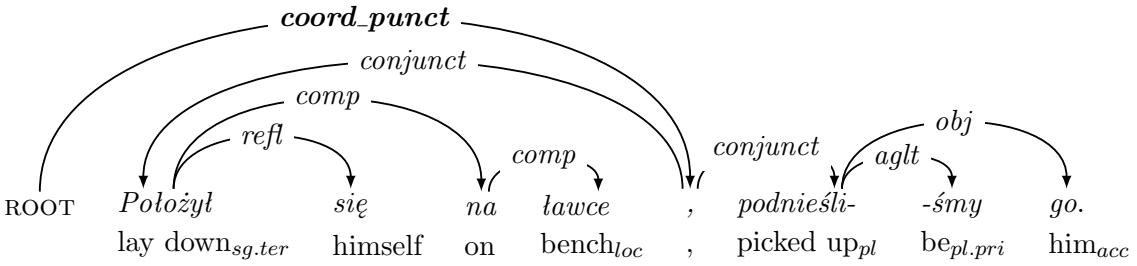


FIGURE 3.50: A dependency structure of the sentence *Położył się na ławce, podnieśliśmy go.* (Eng. 'He lay down on the bench, we picked him up.')

***coord_punct* (punctuation conjunction)**

If no coordinating conjunction is used to coordinate two sentences, the punctuation mark (e.g., comma, colon or hyphen) is used as a coordinating element (see Figure 3.50). The punctuation mark with the coordinating status depends on the ROOT node. Such relation is labelled with the *coord_punct* function. However, if there are two coordinated predicates sharing the same subject (possibly a pro-drop pronoun), the relation between

the ROOT node and the coordinating punctuation mark is labelled with the *pred* function (see Figure 3.51).[11]

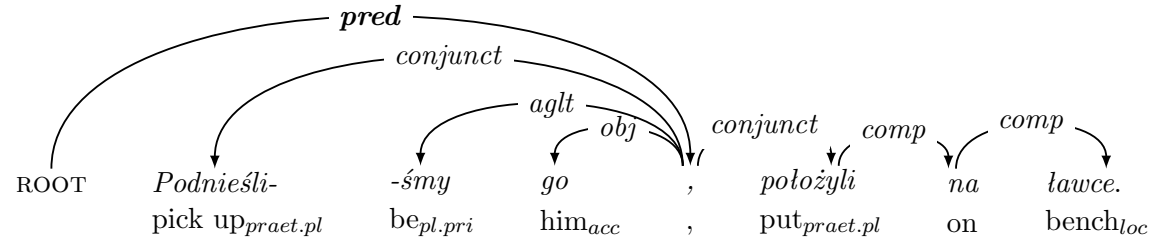


FIGURE 3.51: A dependency structure of the sentence *Podnieśliśmy go, położyli na ławce.* (Eng. 'We picked him up, we put him on the bench.')

***pre_coord* (pre-conjunction)**

Some two-part correlative conjunctions are possible in Polish, e.g., *albo... albo...* (Eng. 'either... or...'), *ani... ani...* (Eng. 'neither... nor...'). The first part of the correlative conjunction depends on the second one. If the correlative conjunction contains more than two parts, e.g., *albo... albo... albo*, all preceding conjunction parts depend on the last one. The function *pre_coord* is used to label this relation (see Figure 3.52).

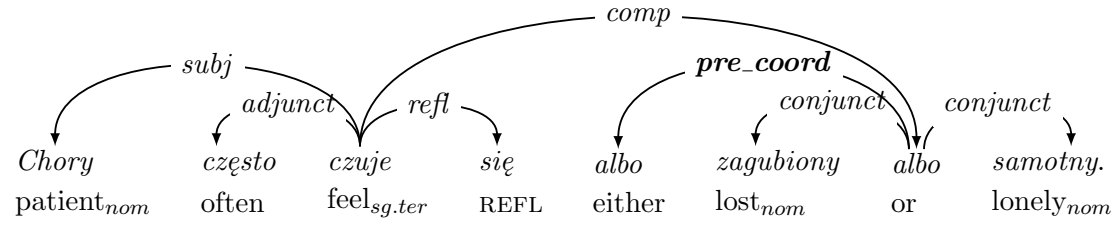
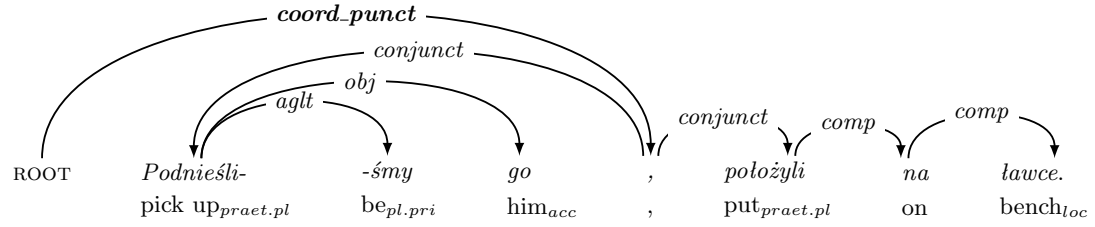


FIGURE 3.52: A dependency structure of the sentence *Pacjent czuje się albo zagubiony albo samotny.* (Eng. 'The patient often feels either lost or lonely.').

[11] An alternative dependency tree for the example sentence *Podnieśliśmy go, położyli na ławce.* may look like the tree below:



This analysis indicates that there is a coordination of two independent sentences. However, from the semantic point of view, there is only one Agent represented by the group of people who pick somebody up and put him on a bench. Hence, there is only one subject which is represented by a pro-drop pronoun 'we' and this subject is shared by both coordinated predicates. Furthermore, the mobile inflection *-śmy* should also be syntactically shared by both coordinated sentence predicates and thus governed by the coordinating punctuation mark, so that the forms of the predicates agree. Similarly, Patient of these two actions (picking up and putting down), which is represented by the direct object *go*, should also be shared by both predicates.

3.3 Syntactic Annotation Schemata: Related Work

The number of syntactically annotated corpora (treebanks) which are available for various languages is growing constantly. This process is accompanied by designing and developing different constituency- or dependency-based annotation schemata. The growing significance of dependency approaches has affected constituency treebanks developed in recent years, e.g., phrase structure trees in some treebanks have been extended with grammatical functions (e.g., the English *Penn Treebank II*, Marcus et al., 1994; Bies et al., 1995). Another idea consists in integration of constituent and dependency information in a graph whose nodes and edges represent phrasal categories and grammatical functions respectively (e.g., the German *TIGER Treebank*, Brants et al., 2002) or whose nodes incorporate phrasal and dependency information (e.g., the Polish treebank – *Składnica*, Świdziński and Woliński, 2010; Woliński et al., 2011).

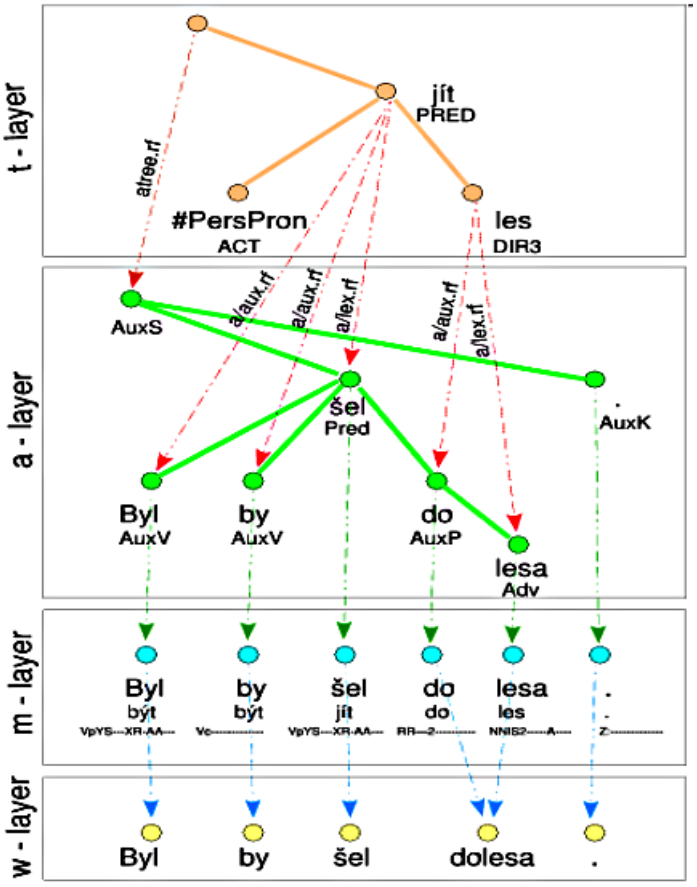


FIGURE 3.53: PDT annotation of the Czech sentence *Byl by šel dolesa.* (Eng. 'He would go into the forest.') found in the PDF presentation by Eva Hajičová publicly available on `http://www.lumii.lv/ngslt/hajicova/Riga-1-definitivni.pdf`.

Apart from constituency treebanks extended with dependency information, there are also some treebanks containing 'pure' dependency structures. The leading example of a dependency annotated corpus is the *Prague Dependency Treebank* (PDT, Hajič, 1998;

Böhmová et al., 2003) based on *Functional Generative Description* (Sgall et al., 1986). Treebank sentences are randomly selected from the Czech National Corpus.[12] In the PDT schema, there are three annotation levels superstructured on each other: morphological layer (m-layer), analytical layer (a-layer) and tectogrammatical layer (t-layer) (see Figure 3.53). The morphological layer of an annotated sentence consists of lemmata and morphosyntactic tags with which the subsequent tokens are supplemented. The analytical layer corresponds to a labelled dependency structure which is represented as a connected acyclic directed graph. The analytical layer encodes 24 grammatical functions (e.g., *Sb* – subject, *Obj* – object, *Atr* – attribute in noun phrases, *AtvV* – verbal attribute/complement). In comparison to nodes of the a-layer which correspond to all morphosyntactically annotated segments (tokens) of the m-layer, nodes of the tectogrammatical layer represent only semantically essential/full (lexical) items including those which are elided in the surface representation (w-layer). The core of the tectogrammatical representation is a dependency tree rooted at the predicate governing some arguments (e.g., Actor, Objective, Addressee, Origin or Effect) or adjuncts (e.g., of location, direction, time, cause or manner). Apart from the predicate-argument structure, the tectogrammatical layer encodes semantic roles (functors), topic-focus information, named entities, coreference relations, etc.

While designing the Polish dependency annotation schema, we took into consideration annotation solutions proposed in the PDT schema. Polish and Czech belong to the group of West Slavic languages and similar syntactic constructions can be found in both languages. Therefore, the annotation of many Polish relations (e.g., subject, object) and syntactic constructions (e.g., coordination, complex verb forms) is consistent with the PDT annotation proposals, even if our labelling nomenclature differs from the PDT labels.

The definition of the Polish schema has also been influenced by the *PARC 700 Dependency Bank* (DEPBANK, King et al., 2003). The English DEPBANK consists of 700 dependency structures automatically converted from LFG-parsed sentences[13] and manually corrected and extended by human validators. Fully connected dependency structures are annotated as sets of relation triples of the form ⟨`relation type`⟩(⟨`indexed governor`⟩, ⟨`governor's value or indexed governor's dependent`⟩). Each token of a sentence has an assigned index. The top-level predicate is assigned the index 0. There are two kinds of relation triples. First, predicate-argument/adjunct relations are established between lemma forms of a governor and its dependent and are labelled with grammatical functions. For example, the subject relation connects 'replace' with 'device' (`subj(replace~0, device~1)`) as shown in Figure 3.54. Second, grammatical features (e.g., number, case, passive) of individual tokens are also encoded as relation triples. In the DEPBANK example in Figure 3.54, triples `tense(replace~0, past)` and

[12] The Czech National Corpus is published on `http://www.korpus.cz`.

[13] Sentences were parsed with a broad-coverage English LFG using the XLE system (Maxwell III and Kaplan, 1993).

`passive(replace∼0, +)` indicate a past-tensed and passive-voiced surface realisation of 'replace'.

```
sentence(
   id(wsj_2356.19, parc_23.34)
   date(2002.6.12)
   validators(T.H. King, J.-P. Marcotte)
sentence_form(The device was replaced.)
structure(
   mood(replace∼0, indicative)
   tense(replace∼0, past)
   passive(replace∼0, +)
   stmt_type(replace∼0, declarative)
   subj(replace∼0, device∼1)
   vtype(replace∼0, main)
   det_form(device∼1, the)
   det_type(device∼1, def)
   num(device∼1, sg)
   pers(device∼1, 3)))
```

FIGURE 3.54: DEPBANK annotation of the sentence *The device was replaced.* found on `http://www2.parc.com/isl/groups/nltt/fsbank/triplesdoc.html`.

The Polish dependency annotation schema adapts many grammatical functions employed to label dependency relations in the DEPBANK structures. Some of the labels are directly taken over (e.g., *subj*, *obj*, *obj_th* or *adjunct*) and other are derived from DEPBANK dependency relation types (e.g., *coord* from *coord-form*, *pre_coord* from *precoord-form*, *comp_fin* from *comp*, *comp_inf* from *xcomp*, *comp_ag* from *obl-ag* and *complm* from *comp-form*).

The Stanford dependency schema (de Marneffe and Manning, 2008b,a) represents dependency relations as triples of the form ⟨`relation type`⟩(⟨`indexed governor-word`⟩, ⟨`indexed dependent-word`⟩), similarly as in the DEPBANK format. The Stanford schema assumes that dependencies are established between content words in a sentence. It defines 53 various dependency relations types. Some Stanford dependencies involving certain part of speech tags may be collapsed. In collapsed relations, parts of speech and surface realisations of tokens are propagated to dependency labels. Collapsing and propagation are typically employed for prepositions and coordinating conjunctions in English (see collapsed and propagated dependencies `prep_in(based-7, LA-9)` and `conj_and(makes-11, distributes-13)` in Figure 3.55), but not for other function words (e.g., auxiliary verbs, determiners). An example of a Stanford dependency structure annotation is given in Figure 3.55. The Stanford schema does not affect the designing of the Polish annotation schema. Nevertheless, we mention it to maintain a comprehensive overview of all dominating and widely adapted annotation schemata.

A common feature of the presented dependency annotation schemata is that they tend to be semantically oriented, i.e., they presuppose that content words (e.g., main verbs, nouns) dominate function words (e.g., auxiliary verbs, determiners). This view may be motivated by the intended use of dependency annotated treebanks to support semantic

```
nsubj(makes-11, Bell-1)
det(company-4, a-3)
appos(Bell-1, company-4)
nsubjpass(based-7, which-5)
auxpass(based-7, is-6)
rcmod(company-4, based-7)
prep_in(based-7, LA-9)
root(ROOT-0, makes-11)
conj_and(makes-11, distributes-13)
nn(products-15, computer-14)
dobj(makes-11, products-15)
```

FIGURE 3.55: Stanford dependency annotation of the sentence *Bell, a company which is based in LA, makes and distributes computer products.* found in de Marneffe and Manning (2008a).

parsing. The Polish dependency schema continues this annotation trend and defines dependency relations so that they are semantically oriented to some extent.

There are also syntactically oriented annotation schemata in which function words tend to dominate content words. These schemata are less relevant for defining the Polish dependency annotation schema, but we refer to them to keep the overview of schemata as complete as possible. One of the syntactically oriented schemata is the schema designed for the *Danish Dependency Treebank* (DDT, Kromann, 2003) and based on the *Discontinuous Grammar* (Buch-Kromann, 2006). The DDT annotation schema assumes that there are two types of dependency relations between tokens: primary complement and adjunct relations building a fully connected dependency tree (arcs above tokens in Figure 3.56), and secondary relations (arcs below tokens in Figure 3.56). A secondary relation can be established between a non-finite verb form and its subject. A token may have multiple incoming arcs representing secondary relations.

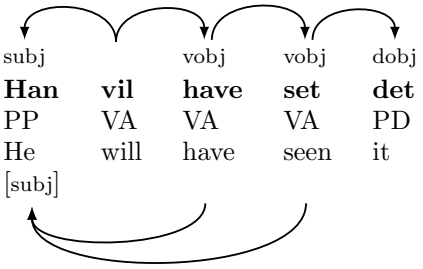


FIGURE 3.56: DDT annotation of a sentence *Han vil have set det* (Eng. 'He will have seen it') found in Kromann (2003).

Recently, interoperable syntactic annotation schemata are created or existing annotation schemata are adapted to annotate corpora in other languages. The Penn Treebank annotation schema has been adapted for Arabic[14] (Maamouri et al., 2011) and Chinese[15] (Xue et al., 2005). The PDT annotation schema has been used to annotate corpora in Arabic[16] (Hajič et al., 2004), Croatian[17] (Berovic et al., 2012), Slovene[18] (Džeroski et al., 2006), Tamil[19] (Ramasamy and Žabokrtský, 2011), and a Czech-English parallel corpus[20] (Hajič et al., 2012). A German dependency bank[21] (Forst et al., 2004) has adapted the DEPBANK triple annotation schema. The Stanford dependency schema has been adapted for Finish[22] (Haverinen et al., 2010). The DDT schema is used in the currently developed *Copenhagen Dependency Treebanks*[23] (CDT, Buch-Kromann and Korzen, 2010; Buch-Kromann et al., 2010) which consists of parallel corpora in Danish, English, German, Italian, and Spanish.

The question arises whether an annotation schema may be really universal and applicable to other languages. Some studies have compared various annotation schemata and measuring their impact on NLP applications (e.g., parsing performance). According to Kübler et al. (2008), whose comparative study concerns annotation schemata originally designed for German, the design of a syntactic annotation schema significantly influences performance of a parser trained on sentences annotated in accordance with this schema. This may suggest that the application of a schema to a new language without any adaptations motivated by the language specificity may decrease parsing performance. However, we are not familiar with studies measuring an impact of an annotation schema applied to a new language on performance of a parser trained on trees annotated in accordance with this schema. Hence, we may not be positive about performance of a parser trained on Polish data annotated according to a schema which is not attuned to the specificity of Polish. Since an annotation schema is typically designed for a particular language, it does not have to take into account all linguistic phenomena occurring in other languages. An annotation schema may thus be insufficient to annotate some Polish-specific linguistic facts, e.g., mobile inflection, and it need to be at least adjusted for Polish. Finally, there is a long tradition of the grammatical description of Polish which motivates some of our annotation solutions making them inconsistent with other annotation schemata. Since we have not found any annotation schema that would perfectly cover all Polish phenomena, this chapter described the new dependency annotation schema conceptualised for Polish.

[14]Penn Arabic Treebank `http://www.ircs.upenn.edu/arabic`.
[15]Penn Chinese Treebank `http://www.cis.upenn.edu/~chinese/ctb.html`.
[16]Prague Arabic Dependency Treebank `ufal.mff.cuni.cz/padt/PADT_1.0/docs/index.html`.
[17]Croatian Dependency Treebank `http://hobs.ffzg.hr/default_en.html`.
[18]Slovene Dependency Treebank `http://nl.ijs.si/sdt/`.
[19]Tamil Dependency Treebank `http://ufal.mff.cuni.cz/~ramasamy/tamiltb/0.1/`.
[20]Prague Czech-English Dependency Treebank `http://ufal.mff.cuni.cz/pcedt/`.
[21]TiGer Dependency Bank `http://www.ims.uni-stuttgart.de/projekte/TIGER/`.
[22]Turku Dependency Treebank `http://bionlp.utu.fi/fintreebank.html`.
[23]Copenhagen Dependency Treebank `code.google.com/p/copenhagen-dependency-treebank`.

Chapter 4

Conversion-based Dependency Bank

This chapter outlines the creation of the Polish dependency bank that contains 8227 dependency structures. The treebank structures are converted from constituent trees and annotated in accordance with the schema described in Chapter 3 *Polish Dependency Annotation Schema*. The conversion is an entirely automatic process. The converter takes manually disambiguated constituent trees encoded in the XML format as input (see Section 4.1 *Składnica – Polish Constituency Treebank*) and outputs dependency structures encoded in the column-based CoNLL format (Buchholz and Marsi, 2006). The conversion process is relatively straightforward since constituents have their syntactic centres marked in most cases (see Section 4.2 *Conversion Procedure*). To convert phrase structures without identified heads, we define some head selection heuristics (see Section 4.3 *Head Selection*). Moreover, some dependency trees directly converted from constituent trees require reorganisation of their arcs in order to make them meet annotation principles (see Section 4.4 *Rearrangement of Dependency Structures*). In the experimental part, the conversion-based dependency treebank is used for training and evaluation of the Polish dependency parser (see Sections 4.5 *Experimental Setup* and 4.6 *Experiments and Results*). An overview of existing constituency-to-dependency conversion approaches (see Section 4.7 *Constituency-to-Dependency Conversion: Related Work*) and a summary of results (see Section 4.8 *Partial Conclusions*) close this chapter.

4.1 *Składnica* – Polish Constituency Treebank

Dependency structures are derived from the Polish constituency treebank *Składnica* (Woliński et al., 2011). Sentences in *Składnica* are selected from the hand-annotated balanced subcorpus of the National Corpus of Polish (NKJP, Przepiórkowski et al., 2012). The selected sentences are semi-automatically annotated with constituent trees.

First, the *Świgra* parser (Woliński, 2004) automatically generates candidate trees for each sentence. Then, the generated parse trees are validated by human annotators, who either select a tree which best conforms to the linguistic constraints of Polish or reject the sentence if no correct tree is generated for it.

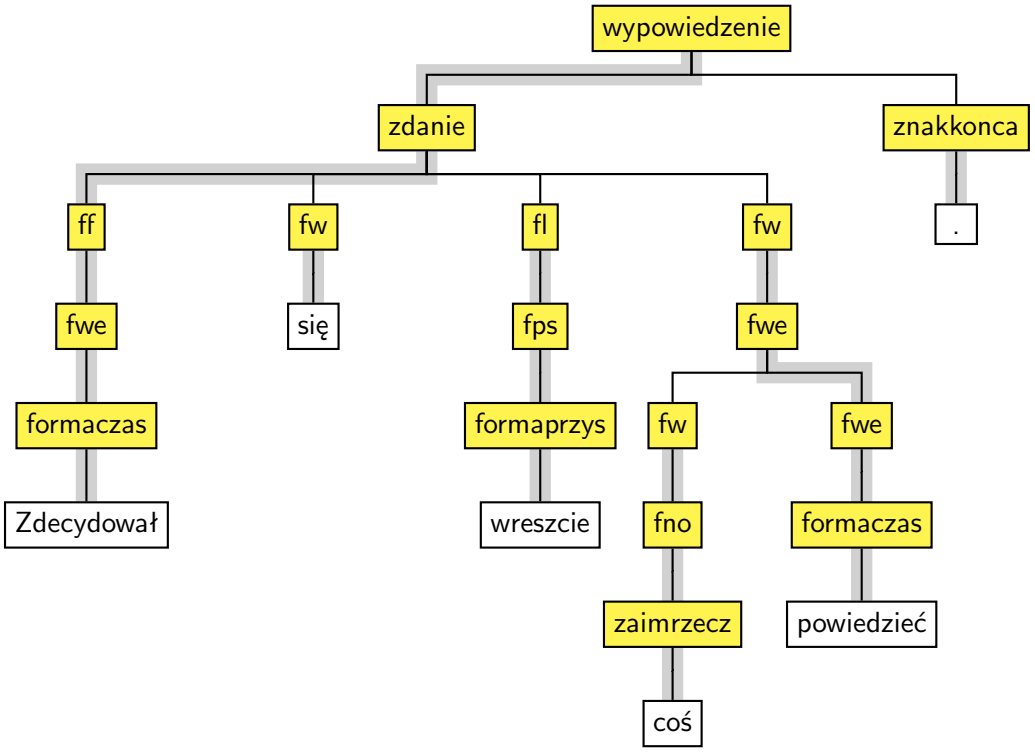


FIGURE 4.1: A phrase structure representation of the sentence *Zdecydował się wreszcie coś powiedzieć.* (Eng. 'He finally decided to say something.').

The syntactic structure of a sentence is represented as a constituent tree which is built in accordance with the Polish constituent grammar (Świdziński, 1992). Leaf and internal nodes of constituent trees are associated with terminals and non-terminals of the grammar respectively. Terminals correspond to vocabulary items (tokens). Apart from tokens, leaves encode information about lemmata, part of speech tags and morphological features of tokens. Non-terminals, in turn, represent the following syntactic categories:

- pre-terminals (or syntactic words), e.g., noun form (Pol. 'forma rzeczownikowa', *formarzecz*), verb form (Pol. 'forma czasownikowa', *formaczas* in Figure 4.1), adverbial form (Pol. 'forma przysłówkowa', *formaprzys* in Figure 4.1), or personal pronoun (Pol. 'zaimek rzeczowny', *zaimrzecz* in Figure 4.1). Strings of multiple tokens (e.g., analytical verb forms, complex prepositions) can also be regarded as pre-terminals in *Składnica* trees,

- constituent types, e.g., verbal phrase (Pol. 'fraza werbalna', *fwe* in Figure 4.1), nominal phrase (Pol. 'fraza nominalna', *fno* in Figure 4.1), adverbial phrase (Pol. 'fraza przysłówkowa', *fps* in Figure 4.1), adjectival phrase (Pol. 'fraza

przymiotnikowa', *fpt*), prepositional phrase (Pol. 'fraza przyimkowa', *fpm*), or clausal phrase (Pol. 'fraza zdaniowa', *fzd*),

- subcategorisation types determining whether a phrase is an argument, i.e., required phrase (Pol. 'fraza wymagana', *fw* in Figure 4.1) subcategorised by a sentence predicate or an optional adjunct, i.e., free phrase (Pol. 'fraza luźna', *fl* in Figure 4.1).

Internal nodes encode some grammatical features (e.g., case, number, gender, person, tense, aspect, mood, or government), possibly the subject function, and the information about heads of constituents. An example of a Polish constituent tree is given in Figure 4.1. In this example tree any constituent head-child is connected to its parent with a thick grey edge, e.g., *zdanie* is immediately headed by *ff*.

4.2 Conversion Procedure

In order to induce a bank of labelled dependency structures, we apply the constituent-to-dependency conversion procedure that makes use of information encoded in the Polish constituency treebank *Składnica*. The main idea behind the conversion is to cover all language-specific syntactic phenomena encoded in the Polish constituent trees and to annotate these phenomena with appropriate dependencies. Explicitly marked heads of constituents make it relatively straightforward to convert the phrase structure trees into unlabelled dependency structures, without the general requirement for head selection rules.

4.2.1 Lexical Nodes

Lexical nodes of a dependency tree built for a sentence correspond to tokens of this sentence. The spanning property implies that a dependency structure spans over all tokens of an underlying sentence. Since terminal nodes of a source constituent tree encode tokens, these tokens can be directly transferred to nodes of a converted dependency tree.

The column-based CoNLL format (Buchholz and Marsi, 2006) employed to encode converted dependency structures makes it possible to attach some additional morphosyntactic features of individual tokens. Morphosyntactic features are utilised in training a dependency parser and in parsing new sentences. Since the annotation schema of phrase structure trees requires encoding morphosyntactic features (i.e., lemmata, part of speech tags and morphological features) in each terminal node, these features can be directly transferred to lexical nodes of corresponding dependency trees without the need for additional language processing tools. Additional grammatical information enclosed in lexical nodes may have a positive impact on the quality of parsing fusional languages with multiple inflected surface word forms such as Polish.

4.2.2 Unlabelled Dependency Relations

Phrase structures in *Składnica* trees have their syntactic heads marked in most cases. The information about syntactic heads present in the constituent trees is employed to convert these trees into dependency structures.

Dependency relations are established between two tokens. Principles of well-formed dependency trees determine that each token may only have one governor. In order to identify related tokens in a constituent tree, it is essential to find the head of each token encoded in terminal nodes. The head of the current token corresponds to another token or the root node. A head token may be extracted with the following procedure:

1. starting from a terminal node encoding the current token, edges are followed in the bottom-up direction until the first non-head constituent node is reached,
2. then, starting from the parent of the identified non-head constituent node and following edges between parents and their head-children in the top-down direction the head token is found. If the parent node of a constituent node *zdanie* is labelled with *wypowiedzenie*, i.e., the root of the entire constituent tree, the additional ROOT node is determined as the head of the underlying token.

Extracted relations between tokens corresponding to dependents and their head tokens corresponding to dependency governors are transferred to dependency structures.

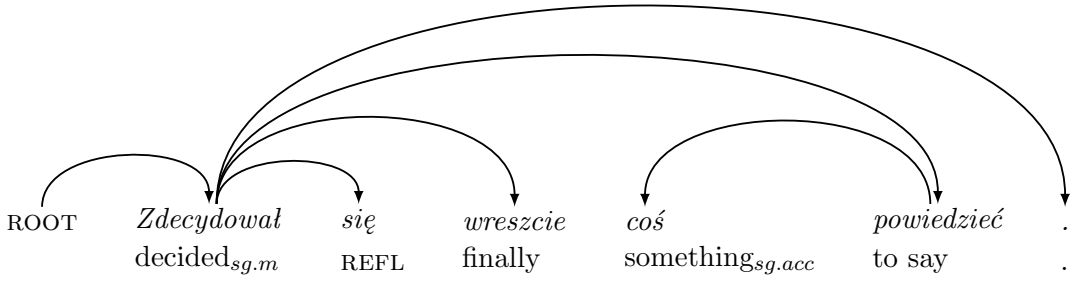


FIGURE 4.2: A dependency tree converted from the constituent tree in Figure 4.1.

For example, the non-head node *fl* is reached as a consequence of following constituent edges in the bottom-up direction starting from the token *wreszcie* (Eng. 'finally') in the tree in Figure 4.1. Then, starting from the parent of *fl* which is labelled with *zdanie*, edges between parents and head-children (i.e., *ff*, *fwe* and *formaczas*) are followed until the terminal node *Zdecydował* is reached. The relation *Zdecydował*$\longrightarrow$*wreszcie* is transferred as an arc to the converted dependency tree. Single arcs coming into all tokens of the sentence constitute the unlabelled dependency tree (see Figure 4.2).

4.2.3 Labelling Dependency Relations

The process of labelling dependency relations is conducted simultaneously with extracting relations from the constituent trees. Hence, an extracted relation transferred to a dependency structure is already labelled with a grammatical function. However, in order to preserve the clarity of the description, the two processes (extracting and labelling dependency relations) are discussed separately. The process of extracting dependency relations has already been presented. Now, we focus on the rule-based procedure of labelling dependency relations.

There are two ways of inferring grammatical functions. First, grammatical functions explicitly encoded in the constituent trees directly label appropriate relations in converted dependency structures. However, there is only one grammatical function encoded in the constituent trees – subject. The subject function is encoded in the highest phrase structure which is headed by the current token and labelled with the subcategorisation type *fw* (*fraza wymagana*, Eng. 'required phrase'). If the current token constitutes the head of a phrase, this phrase is required by the sentence predicate and is annotated with the subject function, the dependency relation between the token and the sentence predicate is labelled with *subj*. The relation $predicate \xrightarrow{subj} token$ may thus be directly transferred to a converted dependency tree.

Second, other grammatical functions are inferred based on information encoded in the phrase structure trees: subcategorisation information, phrase structure types, morphosyntactic features (e.g., part of speech, case, number, gender or person), and types of construction rules used to build the constituent trees. In the course of identifying the head of a token, several non-terminal nodes are visited. Information encoded in these non-terminals is considered to derive labels of induced dependency relations.

4.2.3.1 Verb-Dependent Relations

Labelling rules take into account subcategorisation information encoded in non-terminal nodes. The constituency annotation schema distinguishes two subcategorisation types with which dependents of a sentence predicate are annotated: free phrases *fl* (Pol. 'fraza luźna') which are equivalent to adjuncts and required phrases *fw* (Pol. 'fraza wymagana') corresponding predominantly to arguments. If the highest constituent headed by a token is labelled with the category *fw* or *fl*, its parent should be labelled either with the phrase structure category *zdanie* (Eng. 'sentence') or with the category *fwe* (*fraza werbalna*, Eng. 'verb phrase').

Dependency relations between predicates and head tokens of free phrases are always labelled with the *adjunct* function. In the constituent tree in Figure 4.1, there is one terminal *wreszcie* whose highest constituent node is labelled with the category *fl*. We

presented the process of extracting the relation between *wreszcie* and its governor *Zdecydował* in the previous section. Based on the subcategorisation type *fl*, this relation is labelled with the *adjunct* function ($Zdecydował \xrightarrow{adjunct} wreszcie$) and transferred to the final dependency tree (see Figure 4.3).

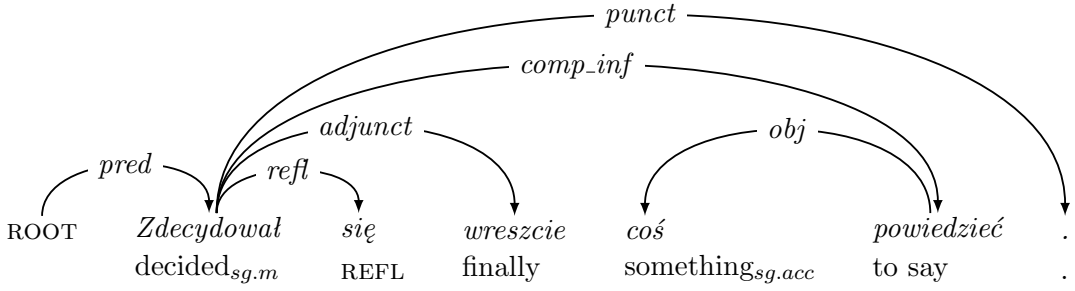


FIGURE 4.3: A labelled dependency tree converted from the constituent tree in Figure 4.1.

The procedure of labelling relations between head tokens of required phrases (except for the subject arguments discussed above) and sentence predicates is rather complex. It consists in inferring appropriate grammatical functions of verb arguments based on phrase structure types and morphosyntactic features encoded in terminal nodes of related tokens.

The constituent type of the current token is a valuable source of knowledge in the procedure of labelling verb-argument relations. If the phrase structure headed by a token is labelled with the category *fno* (*fraza nominalna*, Eng. 'noun phrase'), the relation between the token and the predicate can be labelled with *obj*, *obj_th* or *pd*. The *obj* function labels the relation between a transitive verb and the head noun of a noun phrase. This head noun can alternate to the subject during the passivisation and is marked for accusative, genitive of negation, dative or instrumental. Since information about the transitivity of a verb is not encoded in the constituent trees, we use external lists of transitive verbs that require objects marked for particular cases.[1] In the example tree in Figure 4.1, there is a noun phrase *coś* that is required by the transitive verb *powiedzieć*. Hence, the relation $coś \xleftarrow{obj} powiedzieć$ is transferred to the final dependency tree (see Figure 4.3). The *obj_th* function is assigned to relations established between non-predicative verbs form and heads of noun phrases. The head noun cannot be marked for nominative or locative and may not alternate to the subject during the passivisation. The *pd* function labels relations in which a predicative verb governs the head of a nominal phrase which is marked for instrumental.

[1] In order to separate objects from other nominal complements, the best solution would be to use the valency dictionary by Przepiórkowski et al. (2013). However, at the time of conducting our experiments with the treebank conversion, this dictionary did not exist. Hence, we compiled lists of the most common transitive verbs available in *Składnica* trees based on the linguistic literature and dictionaries of Polish.

If the current token is an infinitive that heads a phrase structure labelled with the category *fzd* (*fraza zdaniowa*, Eng. 'clause') or *fwe* (*fraza werbalna*, Eng. 'verb phrase'), the relation between the infinitive and its governing sentence predicate is labelled with the *comp_inf* function. In the example tree in Figure 4.1, there is the head infinitive *powiedzieć* of a verb phrase which is required by the sentence predicate *Zdecydował*. Hence, the relation *Zdecydował* $\xrightarrow{comp\_inf}$ *powiedzieć* is transferred to the final dependency tree (see Figure 4.3).

The relation between a sentence predicate and a finite verb whose highest constituent node is assigned the type *fzd* is labelled with the *comp_fin* function. The relation between a predicate and a preposition (prepositional phrase category *fpm*) is labelled with the *comp_ag* function if it is a demoted subject realised as the *przez*-prepositional phrase and if the predicate is realised as a passive adjectival participle. Otherwise, the relation is labelled with the *comp* function. The relation between a predicate and an adjective (adjective phrase category *fpm*) is labelled either with the *pd* function if the predicate has the form of a predicative verb or with the *comp* function. The relation between a predicate and an adverb (adverbial phrase category *fpt*) is labelled with the *comp* function.

There are also some non-arguments that may depend on a sentence predicate, e.g., an auxiliary verb, a complementiser, a conditional clitic, an imperative marker, a mobile inflection, a negation marker, a punctuation marker or a reflexive marker. Relations between tokens, which are identified by their morphosyntactic properties as members of the specified part of speech groups, and their governing verbs are labelled according to the dependency annotation schema (see Chapter 3 *Polish Dependency Annotation Schema*). In the example tree in Figure 4.1, there is a reflexive marker *się* that is required by the predicate *Zdecydował*. Hence, the relation *Zdecydował* $\xrightarrow{refl}$ *się* is transferred to the final dependency tree (see Figure 4.3). Furthermore, in this example tree (Figure 4.1), there is also a punctuation marker realised as the full stop which depends on the sentence predicate *Zdecydował*. The relation *Zdecydował* $\xrightarrow{punct}$. is thus transferred to the final dependency tree (see Figure 4.3).

4.2.3.2 Other Relations

The first case concerns sentence predicates realised as verb forms marked with the category *ff* (*fraza finitywna*, Eng. 'finite phrase'). A constituent tree is rooted at the highest non-terminal node of the sentence predicate which is labelled with the phrase structure category *wypowiedzenie* (Eng. 'utterance'). The predicate is thus transferred as the top lexical node of a dependency tree governed by an artificial ROOT node. The dependency relation between the predicate and the ROOT node is labelled with the *pred* function. In the example tree in Figure 4.1, the highest constituent node headed by the sentence predicate *Zdecydowł* is labelled with the constituent category *wypowiedzenie*. Hence,

the relation ROOT$\xrightarrow{pred}$*Zdecydował* is transferred to the final dependency tree (see Figure 4.3). Since the constituent treebank contains only proper sentences with finite verb forms or coordinated verb forms functioning as sentence predicates, the ROOT node may also govern a coordinating conjunction or a coordinating punctuation mark. In such cases, the relation is labelled with *coord* or *coord_punct* if it is a coordination of two sentences, or with the *pred* function if it is a coordination of sentence predicates with a shared subject and possibly other shared arguments.

Labelling rules also cover dependency relations between a preposition and its dependents within the scope of a prepositional phrase (*fpm*) or a prepositional-adjective phrase (*fpmpt*). Dependents mostly bear the function of the prepositional complement (*comp*). However, if dependents of prepositions are realised as modifying particles (e.g., *już*, Eng. 'already', *nawet*, Eng. 'even', *zwłaszcza*, Eng. 'especially') or adverbs (e.g., *włącznie*, Eng. 'inclusive'), relations between prepositions and their dependents are labelled with the *adjunct* function.

Within the scope of a noun phrase *fno*, dependents of the head noun bear either the *adjunct* function or the *app* function if the type of a construction rule is specified as *noap*. However, if a head noun is realised as a gerund, its dependents may bear both argument and adjunct functions.

Relations between an adverb or an adjective and their dependents in an adverbial phrase (*fps*) or an adjectival phrase (*fpt*) are mostly labelled with the *adjunct* function. The *adjunct* function is also treated as the default label of relations that cannot be covered with labelling rules.

4.3 Head Selection

In accordance with what has been said above, constituents have their syntactic heads marked in *Składnica* trees and head selection rules seem to be redundant. However, there are some phrase structures (predominantly syntactic words) in which not only one but several elements are marked as syntactic heads. In the case of constituents with multiple head-children, it is necessary to decide which child node should be selected as the proper head in order to find an unequivocal head of each token. The current section presents multi-headed scenarios and head-selection heuristics designed to handle such scenarios.

The first case refers to conditional verb forms which consist of a verbal stem and the conditional particle *by*. These two tokens are annotated as head-children of a conditional verbal phrase in the constituent trees. The dependency annotation schema, in turn, implies that the conditional particle should depend on the verbal stem (see *cond* in Section 3.2.3, p. 41). Hence, the verbal stem is selected as the head of the conditional verb phrase during constituency-to-dependency conversion. The relation between the verbal

stem *chciał* (governor) and the conditional particle *by* (dependent) labelled with the *cond* function is transferred to the dependency tree in Figure 4.4.[2]

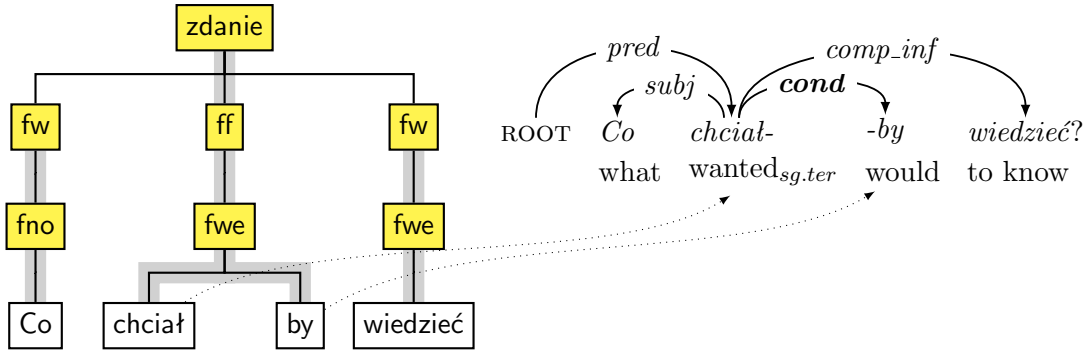


FIGURE 4.4: Conversion of the constituent tree (left tree) of the sentence *Co chciałby wiedzieć?* (Eng. 'What would he want to know?') into a dependency structure (right tree). Heads are marked with shaded lines. The transfer of the dependency relation between two parts of the conditional verb form *chciałby* is pointed out with dotted arrows.

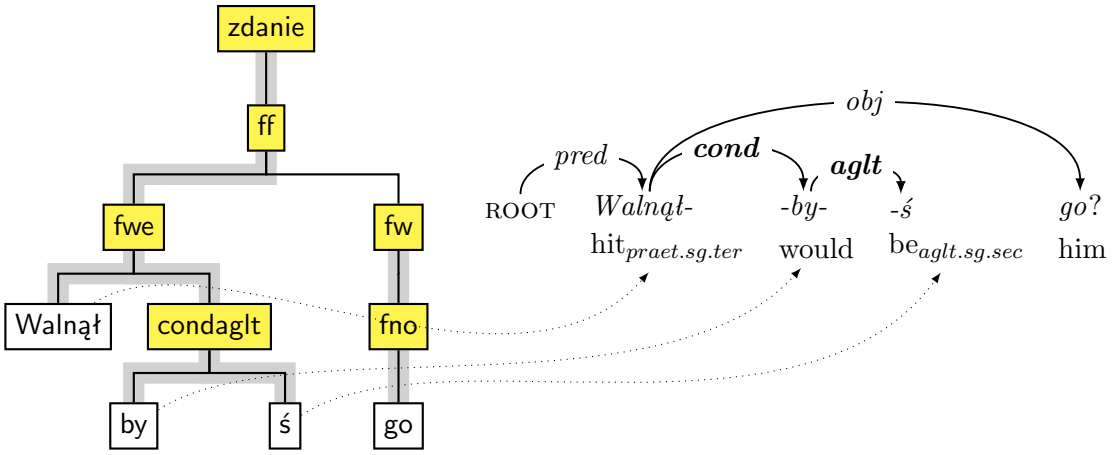


FIGURE 4.5: Conversion of the constituent tree (left tree) of the sentence *Walnąłbyś go?* (Eng. 'Would you hit him?') into the dependency structure (right tree). The transfer of dependency relations between three parts of the conditional verb form *walnąłbyś* is pointed out with dotted arrows.

Similarly, a verb form and a mobile inflection, which is regarded as an independent syntactic element in Polish, are annotated as head-children of a verb phrase. However, the dependency annotation schema assumes that the mobile inflection depends on the verbal stem (see *aglt* in Section 3.2.3, p. 39). Therefore, the relation between the verbal stem (governor) and the mobile inflection (dependent) is transferred to the converted dependency tree and labelled with the *aglt* function. The mobile inflection may also be part of a conditional verb form. All three head-marked children (a verb stem, the conditional particle *by* and a mobile inflection) build one phrase structure in the constituent

[2]For clarity's sake, some constituent levels are not displayed in the diagrams of constituent trees. For the presentation of entire constituent trees refer to the search engine of *Składnica* trees at `http://nlp.ipipan.waw.pl:8000/ui.xhtml`.

trees. In converted dependency structures, the mobile inflection depends on the conditional particle and the particle, in turn, depends on the verbal stem (see Figure 4.5).

The second case of multi-headed constituents refers to analytical verb and quasi-verb forms. An analytical verb form consists of an auxiliary verb and a main verb form (e.g., infinitive, participle or quasi-verb). There are two auxiliary verbs in Polish: *zostać* (Eng. 'to be') used in passive constructions and *być* (Eng. 'to be') used to build the imperfect future tense, the analytical past conditional, analytical forms of quasi-verbs, analytical forms of the predicative *to*[3] and passive constructions. All parts of analytical verb and quasi-verb forms[4] are head-marked in the phrase structure trees. The semantically oriented dependency annotation schema assumes that the main verb form governs the auxiliary verb (see *aux* in Section 3.2.3, p. 40). Hence, the main verb form is selected as the head child during constituency-to-dependency conversion. The relation between the governing main verb and the auxiliary verb is labelled with the *aux* function and is transferred to the resulting dependency tree (see Figure 4.6).

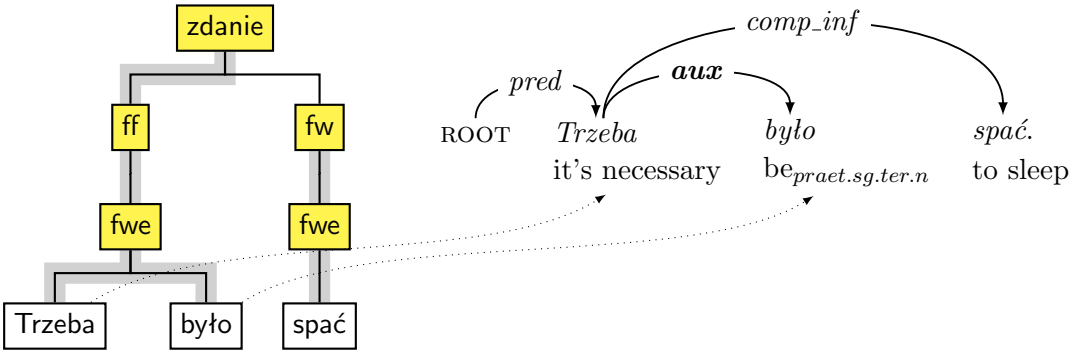


FIGURE 4.6: Conversion of the constituent tree (left tree) of the sentence *Trzeba było spać.* (Eng. 'We had to sleep.') into the dependency structure (right tree). The transfer of the dependency relation between the quasi-verb *trzeba* and the auxiliary verb *było* is pointed out with dotted arrows.

Since Polish distinguishes between simple and complex conjunctions, the third case concerns complex coordinating or subordinating conjunctions whose parts are annotated as constituents' heads in the phrase structure trees. In two-part coordinating conjunctions, e.g., *albo... albo...* (Eng. 'either... or...'), *ani... ani...* (Eng. 'neither... nor...'), *nie tylko... ale także* (Eng. 'not only... but also'), the first element (or multiword element) is converted as the dependent of the second one and the relation is labelled with the *pre_coord*

[3]The predicative *to* (Eng. 'it, it's') fulfils the role of a sentence predicate in Polish. Similarly to other quasi-verb forms, *to* may be combined with an auxiliary verb to express future tense (e.g., ***To będzie*** *dobry moment.*, Eng. 'It will be a good time.'), present tense (e.g., ***To jest*** *złe myślenie.*, Eng. 'It is a wrong way of thinking.') or past tense (e.g., ***To było*** *przypadkowe uderzenie.* Eng. 'It was a random attack.'). Sometimes, it is difficult to distinguish between the predicative *to* and *to* treated as the subject (e.g., *To byłoby śmieszne, groteskowe.*, Eng. 'It would be ridiculous, grotesque.'). The subject function is explicitly encoded in the manually checked constituent trees. Therefore, *to* is converted as the subject only if it is annotated as the subject in a constituent tree.

[4]Passive constructions are considered in Section 4.4.2 *Passive Construction* since their phrase structure annotations are different from annotations of analytical verb forms.

function. In two-part subordinating conjunctions, e.g., *mimo że* (Eng. 'although'), *podczas gdy* (Eng. 'whereas, while'), the first token is selected as the governor of the second one and the relation is labelled with the *mwe* function (see Figure 4.7).

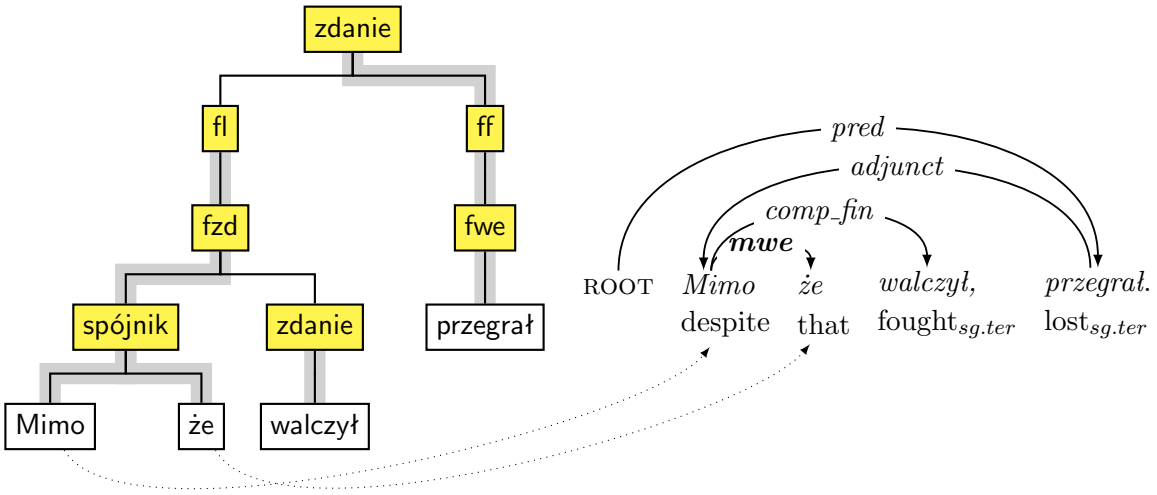


FIGURE 4.7: Conversion of the constituent tree (left tree) of the sentence *Mimo że walczył, przegrał.* (Eng. 'Although he fought, he lost.') into the dependency structure (right tree). The transfer of the dependency relation between two parts of the complex subordinating conjunction *mimo że* is pointed out with dotted arrows.

In the case of other multi-headed phrase structures (e.g., abbreviations, multiword expressions, series of punctuation marks), they are converted in accordance with their linear word order. The first element is selected as the governor of the second element, which is, in turn, the governor of the next one, and so on. Relations are labelled with appropriate grammatical functions. An example of converting a constituent tree with a series of punctuation marks annotated as heads into a dependency structure is given in Figure 4.8.

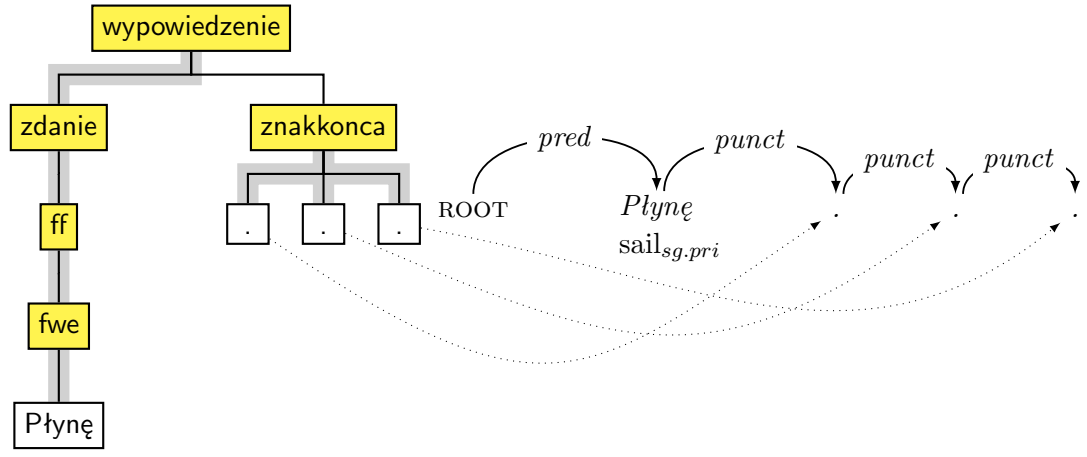


FIGURE 4.8: Conversion of the constituent tree (left tree) of the sentence *Płynę...* (Eng. 'I am sailing...') into the dependency structure (right tree). The transfer of dependency relation between individual punctuation marks is pointed out with dotted arrows.

4.4 Rearrangement of Dependency Structures

As discussed above, the conversion is a relatively straightforward process. However, rearrangement of particular arcs in converted dependency trees is essential to meet the annotation principles. The current section describes some linguistic constructions whose phrase structure annotations are transferred to dependency structures with modifications. Besides reordering described below we do not interfere in the internal structure of the constituent trees, but we take syntactic facts encoded in these trees as they are.

4.4.1 Discontinuous Constituents

The constituent annotation schema does not cover discontinuous constituents. Therefore, unconnected constituent parts are encoded as quasi-independent constituents (see annotation of *kilka wniosków*, Eng. 'a few applications' in the constituent tree in Figure 4.9). The direct conversion of discontinuous constituents would result in dependency structures which are incompatible with the annotation principles. Converted dependency structures are thus reordered and discontinuous dependencies are annotated in accordance with the dependency annotation schema even if it results in non-projective trees. Currently, only discontinuous dependencies within the scope of numeral phrases are identified in *Składnica* trees. However, we cannot rule out that other discontinuous constituents are encoded in these trees.

Rules of annotating Polish numeral phrases require that numerals are treated as syntactic governors of depending noun phrases. Case of the depending noun phrase either agrees with case of the governing numeral marked for dative, instrumental or locative, or is determined as genitive if the governing numeral is marked for nominative, accusative, vocative or genitive. On the other hand, the depending noun phrase imposes the gender feature on the governing numeral. Hence, morphological clues can be employed to identify a nominal dependent of a numeral even if the noun phrase is annotated as an independent constituent in a constituent tree.

In the example tree in Figure 4.9, the genitive noun phrase *Wniosków* (Eng. 'Applications') initiating the sentence is annotated as an independent constituent required by the sentence predicate. Based on the morphological evidence and the fact that a numeral typically requires a complement, the noun *Wniosków* is identified as the complement of the numeral *kilka* (Eng. 'a few') marked for nominative.[5] The converted dependency structure is appropriately reorganised and the resulting dependency tree is shown in Figure 4.9 (the dependency relation marked with the dotted line on the schema is not present in the final dependency structure).

[5] In this example, the numeral *kilka* (Eng. 'a few') may also be analysed as an accusative.

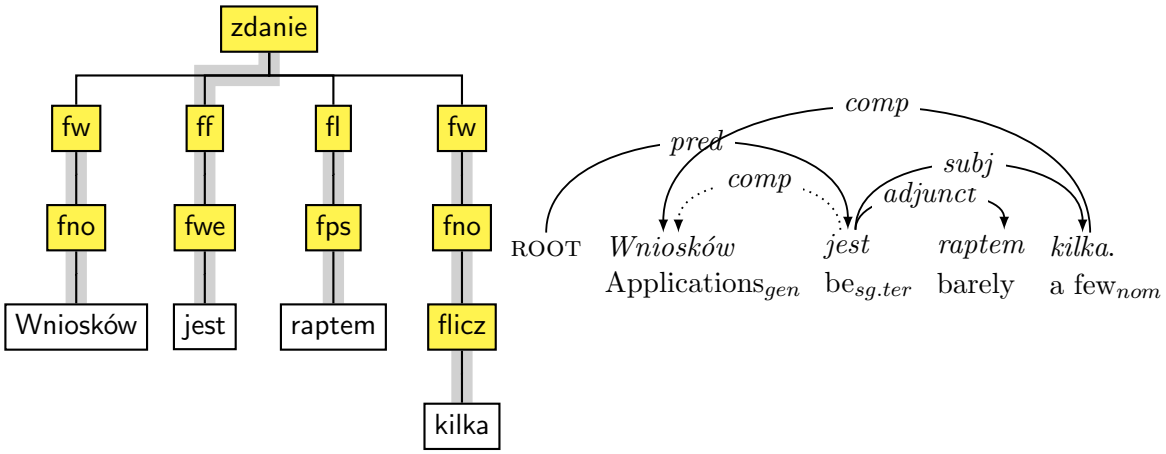


FIGURE 4.9: Conversion of the constituent tree (left tree) of the sentence *Wniosków jest raptem kilka.* (Eng. 'There is barely a few applications.') into the dependency structure (right tree). The dotted line indicates the relation before reorganising.

4.4.2 Passive Construction

The passive voice is indicated in Polish by a conjugated auxiliary verb combined with a passive adjectival participle. In the constituent trees, auxiliary verbs constitute heads of passive constructions and participles are annotated as adjectival phrases required by auxiliaries. Nevertheless, we annotate passive constructions by analogy to analytical future or past conditional constructions. The participle is annotated as the head of a passive sentence. If it is governed by the ROOT node, the relation between ROOT and the participle is labelled with the *pred* function. The auxiliary verb depends on the participle and the relation is labelled with the *aux* function. Required arguments and non-subcategorised adjuncts depend on the sentence predicate.

A passive adjectival participle may also occur without an auxiliary verb as the dependent of a nominal phrase, e.g., *Parlament rozwiązany 30 sierpnia 1930 r.* (Eng. 'Parliament dissolved on August 30, 1930.') as shown in Figure 4.10. Such constructions can be treated as 'reduced' relative clauses in contrast to relative clauses like the one shown in Figure 4.11. Since both constructions have the same meaning, the decision of annotating them with similar dependency structures seems to be well-founded.

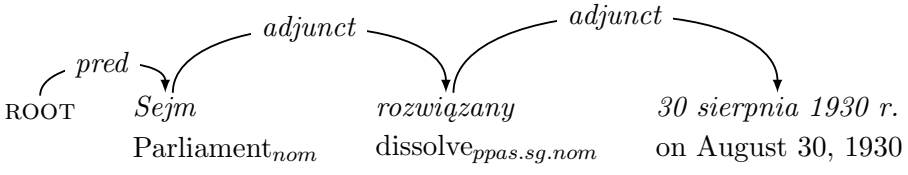


FIGURE 4.10: A dependency structure of the noun phrase *Sejm rozwiązany 30 sierpnia 1930 r.* (Eng. 'Parliament dissolved on August 30, 1930').

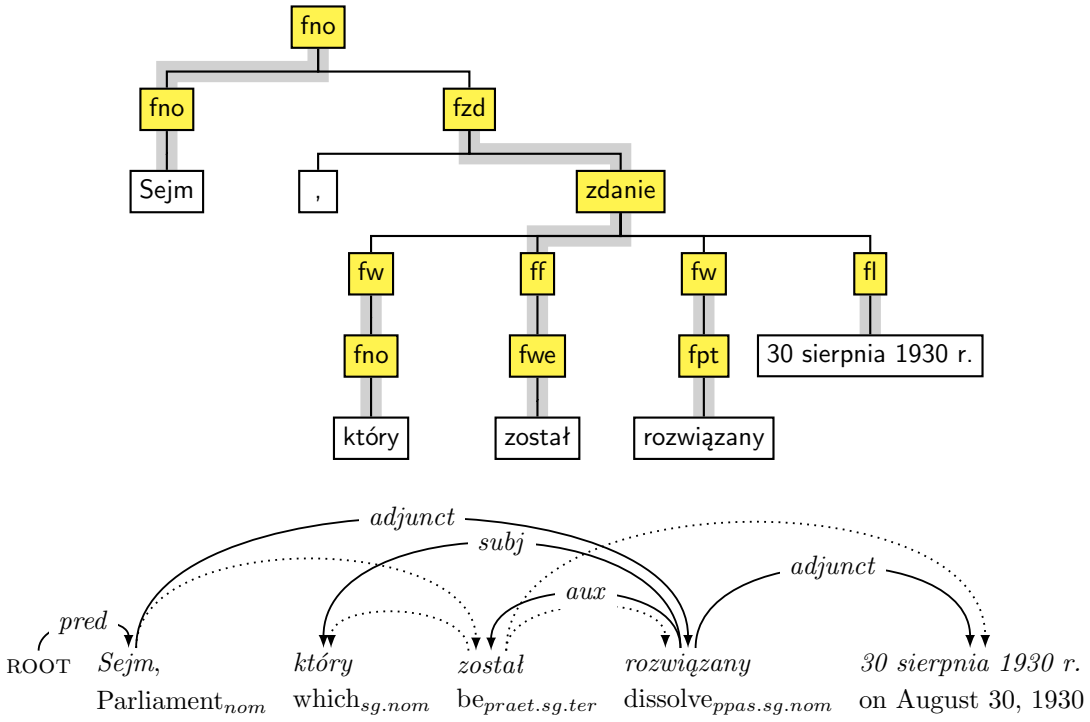


FIGURE 4.11: Conversion of the constituent tree (top tree) of the noun phrase *Sejm, który został rozwiązany 30 sierpnia 1930 r.* (Eng. 'Parliament, which was dissolved on August 30, 1930') into the dependency structure (bottom tree). Dependency arcs which are directly transferred but do not appear in the final tree are marked with dotted lines. Reordered arcs and other arcs which are directly transferred but not modified are marked with solid lines.

4.4.3 Subordinate Clauses

Various types of subordinate clauses functioning as adjuncts are distinguished in the constituent treebank. However, we found out that subordinate clauses of the same type are differently annotated in the treebank. Divergent trees with subordinate clauses result from the annotation procedure which consists in simultaneous annotation of trees and modification of the underlying grammar. As the Polish constituent grammar improved, annotation rules changed and new sentences were annotated with the amended rules. However, some trees annotated before the modification of the grammar could be incompatible with the improved version of the grammar, but they were not upgraded.

Incompatible constituent analyses of complex sentences are rather isolated cases, but they violate our conversion rules. Conversion rules designed for transferring relations from constituent trees annotated in accordance with one grammar version may be unsuitable for converting other constituent trees. Therefore, some additional rules are designed to annotate complex sentences of the same type (identified by categories of phrase structure rules) with uniform dependency structures. This rearrangement process does not apply to all complex sentences, but only to those whose annotations deviate from the following annotation principles.

In subordinate clauses introduced by a conjunction, e.g., *albowiem, bo* or *gdyż* (Eng. 'because, since'), the conjunction constitutes the head of the subordinate clause and depends on the sentence predicate of a superordinate clause. Similarly, conjunctions *jeśli* or *gdyby* (Eng. 'if') introducing subordinate clauses depend on the sentence predicate of a superordinate clause. If an optional particle *to* (Eng. 'then') introduces such superordinate clause, this particle depends on the sentence predicate of the superordinate clause and the relation is labelled with the *adjunct* function (see Figure 4.12).[6]

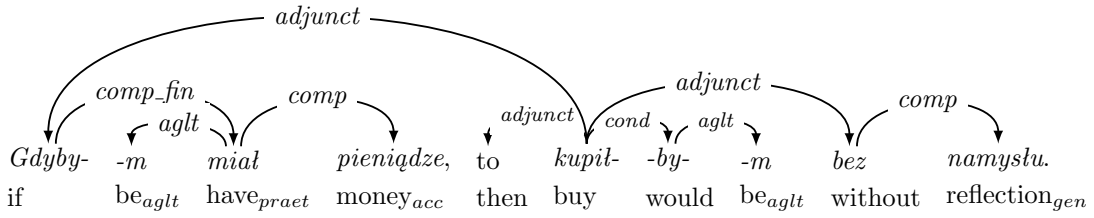


FIGURE 4.12: A dependency structure of the sentence with a subordinated clause *Gdybym miał pieniądze, to kupiłbym bez namysłu.* (Eng. 'If I had money I would buy it immediately.').

4.4.4 Incorporated Conjunction

Polish admits complex sentences with a conjunction taking a non-initial position in a clause. We distinguish coordinating constructions with conjunctions (e.g., *przeto, więc, zatem*, Eng. 'therefore, then') incorporated into the second of coordinated clauses. Furthermore, there are also subordinating constructions with conjunctions (e.g., *bowiem*, Eng. 'since, as') which may appear anywhere in the subordinate clause, subject to various island constraints. In the constituent trees, an incorporated conjunction depends on the immediately preceding constituent, e.g., verb, adverb or noun. Even if the conversion results in a non-projective dependency structure, an incorporated conjunction is annotated either as the governor of coordinated clauses (see Figure 4.13) or as the governor of a subordinate clause.

4.4.5 Clauses with Correlative Pronouns

The correlative pronoun (Pol. 'korelat', Świdziński, 1992) is a pronoun (also a pronoun in a prepositional phrase) that correlates with a succeeding relative clause. In the constituent trees, the predicate of the relative clause with the correlative pronoun is annotated as the phrase structure head. Direct conversion of such constructions would require an additional dependency type which would be equivalent to the phrase structure category *korelat*. As clauses with correlative pronouns can be handled with relation types

[6]Since the example sentence is long, the ROOT node is not displayed here.

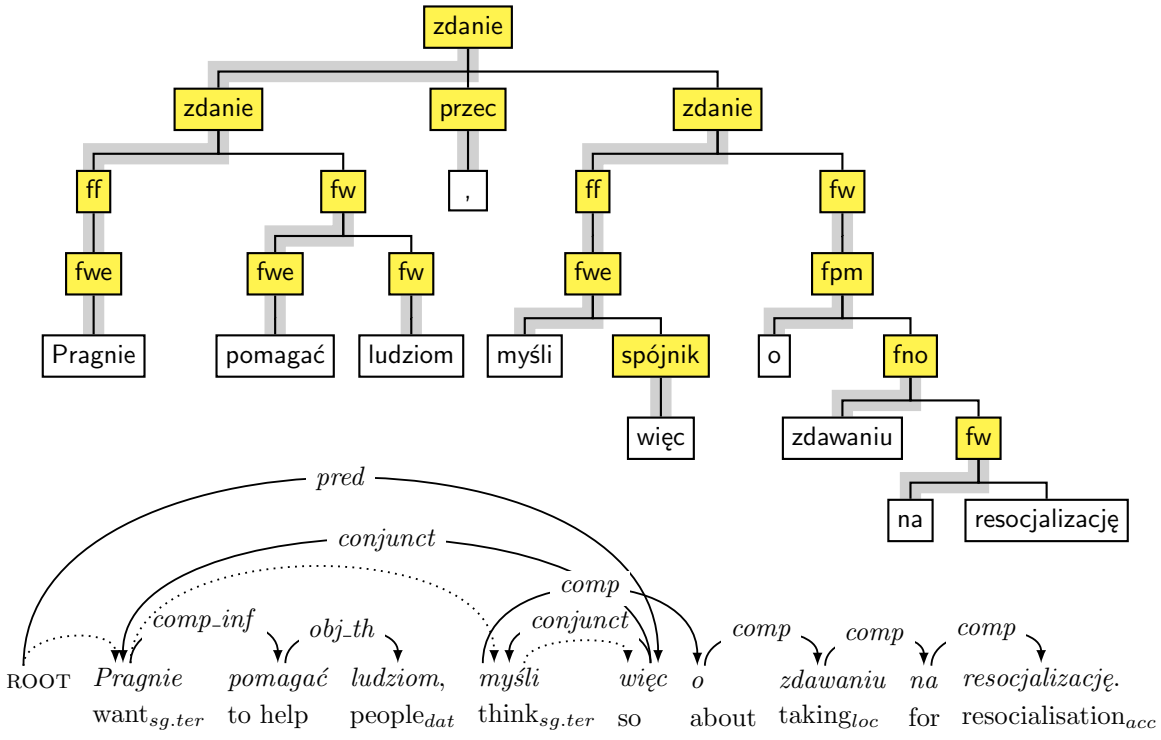


FIGURE 4.13: Conversion of the constituent tree with an incorporated conjunction (top tree) of the sentence *Pragnie pomagać ludziom, myśli więc o zdawaniu na resocjalizację.* (Eng. 'He wants to help people, so he is thinking about taking an entrance exam for resocialisation.') into the dependency structure (bottom tree). Dependency arcs which are directly transferred but do not appear in the final tree are marked with dotted lines. Reordered arcs and other arcs which are directly transferred but not modified are marked with solid lines.

already defined in the dependency annotation schema, an additional type is unnecessary in the context of dependency structures. Such constructions are thus converted as nominal phrases realised as correlative pronouns with depending relative clauses.

In the constituent tree in Figure 4.14, there is a clause with a correlative prepositional phrase. The correlative pronoun *tym* is converted as the governor of the relative clause. The relation between the correlative pronoun and the verbal head of the relative clause is labelled with the *comp_fin* function (see the dependency tree in Figure 4.14). The correlative pronoun depends on the preposition (the relation $O \xrightarrow{comp} tym$ in the dependency tree in Figure 4.14) and the preposition, in turn, is governed by the predicate of the superstructured clause (the relation $O \xleftarrow{comp} wiedzieli$ in the dependency tree in Figure 4.14).

4.5 Experimental Setup

The conversion-based dependency treebank is used to train Polish dependency parsers with publicly available parser generation systems.

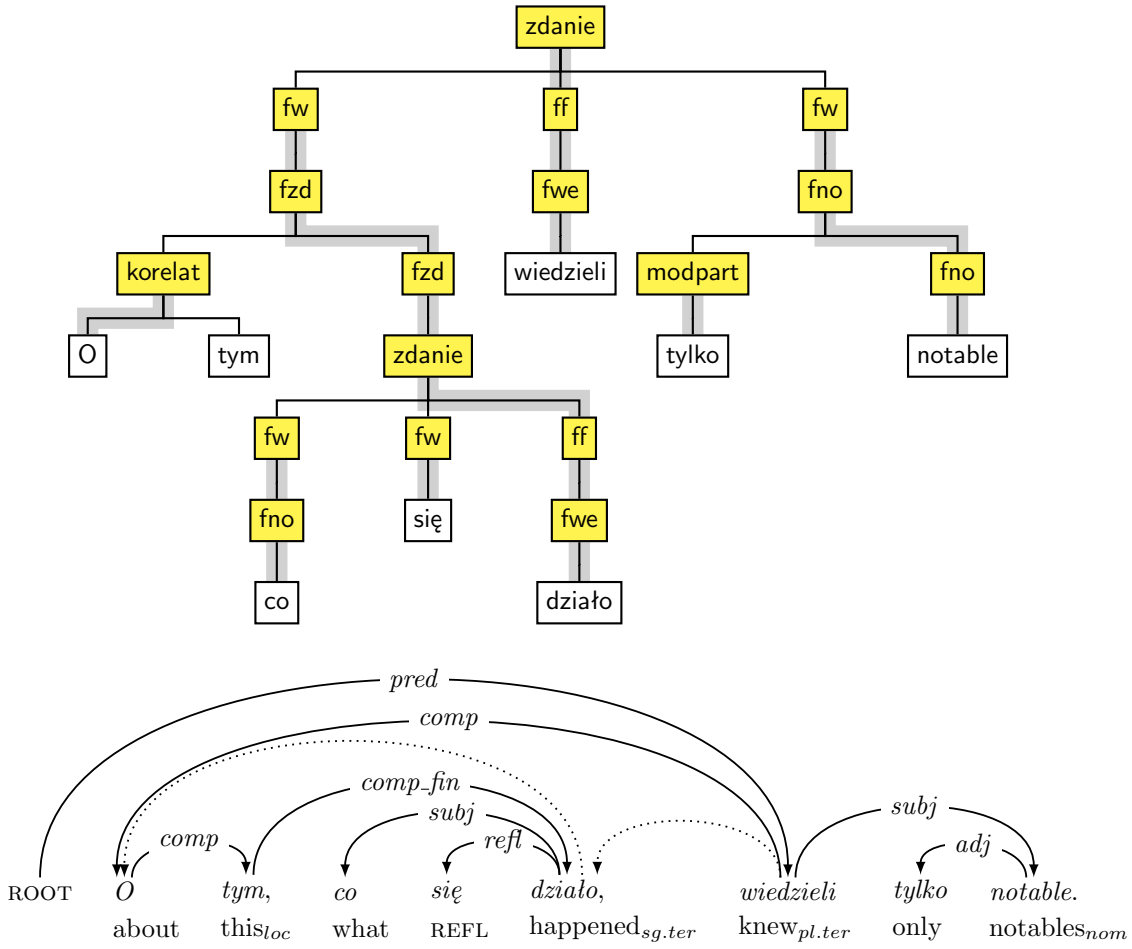


FIGURE 4.14: Conversion of the constituent tree (top tree) of the sentence with a correlative pronoun *O tym, co się działo, wiedzieli tylko notable.* (Eng. 'Only notables knew what was happening.') into the dependency structure (bottom tree). Dependency arcs which are directly transferred but do not appear in the final tree are marked with dotted lines. Reordered arcs and other arcs which are directly transferred but not modified are marked with solid lines.

4.5.1 Data

A dependency treebank acquired with the constituency-to-dependency conversion method consists of 8227 dependency trees. We are aware that this treebank is relatively small, so we will verify if it is sufficient to train a dependency parser. Moreover, sentences in the treebank are rather short and consist of 10.16 tokens on average. The entire treebank contains only 125 non-projective arcs (0.15% of all arcs). Therefore, it is very likely that sentences have relatively simple syntactic structures in many cases.[7] Furthermore,

[7]The selection of sentences with relatively simple syntactic structures results from the procedure of annotating source constituent trees. This procedure starts with generation of candidate parse trees for a sentence with the *Świgra* parser (Woliński, 2004, 2005b). Then, the candidate trees are disambiguated and validated by human annotators in order to select an appropriate constituent tree. However, only 8227 sentences from the entire set of 20,000 sentences randomly selected from NKJP are in the version of *Składnica* which is employed in our experiments. There are two reasons for rejecting a large part of sentences. First, 42.3% of sentences were not accepted by the constituent grammar. The rejected sentences contained grammatical errors, they did not contain a finite verb or they were too complex to

the procedure of annotating source constituent trees was iterative,[8] and some linguistic phenomena can be differently annotated in *Składnica* trees. It is thus possible that the converted dependency structures vary since the conversion is an automatic process.

In view of possible errors in the converted dependency structures, we decided to manually verify at least a part of the treebank. Some of the automatically converted dependency trees are manually corrected by a linguist experienced in the Polish syntax. The first 1,000 trees are thoroughly checked for errors. Other trees are skimmed through focusing on potentially recurring errors. Hence, the converted dependency structures may still contain errors. Even though we investigate whether they are sufficient for training a Polish dependency parser.

According to our knowledge, this is the first attempt at building a Polish dependency treebank using constituency-to-dependency conversion. Moreover, we are not aware of the existence of an already annotated dependency treebank of Polish. Therefore, we use the converted treebank both for training and for evaluation purposes. The entire dependency treebank is split into a validation set with 822 trees (10.08 tokens per sentence on average, 0.46% of non-projective arcs) and a training set with 7405 trees (10.17 tokens per sentence on average, 0.12% of non-projective arcs).

4.5.2 Dependency Parsing Systems

Two dependency parsing systems are selected for training dependency parsers for Polish: the transition-based *MaltParser* (Nivre et al., 2006a) and the graph-based *Mate* parser (Bohnet, 2010). We start with a short description of both parsing systems.

The transition-based dependency parser *MaltParser*[9] uses a deterministic parsing algorithm that builds a dependency structure of an input sentence based on transitions (shift-reduce actions) predicted by a classifier. The classifier learns to predict the next transition given training data and the parse history. The architecture of the deterministic *MaltParser* consists of three main components: a parsing algorithm that derives a labelled dependency structure from an input sentence, a treebank-induced classifier that deterministically predicts an optimal next transition given the feature representation in the current configuration of the parser, and a feature model that supports the prediction of the next parser transition.

be parsed. Second, 27.7% of parsed sentences were rejected because human validators could not select any appropriate tree from the candidate parse trees. We suppose that the rejected sentences could have more complex syntactic structures.

[8] The iterative annotation process consisted in the parallel development of the treebank and the underlying grammar. If human validators asserted that there was no appropriate tree for a sentence, grammar rules were corrected or new grammar rules were defined to cover syntactic phenomena in this sentence. These rules were then used to parse new sentences. Since previously annotated trees were not improved, some inconsistencies could arise in the treebank.

[9] We use *MaltParser* 1.7.2 downloaded from `http://maltparser.org`.

The *MaltParser* system provides some built-in implementations of parsing algorithms. There are some algorithms for projective dependency structures: *nivreeager*, *nivre-standard*, *covproj* (Nivre, 2008) and *stackproj* (Nivre, 2009). Besides, there are some algorithms for non-projective dependency structures: *covnonproj* (Nivre, 2008), *stack-lazy* (Nivre et al., 2009), and *stackeager* (Nivre, 2009), and for m-planar[10] dependency structures: *planar* and *2planar* (Gómez-Rodríguez and Nivre, 2010)

The *MaltParser* system enables switching between two implementations of machine learning algorithms used to train a classifier given training data: the LIBSVM library (Chang and Lin, 2001) being an implementation of *support vector machines* and the LIBLINEAR package (Fan et al., 2008) with different *linear classifiers*.

The history-based feature model is employed by the *MaltParser* classifier to predict next actions at non-deterministic choice points. Features are defined in terms of token attributes, i.e., word form (`FORM`), part of speech (`POS`), morphological features (`FEATS`), and lemma (`LEMMA`) available in input data or dependency types (`DEPREL`) extracted from partially built dependency graphs and updated during parsing.

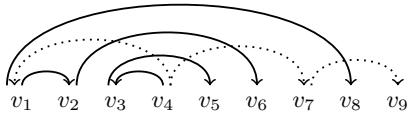
The *Mate* parser (Bohnet, 2010)[11] is a graph-based parser that improves parsing time simultaneously maintaining the accuracy level. There are two main components of the *Mate* parser – a decoder and a feature model. As the decoder, the parser applies the implementation of the second-order MST parsing algorithm by Carreras (2007). The model, in turn, is a linear multi-class classifier trained with the passive-aggressive perceptron algorithm (McDonald et al., 2005a; Crammer et al., 2006) implemented as the hash kernel.

Since the employed parsing algorithm produces only projective dependency structures, Bohnet (2010) applies the Non-Projective Approximation Algorithm (McDonald and Pereira, 2006) for non-projective parsing. The idea behind this algorithm is to find the highest scoring projective tree and to rearrange arcs in this tree. Rearrangements should increase the tree score and may not lead to violate constraints on dependency trees. Transformed trees may be non-projective. Bohnet (2009) adds a further parameter

[10] A planar dependency structure can be drawn on the plane without crossing arcs, i.e., it corresponds to a projective dependency tree. Gómez-Rodríguez and Nivre (2010) gives the following definition of a multiplanar (m-planar) graph:

A dependency graph $G = (V, E)$ is **m-planar** iff there exist planar dependency graphs $G_1 = (V, E_1), ..., G_m = (V, E_m)$ (called **planes**) such that $E = E_1 \cup ... \cup E_m$.

The following dependency structure (taken from Gómez-Rodríguez and Nivre, 2010) is 2-planar, because there are two planes with two distinct sets of non-crossing arcs (represented by solid and dotted lines):



[11] We use the *Mate* dependency parser (version 3.6) downloaded from `http://code.google.com/p/mate-tools/`.

to the algorithm – the threshold of the non-projective approximation which indicates a minimal increase of the tree score after rearrangement of arcs.

Compared to a baseline parser with the MST-parser architecture presented in Bohnet (2010), the hash kernel implementation significantly shortens the parsing time by a faster estimation of feature weights. Furthermore, the *Mate* parser improves the accuracy by taking into account features of negative examples which are built during the training process. Moreover, this parser applies parallelisation across several CPUs. The parallelisation significantly accelerates the feature extraction and parsing.

4.5.3 Evaluation Methodology

To evaluate the quality of the converted dependency trees, we employ an extrinsic evaluation that consists in training a parser on converted data (the training set) and evaluating to what extent the converted trees affect performance of the parser. The trained parser is evaluated using the standard evaluation metrics: *labelled attachment score* (LAS) and *unlabelled attachment score* (UAS). LAS is the percentage of tokens that are assigned the correct governor and the correct dependency type. UAS, in turn, is the percentage of tokens that are assigned the correct governor.

In the standard evaluation setting, parsers are evaluated against 822 gold standard dependency structures (the validation set) with manually annotated part of speech tags and morphosyntactic features (Manual Test). Furthermore, in a more realistic scenario, parsers are evaluated against the validation trees with automatically assigned part of speech tags and morphosyntactic features (Automatic Test). For purposes of our experiments, an additional set of 100 sentences was manually annotated with dependency trees (Additional Test). Sentences in this set are quite long (16.6 tokens per sentence on average). Additional trees contain 2.8% of non-projective arcs. 50 sentences were randomly selected from the Polish part of the parallel corpus (see Section 5.4.1 *Data*) and 50 sentences were randomly selected from NKJP (33 sentences) and two Polish magazines. We annotated two excerpts from *Newsweek Polska* (13 sentences) and from *Zwierciadlo* (4 sentences). Sentences from magazines and the parallel corpus were automatically tokenised and tagged. The consequent morphosyntactic annotations were manually validated and corrected by two experienced linguists. NKJP sentences were already assigned morphosyntactic annotations. The same linguists manually annotated sentences with dependency trees.

4.6 Experiments and Results

4.6.1 Experiment 1 – *MaltParser*

For purposes of our experiments on training a *MaltParser* model, we used the built-in non-projective parsing algorithm *stackeager* and the LIBLINEAR learning algorithm. The results of experiments on dependency parser training reported in Wróblewska and Woliński (2012) show that the learning algorithm has a relatively insignificant impact on parser performance. An important advantage of the LIBLINEAR algorithm is its speed. The implementation of a machine learning algorithm based on linear classifiers (the LIBLINEAR package) is considerably faster than the implementation based on support vector machines (the LIBSVM library).[12] Therefore, the LIBLINEAR algorithm was used in our experiments.

Model	**Manual Test**		**Automatic Test**		**Additional Test**	
	UAS	LAS	UAS	LAS	UAS	LAS
Malt default	88.2	80.4	84.6	76.1	72.7	63.3
Malt optimised	90.5	85.4	85.3	78.4	73.3	66.1
Malt automatic	88.4	82.9	87.8	81.6	75.8	68.0
Mate default	**92.7**	**87.2**	88.4	81.0	76.0	69.5
Mate automatic	91.2	85.6	**90.8**	**84.7**	**76.6**	**70.1**

TABLE 4.1: Performance of parsers trained with *MaltParser* and *Mate* parsing systems on the Polish dependency structures converted from the constituent trees. Settings of model training: *Malt* default – the *MaltParser* model trained on trees with manual morphosyntactic annotations using the built-in *StackSwap* feature model; *Malt* optimised – the *MaltParser* model trained on trees with manual morphosyntactic annotations using the optimised feature model; *Malt* automatic – the *MaltParser* model trained on trees with automatic morphosyntactic annotations using the optimised feature model; *Mate* default – the *Mate* model trained on trees with manual morphosyntactic annotations in 10 iterations, with the default heap size and the threshold of 0.3; *Mate* automatic – the *Mate* model trained on trees with automatic morphosyntactic annotations in 10 iterations, with the default heap size and the threshold of 0.3. Validation data sets: Manual Test – the set of 822 conversion-based test trees; Automatic Test – the set of 822 conversion-based test trees with automatically assigned morphosyntactic annotations; Additional Test – the set of 100 sentences manually annotated with dependency trees.

[12] Learning a parsing model on 7405 dependency structures took 11 seconds when the LIBLINEAR learning algorithm was used and 103 seconds when the LIBSVM algorithm was used. Furthermore, the learning algorithm has also an indirect impact on the parsing time. Parsing of 822 test dependency structures with the LIBLINEAR-trained model took 2 seconds, while parsing of the same test set with the LIBSVM-trained model took 17 seconds.

The model *Malt default* was learnt with the LIBLINEAR algorithm using the built-in *stackeager* parsing algorithm and the *StackSwap* feature model included in the *MaltParser* distribution. Performance of the default *MaltParser* is given in the first row in Table 4.1.[13]

		FORM	POS	DEPREL	LEMMA	FEATS
Stack:	TOP	⊕	⊕		+	+
Stack:	TOP-1	⊕	⊕		+	+
Stack:	TOP-2		⊕			∗
Stack:	PRED(TOP)	∗	+			
Stack:	PRED(TOP-1)		+			
Stack:	SUCC(TOP)	∗	+			
Stack:	SUCC(TOP-1)		+			
Input:	NEXT		⊕		∗	∗
Input:	NEXT+1					∗
Input:	PRED(NEXT)	∗				
Lookahead:	LOOK	⊕	⊕		+	+
Lookahead:	LOOK+1		⊕			∗
Lookahead:	LOOK+2		⊕			∗
Lookahead:	PRED(LOOK)		+			
Lookahead:	SUCC(LOOK)		+			
Tree:	HEAD(TOP)	∗	∗	+		
Tree:	LDEP(TOP)	∗	∗	⊕		
Tree:	RDEP(TOP)	∗	∗	⊕		
Tree:	HEAD(TOP-1)	∗	∗	+		
Tree:	LDEP(TOP-1)	∗	∗	⊕		
Tree:	RDEP(TOP-1)	∗	∗	⊕		

TABLE 4.2: Repertoire of history-based features. Rows correspond to lexical nodes in a parser configuration: TOP – the lexical node on the top of the stack, NEXT – the next lexical node in the remaining input, PRED(NEXT/TOP) – the lexical node immediately preceding NEXT or TOP, SUCC(NEXT/TOP) – the lexical node immediately succeeding NEXT or TOP, LOOK – the next plus one input lexical node, HEAD(TOP) – the head of TOP in the partially built tree, LDEP(TOP/NEXT) – the leftmost dependent of TOP or NEXT), RDEP(TOP/NEXT – the rightmost dependent of TOP or NEXT. Columns correspond to feature types: FORM – word form, POS – part of speech, DEPREL – dependency relation, LEMMA – lemma, FEATS – morphological features. ⊕ – a default feature for the *StackSwap* feature model; + – an additional feature in the optimised feature model; ∗ – an additional feature tested in extended feature models.

The history-based feature model combines static features (word forms, lemmata, part of speech tags, morphological features) available in input data and dynamic features (dependency relations) extracted from partially built dependency trees and updated during parsing. The baseline feature model *StackSwap* contains the following features: FORM, POS and DEPREL. This baseline feature model can be extended with LEMMA and/or FEATS features. In order to improve performance of the Polish *MaltParser*, we conducted an experiment on adjusting model features. The default and additional features used in the experiment are presented in Table 4.2. Our results indicate that the best performing parser uses the feature model combining default features (marked with ⊕) with additional features (marked with +). Features marked with the asterisk (∗) were tested, but

[13] In this and the following paragraphs we will only consider the columns under Manual Test. The results from other columns will be referred to in the following sections.

they are not included in the final feature model. The feature model extended with some additional features contributes to the improvement of Polish parsing.

As shown in the second row in Table 4.1, the optimised *MaltParser* achieves 90.5% UAS and 85.4% LAS if tested against the validation trees with manual morphosyntactic annotations. There is a significant improvement of 5 percentage points over the default *MaltParser* in terms of LAS and of 2 percentage points in terms of UAS.

4.6.2 Experiment 2 – *Mate* Parser

In addition to experiments with the transition-based *MaltParser*, we conducted some experiments with a graph-based parser. We applied the *Mate* dependency parser for purposes of a comparative analysis between transition-based and graph-based parsing systems. We used the *Mate* parser with standard settings – 10 iterations and the default size of the weight vector (hence, *Mate* default). Since Polish admits non-projective dependency structures, we also set *Mate* parameters to decode non-projective arcs. The threshold of the non-projective approximation was set to 0.3.

The *Mate* parser trained on the converted Polish dependency structures outperforms the optimised *MaltParser* by nearly 2 percentage points in terms of LAS (see the third row in Table 4.1). As mentioned above, we used the default size of the weight vector (134,217,727). However, since there are only 3,132,538 nonzero values in the feature vector, it may indicate that training data is relatively sparse.

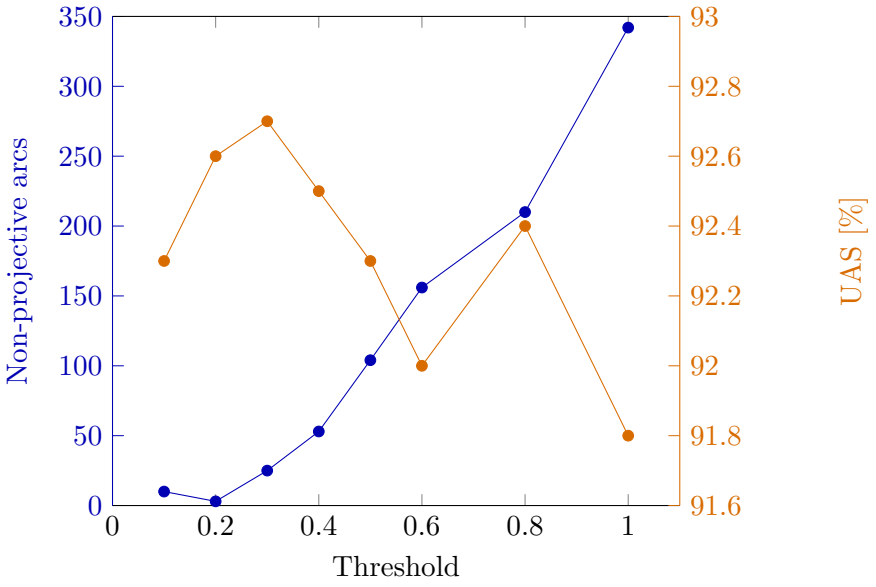


FIGURE 4.15: The Impact of the non-projective approximation thresholds (x-axis) on the number of non-projective arcs in the parsed test trees (blue y-axis on the left) and on parsing performance measured in UAS (orange y-axis on the right).

We also conducted an experiment to determine whether the approximation threshold of 0.3 is the best choice for Polish. Experiments carried out by Bohnet (2010) indicate that a threshold of about 0.3 is the best choice for German, English and Czech. Since both Czech and Polish belong to the group of West Slavic languages, we assume that they are similar in terms of their syntactic structures and should allow for a similar amount of non-projective dependency arcs. In accordance with the information on the *Mate* website (`http://code.google.com/p/mate-tools/wiki/Training`), a higher threshold causes fewer non-projective arcs. However, our experiments show the opposite (see Figure 4.15). Along with increasing the threshold value, we may notice an increase in the number of non-projective arcs in parsed dependency structures. Since our test set of 822 dependency trees contains only 38 non-projective arcs (i.e., about 0.46% of all 8289 arcs), the smaller thresholds ensure better parsing results.

We have to admit that differences in performance (UAS scores) of parsers trained with different thresholds are relatively insignificant. The *Mate* parser trained with the threshold of 0.3 achieves the best parsing results. It might be assumed at first that it is due to the number of reordered arcs (i.e., 25) which is closest to the number of gold standard non-projective arcs (38). However, a cursory analysis showed that non-projective arcs were correctly reorganised only in one test dependency structure. We also analysed dependency trees which were parsed with the *Mate* parser trained with the threshold of 0.5. These trees contain 104 non-projective arcs. Even if the number of non-projective arcs in parsed trees is significantly higher, we did not find any tree in which arcs were reorganised correctly. Therefore, even if the number of automatically reorganised non-projective arcs is relatively high, they do not cover the gold standard non-projective arcs. This confirms our conjecture that training data may not be sufficient to train a parser that correctly reorganises arcs in dependency trees.

4.6.3 Evaluation against Automatic and Additional Test Sets

Supervised dependency parsers build on correct dependency trees in the first place, but also on correct parts of speech and morphological features underlying dependency trees. Correct trees are essential for training parsers. Correct morphosyntactic annotations of tokens, in turn, are substantial both for training and parsing. But what happens in the parsing scenario if a raw text is first automatically tokenised and annotated with part of speech tags and morphological features, and then processed by a dependency parser trained on perfectly annotated data? It may not be an important issue for English, which is a language with a wide range of highly accurate tools. However, for languages like Polish it is certainly a problem since state-of-the-art tokenisers and taggers are still far from being accurate.

The evaluation scenario presented above assumes that sentences given to the parser are annotated with correct parts of speech, lemmata and morphological features. However,

sentences which are parsed in real NLP tasks are typically annotated automatically with state-of-the-art taggers. In order to evaluate parser performance in a more realistic scenario, manual morphosyntactic annotations in the test trees are replaced with annotations generated automatically by the *Pantera* tagger (Acedański, 2010).

The results of this evaluation task are given in the second column (Automatic Test) in Table 4.1. When tested against the set of 822 converted sentences with automatically assigned morphosyntactic annotations, a significant decrease in parsing performance can be noticed in comparison to the results of evaluation against the test trees with manual morphosyntactic annotations. The optimised *MaltParser* tested against the Automatic Test trees achieves 85.3% UAS and 78.4% LAS. The default *Mate* parser performs better (88.4% UAS and 81% LAS) than the optimised *MaltParser*. Parsing performance in terms of LAS may even decrease by 7 percentage points if there is noise in token annotations. It indicates that the quality of tagging and morphological analysis of individual tokens has a crucial impact on the dependency parsing of Polish and possibly other fusional languages. Nevertheless, a dependency parser which is trained on the conversion-based trees may assign a correct governor and a correct grammatical function to about 80% of all tokens from the Automatic Test set (LAS of 81% for the *Mate* parser and of 78.4% for *MaltParser*).

Dependency parsers are trained and evaluated on trees from the same source – the conversion-based treebank which consists of relative simple dependency structures. In order to validate the parsers on real data, we employ the set of 100 manually annotated test trees. The results of parser evaluation against the Additional Test trees are presented in the third column in Table 4.1. The parsing results are generally worse than those reported above. The default *Mate* parser obtains 76% UAS and 69.5% LAS and thereby it outperforms the optimised *MaltParser* which achieves 73.3% UAS and 66.1% LAS.

4.6.4 Experiment 3 – Automatic *Malt* and *Mate* Models

There is a significant decrease in performance of parsers evaluated against trees with noisy morphosyntactic annotations. One solution would be to enhance the underlying tools so that they could produce more adequate morphosyntactic analyses. Since this solution is beyond the scope of this dissertation, we verify whether it is possible to train a dependency parser on noisy data and whether such parser could predict better dependency structures. Hence, we conducted another experiment that consists in training a dependency parser on noisy data, i.e., the converted trees with automatic part of speech tags and morphological features assigned to tokens. As previously, *MaltParser* was trained using the optimised feature model and the *Mate* parser was trained in 10 iterations, with the default heap size and the non-projective approximation threshold of 0.3.

The results achieved by these parsers are given in Table 4.1 – in the third row for *MaltParser* (*Malt* automatic) and in the fifth row for the *Mate* parser (*Mate* automatic). There is no significant difference between results of evaluation against the Manual Test trees and the Automatic Test trees. The automatic *MaltParser* achieves 88.4% UAS and 82.9% LAS if evaluated against the Manual Test trees and 87.8% UAS and 81.6% LAS if evaluated against the Automatic Test trees. The automatic *Mate* parser, in turn, achieves 91.2% UAS and 85.6% LAS if evaluated against the Manual Test trees and 90.8% UAS and 84.7% LAS if evaluated against the Automatic Test trees. When evaluated against the Manually Test trees, the 'automatic' parsers perform worse than the parsers trained on fully correct data (*Malt* optimised and *Mate* default). When evaluated against the Automatic Test trees, the 'automatic' parsers outperform parsers trained on trees with correct morphosyntactic annotations of tokens (*Malt* optimised and *Mate* default). The 'automatic' parsers evaluated against the Additional Test trees perform rather poorly but at least better than the default *Mate* parser and the optimised *MaltParser*.

These results show that it is possible to train dependency parsers on possibly noisy data. The parser trained on the converted trees with automatically annotated tokens analyses both manually and automatically annotated sentences almost equally well.

4.6.5 Evaluation of Individual Relation Labels

As the right choice of dependency labels affects the parsing quality measured with the labelled attachment score (LAS), we evaluate parsers for appropriateness of labels assigned to dependency relations in parse trees. Individual dependency labels are evaluated in the Manual Test trees and in the Automatic Test trees in terms of precision, recall and f-measure. Precision is the empirical probability that a label induced by a parser is correct. Recall is the empirical probability that a correct label is induced. F-measure, in turn, is the harmonic mean of precision and recall. Results are presented in two tables: Table 4.3 for dependency labels assigned by parsers trained on the converted trees with manually annotated tokens (*Malt* optimised and *Mate* default) and Table 4.4 for labels assigned by parsers trained on the converted trees with automatically annotated tokens (*Malt* automatic and *Mate* automatic).

Dependency Relation		Manual Test						Automatic Test					
		Malt optimised			Mate default			Malt optimised			Mate default		
Label	**Frequency**	precision	recall	f-score	precision	recall	f-score	precision	recall	f-score	precision	recall	f-score
adjunct	2185	0.80	0.83	0.81	0.82	0.85	0.83	0.72	0.78	0.75	0.72	0.81	0.76
comp	1311	0.89	0.90	0.90	0.90	0.90	0.90	0.88	0.85	0.86	0.87	0.83	0.85
punct	1159	0.85	0.87	0.86	0.87	0.90	0.89	0.78	0.81	0.80	0.83	0.87	0.85
pred	770	0.95	0.92	0.94	0.96	0.94	0.95	0.91	0.88	0.89	0.94	0.92	0.93
subj	608	0.91	0.90	0.90	0.92	0.93	0.92	0.80	0.74	0.77	0.83	0.78	0.80
conjunct	455	0.77	0.81	0.79	0.85	0.84	0.84	0.68	0.76	0.72	0.80	0.80	0.80
obj	419	0.87	0.90	0.88	0.89	0.91	0.90	0.72	0.78	0.75	0.79	0.78	0.78
obj_th	197	0.87	0.76	0.81	0.87	0.76	0.81	0.73	0.65	0.69	0.80	0.68	0.73
refl	166	0.96	0.99	0.97	0.96	0.99	0.98	0.95	0.97	0.96	0.96	0.98	0.97
ne	120	0.78	0.53	0.63	0.80	0.50	0.61	0.54	0.22	0.31	0.74	0.27	0.39
neg	119	1.00	0.99	1.00	0.99	0.99	0.99	1.00	0.99	1.00	0.99	0.99	0.99
comp_inf	112	0.94	0.91	0.92	0.94	0.91	0.93	0.88	0.82	0.85	0.92	0.89	0.90
comp_fin	101	0.78	0.85	0.81	0.84	0.81	0.83	0.69	0.74	0.72	0.80	0.74	0.77
pd	101	0.84	0.67	0.75	0.82	0.65	0.73	0.76	0.65	0.70	0.76	0.61	0.68
mwe	86	0.94	0.79	0.86	0.97	0.86	0.91	0.80	0.45	0.58	0.85	0.51	0.64
item	65	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
complm	60	0.87	0.87	0.87	0.89	0.92	0.90	0.87	0.87	0.87	0.88	0.90	0.89
aglt	59	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.96	0.95	0.96
aux	51	0.83	0.96	0.89	0.84	0.94	0.89	0.80	0.92	0.85	0.84	0.92	0.88
app	43	0.68	0.44	0.53	0.69	0.51	0.59	0.54	0.28	0.37	0.61	0.32	0.42
abbrev_punct	29	1.00	1.00	1.00	1.00	1.00	1.00	0.90	0.62	0.73	0.90	0.62	0.73
coord	26	0.64	0.61	0.63	0.70	0.61	0.65	0.56	0.58	0.57	0.64	0.54	0.58
coord_punct	26	0.31	0.77	0.44	0.31	0.73	0.44	0.25	0.58	0.34	0.25	0.61	0.35
cond	11	0.91	0.91	0.91	0.91	0.91	0.91	0.91	0.91	0.91	0.91	0.91	0.91
comp_ag	9	0.87	0.78	0.82	0.87	0.78	0.82	0.87	0.78	0.82	0.87	0.78	0.82
pre_coord	1	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	1.00	1.00	1.00
average:		0.85	0.85	0.85	0.87	0.87	0.87	0.78	0.79	0.78	0.81	0.81	0.80

TABLE 4.3: Parsing performance in terms of accuracy of individual dependency types assigned by *MaltParser* (*Malt* optimised) and the *Mate* parser (*Mate* default) trained on the converted dependency structures with manually annotated tokens. Validation sets: Manual Test – the set of 822 conversion-based dependency structures with manual morphosyntactic annotations of tokens; Automatic Test – the set of 822 converted dependency structures with automatic morphosyntactic annotations of tokens; average – weighted arithmetic mean.

Dependency Relation		Manual Test						Automatic Test					
		Malt automatic			Mate automatic			Malt automatic			Mate automatic		
Label	**Frequency**	precision	recall	f-score	precision	recall	f-score	precision	recall	f-score	precision	recall	f-score
adjunct	2185	0.77	0.81	0.79	0.79	0.85	0.82	0.76	0.79	0.78	0.79	0.83	0.81
comp	1311	0.88	0.89	0.88	0.90	0.90	0.90	0.87	0.88	0.87	0.91	0.89	0.90
punct	1159	0.82	0.86	0.84	0.85	0.91	0.88	0.81	0.85	0.83	0.85	0.90	0.88
pred	770	0.92	0.90	0.91	0.95	0.94	0.94	0.92	0.90	0.91	0.95	0.93	0.94
subj	608	0.87	0.88	0.87	0.89	0.90	0.90	0.81	0.82	0.81	0.84	0.87	0.85
conjunct	455	0.75	0.76	0.76	0.82	0.81	0.82	0.74	0.74	0.74	0.80	0.80	0.80
obj	419	0.83	0.86	0.85	0.85	0.85	0.85	0.76	0.80	0.78	0.82	0.77	0.79
obj_th	197	0.81	0.69	0.75	0.81	0.73	0.77	0.83	0.64	0.72	0.79	0.69	0.73
refl	166	0.96	0.99	0.97	0.96	0.99	0.98	0.96	0.97	0.97	0.96	0.99	0.97
ne	120	0.75	0.31	0.44	0.79	0.37	0.50	0.78	0.47	0.58	0.82	0.53	0.65
neg	119	1.00	0.99	1.00	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
comp_inf	112	0.90	0.87	0.89	0.93	0.90	0.91	0.91	0.87	0.89	0.92	0.90	0.91
comp_fin	101	0.72	0.82	0.77	0.81	0.78	0.79	0.74	0.80	0.77	0.78	0.78	0.78
pd	101	0.84	0.67	0.75	0.80	0.67	0.73	0.78	0.62	0.69	0.79	0.63	0.70
mwe	86	0.99	0.52	0.68	0.91	0.56	0.69	0.94	0.70	0.80	0.96	0.84	0.89
item	65	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
complm	60	0.85	0.87	0.86	0.82	0.82	0.82	0.86	0.85	0.86	0.82	0.83	0.83
aglt	59	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
aux	51	0.79	0.98	0.88	0.81	0.90	0.85	0.79	0.96	0.87	0.80	0.88	0.84
app	43	0.58	0.44	0.50	0.76	0.44	0.56	0.70	0.44	0.54	0.69	0.51	0.59
abbrev_punct	29	0.96	0.96	0.96	0.93	1.00	0.97	0.93	1.00	0.97	1.00	1.00	1.00
coord	26	0.62	0.58	0.60	0.82	0.54	0.65	0.67	0.61	0.64	0.81	0.50	0.62
coord_punct	26	0.38	0.65	0.48	0.41	0.69	0.51	0.32	0.54	0.41	0.40	0.73	0.52
cond	11	0.91	0.91	0.91	0.69	0.82	0.75	0.91	0.91	0.91	0.75	0.82	0.78
comp_ag	9	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89
pre_coord	1	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	1.00	1.00	1.00
average:		0.82	0.83	0.82	0.85	0.86	0.85	0.81	0.81	0.81	0.84	0.84	0.84

TABLE 4.4: Parsing performance in terms of accuracy of individual dependency types assigned by *MaltParser* (*Malt* automatic) and the *Mate* parser (*Mate* automatic) trained on the converted dependency structures with automatically annotated tokens. Validation sets: Manual Test – 822 converted dependency structures with manually assigned parts of speech and morphological features; Automatic Test – the same set of 822 converted dependency structures with parts of speech and morphological features automatically automatically predicted by the *Pantera* tagger; average – weighted arithmetic mean.

Generally, the *Mate* parser assigns labels more precisely than *MaltParser* – the weighted arithmetic means of precision and recall of the *Mate* parser are higher than the weighted arithmetic means of precision and recall of *MaltParser*. Furthermore, regardless of data which the parser is trained on (i.e., trees with noise or without noise in token annotations), parsers assign more accurate labels to arcs in trees predicted for manually annotated sentences (Manual Test) than to arcs in trees predicted for automatically annotated sentences (Automatic Test). However, the difference is insignificant if we compare average F-scores obtained by *MaltParser* – 0.82 vs. 0.81 – and the *Mate* parser – 0.85 vs. 0.84 – trained on the trees with automatically annotated tokens. The worst labelling results are obtained in the parsing scenario in which parsers were trained on gold standard trees and evaluated against the Automatic Test trees. It is important to note that this parsing scenario is most commonly applied to sophisticated NLP tasks.

Dependency relations occurring more than 100 times in the test trees (except for the *ne* relation type) are assigned labels quite accurately with precision and recall often above 80% or even 90%. Furthermore, a lot of less frequent labels have precision and recall above 70%. The results show that there are some dependency relations particularly easy to label, e.g., *refl*, *neg*, or *aglt*. On the other hand, there are also syntactic phenomena which are problematic to annotate and to label, e.g., apposition (*app*), coordination (*coord*, *coord_punct*, *pre_coord*) or named entites (*ne*). The reasons for the poor quality of labelling these constructions could be that they are too sparsely represented in the training corpus or they are not uniquely identified based on their morphosyntactic characteristics.

Precision and recall of several dependency labels are not only relatively high but also often balanced. Hence, the assigned labels are accurate as the majority of them are appropriate and complete as the majority of appropriate labels are induced. However, there are some dependency labels with unbalanced precision and recall values. In some cases, e.g., *coord*, *ne*, *pd*, *mwe*, or *app*, precision is much higher than recall. It suggests that only a small part of relations of a particular type is assigned a relevant label. Other relations of this type are labelled with incorrect grammatical functions. In the case of *coord_punct*, in turn, precision is much lower than recall. It means that multiple relations are labelled with the grammatical function *coord_punct*, but many of these labels are wrongly assigned.

There are also two dependency labels – *item* and *pre_coord* – which obtain precision and recall of 0. The grammatical function *item* was not automatically assigned during constituency-to-dependency conversion. It was added later when a part of the treebank was manually corrected. Thus, the *item* label only appears in the first 1000 dependency trees. Since 822 of these manually corrected trees constitute the validation set, it is probable that there are only a few (or none) *item* instances in the training set. The *pre_coord* relation has only one realisation in the test trees. It is identified by the *Mate* parser but not by *MaltParser*.

4.7 Constituency-to-Dependency Conversion: Related Work

With increasing interest in data-driven dependency parsing, new methods of gathering annotated data have been explored since manual annotation of large corpora is a very expensive and time-consuming process. As constituency treebanks extended with dependency information are available for some languages, e.g., the *Penn Treebank II* for English (Marcus et al., 1994) or the *TIGER Treebank* for German (Brants et al., 2002), it may be a reasonable solution to convert extended phrase structure trees into dependency structures.

The basic constituency-to-dependency conversion procedure usually consists of three steps. First, heads of all constituents are identified in a phrase structure tree. In order to find a head of a constituent, a *head percolation table* that consists of priority lists based on constituent types is employed. The concept of a head percolation table was introduced by Magerman (1994, 1995) to improve performance of a constituent parser. Afterwards, Magerman's rules were modified by Collins (1999, 2003) to further improve the parsing performance. An example rule (taken from Collins, 1999, Appendix A) – `VP` → `TO VBD VBN MD VBZ VB VBG VBP VP ADJP NN NNS NP` – determines which of VP's children should be its head. The head selection algorithm iterates over VP's children starting from the left-most child and tries to find a child of the type `TO`. If it does not find any `TO` child, it searches for the next type – `VBD`. The algorithm continues searching until the first head child with a type from the priority list is found. If none of the types are found, the left-most child is chosen as the default head.

Yamada and Matsumoto (2003) propose some further modifications of the table leading to an improvement of the conversion procedure. Choi and Palmer (2010) design a new set of head percolation rules to better identification of heads. Gelbukh et al. (2005), in turn, define some heuristic rules which mark a head in automatically extracted patterns. These patterns are actually combinations of phrase structure categories of nodes and of their children nodes. The concept of a pattern is similar to the idea of the head percolation table, but patterns are obtained automatically from constituent trees.

In the procedure of converting Polish constituent trees into dependency trees, which is described in the previous sections of this chapter, we did not have to define head selection rules for all constituents since heads of many phrase structures were already denoted. Head selection heuristics covering other cases are described in Section 4.3 *Head Selection*.

Second, each token of a constituent tree is connected with its head token. Starting from a token and following child-parent links in the bottom-up direction, the highest constituent headed by this token is found. The current token depends on the head token of the parent of the found constituent and this relation is transferred to a dependency

tree. For example, in order to find a governor of the token *they* in the constituent tree in Figure 4.16, the highest constituent `NP` headed by this token is identified. The parent of the constituent `NP` is labelled with `S`. The head token *wonder* of the `S` constituent is identified as the governor of *they*. Hence, the dependency relation $they \xleftarrow{\text{SUB}} wonder$ is transferred to the dependency tree (the label of the dependency relation SUB is derived from the grammatical function SBJ encoded in the Penn tree). Section 4.2.2 *Unlabelled Dependency Relations* describes the implementation of this step for Polish.

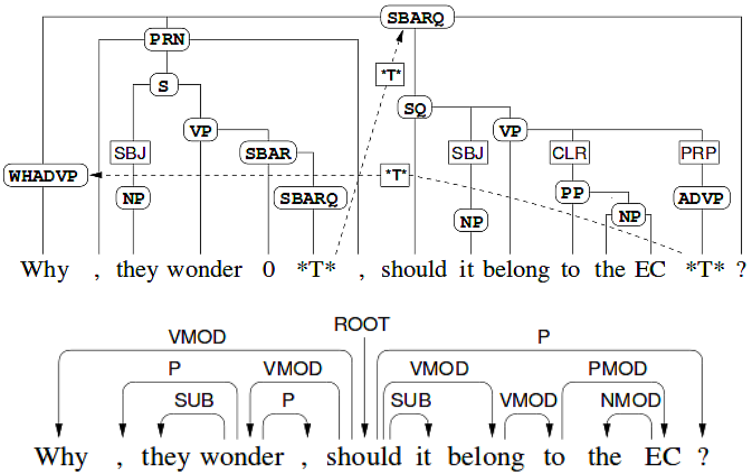


FIGURE 4.16: The conversion of the top constituent tree from the *Penn Treebank* into the bottom dependency tree using the PENN2MALT procedure. The example trees are taken from Johansson and Nugues (2007).

Third, converted dependency relations are labelled with appropriate grammatical functions. The earliest research on constituency-to-dependency conversion paid more attention to the accurate selection of constituents' heads and kept aside the issue of appropriate labelling dependency relations. Therefore, converted dependency structures remained unlabelled. First methods of inferring grammatical functions used to annotate arcs in converted dependency structures were proposed by Nivre (2006) and de Marneffe et al. (2006). The PENN2MALT converter by Nivre (2006) is a reimplementation of the conversion model by Yamada and Matsumoto (2003) with some heuristic rules to label arcs with appropriate dependency types. In order to label dependencies with grammatical functions, de Marneffe et al. (2006) propose a set of hand-crafted rules using *tregex* patterns (Levy and Andrew, 2006). The conversion-based Polish dependency structures were labelled with rules described in Section 4.2.3 *Labelling Dependency Relations*.

Further modifications of the conversion procedure by Yamada and Matsumoto (2003) are proposed by Johansson and Nugues (2007). Johansson and Nugues notice that not all of the syntactic facts encoded in the *Penn Treebank II* are taken into account in the previous conversion approaches. For example, phenomena such as wh-movement, topicalisation, discontinuous constituents, it-clefts, expletives, gaping, etc., annotated with secondary edges in the *Penn Treebank II* are not converted into dependency structures. Hence, they propose a method of converting these phenomena. However, as secondary

edges are hard to encode in the format used by *MaltParser* and *MST* parser employed in their experiments, the conversion is conducted at the expense of primary edges. A modification of dependency structures consists in converting some of the phrase structure's secondary edges into dependency arcs (non-projective dependency structures are permitted) and deleting some primary edges to keep dependency graphs in accordance with the well-formedness dependency principles. Johansson and Nugues (2007) also introduce some further modifications to head percolation rules defined by Yamada and Matsumoto (2003). First of all, rules are based not only on constituent types but also on grammatical functions and the context of a phrase. Furthermore, the presented repertoire of dependency labels is extended in comparison to the set of labels used by the PENN2MALT converter. Property labels (e.g., LOC – location, MNR – manner) describing properties of constituents in Penn trees and structural labels (e.g., EXP – expletive, CLF – cleft, GAP – ellipses in coordinate structures) are established as dependency types. Results of experiments performed by Johansson and Nugues (2007) indicate that the modified conversion procedure and the extended set of grammatical functions cause a decrease in parsing performance, but they have a positive impact on semantic role classification.

Apart from experiments on converting the English *Penn Treebank* (cf. Yamada and Matsumoto, 2003; Nivre, 2006; Johansson and Nugues, 2007), constituency-to-dependency conversion is employed to derive dependency structures from constituency treebanks in other languages, e.g., from the German *NEGRA* corpus (Bohnet, 2003), from the Spanish *Cast3LB* treebank (Gelbukh et al., 2005), from the Swedish *Talbanken* treebank (Nivre et al., 2006b) and from the Bulgarian *BulTreeBank* (Chanev et al., 2006; Marinov, 2009).

4.8 Partial Conclusions

The procedure of converting Polish constituent trees described in this chapter is consistent, to a certain degree, with conversion procedures proposed for English and many other languages. However, as considered languages vary, the head selection rules and labelling rules had to be designed for Polish anew. We would like to draw attention to the structural complexity of the labelling rules. Since Polish is a fusional language with numerous inflected forms, rules could not solely rely on parts of speech and topological features as in English or other isolated languages, but they also took into account morphological features of related tokens, subcategorisation information and types of construction rules used to build the source constituent trees.

Unlike previous conversion approaches, we did not have to focus first and foremost on the definition of head selection rules since information about the head of the majority of phrase structures was already encoded in *Składnica* trees. The availability of phrase structure heads was undoubtedly an important advantage in the first phase of the conversion procedure. On the other hand, the pre-defined phrase structure heads made it

necessary to rearrange acquired dependency trees, so that they became as consistent with the dependency annotation schema as possible. Application of the rearrangement rules is a novelty compared to the aforementioned conversion approaches.

In order to evaluate the acquired dependency structures, we performed an extrinsic evaluation that consisted in training a dependency parser on the converted trees and in evaluating performance of the parser. As the evaluation experiments showed, it is possible to train a Polish dependency parser on a relatively small set of constituency-to-dependency converted trees. The best Polish parser trained with the *Mate* parsing system achieved quite high performance: 92.7% UAS and 87.2% LAS.

However, there was a precondition that the *Mate* parser was trained and evaluated on trees with manually annotated tokens. As this precondition cannot be met in the case of Polish sentences which are fully automatically processed, we conducted an evaluation experiment that consisted in evaluation of the parser against the conversion-based test trees with tokens annotated with automatically predicted part of speech tags and morphological features. In this evaluation setting, performance of the parser decreased even by 6 percentage points. It indicates that morphological analysis and tagging that precede dependency parsing have a significant impact on parser performance.

Furthermore, if a dependency parser was trained on the conversion-based trees with automatically annotated tokens, it parsed sentences with manually and automatically annotated tokens almost equally well. The parser trained on the converted trees with automatically annotated tokens performed as well as the parser trained on the converted trees with manually annotated tokens if evaluated against the test trees with automatically annotated tokens.

Presented parsing models were also evaluated against an additional validation set of 100 complex dependency structures. In this more realistic evaluation scenario, the *Mate* parser trained on the converted trees with automatically assigned morphosyntactic annotations of tokens beat other dependency models, reaching 76.6% UAS and 70.1% LAS. To the best of our knowledge, this kind of comparative analysis has not been carried out with respect to Polish syntactic parsers.

Chapter 5

Projection-based Dependency Bank

Supervised methods are very well-established in data-driven dependency parsing and they give the best results so far. However, the manual annotation of training data required by supervised frameworks is a very time-consuming and expensive process. For this reason, intensive research has been conducted on unsupervised grammar induction. However, performance of unsupervised dependency parsers is still significantly below performance of supervised systems. Moreover, performance of unsupervised parsers is also substantially below performance of systems based on cross-lingual projection methods (McDonald et al., 2011).

The cross-lingual projection of linguistic information (so-called *annotation projection*) is an alternative method of annotating sentences with dependency trees in less researched languages. The method builds on the assumption that the dependency analysis of a sentence largely carries over to its translation since valency relations encoded in dependency structures are relatively invariant across languages. A sentence in one language and its translation in another language tend to have not only parallel semantic structures but also correlated syntactic structures. The cross-lingual projection of source language dependencies to the target language does not take into account the order of words. It is thus perfectly suited for projection between languages with different word order.

The main idea behind the cross-lingual dependency projection is to automatically parse source sentences and to project acquired dependency trees to equivalent target sentences. Since relations encoded in dependency structures connect tokens, projection of these relations may be sufficiently guided by word alignment which links corresponding tokens in parallel sentences. In the ideal case, projected dependency relations constitute valid dependency structures of the target sentences. Projected annotations can be used to train dependency parsers for the target language.

This chapter describes a novel weighted induction method of obtaining Polish dependency structures. In a parallel English-Polish corpus, the English side is automatically annotated with a syntactic parser and resulting annotations are transferred to Polish equivalent sentences. Dependencies are projected via an extended set of word alignment links. Projected arcs are initially weighted according to the certainty of word alignment links used in projection. In the induction step, initial arc weights are recalculated using a method based on the expectation maximisation (EM) algorithm. Maximum spanning trees that conform to requirements of well-formed dependency trees are then extracted from digraphs containing all projected arcs with recalculated weights. Dependency trees selected from EM-scored digraphs constitute a treebank that may be employed to train dependency parsers for Polish. The novelty of the method proposed here consists in involving a weighting factor in processes of projecting dependency relations and inducing dependency trees.

The weighted induction procedure consists of two successive processes – projection of dependency relations (Section 5.1 *Weighted Projection*) followed by inference of dependency structures from projected directed graphs (Section 5.2 *Weighted Induction*). Induced trees constitute a bank of unlabelled Polish dependency structures. To create a bank of labelled dependency structures, arcs of induced trees are assigned labels corresponding to grammatical functions of dependents (Section 5.3 *Rule-based Adaptation of Polish Dependency Structures*). The final dependency treebank is employed for training a Polish dependency parser (Section 5.4 *Experimental Setup* and Section 5.5 *Experiments and Results*). This work is set within the mainstream of the study on cross-lingual information projection. Some relevant or important dependency projection frameworks are thus outlined in Section 5.6 *Annotation Projection: Related Work*. Section 5.7 *Partial Conclusions* closes this chapter.

5.1 Weighted Projection

This section describes weighted projection which is the first step in the entire process of acquiring valid Polish dependency structures. As word alignment is a crucial issue in projection of linguistic information, Section 5.1.1 *Bipartite Alignment Graph* starts with a presentation of the idea of word alignment and some related issues. Subsequently, the concept of *bipartite alignment graph* is outlined. Instead of projecting relations via links of single word alignment, dependency relations are projected via links gathered from different automatic word alignments and extended with some additional links. The extended set of alignment links that connect tokens of parallel sentences constitutes a bipartite alignment graph. Bipartite edges are weighted with scores indicating their certainty. The next main topic of this section is the weighted projection procedure outlined in Section 5.1.2 *Projection of Dependency Relations*. English relations are

projected via weighted edges of bipartite alignment graphs to the equivalent Polish sentences even if it results in directed graphs with multiple arcs between two tokens, i.e., multi-digraphs. A multi-digraph spans over all tokens of a Polish sentence. All projected arcs in the multi-digraph are weighted with initial scores using an intuitive weighting method which is almost solely based on weights of bipartite edges and the projection frequency. The heuristic of assigning initial weights to arcs of the projected multi-digraph is described in Section 5.1.3 *Intuitive Weighting Method*.

5.1.1 Bipartite Alignment Graph

The idea of word alignment comes from the large field of statistical machine translation. Word alignment as introduced by Brown et al. (1993) is an intermediate result of statistical machine translation. Different alignment models have been defined since then, e.g., IBM models (Brown et al., 1993) or a hidden Markov model (Vogel et al., 1996). An alignment model determines minimal translational correspondences between words of a sentence in one language and words of a sentence in another language. However, alignments are not given in advance. They need to be revealed in the translation process. Hence, alignments between corresponding sentences are treated as hidden variables in the alignment model. Parameters of the alignment model with hidden variables may be estimated with the expectation maximisation algorithm (Dempster et al., 1977). In the expectation step, the current model is applied to data and missing alignment links between foreign and target words are filled with the most likely values. In the maximisation step, counts for word translations over all weighted alignments are collected and a new probability distribution is estimated given these counts. The model parameters are updated and the updated model can be used in the next iteration. IBM models may run successively, i.e., parameters of the current model are given as prior parameters of a higher-order model. The Viterbi alignment, which is assigned the highest probability in the last iteration, constitutes the output of model learning.

The bilingual word alignment is a mapping between any foreign sentence $\mathbf{f} = f_1, ..., f_m$ and its target translation $\mathbf{e} = e_1, ..., e_l$ in a bitext.[1] An alignment $\mathbf{a}$ indicates which target word e originates from which foreign word f. An alignment $\mathbf{a}$ may be defined as a sequence of values $\{a_1, a_2, ..., a_m\}$ for each foreign word f_j, where each a_j indicates the index of the aligned target word e_i, for $i \in \{1, ..., l\}$. Any of alignment variables a_j can take any value in $\{0, 1, ..., l\}$, where 0 corresponds to an extra word NULL. Since

[1] Tiedemann (2011, p. 7) gives the following description of the term *bitext*:

A bitext $\boldsymbol{B} = (\boldsymbol{B}_{src}, \boldsymbol{B}_{trg})$ is a pair of texts $\boldsymbol{B}_{src}$ and $\boldsymbol{B}_{trg}$ that correspond to each other in one way or another. Correspondence can, for example, be *translational equivalence* in the case of bilingual bitexts. Even though the indexes *src* and *trg* commonly refer to *source language* and *target language* respectively, we usually do not require that one half of the bitext is the original source text and the other half is the target text that has been produced on the basis of that source text. However, it is often convenient to think of source and target texts when talking about bitexts. Correspondence between the texts is treated as a symmetric relation between both halves of a bitext.

each foreign word f_j can be aligned with a single target word e_i or the additional NULL token, there are $l+1$ possible alignments for each foreign word. As there are m words in the foreign sentence, there are $(l+1)^m$ possible values for $\mathbf{a}$, i.e., $(l+1)^m$ possible alignments between $\mathbf{f}$ and $\mathbf{e}$. The idea behind the alignment model is to define the probability distribution over the set $\mathcal{A}$ of all possible alignments between $\mathbf{f}$ and $\mathbf{e}$ (see Equation (5.1)) and to compute the most likely alignment $\mathbf{a'}$ (see Equation (5.2)).

$$p(\mathbf{a}|\mathbf{f},\mathbf{e},m) = \frac{p(\mathbf{f},\mathbf{a}|\mathbf{e},m)}{\sum_{\mathbf{a}\in\mathcal{A}} p(\mathbf{f},\mathbf{a}|\mathbf{e},m)} \tag{5.1}$$

$$\mathbf{a'} = \underset{\mathbf{a}}{\operatorname{argmax}}\, p(\mathbf{a}|\mathbf{f},\mathbf{e},m) \tag{5.2}$$

We have already said that each foreign word f may be aligned with one target word e or the NULL word. However, such alignment links are not sufficient to cover all translational correspondences. We illustrate this with the example given in Figure 5.1. Two top matrices correspond to the Polish-to-English word alignment (the top left matrix) and the English-to-Polish word alignment (the top right matrix). Assuming that *jest* is a translation of *he's*, it is desirable to align the Polish token *jest* with two English tokens *he* and *'s*. It is possible in the English-to-Polish word alignment since any English token may be aligned with one Polish token (or the NULL token), i.e., *he* is aligned with *jest* and *'s* is aligned with *jest*. On the other hand, it is not possible to align *jest* with *he* and *'s* in the Polish-to-English word alignment since a Polish token may be aligned with only one English token (or the NULL token). Hence, the English-to-Polish word alignment should be selected to cover translational correspondences in this sentence pair. However, the English-to-Polish word alignment cannot cover other correspondences between these languages since one English token may not be aligned with multiple Polish tokens (e.g., Eng. *simply* vs. Pol. *po prostu*, literary *by* $just_{adj}$). There are well-known typological differences between considered languages[2] and unidirectional word alignments are not sufficient to cover relevant linguistic phenomena. For example, a Polish noun marked for genitive may correspond to an *of*-prepositional phrase in English (e.g., Pol. *Statua Wolności*, lit. $Statue_{nom}$ $Liberty_{gen}$, vs. Eng. *Statue of Liberty*), a Polish prepositional phrase may correspond to an adverb in English (e.g., aforementioned Pol. *po prostu* vs. Eng. *simply*), the number of tokens in multiword expressions may differ (e.g., Pol. *zgodnie z*, lit. $appropriately_{adv}$ *with*, vs. Eng. *in accordance with*), etc. It follows from the above that unidirectional word alignment cannot cover translational correspondences between one source token and multiple target tokens, or multiple source tokens and multiple target tokens. It is a major limitation of IBM models that they are not able to generate such links even if translational discrepancies between languages are frequent.

[2]Polish is an inflecting language with a relatively free word order; English is an isolating language with the topological argument marking.

Since their conceptualisation by Brown et al. (1993), IBM models have been extended and word alignment induced with these models has improved. In order to improve word alignment quality and overcome the limitation of unidirectional word alignment, a method of symmetrisation of unidirectional alignments was proposed by Och and Ney (2003). The symmetrisation heuristic assumes that two unidirectional word alignments ($A_{\mathbf{e}\to\mathbf{f}}$ and $A_{\mathbf{f}\to\mathbf{e}}$) are combined together. If symmetrisation is applied as a post-processing step of model learning, the quality of word alignment improves.

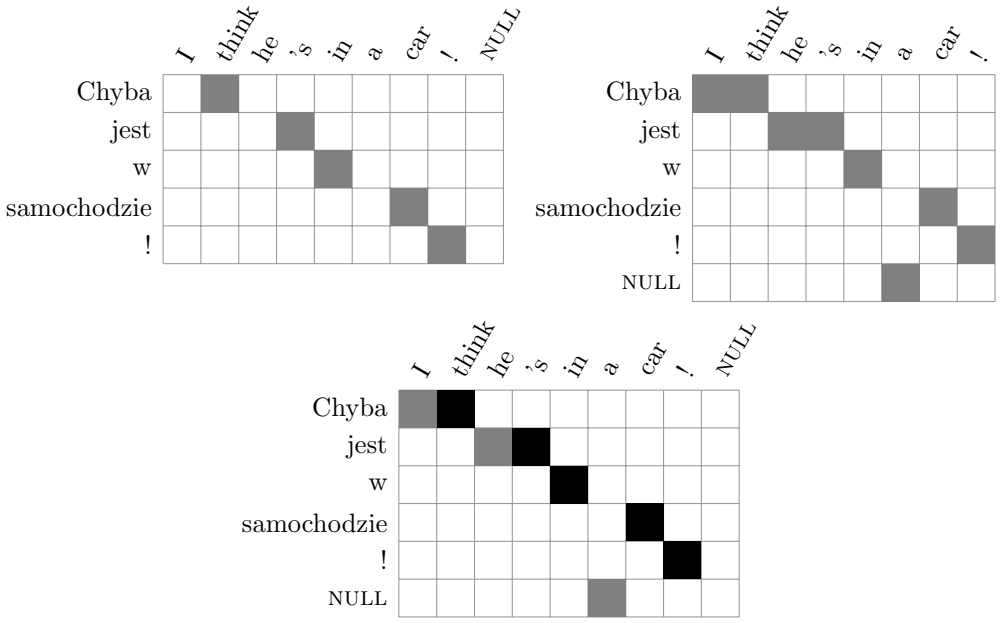


FIGURE 5.1: Symmetrisation of unidirectional word alignments: Polish-to-English word alignment (top left matrix); English-to-Polish word alignment (top right matrix); the symmetrisation (bottom matrix) of both unidirectional word alignments: intersection links – black squares, union links – black and grey squares.

The most fundamental symmetrisation methods are union and intersection (Och and Ney, 2003). The union set contains all alignment points that occur in either of the unidirectional word alignments ($A_{\cup} = A_{\mathbf{e}\to\mathbf{f}} \cup A_{\mathbf{f}\to\mathbf{e}}$). The intersection set contains all alignment points coexisting in both unidirectional word alignments ($A_{\cap} = A_{\mathbf{e}\to\mathbf{f}} \cap A_{\mathbf{f}\to\mathbf{e}}$). The union set is characterised by high recall and low precision scores. The intersection links, in turn, have high precision and low recall. An example of symmetrising unidirectional word alignments with the union and intersection heuristics is given in the bottom matrix in Figure 5.1.

Next to union and intersection, there exist some rather complex symmetrisation concepts. The `grow-diag-final-and` symmetrisation method (Koehn, 2010) iteratively extends the set of established alignment links. In the first step, all intersection alignment links are selected. Then, the set of intersection alignment links is extended with other links from union of unidirectional word alignments. The `grow` step adds some links connecting words at least one of which is currently unaligned. The `grow`-links have to be adjacent (left, right, top, bottom) to established alignment links. Similarly, the `diag`

step adds some links that connect words at least one of which is currently unaligned and they are diagonally adjacent to already established links. The `final` step adds alignment links not adjacent to established alignment links. The `final`-links relate two words at least one of which is currently unaligned. In the `and` step, not adjacent alignment links between two unaligned words are added. The growing method that symmetrises unidirectional word alignments derived from the IBM model training is implemented in the statistical machine translation system MOSES (Koehn et al., 2007).

Since we aim to project relations which are restricted to sentence boundaries, only word alignment links within a pair of aligned parallel sentences are considered in projection of dependency structures. In our projection scenario, we make use of both unidirectional word alignments and the set of bidirectional word alignment links symmetrised with the `grow-diag-final-and` heuristic (henceforth referred to as **gdfa**). The three word alignment sets ($A_{\mathbf{pl}\rightarrow\mathbf{en}}$, $A_{\mathbf{en}\rightarrow\mathbf{pl}}$ and $A_{\mathbf{gdfa}}$) are referred to while scoring edges $E = V_{\mathbf{en}} \times V_{\mathbf{pl}}$ of a complete bipartite graph $BG = (V_{\mathbf{en}} \cup V_{\mathbf{pl}}, E)$ which is built for each sentence pair. Since edges of the graph BG partially correspond to links from alignment sets, we refer to BG as *bipartite alignment graph*. Bipartite alignment graphs are used to project English dependencies to Polish corresponding sentences.

Vertices in a bipartite alignment graph are decomposed into two disjoint sets $V_{\mathbf{en}}$ and $V_{\mathbf{pl}}$ corresponding to English tokens together with an additional ROOT node and Polish tokens with a ROOT node respectively. Each edge (u, v) of BG has its one end u in $V_{\mathbf{en}}$ and another one v in $V_{\mathbf{pl}}$. Any two distinct vertices u and v, for $u \in V_{\mathbf{en}}$ and $v \in V_{\mathbf{pl}}$, are joined by an edge, i.e., every pair of graph vertices from $V_{\mathbf{en}} \times V_{\mathbf{pl}}$ is adjacent.

Edges in a bipartite alignment graph are labelled in accordance with their occurrence in the three word alignment sets. If an edge connecting an English token with a Polish token occurs in one of the word alignment sets, it is presumed to be labelled as *true*. Otherwise, it is labelled as *false*. Since there are 3 sets of alignment links, any edge in the bipartite alignment graph is labelled with a triple of Boolean values. The first value corresponds to the occurrence of an edge in the $A_{\mathbf{pl}\rightarrow\mathbf{en}}$ set. The second one denotes the occurrence of an edge in the $A_{\mathbf{en}\rightarrow\mathbf{pl}}$ set. The third value stands for the occurrence of an edge in the $A_{\mathbf{gdfa}}$ set. For example, if an edge is present in all alignment sets, it is labelled with `111`; if an edge is not present in any alignment set, it is labelled with `000`; if an edge only occurs in $A_{\mathbf{pl}\rightarrow\mathbf{en}}$, it is labelled with `100`. An example of a complete bipartite alignment graph is given in Figure 5.2.

There seem to be 8 possible labels for edges in bipartite graphs. However, as $A_{\mathbf{gdfa}}$ links are selected from union of both unidirectional word alignments, the label `001`, which indicates that an edge occurs in $A_{\mathbf{gdfa}}$ set but not in unidirectional word alignments, is not possible. Moreover, the label `110` is also not possible since any link occurring in both unidirectional word alignments (therefore, in intersection of unidirectional word

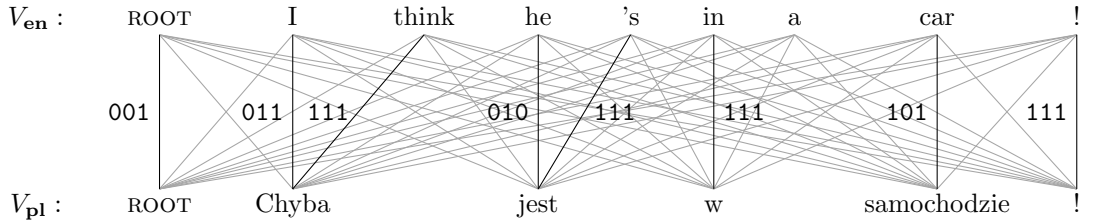


FIGURE 5.2: A complete bipartite alignment graph $BG := (V_{\mathbf{en}} \cup V_{\mathbf{pl}}, E)$ for projecting English dependency relations to Polish. Vertices of BG are decomposed into two disjoint sets: $V_{\mathbf{en}}$ of English tokens together with an additional ROOT node and $V_{\mathbf{pl}}$ of Polish tokens with an additional ROOT node. Every pair of graph vertices from different sets are adjacent. An edge (u, v), for $u \in V_{\mathbf{en}}$ and $v \in V_{\mathbf{pl}}$, indicates the alignment correspondence between u and v. Edges E are labelled with sets of Boolean values that indicate presence of an edge in automatically generated word alignment sets. For clarity's sake, edges which are absent in any set of word alignment links (i.e., labelled with 000) are marked grey.

alignments) is an inherent element of the $A_{\mathbf{gdfa}}$ set. The following labels are thus possible in complete bipartite alignment graphs: 111, 101, 011, 100, 010, 000.

Any dependency tree originates from a ROOT node. The relation between the ROOT node and its dependent node should also be projected in order to enable inducing proper dependency trees on the Polish side. Therefore, the additional ROOT nodes are in the disjoint sets of bipartite vertices, i.e., ROOT $\in V_{\mathbf{en}}$ and ROOT $\in V_{\mathbf{pl}}$. Since the edge between ROOT nodes is not present in any word alignment set, it should be labelled with 000. However, as automatic word alignment is prone to errors, we cannot exclude that the predicate of an English sentence is aligned with a Polish token that does not fulfil the function of the sentence predicate. If the bipartite edge between ROOT nodes was labelled with 000, the incorrect dependency relation would only be projected since projection via two edges labelled with 000 is not allowed (see below). Therefore, edges between ROOT nodes are assigned the special label 001 in order to make projection via these edges possible.

Labelled bipartite edges are weighted with the function $w : E \to \{0, 1, 2, 3\}$. The edge weight $w(u, v)$ is the sum of Boolean values on an edge label (see (5.3)). The sum of Boolean values on each edge indicates the certainty of this edge.

$$w(u,v) = \begin{cases} 0, & \text{for edge label } 000 \\ 1, & \text{for edge labels } 100, 010, 001 \\ 2, & \text{for edge labels } 101, 011 \\ 3, & \text{for edge label } 111. \end{cases} \tag{5.3}$$

Weighted complete bipartite alignment graphs as one shown in Figure 5.3 are employed to project English dependency relations to Polish.

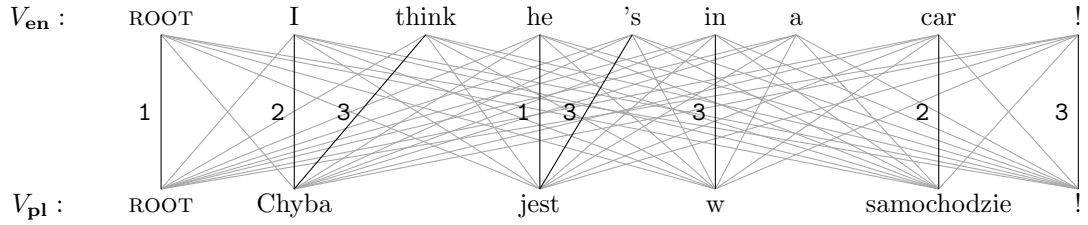


FIGURE 5.3: A weighted complete bipartite alignment graph $BG := (V_{\mathbf{en}} \cup V_{\mathbf{pl}}, E)$ for projecting English dependency relations to Polish. Vertices of BG are decomposed into two disjoint sets: $V_{\mathbf{en}}$ of English tokens with an additional ROOT node and $V_{\mathbf{pl}}$ of Polish tokens with an additional ROOT node. An edge (u, v), for $u \in V_{\mathbf{en}}$ and $v \in V_{\mathbf{pl}}$, indicates the alignment correspondence between u and v. Edges E are weighted with sums of Boolean values on edge labels. For clarity's sake, edges which are absent in any set of word alignment links (i.e., weighted with 0) are marked grey.

5.1.2 Projection of Dependency Relations

English dependencies are projected to Polish according the following procedure. The projection algorithm (see Algorithm 5.1) takes as input an English dependency tree $T_{\mathbf{en}}$ and a weighted bipartite alignment graph BG. The English tree $T_{\mathbf{en}} = (V_{\mathbf{en}}, A_{\mathbf{en}})$ consists of a set of vertices $V_{\mathbf{en}} = \{u_0, u_1, ..., u_m\}$, where u_0 corresponds to the ROOT node, and a set of directed edges (arcs) $A_{\mathbf{en}} \subseteq \{(u', u, gf) | u', u \in V_{\mathbf{en}}, gf \in GF\}$ labelled with grammatical functions from GF. The weighted bipartite alignment graph $BG = (V_{\mathbf{en}} \cup V_{\mathbf{pl}}, E)$ is built of two sets of vertices $V_{\mathbf{en}} \cup V_{\mathbf{pl}}$ and a set of weighted edges $E \subseteq \{(u, v, w) | u \in V_{\mathbf{en}}, v \in V_{\mathbf{pl}}, w \in \{0, 1, 2, 3\}\}$, such that there is exactly one edge between u and v and this edge is assigned a weight from $\{0, 1, 2, 3\}$.

Algorithm 5.1 The projection procedure.

projection$(T_{\mathbf{en}}, BG)$

$T_{\mathbf{en}} = (V_{\mathbf{en}}, A_{\mathbf{en}})$, where $V_{\mathbf{en}} = \{u_0, u_1, ..., u_m\}$ and $A_{\mathbf{en}} \subseteq \{(u', u, gf) | u', u \in V_{\mathbf{en}}, gf \in GF\}$
$BG = (V_{\mathbf{en}} \cup V_{\mathbf{pl}}, E)$, where $V_{\mathbf{en}} = \{u_0, u_1, ..., u_m\}$, $V_{\mathbf{pl}} = \{v_0, v_1, ..., v_n\}$, $E \subseteq \{(u, v, w) | u \in V_{\mathbf{en}}, v \in V_{\mathbf{pl}}, w \in \{0, 1, 2, 3\}\}$ and $\forall u \in V_{\mathbf{en}}\ \forall v \in V_{\mathbf{pl}}\ \exists! w \in \{0, 1, 2, 3\} : (u, v, w) \in E$
$A_{\mathbf{pl}} := \emptyset$

For $(u, v, w) \in E$, such that $u \neq u_0$
  Find $u' \in V_{\mathbf{en}}$, such that $(u', u, l) \in A_{\mathbf{en}}$
  For $v' \in V_{\mathbf{pl}}$, such that $v' \neq v$ and $(u', v', w') \in E$
    If $w > 0$ or $w' > 0$
      If $(v', v, l) \in A_{\mathbf{pl}}$, where $l = (w, w', gf, f)$
        $l := (w, w', gf, f + 1)$
      Else add $(v', v, (w, w', gf, 1))$ to $A_{\mathbf{pl}}$
Return $(V_{\mathbf{pl}}, A_{\mathbf{pl}})$

The algorithm iteratively projects arcs of an English tree to the Polish equivalent sentence. For each Polish lexical node v, the algorithm finds its governor node v' in the following way. First, it looks for an English non-ROOT node u connected with the node v in the bipartite alignment graph BG. Then, the governor node u' of u is found in the English dependency tree $T_{\mathbf{en}}$. Finally, the Polish node v' which is connected with u' in BG is identified and recognised as the governor of v.

The arc (v', v, l), which corresponds to the relation between tokens v' and v and is assigned the label l, may be added to the Polish directed graph. The only restriction is that it is not possible to project relations via bipartite edges which are both weighted with `0`. The reason for this limitation is to avoid projection of relations considered to be the most error prone (e.g., $[I \overset{0}{—} jest] \longleftarrow [samochodzie \overset{0}{—} think]$ in Figure 5.4). However, projection via two edges one of which is weighted with `0` is permitted in order to cover relations between English tokens one of which is not aligned with any Polish token in any of word alignment sets.

The projection algorithm outputs the set of Polish vertices $V_{\mathbf{pl}}$ and the set of arcs $A_{\mathbf{pl}}$ between these vertices. The set of vertices $V_{\mathbf{pl}} = \{v_0, v_1, ..., v_n\}$ consists of an additional ROOT node v_0 and a set of lexical nodes $\{v_1, ..., v_n\}$, for each vertex v_i corresponding to the ith token of a Polish sentence $S = t_1, ..., t_n$. The vertices from $V_{\mathbf{pl}}$ are connected with arcs from the set $A_{\mathbf{pl}} \subseteq \{(v_i, v_j, l) | v_i, v_j \in V_{\mathbf{pl}}, l = (w, w', gf, f)\}$, for $w, w' \in \{0, 1, 2, 3\}, gf \in GF, f \in \mathbb{N}_+$. These two sets constitute the Polish digraph $G_{\mathbf{pl}} = (V_{\mathbf{pl}}, A_{\mathbf{pl}})$ which fulfils the following requirements:

- $G_{\mathbf{pl}}$ originates from the ROOT node v_0, i.e., each node in $V_{\mathbf{pl}}$ directly or indirectly depends on the ROOT node,
- the ROOT node does not have any predecessor, i.e., $(v_i, v_0) \notin A_{\mathbf{pl}}$, for all $v_i \in V_{\mathbf{pl}}$,
- all vertices $V_{\mathbf{pl}}$ are weakly connected,[3] i.e., every vertex is reachable from every other disregarding the direction of arcs (Bang-Jensen and Gutin, 2009),
- each arc (v_i, v_j, l) is assigned an initial score $s(v_i, v_j, l) \in \mathbb{R}$ estimated on the basis of information in the label l (see Section 5.1.3 *Intuitive Weighting Method*).

Any projected arc is assigned a label $l = (w, w', gf, f)$, i.e., a quadruple composed of two integer values w and w', an English grammatical function gf and a projection frequency f. The integer values correspond to weights of bipartite edges E connecting two related

[3]Each projected digraph is weakly connected. Assume that the English dependency tree is connected and there is at least one bipartite edge which is assigned a non-zero score. Then, it is possible to project all English arcs coming in or out of the node connected by this bipartite edge with a positive score as arcs coming to each token of the Polish corresponding sentence. The bipartite edge linking ROOT nodes is always assigned a score of 1. Therefore, in the worst case when all bipartite edges linking Polish and English tokens are assigned a score of 0, we may project at least English arcs coming out of the English ROOT node (see dotted arcs on the schema below).

English nodes with two Polish nodes. These bipartite edges are used to project an English dependency relation to the Polish sentence. The first integer value (the dependent link value) in the quadruple refers to the weight of a bipartite edge connecting English and Polish tokens with the dependent status. The second integer value (the governor link value) in the quadruple refers to the weight of a bipartite edge connecting English and Polish tokens with the governor status. The third element is a string indicating the label of the English dependency relation projected to Polish. The fourth element indicates the frequency of projecting a particular relation to the same Polish tokens via equally weighted edges in the bipartite alignment graph.

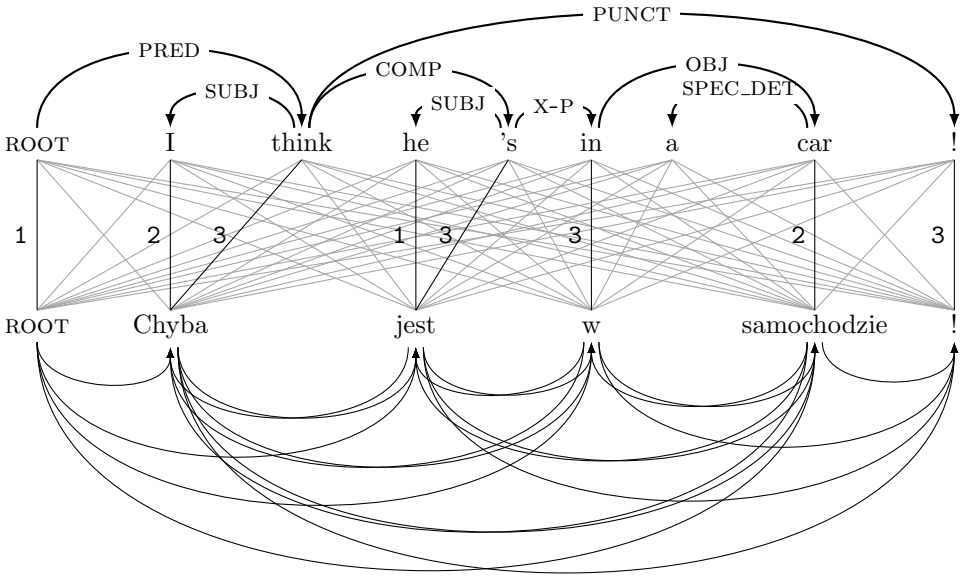
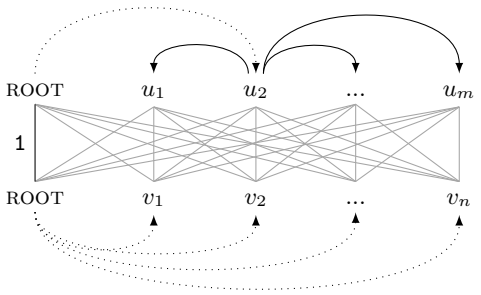


FIGURE 5.4: The Polish digraph (bottom arcs) projected from the English dependency structure (top arcs) via edges of the bipartite alignment graph (the middle edges between English and Polish nodes). The arc labels in the Polish digraph are not displayed to preserve the clarity of presentation. X-P is short for XCOMP-PRED.

For example, the projected arc between the dependent node *samochodzie* governed by the node w (see Figure 5.4)[4] should be labelled with (2, 3, OBJ, 1). This label indicates



[4]Arcs of the projected digraph are assigned labels, but they are not displayed in the schema for clarity's sake.

that the relation is projected via two bipartite edges: the first one with the weight 2 connects dependents (*car* and *samochodzie*) and the second one with the weight 3 connects governors (*in* and *w*). The English grammatical function OBJ, i.e., the object function which is fulfilled by the dependent *car* of the preposition *in*, is directly transferred to the Polish relation and stored as the third element in the labelling quadruple. The last element in the quadruple indicates the frequency of projecting English arcs labelled with the same grammatical function via bipartite edges with the same governor link value and the same dependent link value.

The projection frequency is typically equal to 1, but we may not rule out higher frequency values. In Figure 5.5, the arc *Historia* $\xrightarrow{(0,3,\text{ADJUNCT},3)}$ *średniowiecza* corresponds to three ADJUNCT arcs of the English tree: *History* $\xrightarrow{\text{ADJUNCT}}$ *of*, *History* $\xrightarrow{\text{ADJUNCT}}$ *from*, *History* $\xrightarrow{\text{ADJUNCT}}$ *to*, all of which are projected via (0,3)-weighted pairs of links. The arc *Historia* $\xrightarrow{(0,3,\text{ADJUNCT},2)}$ *po*, in turn, corresponds to two ADJUNCT arcs of the English tree: *History* $\xrightarrow{\text{ADJUNCT}}$ *of* and *History* $\xrightarrow{\text{ADJUNCT}}$ *from*, which are projected via (0,3)-weighted pairs of links.

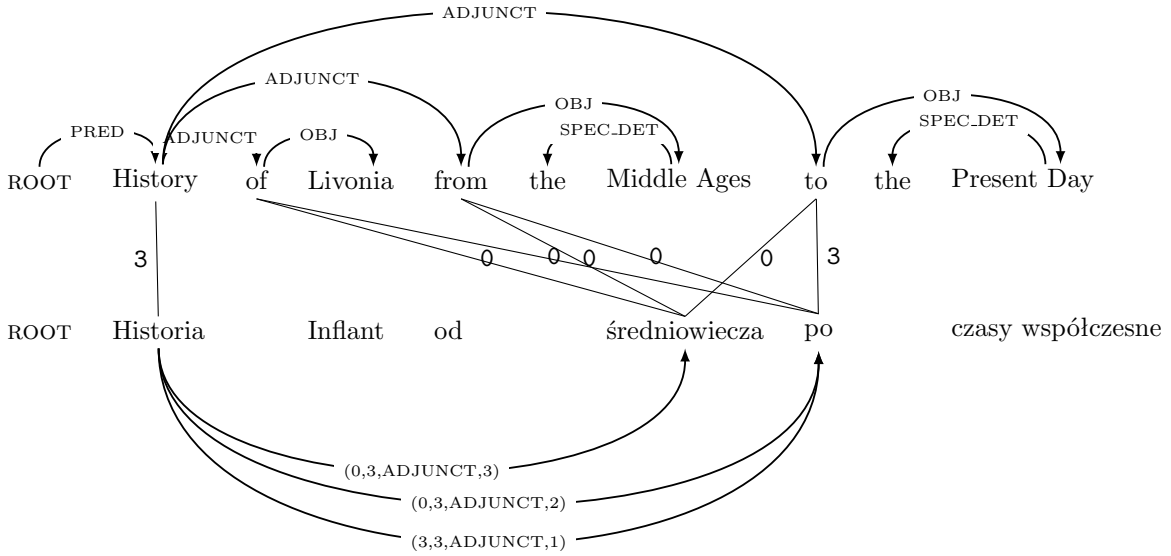


FIGURE 5.5: Estimation of a higher frequency value which is the last integer in quadruples labelling Polish arcs. For clarity, only bipartite edges used to project the three Polish dependency relations are displayed.

A projected digraph may have several arcs between the same two vertices (multiple arcs). Since English relations are projected through all possible pairs of alignment links, projected digraphs may contain differently labelled multiple arcs between the same Polish nodes (see two relations between *Historia* and *po* in Figure 5.5). Therefore, we refer to them as *projected multi-digraphs*.

In comparison to the number of relations directly projected via word alignment links, projection via two edges of a bipartite alignment graph one of which is assigned a score

of 0 significantly increases the number of projected relations encoded as arcs in multi-digraphs. Some of these arcs correspond to invalid Polish dependency relations, cause noise in projected multi-digraphs and need to be filtered out. On the other hand, we want to eliminate some translational discrepancies by extending the set of possible arcs. For example, if we applied projection via links present in symmetrised word alignments only, it would not be possible to acquire the correct Polish dependency tree marked with solid bottom arcs in Figure 5.6. The English sentence *I think he's in a car!* was translated into Polish as a simple sentence *Chyba jest w samochodzie!*. The particle *chyba* expressing a presumption corresponds to the English matrix clause *I think*. However, the particle *chyba* cannot function as the predicate of the Polish sentence and the projected tree marked with dotted arcs in Figure 5.6 is incorrect. In contrast to the particle *chyba*, the finite verb form *jest* can be the sentence predicate and the arc ROOT $\xrightarrow{pred}$ *jest* should thus appear in the final dependency tree. This arc could be acquired if the English relations are projected via edges of the bipartite alignment graph.

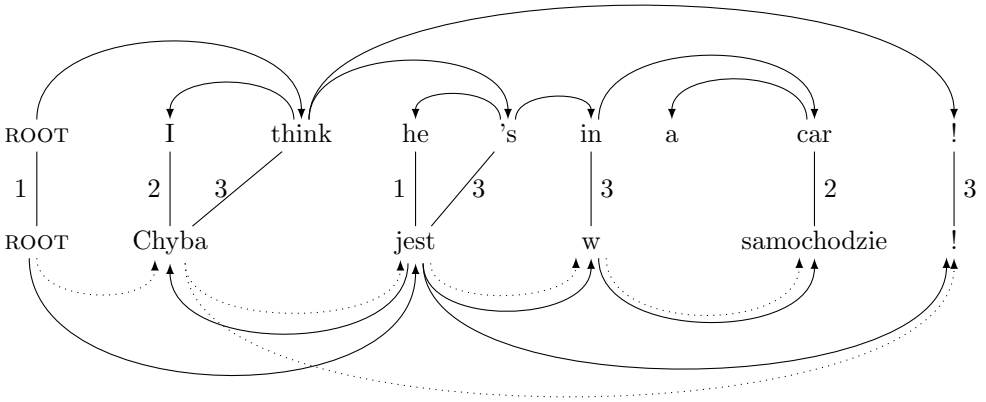


FIGURE 5.6: Discrepancies between the gold standard Polish dependency tree (solid bottom arcs) and the Polish dependency tree (dotted bottom arcs) projected from the English dependency structure (top arcs) via automatically generated word alignment links.

A projected multi-digraph not only spans over all Polish tokens but also connects all tokens (connected digraph). A spanning tree can be found in any connected directed graph. Since some spanning trees are equivalent to dependency structures, the idea is to find such spanning trees in projected multi-digraphs. The idea of searching for maximum spanning trees in directed graphs comes from MST-based dependency parsing (McDonald et al., 2005b; Kübler et al., 2009). In graph-based dependency parsing, a pre-trained parsing model is applied to score arcs in candidate spanning trees. Then, the highest scored spanning tree is selected as the valid dependency structure of an input sentence. Details of the MST-based parsing are provided in Section 2.4.2 *Graph-based Dependency Parsing*.

In our approach, we assume that there is no manually annotated data to train a model scoring arcs in projected multi-digraphs. All arcs are projected with the same significance and all of them may equally likely be selected as edges of a final dependency tree. Since only some of projected arcs correspond to correct dependency relations, it is essential to identify the most probable arcs and assign them appropriate scores. To do this, we first assign initial weights to arcs in projected multi-digraphs (see Section 5.1.3 *Intuitive Weighting Method*). The initial weights are then optimised in order to select the most probable arcs that can constitute dependency trees (see Section 5.2 *Weighted Induction*).

5.1.3 Intuitive Weighting Method

The main goal of the intuitive weighting method is to score arcs in projected digraphs assuming that the weight of an arc indicates how likely is a corresponding relation. Intuitively, a relation between two tokens might be more important than relations between other tokens if it is projected via bipartite edges categorised as more certain. Hence, the intuitive weighting method is based on the hypothesis that the strength of a relation primarily depends on scores of bipartite edges which the relation was projected through.

There are two bipartite edges which are involved in projection of a dependency relation – one with the score w_d connects dependents and another one with the score w_g connects governors. These two scores are included in the label assigned to each projected arc. Apart from dependent and governor scores, the arc label contains information about the English grammatical function *gf* and the projection frequency *f*. As the dependent edge score, the governor edge score and the projection frequency are the only clues to identify fundamental arcs, they are taken into account in estimation of initial weights of projected arcs. The scoring function s (see (5.4)) is defined to estimate an initial weight for any arc $(v_i, v_j, (w_d, w_g, gf, f)) \in A$ in a projected multi-digraph $G = (V, A)$:

$$s(v_i, v_j, (w_d, w_g, gf, f)) = w_d + w_g + 2w_d w_g f \tag{5.4}$$

The intuition behind this definition is as follows. Since dependency relations projected via two bipartite edges one of which is assigned a score of 0 are predominantly unreliable, they should be assigned significantly lower scores than dependency relations projected via two bipartite edges with positive scores. Furthermore, the frequency f should not considerable increase scores of unreliable relations which are numerous and may dominate other projected relations. The initially weighted multi-digraphs provide a starting point to induce dependency trees.

In the example in Figure 5.7, the English relation $in \xrightarrow{\text{OBJ}} car$ is projected to the Polish corresponding sentence via all approved bipartite edge pairs. There are, among others, the projected arc $w \xrightarrow{(2,3,\text{OBJ},1)} samochodzie$ and the arc $jest \xleftarrow{(0,3,\text{OBJ},1)} w$. The first arc is

projected via bipartite edges with scores higher than 0. The second arc, in turn, is projected via two bipartite edges one of which is assigned a weight of 0. The function s scores the unreliable second arc significantly lower than the first arc. Zero score of the bipartite edge diminishes the weight of the second arc, but this weight is not reduced to 0.

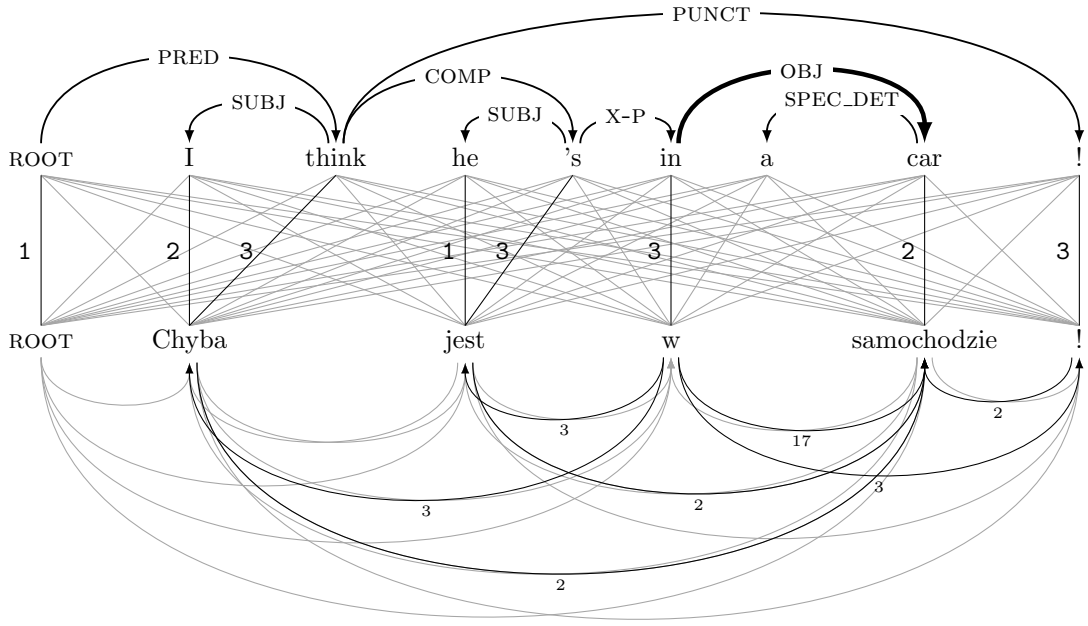


FIGURE 5.7: The initially weighted projected multi-digraph (bottom arcs). For clarity's sake, only arcs projected from the English arc $in \xrightarrow{\text{OBJ}} car$ (marked with a thick line) are displayed on the schema, whereas projected arcs with a score of 0 are marked with grey lines. The projection frequency is equal to 1 in this case. X-P is short for XCOMP-PRED.

5.2 Weighted Induction

This section presents weighted induction which is the second step in the process of acquiring Polish dependency structures. The main idea behind weighted induction is to identify the most likely arcs in initially scored projected multi-digraphs and to assign them appropriate weights. Using methods of selecting maximum spanning trees from weighted directed graphs, final well-formed dependency structures (i.e., *maximum spanning dependency trees*, MSDTs) are inferred from weighted projected multi-digraphs. Induced dependency structures may be used as data for training a Polish dependency parser.

As noted above, projected multi-digraphs contain unreliable arcs which are predominantly projected via bipartite edges one of which is assigned a score of 0. These arcs may cause noise in projected multi-digraphs and they are less likely to enter into final dependency trees. Since they are numerous and may quantitatively dominate correct

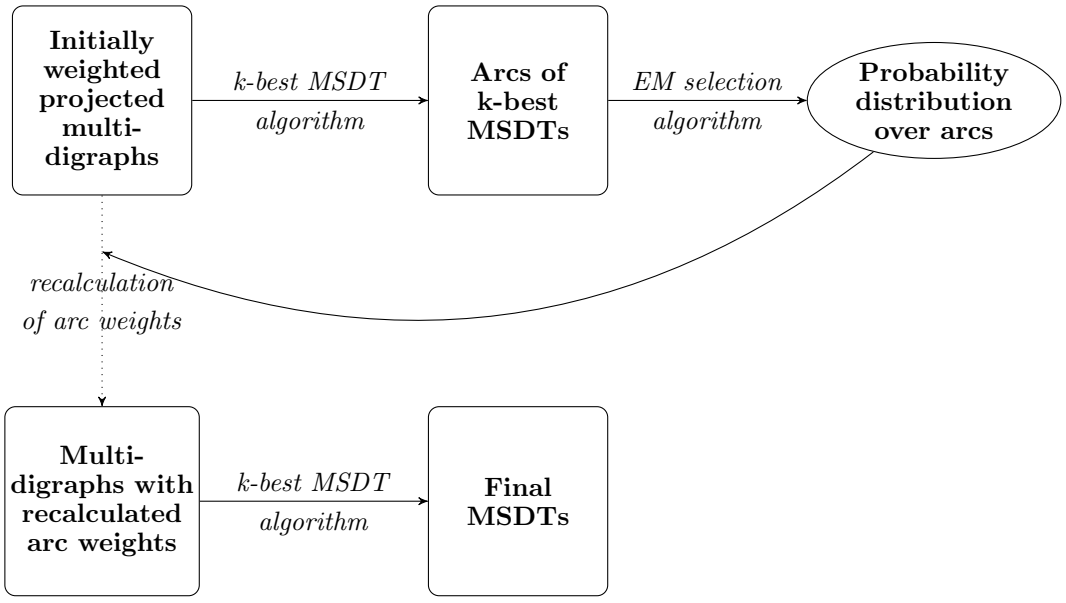


FIGURE 5.8: Schema of the weighted induction procedure.

arcs, methods based on the frequency of arcs in projected multi-digraphs may not lead to the identification of higher priority arcs. Therefore, we propose a method of the identification of higher priority arcs based on an EM-inspired algorithm which is trained on a restricted set of projected arcs. A schema of the weighted induction procedure is shown in Figure 5.8.

The set of arcs used in the EM training is restricted to arcs of k-best maximum spanning dependency trees found in initially weighted projected multi-digraphs. Details of the MSDT selection procedure are described in Section 5.2.1 *Maximum Spanning Dependency Trees*. Then, the probability distribution is estimated over arcs of k-best MSDTs using the EM-inspired selection algorithm. It should be noted that arcs are identified by their feature representations (see Section 5.2.2 *Feature Representations of Arcs*). The probability distribution over arcs, in turn, is employed to recalculate initial weights on arcs in projected multi-digraphs. The procedures of estimating the probability distribution over arcs and optimising weights of arcs in projected multi-digraphs are described in Section 5.2.3 *Recalculation of Arc Weights in Projected Multi-Digraphs*. Projected multi-digraphs with recalculated arc weights are used to induce final dependency structures. A final dependency structure of a sentence is the first well-formed dependency tree on the list of k-best maximum spanning trees. The entire induction procedure results in a bank of Polish unlabelled dependency structures. The following subsections describe relevant parts of the weighted induction procedure in detail.

5.2.1 Maximum Spanning Dependency Trees

As known from graph theory (e.g., Diestel, 2000, p. 14), every connected graph contains a normal spanning tree with any vertex specified as its root. The spanning tree $G' = (V', E')$ of the connected graph $G = (V, E)$ is a tree, vertices V' of which span all of G, i.e., $V' = V$, and edges E' constitute a subset of E, i.e., $E' \subseteq E$. If edges E in G are weighted, it is possible to select a maximum spanning tree, i.e., a spanning tree that maximises the sum of edge weights.

Therefore, we may search for a maximum spanning tree in a weighted projected multi-digraph $G = (V, A)$ which fulfils the weak connectedness criterion. A maximum spanning tree that meets the criteria of a well-formed dependency tree is referred to as *maximum spanning dependency tree* (see Definition 5.1 of MSDT).

Definition 5.1. A maximum spanning tree $T = (V', A')$ extracted from a weighted projected multi-digraph $G = (V, A)$, for $A' \subseteq A$ and $V' = V = \{v_0, v_1, ..., v_n\}$, where v_i corresponds to the ith token of a sentence $S = t_1, ..., t_n$ and v_0 is an additional ROOT node, corresponds to a valid dependency structure if v_0 is the ROOT of T, i.e., $(v_i, v_0, l) \notin A'$, for $v_i \in V'$, $l \in L$, and v_0 has only one successor, i.e., if $(v_0, v_i, l) \in A'$, then $(v_0, v_j, l') \notin A'$, for $v_i \neq v_j$. Such T is called *maximum spanning dependency tree* of G.

A maximum spanning tree extracted from a weighted projected multi-digraph coincides with a dependency structure of a sentence if it is rooted, acyclic and connected, spans over all lexical nodes, and contains nodes which have exactly one incoming arc, except for the ROOT node which has no predecessor and only one successor. Hence, finding a maximum spanning dependency tree in a weighted projected multi-digraph is equivalent to inducing a dependency structure of a sentence.

In order to find a maximum spanning tree in a weighted projected multi-digraph, we might apply the Chu-Liu-Edmonds algorithm adapted to purposes of dependency parsing by McDonald et al. (2005a). The pseudocode of this algorithm is given in Algorithm 2.1, p. 18 of this dissertation. The maximum spanning tree selected with the Chu-Liu-Edmonds algorithm (see the top tree in Figure 5.9) meets the properties of a well-formed dependency tree. However, this MST does not correspond to a dependency structure which we want to acquire (see the bottom tree in Figure 5.9). The extracted MST lacks some arcs corresponding to appropriate relations. Furthermore, it contains arcs that are highly scored but invalid. In order to improve initial arc weights in projected multi-digraphs and thereby to enable induction of more adequate dependency trees, we propose a procedure of recalculating initial arc scores (see Section 5.2.3 *Recalculation of Arc Weights in Projected Multi-Digraphs*).

The general idea behind the recalculation consists in estimation of the probability distribution over reliable arcs selected from initially weighted projected multi-digraphs and

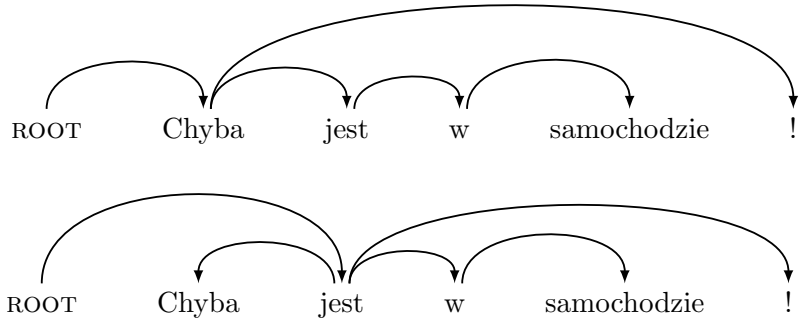


FIGURE 5.9: Two dependency trees of the Polish sentence *Chyba jest w samochodzie!* (Eng. 'I think he is in a car!'): the top tree is automatically found in the initially weighted projected multi-digraph using the Chu-Liu-Edmonds algorithm and the bottom one is the gold standard tree.

then, in recalculation of all arc weights in projected multi-digraphs based on the estimated probability distribution. A candidate for a restricted set of reliable arcs could be a maximum spanning tree. A MST contains some reliable arcs but not necessarily all of them. Therefore, it is essential to enlarge the set of reliable arcs which are taken into account in estimation of the probability distribution. Even if the maximum spanning tree does not correspond to an appropriate dependency structure of a sentence (i.e., it does not encode all required dependency relations), a proper tree may be selected as second or next best spanning tree. In order to increase the chance of selecting all accurate arcs that constitute a dependency tree, we do not confine the set of arcs to those building the maximum spanning tree. Instead, we enlarge this set with arcs which are in k-best maximum spanning dependency trees assuming that each required arc is contained in at least one of these trees. The set of arcs in k-best maximum spanning dependency trees is larger than the set of arcs in a maximum spanning tree. It may even contain all arcs required to build a final dependency tree. Moreover, k-best MSDTs contain less noisy arcs then projected multi-digraphs. Therefore, arcs in k-best MSDTs are more suitable for estimation of the probability distribution than entire projected multi-digraphs. An example of k-best maximum spanning dependency trees selected from a initially weighted projected multi-digraph is given in Figure 5.10.

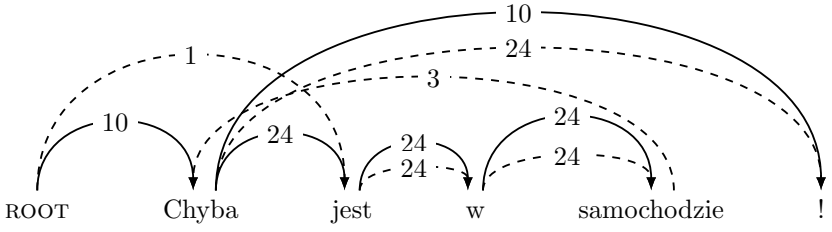


FIGURE 5.10: k-best MSDTs (for $k = 50$) selected from the initially weighted projected multi-digraph built for the Polish sentence *Chyba jest w samochodzie!* (Eng. 'I think he is in a car!'). For clarity's sake, arcs from the 1st MSDT (marked with solid lines) and from the 33rd MSDT (marked with dashed lines) are only displayed.

In order to select k-best maximum spanning dependency trees, we apply a slightly modified version of the algorithm by Camerini et al. (1980). The pseudocode of the k-best MSDT selection algorithm and its explanation are presented in Appendix D, p. 183 of this dissertation. The algorithm finds k-best MSDTs in a weighted directed graph $G = (V_G, E_G)$ with the set of vertices $V_G = \{v_0, ..., v_n\}$, where v_0 is the root node, and the set of edges $E_G = \{e_1, ..., e_m\}$. All found k-best MSDTs are rooted at the same vertex v_0.

A list of k-best MSDTs of the digraph G is computed with the ranking function **rank** of the k-best MSDT selection algorithm. This function is slightly modified in relation to the original function **rank** which outputs a list of k-best MSTs (c.f., Camerini et al., 1980, p. 107). In our version, some additional conditions are imposed on candidate MSTs so that they meet properties of well-formed dependency trees. Since not all MSTs fulfil properties of well-formed dependency trees, they are not taken into account in estimation of the probability distribution. We reject all MSTs with multiple dependents of the ROOT node (see Figure 5.11) since they do not correspond to valid dependency trees. Furthermore, if the best MST found in a multi-digraph G is ill-formed (i.e., it has more than one arc coming out of the ROOT node), then k-best MSDTs are not selected from this multi-digraph at all. A cursory analysis shows that if the best MST has multiple ROOT dependents, it typically corresponds to a noisy dependency analysis of a sentence. Noise may result from incorrect word alignment or an inaccurate English tree. In order to reduce noise in training data, invalid MSTs are rejected.

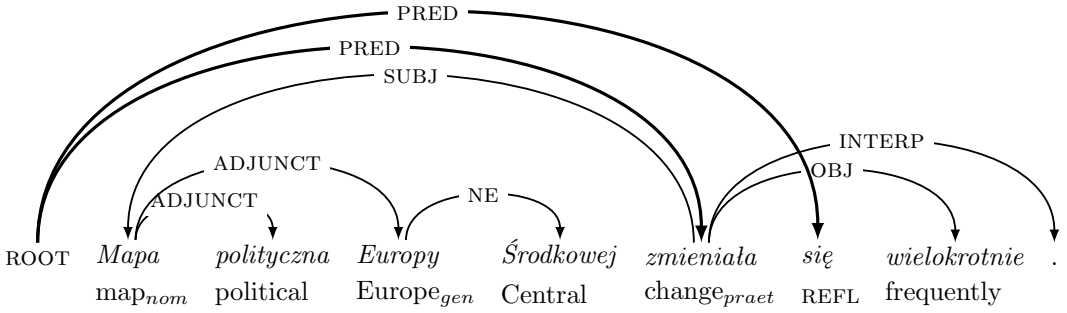


FIGURE 5.11: An invalid dependency structure of the Polish sentence *Mapa polityczna Europy Środkowej zmieniała się wielokrotnie.* (Eng. 'The political map of Central Europe has changed numerous times.') with multiple dependents of the ROOT node (marked with thick arrows).

The weighted induction procedure starts with the selection of P sets of k-best MSDTs $\mathcal{T} = \{T_1, T_2, ..., T_P\}$ from M initially weighted projected multi-digraphs $\mathcal{G} = \{G_1, ..., G_M\}$ (see the function **extractMSDTs** in Algorithm 5.2). k-best MSDTs are extracted with the k-best MSDT selection algorithm. The number M of initially weighted projected multi-digraphs differs from the number P of sets of selected k-best MSDTs since some MSTs do not fulfil requirements of well-formed dependency trees and are

rejected from the EM training. Any set T_p contains at most k-best MSDTs. Since it may not be possible to extract k spanning trees for short sentences, k is thus the upper bound of the number of selected MSDTs. Nevertheless, we refer to this set as k-best MSDTs in this dissertation.

Arcs from any set of k-best MSDTs are then composed into sets of individual arcs coming into particular nodes $\mathcal{B} = \{B_1, ..., B_N\}$. The number N of sets in $\mathcal{B}$ is equal to the number of individual vertices in $\mathcal{T}$ (see the function **prepareTrainingData** in Algorithm 5.2). The training data $\mathcal{B}$ is used to estimate the probability distribution over arcs with the EM-inspired selection algorithm.

5.2.2 Feature Representations of Arcs

Arcs which are used in the EM training are represented with their features. Any projected arc $(v_h, v_i, (w_d, w_g, gf, f))$ representing a dependency relation connects two vertices one of which v_h corresponds to the governor of the relation, while the other one v_i is its dependent. Each of related vertices represents a token in a sentence and encodes information about the token's lemma, part of speech tag, and morphological features. Furthermore, any arc is assigned a label (w_d, w_g, gf, f) and an initial weight $s(v_h, v_i, (w_d, w_g, gf, f))$. The information available in the arc label and in the related vertices may be used in the feature representation j of the arc. The set of features identifying an arc is given with the function fr, i.e., $fr(v_h, v_i, (w_d, w_g, gf, f)) = j$. Particular arcs may be denoted by the following sets of features (or combinations of these sets):

1. surface forms of the dependent and the governor, and the grammatical function labelling the dependency relation,
2. lemmata of the dependent and the governor, and the grammatical function,
3. part of speech tags of the dependent and the governor, and the grammatical function.

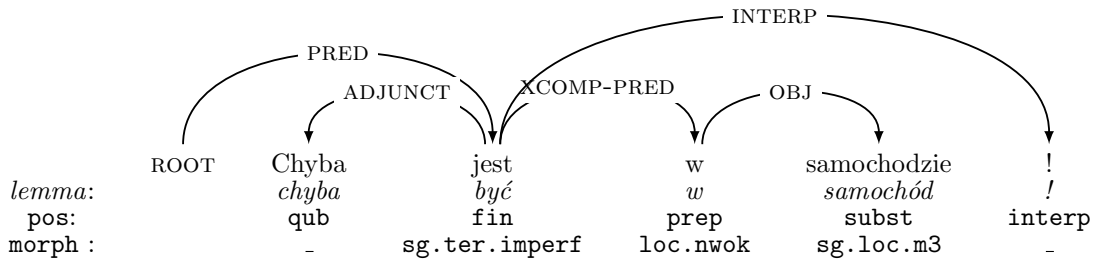


FIGURE 5.12: A structurally valid dependency tree of the Polish sentence *Chyba jest w samochodzie!* (Eng. 'I think he is in a car!') with arcs labelled with English grammatical functions and with available token annotations: lemmata, part of speech tags and morphological features.

Algorithm 5.2 The induction procedure of the final dependency tree.

Algorithm:
$\mathcal{T} :=$ **extractMSDTs**($\mathcal{G}$, k)
$\mathcal{B} :=$ **prepareTrainingData**($\mathcal{T}$)
$\mathcal{G}' :=$ **updateWeights**($\mathcal{G}$, $\mathcal{B}$, t)
$\mathcal{T}' :=$ **extractMSDTs**($\mathcal{G}'$, 1)
Return $\mathcal{T}'$ *# the final set of maximum spanning dependency trees*

extractMSDTs($\mathcal{G}$, k)
$\mathcal{G} = \{G_1, ..., G_M\}$, where $G_x = (V_x, A_x)$, for $x \in \{1, ..., M\}$; $V_x = \{v_0, ..., v_n\}$; $A_x \subseteq \{(v_h, v_i, l) | v_h, v_i \in V_x, l = (w, w', gf, f)\}$, for $w, w' \in \{0, 1, 2, 3\}$, $gf \in GF$, $f \in \mathbb{N}_+$
k is the number of well-formed MSDTs to extract
$\mathcal{T} := \emptyset$

For $G_x \in \mathcal{G}$
  $T_x :=$ **rank**(G_x, k) *# the function **rank** is outlined in Appendix D*
  If $T_x \neq \emptyset$
    Add T_x to $\mathcal{T}$
Return $\mathcal{T}$

prepareTrainingData($\mathcal{T}$)
$\mathcal{T} = \{T_1, ..., T_P\}$
$i := 1$
$\mathcal{B} := \emptyset$

For $T_p \in \mathcal{T}$, where $T_p = \{MSDT_1, ..., MSDT_r\}$, for $r \leq k$
  For $v_z \in V_{MSDT_1}$
    $B_i := \emptyset$
    For $MSDT_y \in T_p$, where $MSDT_y = (V_y, A_y)$
      Select from A_y all arcs coming into v_z and add these arcs to B_i
    Add B_i to $\mathcal{B}$ and set $i := i + 1$
Return $\mathcal{B}$

updateWeights($\mathcal{G}$, $\mathcal{B}$, t)
$\mathcal{G} = \{G_1, ..., G_M\}$
$\mathcal{B} = \{B_1, ..., B_N\}$, for N being the number of individual nodes in $\mathcal{T}$
t is the number of EM iterations
$(p_j) :=$ **EM**($\mathcal{B}$, t) *# estimation of probability distribution over arcs with Algorithm 5.3*
$\mathcal{G}' := \emptyset$

For $G_x \in \mathcal{G}$, where $G_x = (V_x, A_x)$
  $A'_x := \emptyset$
  For $(v_h, v_i, l) \in A_x$
    If the feature representation j of the arc (v_h, v_i, l) is represented in (p_j)
      $s^* := \sqrt{s(v_h, v_i, l)} \times p_j$
    Else $s^* := \sqrt{s(v_h, v_i, l)} \times \min_j p_j \times \alpha$, for some $0 < \alpha < 1$
    Add (v_h, v_i, l) with the weight s^* to A'_x
  Add (V_x, A'_x) to $\mathcal{G}'$
Return $\mathcal{G}'$

An example of the first feature representation could be *fr*(*jest*, *w*, (3, 3, XCOMP-PRED, 1)) = {*w*, *jest*, XCOMP-PRED}, which corresponds to a projected dependency relation presented in the dependency tree in Figure 5.12.[5] The dependency relation between the preposition *w* (Eng. 'in') and its governor realised as the verb form *jest* (Eng. 'be'$_{3.sg.pres}$) is labelled with the English grammatical function XCOMP-PRED (an open complement in predicative position). Since Polish is an inflecting language, the diversity of word forms on the one hand and alternative meanings of the same word forms on the other hand may cause problems (e.g., the data sparseness problem) in estimation of the probability distribution over arcs identified by their surface forms.

The data sparseness problem should be less severe in the case of the second and the third type of feature sets. The probability distribution may be estimated over arcs represented by lemmata or part of speech tags of related tokens. Taking into account the previously considered dependency relation, the second feature representation would be {*w*, *być*, XCOMP-PRED}, where *w* and *być* are lemmata of tokens *w* and *jest* respectively. The third feature representation of the same dependency relation is {*prep*, *fin*, XCOMP-PRED}, where *prep* (preposition) and *fin* (non-past finite verb form) constitute part of speech tags of tokens *w* and *jest* respectively.

With respect to feature sets based on lemmata, we apply a common heuristic of generalising lemma forms of tokens not recognised by the tagger (annotated with the *ign* part of speech tag), similarly as in Finkel et al. (2005). The general idea is to replace digits,[6] lowercase characters and uppercase characters in unrecognised strings with 'd', 'l' and 'u' respectively. There are some particularly frequent dependency relations between two tokens one of which is a string of digits. Occurrences of these relations are generalised by substituting each digit in the string of digits with the letter 'd', e.g., *w* ⟶ *2013* ⟶ *roku* (Eng. 'in 2013') and *w* ⟶ *1947* ⟶ *roku* (Eng. 'in 1947') are generalised to *w* ⟶ dddd ⟶ *rok*. The digit-based generalisations are not reduced to a single 'd' instance, but we keep their decimal structure. Other unrecognised strings are replaced with their generalised representations and reduced, e.g., 'PZKosz'[7] is generalised to 'uuulll' and reduced to 'ul'. Furthermore, we also replace Roman numerals, which are numerously represented in the corpus, but they are not recognised by the *Pantera* tagger. Roman numerals represented by strings of lowercase letters are replaced with 'r' and roman numerals represented by strings of uppercase letters are replaced with 'R', e.g., *XXI* ⟶ *wiek* (Eng. '21st century') is reduced to R ⟶ *wiek*. This solution should significantly reduce the number of possible lemma-based feature sets.

[5] For clarity, labels on arcs in the example tree are reduced to English grammatical functions.
[6] The state-of-the-art *Pantera* tagger recognises neither Arabic nor Roman numerals.
[7] *PZKosz* is an acronym for *Polski Związek Koszykówki* (Eng. 'Polish Basketball Federation').

5.2.3 Recalculation of Arc Weights in Projected Multi-Digraphs

Arcs of projected multi-digraphs encoding dependency relations are assigned initial weights. Initial arc weights are calculated on the basis of the certainty of bipartite edges used in projection of these arcs (see Section 5.1.3 *Intuitive Weighting Method*). Weights of bipartite edges, in turn, are estimated based on automatic word alignment which is prone to errors (see Section 5.4.2 *Experiments on Word Alignment*). We therefore propose a heuristic of recalculating initial arc weights in projected multi-digraphs. Recalculation of initial arc weights is based on the probability distribution over arcs in k-best maximum spanning dependency trees selected from initially weighted projected multi-digraphs. The probability distribution over arcs is estimated with a version of the expectation maximisation algorithm[8] defined by Dębowski (2009).

The EM selection algorithm by Dębowski (2009) was originally designed to select the most probable valency frames from sets of valency frame candidates. First, a language corpus is parsed with a rule-based parser (Woliński, 2005b) and parses of all sentential clauses are reduced to valency frames (reduced parses). This process results in a training set of valency frame forests with the number of reduced parses ≤ 40 per clause. The EM selection algorithm disambiguates valency frame forests and extracts a single valency frame with the largest conditional probability. Dębowski's algorithm is adapted for our purposes of identifying the most reliable arcs in sets of arcs in k-best MSDTs found in initially weighted projected multi-digraphs. The pseudocode of the EM-inspired selection algorithm is shown in Algorithm 5.3.

Assume we have a training set $\mathcal{B} = \{B_1, ..., B_N\}$, where B_i is a set of arcs of k-best MSDTs coming into the ith vertex, for $i = 1, ..., N$. In this setting, model parameters $\theta_t = \left(p_j^{(t)}\right)_{j \in J}$, where t is the iteration number, for $t = 2, ..., T$, and j is a feature representation of an arc, for $j \in J$ and J being a set of all possible feature representations of arcs in $\mathcal{B}$ (see Section 5.2.2 *Feature Representations of Arcs*), are estimated with the EM selection algorithm by Dębowski (2009).

The original EM selection algorithm by Dębowski (2009) iterates over the formulae in (5.5) and (5.7), and defines a series of parameter values θ_2, ..., θ_t until the last iteration. In the first step of each iteration, new parameter values $p_j^{(t)} = P(j|\theta_t)$ are estimated. The parameter value $p_j^{(t)}$ is the probability of the feature representation j in the tth iteration (for $t \geq 2$). The parameter value $p_j^{(t)}$ satisfies the following constraints: $p_j^{(t)} \geq 0$ and $\sum_{j \in J} p_j^{(t)} = 1$, except for the parameter values from the first iteration of the EM selection algorithm. The probability p_j of the jth feature representation is calculated by the division of the sum of p_{ij} (i.e., values for the feature representation j in each set B_i, for $i = 1, ..., N$) by the number of all vertices N (see Equation (5.7)).

[8] The standard expectation maximisation algorithm is presented in Dempster et al. (1977).

Algorithm 5.3 The EM-inspired selection algorithm.

EM($\mathcal{B}$, T)
$\mathcal{B} = \{B_1, ..., B_N\}$, where $B_i \subseteq \{(v_h, v_i, l) | v_h \in \{v_1, ..., v_N\} \setminus v_i\}$ is a set of arcs of k-best MSDTs that come into the vertex v_i
T = number of iterations of the EM selection algorithm
$s(v_h, v_i, l) \to \mathbb{R}$ – an arc weight function returning the initial weight for each arc $(v_h, v_i, l) \in \mathcal{B}$

For $j = 1, ..., |J|$, where J is the set of feature representations of arcs in k-best MSDTs
$p_j^{(1)} := 1$
For $i = 1, ..., N$
For $j = 1, ..., |J|$, calculate p_{ij}:

If $j \in B_i$ then $p_{ij}^{(1)} := \dfrac{p_j^{(1)} \times s(v_h, v_i, l)}{\sum\limits_{j' \in B_i} p_{j'}^{(1)} \times s(v_{h'}, v_i, l')}$

Else $p_{ij}^{(1)} := 0$

For $t = 2, ..., T$
For $j = 1, ..., |J|$ calculate new parameter values:

$$p_j^{(t)} := \frac{1}{N} \sum_{i=1}^{N} p_{ij}^{(t-1)}$$

For $i = 1, ..., N$
For $j = 1, ..., |J|$, calculate $p_{ij}^{(t)}$:

If $j \in B_i$ then $p_{ij}^{(t)} := \dfrac{p_j^{(t)} \times s(v_h, v_i, l)}{\sum\limits_{j' \in B_i} p_{j'}^{(t)} \times s(v_{h'}, v_i, l')}$

Else $p_{ij}^{(t)} := 0$

Return parameter values $\left(p_j^{(T)}\right)$

The second step of each iteration is to estimate values $p_{ij}^{(t)} = P(j|B_i, \theta_t)$, for each $i = 1, ..., N$ and for each possible feature representation of arcs $j \in J$. The value $p_{ij}^{(t)}$ is a conditional probability of an arc with the feature representation j assuming that this arc is in the set B_i and given the parameter values θ_t in the tth iteration. The coefficient $p_{ij}^{(t)}$ fulfils the following constraints: $p_{ij}^{(t)} \geq 0$ and $\sum_{j \in B_i} p_{ij}^{(t)} = 1$, for $i \in \{1, ..., N\}$ and $j \in J$. In the original version of the EM selection algorithm, the coefficient p_{ij} is a quotient of the probability value p_j of the arc j and the sum of probability values $p_{j'}$ of all arcs in the set B_i (see Equation (5.5)). We modify the way of estimating the coefficient p_{ij} in order to take into account the initial weight $s(v_h, v_i, l)$ of the arc represented as j (see Equation (5.6)). The coefficient p_{ij} is a quotient of the probability value p_j of the arc (v_h, v_i, l) with the feature representation j multiplied by the initial arc weight $s(v_h, v_i, l)$, and the sum of probability values $p_{j'}$ of all arcs in the set B_i multiplied by

the initial weights $s(v_{h'}, v_i, l')$ of corresponding arcs.

$$p_{ij}^{(t)} = \begin{cases} \dfrac{p_j^{(t)}}{\sum\limits_{j' \in B_i} p_{j'}^{(t)}} & \text{, if } j \in B_i \\ 0 & \text{, otherwise.} \end{cases} \tag{5.5}$$

$$p_{ij}^{(t)} = \begin{cases} \dfrac{p_j^{(t)} \times s(v_h, v_i, l)}{\sum\limits_{j' \in B_i} p_{j'}^{(t)} \times s(v_{h'}, v_i, l')} & \text{, if } fr(v_h, v_i, l) = j \text{ and } j \in B_i \\ 0 & \text{, otherwise.} \end{cases} \tag{5.6}$$

$$p_j^{(t+1)} = \frac{1}{N} \sum_{i=1}^{N} p_{ij}^{(t)} \tag{5.7}$$

The initial parameter values are set to 1, i.e., $p_j^{(1)} = 1$, as in the original approach by Dębowski (2009). At each iteration, the new parameter values θ_t are calculated as a function of the previous parameter values θ_{t-1} and the training set $\mathcal{B}$. The EM-inspired selection algorithm (iteration (5.6)–(5.7)) iterates until the final iteration T is reached.

According to the original procedure by Dębowski (2009), the most likely arc would be selected from the set of possible arcs B_i. However, the most probable incoming arcs for each lexical node do not have to necessarily constitute a valid dependency tree for a sentence (e.g., the resulting graph may contain a cycle). Therefore, our approach to recalculating weights does not build directly on the selected arcs but on the probability distribution over feature representations of arcs J estimated in the last iteration of the EM selection algorithm. The estimated probabilities of relation types $\left(p_j^{(T)}\right)$ are used to recalculate initial weights on arcs in projected multi-digraphs.

The new weight of an arc (v_h, v_i, l) with the feature representation j is calculated as the product of the square root[9] of the previous arc weight and the value p_j (see Equation (5.8)).

$$s^* = \sqrt{s(v_h, v_i, l)} \times p_j, \qquad \text{for } fr(v_h, v_i, l) = j \tag{5.8}$$

If an arc is not present in any of the extracted k-best MSDTs, its probability value is equal to 0. Because there is a risk that some multi-digraph arcs would be assigned 0 and they would have the same priority in the extraction of final maximum spanning dependency trees, we assign them the following value:

[9] Arcs in projected multi-digraphs are assigned initial weights from $\mathbb{N}_+$. In order to diminish the difference between multiplied initial weights and probability values, and therefore to raise the importance of relatively low probability values, initial weights are square rooted.

$$s^* = \sqrt{s(v_h, v_i, l)} \times \min_j p_j \times \alpha, \qquad \text{for some } 0 < \alpha < 1 \tag{5.9}$$

The new score s^* of an unselected arc (v_h, v_i, l) is the product of the square root of the initial arc weight, the lowest value p_j in $\left(p_j^{(T)}\right)$ and an optimisation factor α (for some $0 < \alpha < 1$) which further decreases weights of unselected arcs.

The idea behind these modifications is to optimise initial weights of arcs in projected multi-digraphs. Arcs with particular feature representations which have multiple instances in sets of k-best MSDTs should get higher scores than other arcs. On the other hand, arcs with particular feature representations which are not in J should get rather low scores. Arcs with higher weights are more likely to be selected as part of final dependency trees.

Finally, maximum spanning dependency trees are selected from projected multi-digraphs with recalculated arc weights. For the purpose of selecting one maximum spanning dependency tree from a multi-digraph with recalculated arc weights, we apply the k-best MSDT algorithm (see Appendix D) with $k = 1$. The entire induction procedure results in a collection of dependency structures labelled with English grammatical functions. Arc labels need to be adapted to Polish dependency types as defined by the schema of annotating Polish dependency structures. The labelling procedure is described in the subsequent sections.

5.3 Rule-based Adaptation of Polish Dependency Structures

This section presents a procedure of labelling automatically induced dependency structures in accordance with the annotation schema presented in Chapter 3 *Polish Dependency Annotation Schema*. Induced dependency structures encode domination relations between Polish tokens accepted by the dependency annotation schema. However, these relations are labelled with projected English grammatical functions which need to be adapted to dependency types defined for Polish. For this purpose, we design a rule-based labeller that annotates dependency relations in induced dependency trees with appropriate Polish grammatical functions. Polish labels are inferred from projected English grammatical functions and morphosyntactic features of Polish related tokens.

5.3.1 Labelling Rules

Relations in induced dependency structures are labelled with English grammatical functions which differ from dependency types defined for Polish in terms of naming convention and their functionality. Regarding the naming convention, differences are rather

insignificant since labels of Polish dependency relations are largely inspired by the English LFG grammar which is used to parse English sentences. Labels of many Polish arguments and non-arguments have their equivalents in *Lexical Functional Grammar* (e.g., *comp_fin* vs. COMP, *comp_inf* vs. XCOMP, *obj* vs. OBJ, *obj_th* vs. OBJ-TH, *pd* vs. XCOMP-PRED, *subj* vs. SUBJ or *adjunct* vs. ADJUNCT). Other Polish labels are derived either from grammatical functions of the English LFG grammar (e.g., *comp_ag* vs. OBL-AG, *adjunct_qt* vs. ADJUNCT-QT or *pre-coord* vs. PRECOORD-FORM) or from dependency types designed for the purpose of converting English f-structures into dependency structures (e.g., *aux* vs. AUX, *complm* vs. COMP-FORM, *conjunct* vs. CONJUNCT, *coord* vs. COORD-FORM, *punct* vs. INTERP, or *neg* vs. NEG). Finally, there are also some Polish dependency types which are language-specific and have no English equivalents (e.g., *imp*, *refl*, *aglt* or *cond*).

Even if Polish and English dependents have similar labels, they may differ in terms of functions they fulfil. In the LFG f-structure of the sentence *He gave her flowers.*, the pronoun *her* is assigned the object function (OBJ) and the noun *flowers* is assigned the thematically restricted object function (OBJ-TH). In the corresponding Polish sentence *Dał jej kwiatki.*, the noun *kwiatki* (Eng. 'flowers') fulfils the object function (*obj*) since it is promoted to subject in passivisation (*Zostały jej dane kwiatki.*, Eng. 'Flowers were given to her.'). The pronoun *jej* (Eng. 'her'), in turn, fulfils the *obj_th* function since it is a non-passivisable and thematically restricted (Recipient) argument realised as a noun phrase. Direct projection of grammatical functions from English dependency structures to Polish results in Polish verb arguments which are assigned the exact opposite functions than required. The Polish thematically restrictive object *jej* is assigned the OBJ function and the object *kwiatki* is annotated with OBJ-TH (see Figure 5.13). Therefore, we need to define labelling rules that cover such syntactic shifts.

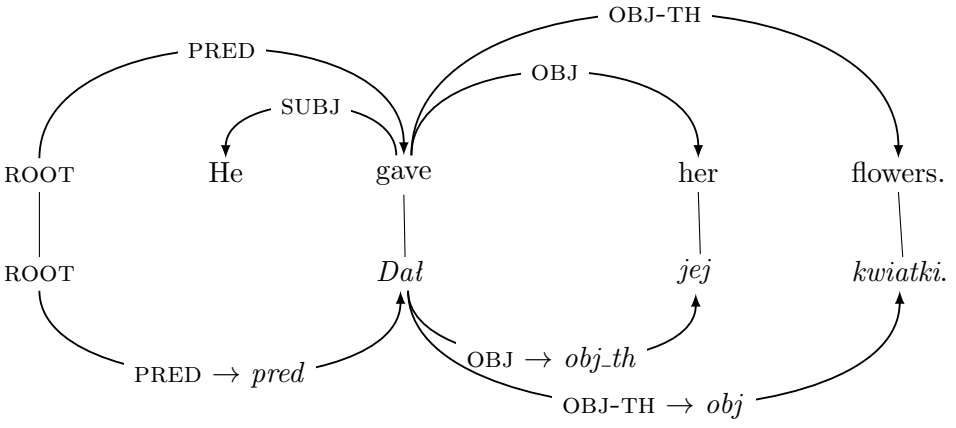


FIGURE 5.13: Divergences in labelling relations in the Polish dependency tree (bottom tree) based on the English dependency tree (top tree).

The definition of labelling rules was preceded by an analysis of the most common configurations of projected English grammatical functions and morphosyntactic properties

of related Polish tokens. The analysis was performed on 30,322 induced dependency structures (developing set) taken from the entire treebank. Furthermore, the labelling procedure also applies information about the number and types of arguments subcategorised by some verbs or quasi-verbal predicates. This information is extracted from the Polish Valency Dictionary *Walenty*.[10]

The labelling process consists of three successive steps in which induced relations are assigned labels. In the first two steps, we consider only relations between verbs which are represented in the valency dictionary and their dependents. In the first step, candidate Polish dependency labels should directly map to projected English grammatical functions as defined in Algorithm 5.4. In the second step, the functional correspondence between English and Polish labels is not necessary. In the third step, we label all relations which remain unlabelled after the first two steps. The three steps are described in more detail below.

In the first step, it is verified whether a dependent of a verb is its argument according to one of valency frames associated with this verb in the valency dictionary. The dependent is considered to be an argument fulfilling a particular grammatical function only if two conditions are satisfied. First, the dependent and its governing verb have some predefined morphosyntactic properties. Second, the candidate grammatical function which is proposed by a valency frame corresponds to the projected English function which is already assigned to the dependent.

Let's look at the following example of the first step of the labelling process. The top dependency tree in Figure 5.14 corresponds to an automatically induced dependency structure of the Polish sentence *Policjant analizuje dowody.* (Eng. 'The policeman analyses the evidence.'). This sentence contains only one verb form *analizuje*, which is the sentence predicate. There are two frames which may be mapped to dependents of *analizować* in the employed version of the valency dictionary:

1. `analizować: imperf: subj{np(str)} + obj{cp(int)}`,
2. `analizować: imperf: subj{np(str)} + obj{np(str)}`.

Both frames contain the *subj* argument which should be realised as a noun phrase marked for the structural case. Since most subjects bearing the structural case are in nominative,

[10]The Polish Valency Dictionary *Walenty* is being developed at the Institute of Computer Science of the Polish Academy of Sciences. We use the version of the dictionary released on 30 January 2013. This version contains subcategorisation frames for 1774 Polish verbs and quasi-verbal predicates. The dictionary is publicly available at `http://clip.ipipan.waw.pl/Walenty`. For the purpose of labelling, entries in the valency dictionary have been slightly modified. The modification consists in assigning grammatical functions to all unlabelled verb arguments: unlabelled noun phrase arguments of predicative verbs are assigned the *pd* function, other unlabelled noun phrase arguments are assigned the *obj_th* function, adjective arguments of predicative verbs are assigned the *pd* function, clausal complements are assigned the *comp_fin* function, infinitival complements are assigned the *comp_inf* function and all other arguments of verbs are assigned the *comp* function.

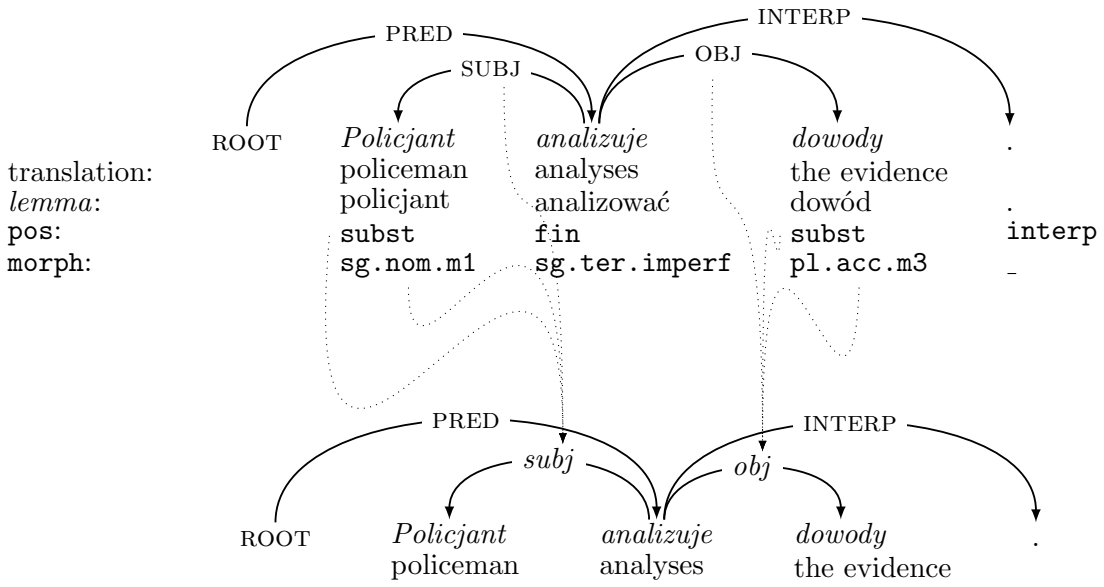


FIGURE 5.14: Labelling dependency relations (1st step). The top tree is an initial dependency tree the relations of which are labelled with projected English grammatical functions. The bottom tree contains some dependency relations labelled according to the Polish dependency annotation schema after the first step of the labelling process. The dotted arrows indicate the dependency labels modified on the basis of information from the tails of arrows.

we look for a dependent of the predicate *analizuje* that is realised as a nominative noun phrase. There is one dependent that fulfils morphological properties of the subject – *Policjant*. The first condition is thus satisfied. It is then checked if the English grammatical function assigned to this dependent corresponds to the *subj* argument. According to Algorithm 5.4, the SUBJ function assigned to the dependent *Policjant* corresponds with the frame argument *subj*. The second condition is thus satisfied.

The second argument of the first frame is an interrogative clausal complement fulfilling the *obj* function. However, the predicate *analizuje* does not have any dependent that could morphosyntactically correspond to this argument. Therefore, there is only one argument in the first frame that maps to dependents of *analizuje*. The second argument of the second frame is an object realised as a noun phrase bearing the structural case (accusative or genitive of negation). There is a dependent of the sentence predicate that is realised as a noun phrase marked for accusative *dowody*. Moreover, according to the functional correspondence rules (see Algorithm 5.4), the English grammatical function OBJ, which the dependent is annotated with, corresponds to the *obj* argument, which is given by the frame. Hence, there are two arguments of the second frame that map to dependents of the predicate *analizuje*. A general strategy for identifying the best frame is to count up mappings between verb dependents and frame arguments, and to select the frame with the largest number of mappings. Since the second frame maps to both verb dependents, the grammatical functions from this frame are used to label appropriate dependents in the tree (see the bottom tree in Figure 5.14).

Algorithm 5.4 Functional correspondence.

functionalCorrespondence($gf_{\mathbf{f}}$, $gf_{\mathbf{d}}$, $pos_{\mathbf{d}}$)

$gf_{\mathbf{f}}$ is a grammatical function given by the valency frame
$gf_{\mathbf{d}}$ is an English grammatical function currently assigned to the dependent
$pos_{\mathbf{d}}$ is a part of speech of the dependent

If $gf_{\mathbf{f}}$ = *comp* and $gf_{\mathbf{d}}$ = OBL
    Return True
Elif $gf_{\mathbf{f}}$ = *comp_fin* and $gf_{\mathbf{d}}$ = COMP
    Return True
Elif $gf_{\mathbf{f}}$ = *comp_inf* and $gf_{\mathbf{d}}$ = XCOMP
    Return True
Elif $gf_{\mathbf{f}}$ = *obj*
    If $gf_{\mathbf{d}}$ = OBJ and $pos_{\mathbf{d}}$ in [`subst, depr, num, numcol, ppron12, ppron3, ger siebie, conj, ign`]
        Return True
    Elif $gf_{\mathbf{d}}$ = COMP and $pos_{\mathbf{d}}$ in [`fin, impt, praet, winien, pred, ppas`]
        Return True
    Elif $gf_{\mathbf{d}}$ = XCOMP and $pos_{\mathbf{d}}$ = `inf`
        Return True
Elif $gf_{\mathbf{f}}$ = *obj_th* and $gf_{\mathbf{d}}$ = OBJ
    Return True
Elif $gf_{\mathbf{f}}$ = *subj* and $gf_{\mathbf{d}}$ in [SUBJ, XCOMP-PRED, OBJ-AG]
    Return True
Elif $gf_{\mathbf{f}}$ = *pd* and $gf_{\mathbf{d}}$ = XCOMP-PRED
    Return True
Return False

In the second step of the labelling process, we once again take into account only relations between verbs which are represented in the valency dictionary and their dependents. However, there are some additional restrictions on this labelling step. First, it is allowed to modify only these relation labels which correspond to English grammatical functions. Second, the set of frames which are taken into account is restricted to those which contain already assigned arguments. Finally, there is a restricted set of Polish arguments (i.e., *subj*, *obj*, *pd*, *comp_fin*, *comp_inf*) which are not allowed to be repeatedly governed by a verb. The idea is to label a dependent of a verb with a candidate Polish grammatical function which is in the set of possible and not multiplied arguments defined by a frame. However, the English grammatical function currently assigned to the dependent doesn't have to correspond to the candidate label as in the first labelling step.

The second step of the labelling process is illustrated in Figure 5.15, which shows an example of a dependency structure automatically induced for the Polish sentence *Innym problemem jest nieobecność kobiet w polityce.* (Eng. 'Another problem

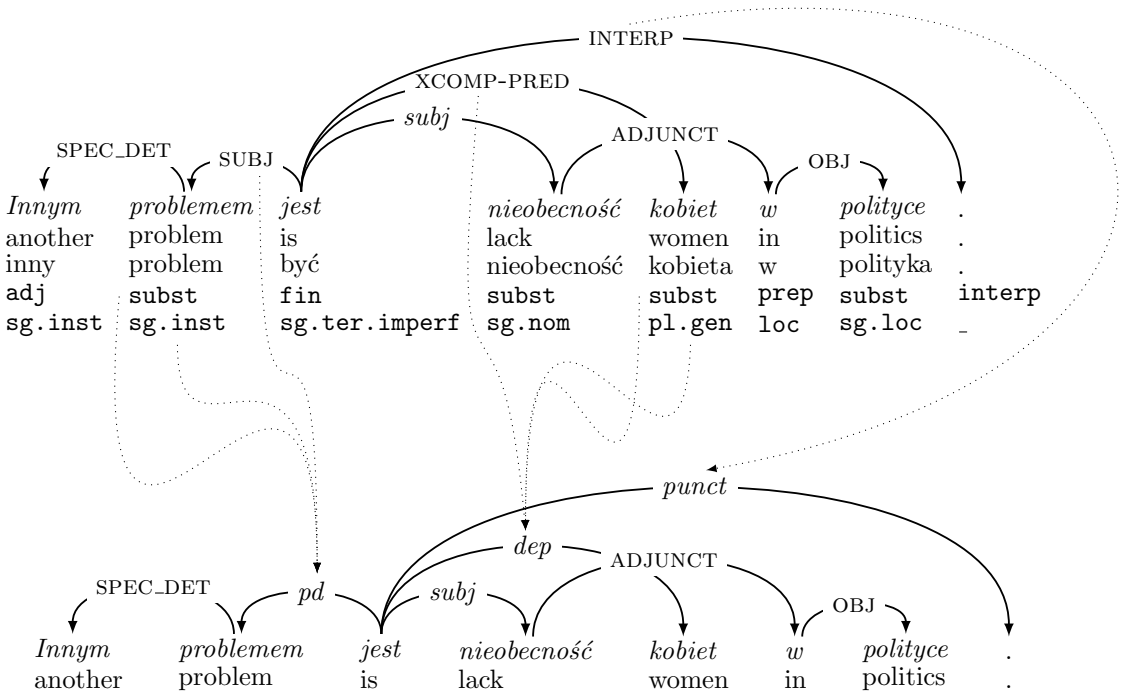


FIGURE 5.15: Labelling dependency relations (2nd step) in the automatically induced dependency structure. The top tree is a dependency tree the relations of which are labelled with both projected English grammatical functions and Polish grammatical functions assigned after the first step of the labelling procedure. The bottom tree contains some dependency relations assigned in the second step of the labelling process. The dotted arrows indicate dependency types modified in the second labelling step based on information indicated by the tails of arrows.

is the lack of women in politics.'). The top tree in Figure 5.15 contains the relation *jest* $\xrightarrow{subj}$ *nieobecność* labelled with the *subj* function in the first step of the labelling process. This tree is considered to be an input in the second step of the labelling process. The input tree contains only one verb form *jest* (the sentence predicate), which has 31 various frames in the valency dictionary. However, only 10 of these frames contain an argument represented as a nominative noun phrase bearing the *subj* function. Hence, these 10 frames are taken into account in this labelling step. Starting from the dependent with the smallest index *problemem*, we try to map it to a possible argument which is in at least one of the selected valency frames. The dependent *problemem* is a noun marked for instrumental and is labelled with the English SUBJ function. Since the *subj* function is already reserved and may not be doubled, other functions (e.g., *obj*, *pd*, *obj_th*) are thus examined. The dependent *problemem* maps to the argument `controlee{np(inst)}` of the valency frame `być: imperf:`

`subj,controller{np(str)} + controlee{np(inst)}`[11] and is assigned the *pd* function. This frame is best suited to labelling arguments of the predicate *jest* in the example sentence.

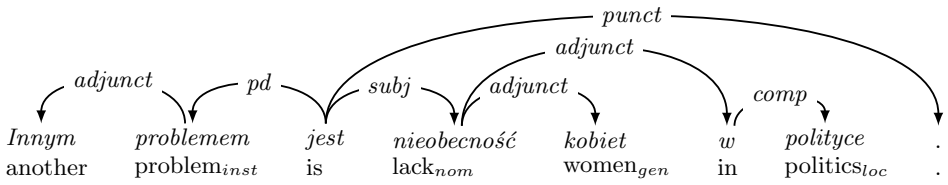
The example structure in Figure 5.15 is taken from the automatically induced treebank without any enhancement in order to illustrate assignment of the default type *dep* to incorrect arcs.[12] In the induced tree, the token *kobiet* is incorrectly annotated as the dependent of the sentence predicate *jest* instead of functioning as an adjunct of *nieobecność*. Since *być* does not admit any argument realised as a genitive noun phrase, the incorrect relation between *jest* and *kobiet* is labelled with the default function *dep*. The full stop is the last dependent of the predicate *jest* in our example tree. It is identified on the basis on its part of speech tag *interp* and is labelled with *punct*.[13] The bottom tree given in Figure 5.15 results from the second labelling step.

In the last step of the labelling process, relations which remained unlabelled are annotated with dependency types. We start with a presentation of rules labelling relations in which the ROOT node fulfils the role of the governor. The ROOT node can govern only one dependent in an induced dependency tree and this dependent should be labelled with one of the following grammatical functions:

- *coord* if the dependent is realised as a conjunction coordinating sentences or clauses, except for the sentence predicate coordination,[14]
- *coord_punct* if the dependent is realised as a punctuation mark coordinating sentences or clauses, except for the sentence predicate coordination,
- *pred* if the dependent is realised as any part of speech including a conjunction or a punctuation mark coordinating sentence predicates (except for a conjunction or a punctuation mark coordinating sentences or clauses).

[11] The control relations – `controller` and `controllee` – represented in the Polish valency dictionary are currently not considered in the labelling process. The `controller` argument controls the syntactic form of another element determined as `controllee`.

[12] The correct dependency structure for the example sentence should looks like this one:



[13] The relation between the predicate *jest* and the full stop is labelled using Rule 20. from Appendix A, p. 164. Note that for conciseness of the description, an overview of defined labelling rules is presented in the further course of this chapter. However, it is important to notice that some of labelling rules apply in this step in order to label non-argument dependents of verbs represented in the valency dictionary.

[14] A construction with the sentence predicate coordination is characterised by the following properties: coordinated sentence predicates agrees in terms of number and possibly person; there is a dependent of the coordinating element which fulfils the subject function and this subject is shared by coordinated sentence predicates.

Now, we present rules used to label other relations in which almost every node except for the ROOT node may function as a governor. First, we shortly describe a set of 31 rules designed to identify language-specific relations on the basis of morphosyntactic properties of related tokens and to label these relations with appropriate dependency types (see Appendix A). Then, we present a set of 14 labelling rules which are primarily based on projected English grammatical functions (see Appendix B).

Labels of dependencies encoding some language-specific phenomena may not be inferred from English grammatical functions since they are assigned a wide variety of functions. For example, mobile inflections, which do not have any structural equivalent in English, are labelled with AUX, SUBJ, ADJUNCT, COMP, OBJ, PART, POSS, PRED, SPEC-DET, SPEC-POSS, XCOMP and XCOMP-PRED in the developing set of induced dependency structures. Relations encoding language-specific phenomena are clearly distinguishable from other relations on the basis of morphosyntactic properties of related tokens, e.g., parts of speech, lemmata, surface forms, preceding or succeeding positions of dependents in relation to their governors and morphological features such as case or number.

We define some rules labelling complement relations, adjunct relations and relations encoding multi-word expressions (see Appendix A *Labelling Rules Based on Morphosyntactic Properties*). Among complement relations (labelled with the *comp* function), there are prepositional phrases functioning as complements of constituents marked for the comparative degree (Rule 1., p. 161), complements of numbers (Rule 2., p. 161), complements of numerals (Rule 4., p. 162), complements of prepositions (Rules 7–10, pp. 162–162) and complements of subordinating conjunctions (Rule 12., p. 163). The following rules label multi-word expressions with the *mwe* function: 3., 5., 6. and 11. (pp. 161–163). Among adjunct relations (labelled with the *adjunct* function), there are adverbial modifiers of adverbs and adjectives (Rule 13., p. 163), modifiers of nouns (Rules 14–18, pp. 163–164), modifiers of unknown tokens (Rule 19., p. 164), and adjuncts of verb forms (Rule 30., p. 166). There are also other rules employed to label relations governed by a verb form.[15] The following relations are covered by these rules: a relation between a main verb and an auxiliary verb (*aux*, Rule 21., p. 164), a relation between a verbal head of a subordinating clause and a complementiser (*complm*, Rule 22., p. 164), a relation between a verb and a conditional clitic (*cond*, Rule 23., p. 165), a relation between a verb and an imperative marker (*imp*, Rule 24., p. 165), a relation between a verb and a mobile inflection (*aglt*, Rule 25., p. 165), a relation between a verb and a negation marker (*neg*, Rules 26 and 27, p. 165), a relation between a verbal head of an interrogative clause and a question marker *czy* (*adjunct*, Rule 28., p. 165), and a relation between a verb and its reflexive marker (*refl*, Rule 29., p. 166). It should also be noted that some of these rules (i.e., Rules 10., 17., 18., 19. and 30.) are treated as endmost rules which do not apply until all other labelling rules (also rules from Appendix B) are exploited.

[15] It should be noted that these rules also apply in the second step of the labelling process to label dependents of verbs not covered with valency frames.

Rules from the second set (see Appendix B *Labelling Rules Based on English Grammatical Functions*) assign labels to relations identified primarily on the basis of their projected English grammatical functions. We distinguish between labelling rules designed for English argument types[16] and other grammatical functions. Labelling rules are defined for the following English argument types: closed complement clause and open complement clause (COMP and XCOMP,[17] Rule 1, p. 167), object and thematically restricted object (OBJ and OBJ-TH, Rule 2., p. 168), oblique argument (OBL, Rule 3., p. 168), oblique agent in passives (OBL-AG, Rule 4., p. 169), comparing oblique in comparatives and equatives (OBL-COMPAR, Rule 5., p. 169), oblique partitive (OBL-PART, Rule 6., p. 169), subject (SUBJ, Rule 7., p. 170) and open complement in predicative position (XCOMP-PRED, Rule 8., p. 170). Other labelling rules cover the following dependents: adjunct (ADJUNCT, Rule 9., p. 171), quotation adjunct (ADJUNCT-QT, Rule 10., p. 171), modifying noun in a noun-noun compound (MOD, Rule 11., p. 171), modifying name in a proper name (NAME-MOD, Rule 12., p. 171), precoordination (PRECOORD, Rule 13., p. 172) and cleft sentence (TOPIC-CLEFT, Rule 14., p. 172).

Mapping of English grammatical functions to Polish arcs should result in completely labelled dependency trees. In the third labelling step, unlabelled arcs of the partially labelled dependency tree (see Figure 5.15) are assigned labels and the final dependency tree is acquired (see Figure 5.16).

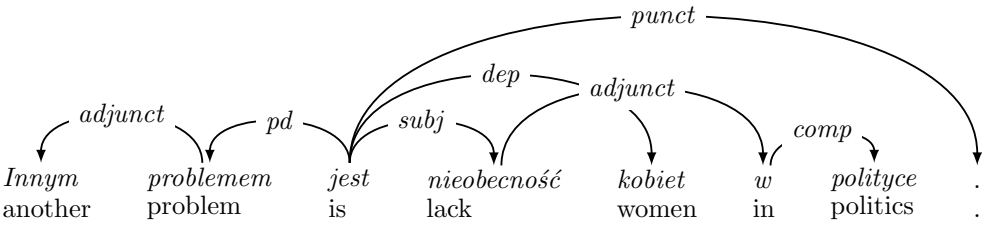


FIGURE 5.16: An induced dependency tree labelled with Polish dependency types (3rd step).

5.3.2 Correction Rules

Parallel sentences in Polish and English tend to have correlated dependency structures. This assumption seems to be largely true if we consider semantic predicate-argument structures of corresponding sentences. However, a syntactic realisation of the semantic predicate-argument structure may differ in both languages. For example, an argument of a predicate may be realised as a noun phrase possibly marked for a particular case in one language and as a prepositional phrase in the other language (see *I'm going* ***home.*** vs. *Idę* ***do domu.***, *Piszę* ***ołówkiem.*** vs. *I write* ***with a pencil.***).

[16]These rules only label relations between verbs which are not in the valency dictionary and their arguments.

[17]An open complement clause is a subordinate clause without an internal subject.

We have already described labelling rules that cover many discrepancies between Polish and English dependency types, e.g., the noun *home* with the OBL function should correspond to the prepositional phrase *do domu* labelled with the *comp* function. However, labelling rules assume that induced dependency structures are correct in terms of domination relations between tokens. Even if many Polish induced dependency structures are correct, there are still some induced structures that are noisy. If the prepositional phrase *do domu* is wrongly annotated as two independent tokens governed by the sentence predicate *idę* in an induced dependency structure of the sentence *Idę do domu.*, it will be impossible to label all relations in this dependency structure with appropriate dependency types. For this reason, we apply the idea by Hwa et al. (2005) to use a predefined set of correction rules.

Even though the induction process is straightforward, there are still some Polish-specific morphosyntactic phenomena or phenomena diversely annotated in both languages whose correct annotations may not be induced based on English dependency structures. Moreover, noise in induced dependency structures may result from erroneous English dependency structures, incorrect word alignment or an inaccurate induction process. A linguistic analysis of trees in the developing set indicates types of errors or divergences that occur most frequently in induced dependency structures. These error types are covered with 31 correction rules presented in Appendix C *Correction Rules*.

The following phenomena not reflected in English dependency structures are identified: conditional clitic, imperative marker, mobile inflection, and reflexive marker. Errors in annotations of these phenomena are corrected with manually designed modification rules. The conditional particle *by* is used to indicate the conditional modality of a sentence. Regardless of whether it is appended to a verb or appears anywhere in the sentence, the conditional particle should be governed by a verb (Rule 1. in Appendix C, p. 173). An imperative marker (e.g., *niech*, *niechaj*, *niechże* or *niechajże*) should also be governed by a verb (Rule 2., p. 174). The mobile inflection is a verbal enclitic marked for number and person. It should depend on a finite verb or a conditional clitic *by* appended to such a verb (Rule 3., p. 174). The reflexive marker realised as the particle *się* should be governed by a verb (4., p. 174).

Apart from rules covering linguistic phenomena which are not reflected in English dependency structures, some correction rules are designed to cover Polish-specific linguistic constructions analysed differently in both languages, i.e., negation markers and numeral complements. The negation marker *nie* should be annotated as the dependent of the subsequent token which is usually a verb (Rule 6., p. 174). In English, a numeral is treated as a number specifier (SPEC-NUMBER) of the governing noun. In Polish, in turn, a numeral governs a noun phrase whose case either agrees with the case of the governing numeral or is determined as genitive. Polish numeral phrases in induced dependency structures are modified in accordance with the dependency annotation schema. Rule 5. (p. 174) covers numeral phrases realised as a numeral plus a noun phrase, e.g., '*dwoje*

dzieci' (Eng. 'two children') and as a number plus a noun phrase, e.g., '*2* dzieci' (Eng. '2 children').

There are also some rules designed to correct errors in induced dependency structures resulting from inaccurate part of speech tagging, incorrect English dependency trees or invalid word alignment. These correction rules modify wrongly annotated abbreviation markers (Rule 7., p. 174), active adjectival participles (Rule 8., p. 175), ad-adjectival adjective phrases (Rule 9., p. 175), appositions (Rule 10., p. 176), auxiliary verbs in complex future constructions (Rule 11., p. 176), auxiliary verbs in passive constructions (Rule 12., p. 176), comparative phrases (Rules 13 and 14, pp. 176–177), complementisers (Rule 15., p. 177), conjunct dependents (Rules 16 and 17, p. 177), various modifiers (Rules 19, 20, 21, 22, and 23, pp. 178–179), partitive phrases (Rule 24., p. 179), post-prepositional adjectives (Rule 25., p. 180), prepositional complements (Rule 27., p. 180), and punctuation dependents (Rule 28., p. 181).

Furthermore, there are also some rules that modify the global structure of sentences headed by *mieć* (Rule 18., p. 178), predicative constructions (Rule 26., p. 180), sentences headed by a non-verb (Rules 29 and 30, p. 181) and simple interrogative sentences (Rule 31., p. 182).

5.4 Experimental Setup

5.4.1 Data

In order to acquire dependency structures with the weighted induction method, a large collection of Polish-English parallel texts was gathered from sources available on the Internet: *European Parliament Proceedings Parallel Corpus* (*Europarl,* Koehn, 2005), *DGT-Translation Memory* (Steinberger et al., 2012), OPUS (Tiedemann, 2012) and *Pelcra Parallel Corpus* (Pęzik et al., 2011). *Europarl* (version 6) consists of parallel texts from the proceedings of the European Parliament in 21 European languages. The corpus of translation memories *DGT-TM* consists of aligned sentences principally taken from the *Acquis Communautaire* in 23 European languages. The open parallel corpus OPUS is a collection of translated texts collected from the web. We use the Polish-English part of the OPUS corpus of the European Medicines Agency documents, the text of the European constitution, a PHP manual, and KDE4 localisation files. From the *Pelcra Parallel Corpus*, we employ aligned sentences from *Academia. The Magazine of the Polish Academy of Sciences*, from CORDIS – a news database, and from RAPID – press releases of the European Union. Furthermore, we also select some parallel texts form the *Official Journal of the European Union* (EUR-Lex), film subtitles, and contemporary literature resources.

Collected documents were split into sentences using the sentence tokenizer *Punkt* (Kiss and Strunk, 2006), which is distributed with the NLTK toolkit (Bird et al., 2009). The sentence-segmented bitexts were then aligned at the sentential level using the sentence aligner *HunAlign* (Varga et al., 2005). The *HunAlign* aligner is based on a hybrid algorithm that combines two methods of aligning sentences in parallel texts: the length-based method (Gale and Church, 1991) and the dictionary-based method (Chen, 1993; Moore, 2002). The process of aligning sentences consists of three phases. *HunAlign* starts with the identification of corresponding segment pairs[18] based on the segment length and the ratio of identical tokens. Then, a simple bilingual lexicon is automatically built from randomly sampled segment pairs. Finally, alignment is rebuilt based on the extracted bilingual lexicon. The number of segment pairs gathered from individual parallel corpora is reported in the column **Sentence Pairs** in Table 5.1.

Text	**Sentence Pairs**	**Filtered Bitexts**		
		Sentence Pairs	**En t/s**	**Pl t/s**
Europarl	448,433	429,929	27.43	24.06
DGT-TM	1,052,136	736,822	26.81	23.96
EMEA (OPUS)	868,808	193,274	19.55	18.98
EU Constitution (OPUS)	9,937	5,271	25.85	21.36
PHP (OPUS)	33,687	4,990	20.51	18.32
KDE4 (OPUS)	165,172	105,609	10.76	10.38
Academia	17,859	12,273	20.90	15.35
CORDIS	176,558	140,005	23.65	22.62
RAPID	146,908	106,713	27.72	26.02
EUR-Lex	4,063,336	2,558,306	31.19	28.69
Subtitles	6,871,465	4,120,543	8.34	6.99
Literature	911,278	854,422	17.02	14.73
TOTAL:	14,765,577	9,268,157		

Table 5.1: A statistical overview of the parallel Polish-English corpus after filtering multiple and unrecognised segment pairs. Explanation: En t/s – the average number of English tokens per sentence after filtering; Pl t/s – the average number of Polish tokens per sentence after filtering.

A cursory analysis showed that many of segment pairs occur repeatedly within collected bitexts aligned at the sentential level, especially in the legislative texts, e.g., terms of political jargon or the legislative terminology such as 'Opening of the sitting', 'Adjournment of the session', or 'Article 1'. Since the idea behind the experiment is to build a bank of non-repetitive dependency structures, all multiple segment pairs were filtered out.

[18]The procedure of aligning documents at the sentential level results in a collection of Polish-English sentence pairs in most cases. However, since it may also result in aligned sequences of tokens that do not correspond to properly built sentences, we thus refer to them as Polish-English segment pairs rather than sentence pairs.

Moreover, some segments appear in the collected texts in languages other than the concerned languages – Polish and English. For example, translations of an expression in all official languages of the European Union may appear in a Polish or/and English document. These segments are not appropriate for our experiment since we aim to create a bank of dependency structures restricted to Polish. Therefore, we filtered out all segment pairs such that the Polish segment contained more than 60% of tokens unrecognised by the Polish part of speech tagger *Pantera* (Acedański, 2010).

After application of the filtering heuristics, 37% of segment pairs were filtered out from the parallel corpus. A statistical overview of the remaining data is presented in the column **Filtered Bitexts** in Table 5.1.

5.4.2 Experiments on Word Alignment

Bitexts aligned at the sentential level were used to generate automatic word alignments. Unidirectional word alignments (Polish-to-English and English-to-Polish) were learnt with the statistical machine translation system MOSES (Koehn et al., 2007) based on statistics captured from the entire corpus. The MOSES system is distributed with modules that symmetrise unidirectional word alignments based on different heuristics, e.g., `union`, `intersection` and `grow-diag-final-and`.

We conducted a series of experiments to achieve the best Polish-English word alignments. The word alignment quality is evaluated against a set of 100 manually aligned sentence pairs[19] with the following metrics: *precision*(A,S) (see Equation (5.10)), *recall*(A,S) (see Equation (5.11)), *alignment error rate*, *AER*(A,S)[20] (see Equation (5.12)), and F_α-*measure*(A,S), which is defined in Fraser and Marcu (2007) (see Equation (5.13)). In these metrics, A is a set of automatic word alignment links, S is a set of gold standard word alignment links, and α is a parameter determining a trade-off between precision and recall ($\alpha = 0.5$ in our evaluation experiments).

$$\textit{precision}(A, S) = \frac{|S \cap A|}{|A|} \tag{5.10}$$

[19]The quality of word alignment is evaluated against a manually annotated gold standard corpus that was created for purposes of the presented experiments. The gold standard corpus consists of 100 sentence pairs randomly selected from the entire parallel corpus. The selected sentence pairs were manually annotated by three annotators (native speakers of Polish proficient in English). The annotators followed guidelines for solving some problematic issues (e.g., annotation of articles, case markers, dropped pronouns, reflexive markers, dates or subordinate clauses). The manually produced annotations were unified according to the following procedure. If at least two annotators assigned the same link, it was determined as the gold/sure alignment link.

[20]The alignment error rate (AER) is commonly used to evaluate word alignments. Indicating the rate of alignment errors, it lays the foundation for improvements of the word alignment quality. However, Fraser and Marcu (2007) criticised the AER metric as not showing any correlation between the measured alignment quality and the statistical machine translation performance.

$$recall(A, S) = \frac{|S \cap A|}{|S|} \tag{5.11}$$

$$AER(A, S) = 1 - \frac{2|S \cap A|}{|S| + |A|} \tag{5.12}$$

$$F_\alpha\text{-measure}(A, S) = \frac{1}{\frac{\alpha}{precision(A,S)} + \frac{(1-\alpha)}{recall(A,S)}} \tag{5.13}$$

In the first experiment, we tested different training configurations of IBM models 1–4 (Brown et al., 1993) and the Hidden Markov model (HMM, Vogel et al., 1996). While the remaining IBM models 5 and 6 are sophisticated, they did not significantly improve the alignment quality. Furthermore, training these models is very time consuming. For this reason, we limited the training procedure to IBM models 1–4 and HMM. The experiment was conducted on parallel sentences with lowercase tokens. Tokenisation and lowercasing were carried out with the standard scripts from the MOSES toolkit. Unidirectional word alignments were symmetrised with the `grow-diag-final-and` heuristic which is also implemented as part of the MOSES toolkit. The best word alignment (76.8 of F_α-score) for the Polish-English language pair was learnt with the following configuration of models: 10 iterations of the IBM model 1 followed by 10 iterations of HMM, 5 iterations of the IBM model 3 and 5 iterations of the IBM model 4. It is also worth noting that the best word alignment achieved not only the highest F_α-score but also quite balanced precision and recall.

The second experiment verified whether the `grow-diag-final-and` symmetrisation heuristic is the best choice for unifying unidirectional Polish-English word alignments. We evaluated word alignments symmetrised with `intersection`, `union`, `grow-diag-final` and `grow-diag-final-and` heuristics. The quality of the `grow-diag-final-and` symmetrised word alignment is better (76.8 F_α-measure) than the quality of word alignments symmetrised with other heuristics. Word alignment symmetrised with the `intersection` heuristic contains the most reliable alignment links (high precision), but not all of them (low recall). In contrast to `intersection`, `union` results are quite surprising. We had expected recall to be significantly higher than precision, but we obtained almost balanced values instead.

Word alignment underlies statistical machine translation which is one of the most important and deeply explored topics in the language processing domain. Therefore, a lot of effort has been invested in the improvement of the quality of word alignment. Some ideas consist in the reduction of the vocabulary size with stemming (Fraser and Marcu, 2005)

or lemmatisation (Bojar and Hajič, 2008). Both referenced works consider inflecting-isolating language pairs, i.e., Romanian-English (Fraser and Marcu, 2005) and Czech-English (Bojar and Hajič, 2008) and report some improvement of the word alignment quality.

Languages used in our experiment also represent two language types: Polish is a fusional language with a large number of inflected word forms; English, in turn, is an isolating language that makes use of function words. Polish often needs fewer words than English to express the same content. On the other hand, Polish lexemes have many more different forms than English lexemes what does substantially increase the Polish vocabulary size. In order to reduce the vocabulary size, the Polish side of the parallel corpus can be lemmatised.

In the third experiment, we tested the impact of lemmatisation on the word alignment quality. There is no crucial improvement while training word alignment on data lemmatised on both sides (76.4 of F_α-score).[21] However, if unidirectional word alignments are trained on the parallel corpus with the lemmatised Polish side and the English side which is only tokenised, an improvement of the word alignment quality is significant: the highest precision of 96.8% for intersected unidirectional word alignments, the highest recall of 81.5% for unified unidirectional word alignments, and precision of 83.5%, recall of 80%, F_α-score of 81.7 and AER of 18.25 for `grow-diag-final-and`-symmetrised bidirectional word alignment.

For each segment pair in our parallel corpus, we extracted three sets of word alignment links (i.e., Polish-to-English word alignment, English-to-Polish word alignment, and `grow-diag-final-and`-symmetrised bidirectional word alignment). These word alignments were trained on parallel corpus with the lemmatised Polish side and the tokenised English side.

5.4.3 Conversion of English Dependency Structures

The English side of the parallel corpus was parsed with the handcrafted wide-coverage English LFG grammar using the *Xerox Linguistic Environment* (XLE, Crouch et al., 2011) as a processing platform. Within the Parallel Grammar Project (ParGram, Butt et al., 2002), grammars for English, French, German, Norwegian, Japanese, Urdu, and further languages have been written in the framework of *Lexical Functional Grammar*. The architecture of *Lexical Functional Grammar*, with its strong lexicon component and multiple levels of representations, seems especially suited for cross-lingual annotation projection. Since LFG f-structures encode dependency relations and are largely invariant across languages, they may serve as the pivot for the cross-lingual projection.

[21] The Polish part of the parallel corpus was lemmatised with the *Pantera* tagger. The English side of the parallel corpus was lemmatised with the lemmatizer from the Stanford CoreNLP package `http://nlp.stanford.edu/software/corenlp.shtml`.

Lexical Functional Grammar is a theory of grammar, the development of which was initiated by Joan Bresnan and Ronald M. Kaplan in the late 1970s (Kaplan and Bresnan, 1995). The LFG formalism focuses on syntax and its relations with morphology, semantics and pragmatics. LFG treats each language as a multidimensional structure with a syntactic dimension, a semantic dimension and a pragmatic dimension among others. The syntactic dimension of a sentence contains three corresponding representations – a constituent structure (c-structure), a functional structure (f-structure) and an argument structure (a-structure), which are different but parallel levels of the syntactic structure. The semantic dimension of a sentence is represented as a semantic structure (s-structure) and the pragmatic dimension is represented as an information structure (i-structure).[22]

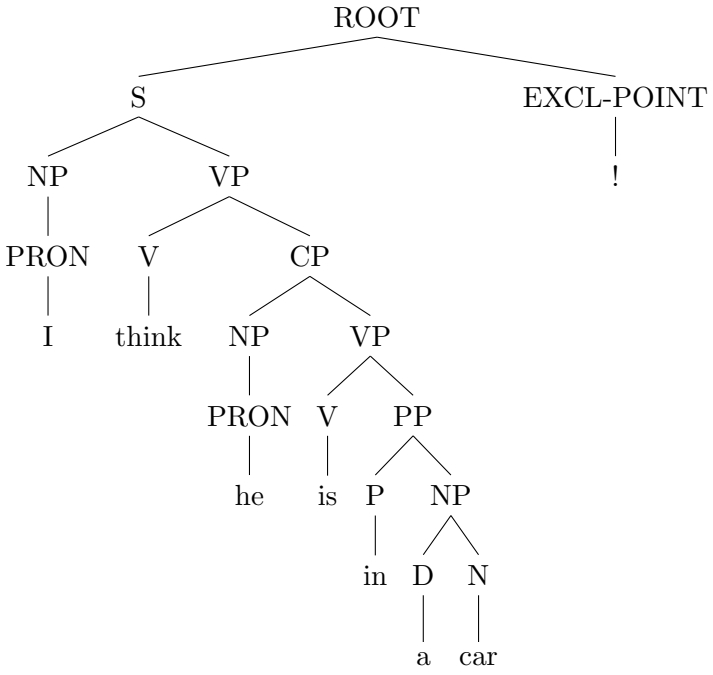


FIGURE 5.17: LFG c-structure of the sentence *I think he is in a car!*.

The concept of a constituent structure (c-structure, see Figure 5.17) is similar to the concept of a context-free phrase structure used, for example, in the transformational grammar. A c-structure encodes language-specific syntactic properties of the surface structure of a sentence: linear word order, syntactic constituency, dominance and precedence relations. The c-structure encodes both lexical categories (e.g., Noun, Verb, Adjective which head the following phrases: NP – noun phrase, VP – verb phrase, AP – adjective phrase) and functional categories (e.g., I – head of a finite clause IP, C – complementiser, CP – a clause with a complementiser C taking an IP as its complement). Lexical and functional categories are not universal across language. Therefore, grammatical rules based

[22] A-structures, s-structures and i-structures are not relevant for the current experiment.

on language-specific categories determine different c-structures as well-formed in various languages. Hence, languages vary because they acknowledge different c-structures as grammatical (*LFG Principle of Variability*, Bresnan, 2001, p. 44).

A c-structure exists simultaneously with a functional structure (f-structure, see Figure 5.18) that integrates information from the c-structure and from the lexicon. An f-structure consists of a finite set of attribute-value pairs that encode functional properties of a sentence and are organised in an attribute-value matrix. There are two main types of attributes in f-structures: grammatical functions and morphosyntactic features.

$$\begin{bmatrix}
\text{PRED} & \text{`think}\langle\boxed{1},\boxed{2}\rangle\text{'} & & \\
\text{SUBJ } \boxed{1} & \begin{bmatrix}\text{PRED} & \text{`I'}\\ \text{CASE} & \text{nom}\\ \text{NUM} & \text{sg}\\ \text{PERS} & 1\end{bmatrix} & & \\
\text{COMP } \boxed{2} & \begin{bmatrix}
\text{PRED} & \text{`be}\langle\boxed{3}\rangle,\boxed{4}\text{'} & \\
\text{SUBJ } \boxed{4} & \begin{bmatrix}\text{PRED} & \text{`he'}\\ \text{CASE} & \text{nom}\\ \text{NUM} & \text{sg}\\ \text{PERS} & 3\end{bmatrix} & \\
\text{XCOMP-PRED } \boxed{3} & \begin{bmatrix}
\text{PRED} & \text{`in}\langle\boxed{4},\boxed{5}\rangle\text{'} & \\
\text{SUBJ} & \boxed{4} & \\
\text{OBJ } \boxed{5} & \begin{bmatrix}
\text{PRED} & \text{`car'} & \\
\text{SPEC} & \begin{bmatrix}\text{DET} & \begin{bmatrix}\text{PRED} & \text{`a'}\end{bmatrix}\end{bmatrix} & \\
\text{CASE} & \text{obl} & \\
\text{NUM} & \text{sg} & \\
\text{PERS} & 3 &
\end{bmatrix} &
\end{bmatrix} &
\end{bmatrix} &
\end{bmatrix}$$

FIGURE 5.18: LFG f-structure of the sentence *I think he is in a car!*.

Grammatical functions are divided into argument functions (e.g., SUBJ, OBJ, OBJ_θ, OBL_θ, COMP or XCOMP), which are governed by a predicate, and non-argument functions (e.g., ADJUNCT, MOD, FOCUS or TOPIC), which are optional and ungovernable. The value of a grammatical function is another subordinate f-structure. Apart from grammatical functions, f-structures encode morphosyntactic features (e.g., NUM, GEN, PERS, CASE, TENSE). The value of a feature may be an f-structure, a symbol or a semantic form in the case of PRED attributes. A semantic form that consists of a predicate and its arguments constitutes an abbreviated representation of the entire f-structure. An f-structure encodes non-context-free syntactic phenomena such as agreement, subcategorisation, discontinuity, or argument selection. While c-structures vary across languages, f-structures are presumed to be largely universal and invariant across many languages (*LFG Principle of Universality,* Bresnan, 2001, p. 45).

The mapping between c- and f-structures is given by a function that connects one c-structure node or a set of nodes with a single f-structure. For example, correspondences between nodes of the c-structure from Figure 5.17 and sub-f-structures building the entire f-structure from Figure 5.18 are displayed in Figure 5.19 with dotted arrows (e.g., the node S corresponds to the top-level f-structure).

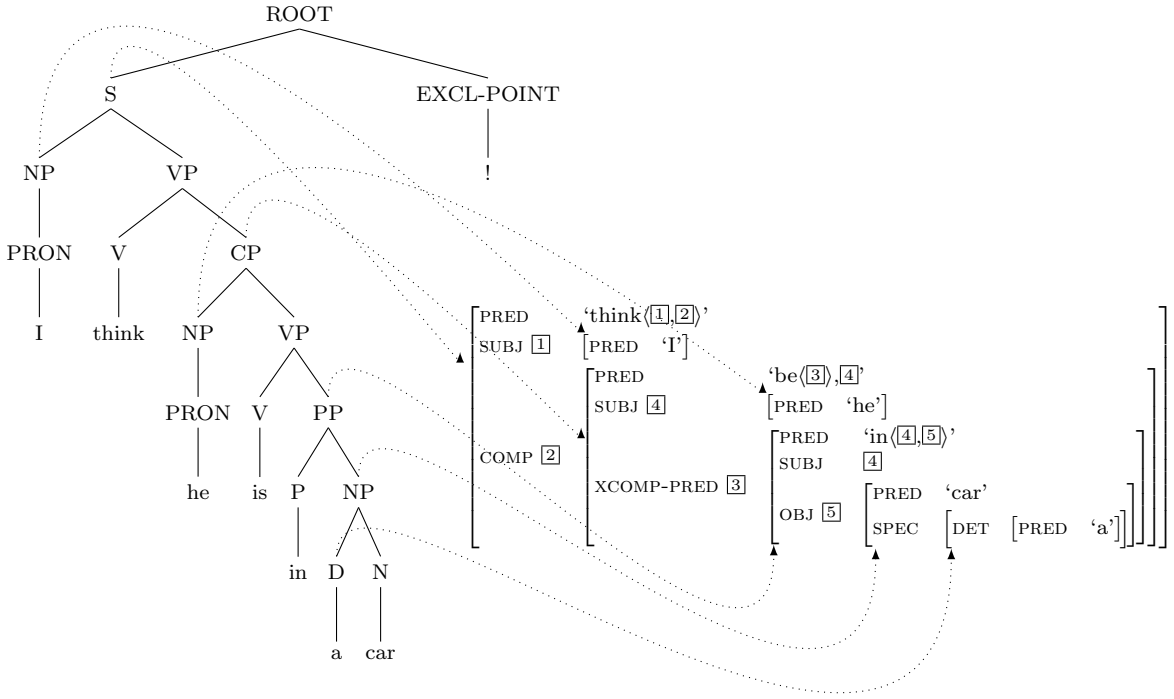


FIGURE 5.19: Correspondence between c-structure nodes and f-structures.

The LFG-driven XLE parser (Crouch et al., 2011) employed to parse the English side of the parallel corpus may output an analysis or a number of analyses of an English sentence. Each analysis consists of a c-structure, an f-structure and correspondences between these structures. Since the parser may output more than one analysis of a sentence, it is necessary to disambiguate these analyses. The XLE parser is enhanced with a statistical disambiguation component (Kaplan et al., 2004) that selects the most probable LFG analysis of an English sentence.

The English LFG grammar may deal with some ungrammatical sentences by applying the shallow parsing (or chunking) technique. The parser identifies well-formed chunks (constituents) in a sentence and then composes them linearly into a FIRST-REST structure marked as FRAGMENTS. Since the aim of the current experiment is to build a bank of Polish dependency structures for strings of tokens considered as grammatical, analyses marked as FRAGMENTS are not taken into account in projection. There are also some ungrammatical sentences or very long sentences which could not be analysed by the XLE parser at all. Sentence pairs in which the English sentence is annotated as

a FIRST-REST structure or is not assigned any analysis are filtered out from the parallel corpus and thus rejected from the further projection experiment. Remaining sentence pairs (4,706,688 sentence pairs) constitute our training corpus. A statistical overview of the parallel corpus after both filtering steps is reported in the column **2nd Filter** in Table 5.2 (information in the column **1st Filter** is repeated from Table 5.1).

Text	**Sentence Pairs**	**1st Filter**	**2nd Filter**		
		Sents pairs	**Sents pairs**	**En t/s**	**Pl t/s**
Europarl	448,433	429,929	247,389	20.71	18.05
DGT-TM	1,052,136	736,822	311,954	16.69	14.58
EMEA (OPUS)	868,808	193,274	105,232	15.01	14.89
EU Const (OPUS)	9,937	5,271	2,759	17.70	13.62
PHP (OPUS)	33,687	4,990	1,487	11.71	10.79
KDE4 (OPUS)	165,172	105,609	69,084	5.02	5.10
Academia	17,859	12,273	10,443	19.28	16.03
CORDIS	176,558	140,005	74,327	19.44	17.96
RAPID	146,908	106,713	57,317	19.98	18.15
EUR-Lex	4,063,336	2,558,306	649,593	18.26	16.18
Subtitles	6,871,465	4,120,543	2,623,280	7.75	7.37
Literature	911,278	854,422	553,823	14.52	12.48
TOTAL:	14,765,577	9,268,157	4,706,688		

TABLE 5.2: A statistical overview of the parallel Polish-English corpus after filtering multiple and unrecognised segment pairs (1st Filter, information transferred from Table 5.1) and after filtering segment pairs in which the English sentence is not fully parsed or has no correct analysis (2nd Filter). Explanation: En t/s – the average number of English tokens per sentence after 2nd filtering; Pl t/s – the average number of Polish tokens per sentence after 2nd filtering.

The decision of applying the XLE parser is motivated by the fact that it is one of the best performing parsers for English. It outputs f-structures which encode predicate-argument structures and are largely equivalent to dependency structures. Moreover, it also outputs c-structures which encode some additional features (e.g., word order, punctuation marks) which are also applied in the conversion of LFG analyses into dependency representations.

The most probable analyses (c- and f-structures) output by the XLE parser are converted into dependency structures and stored in the CoNLL data format (Buchholz and Marsi, 2006). As noticed by Øvrelid et al. (2009), the process of converting f-structures output by the XLE parser into dependency representations is quite straightforward since f-structures can actually be interpreted as dependency structures. The main idea behind the conversion is to extract a relation between values of two PREDs, to label the relation with an English grammatical function, and to transfer the labelled relation to the corresponding lexical nodes of a dependency structure.

The conversion procedure is performed as follows. For each token in a c-structure, its PRED attribute is found in the f-structure using correspondences between c- and f-structures encoded in the analysis. The value of the PRED attribute corresponds to a semantic form, i.e., a lemma (e.g., 'I', 'he', 'a' and 'car' in Figure 5.18) or a lemma with its arguments (e.g., 'think⟨1, 2⟩', 'be⟨3⟩,4' or 'in⟨4,5⟩' in Figure 5.18). The PRED attribute is incorporated into an f-structure. This f-structure may be, in turn, incorporated into an superordinate f-structure as the value of a grammatical function attribute. The grammatical function attribute corresponds to an argument or an adjunct required by the superordinate predicate. The predicate of the superordinate f-structure is the governor of the current token and the relation between them is labelled with the grammatical function. Hence, the labelled relation between the current token and the surface form of the governing PRED, which is encoded in the c-structure, is transferred to the equivalent lexical nodes of the converted English dependency structure. If the f-structure of the current token is the top-level f-structure, the relation between the current token and the additional ROOT node is added to the English dependency structure.

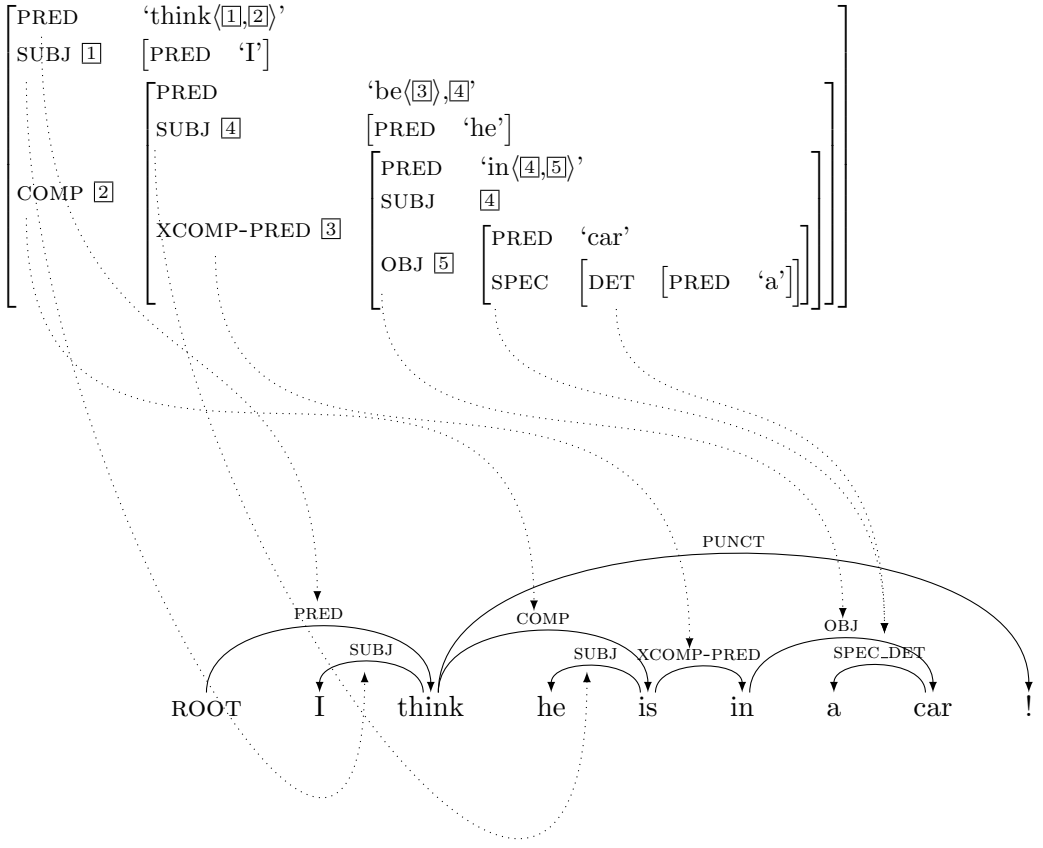


FIGURE 5.20: The procedure of converting an LFG f-structure into a dependency structure of a sentence *I think he is in a car!*. Projection of grammatical functions is marked with dotted arrows.

However, some tokens do not correspond to PREDs but rather to language-specific values of the following FORM features: COMP-FORM (the form of a complementiser, e.g., *that*), COORD-FORM (the form of a coordinating conjunction), PRECOORD-FORM (the form of

the first part of a correlative conjunction, e.g., *neither*), POSTCOORD-FORM (the form of a conjunction closing a list, e.g., *etc.*, *et al.*), PFORM (the form of a preposition that does not have a PRED, e.g., *by*), PRT-FORM (the form of a particle). We defined a set of rules to cover these cases. Generally, a complementiser is annotated as the dependent of the predicate of a closed complement clause. This relation is labelled with COMP-FORM. In the case of coordinating constructions annotated as sets of single f-structures for each coordinated element, the coordinating conjunction is annotated as the governor of coordinated elements and as the dependent of the PRED encoded in the superordinate f-structure. Prepositions encoded in f-structures as PFORMs do not have lexical meaning (e.g., *He looks* ***at*** *me.*, *It is made* ***by*** *hand.*). We annotate them as dependents of preceding verbs and label these relations with PFORM. A word form annotated as a particle PRT-FORM depends on the PRED of the current f-structure and the relation is labelled with PRT-FORM.

All transferred relations constitute an output dependency structure. The scheme of converting an LFG f-structure into a dependency structure is illustrated in Figure 5.20. For example, the top-level predicate *think* subcategorises two arguments – the subject and the closed complement clause. These two arguments are encoded as grammatical function attributes SUBJ and COMP with values realised as f-structure $\boxed{1}$ and f-structure $\boxed{2}$ respectively. The subject is represented by the pronoun *I*, so the relation $I \xleftarrow{\text{SUBJ}} think$ may be transferred to the dependency tree. Since a closed complement clause is headed by a verb in LFG, the COMP argument is represented by the verb form *is* encoded in the c-structure. The relation $think \xrightarrow{\text{COMP}} is$ is thus transferred to the equivalent dependency structure.

The conversion of all proper LFG analyses output by the XLE parser results in a bank of 4,706,688 English dependency structures. These dependency structures are projected to Polish.

5.5 Experiments and Results

Induction of Polish dependency trees is a two-stage process. First, the projection module outputs initially weighted multi-digraphs given bipartite alignment graphs, English dependency structures and Polish sentences. Second, the induction module acquires Polish dependency structures from projected multi-digraphs with recalculated arc weights. In this section, we provide some implementation details and results of an extrinsic evaluation.

Application of the projection procedure described in Section 5.1 *Weighted Projection* led to a large set of initially weighted multi-digraphs. Given unidirectional word alignments (Polish-to-English and English-to-Polish) and `grow-diag-final-and`-symmetrised bidirectional word alignment (see Section 5.4.2 *Experiments on Word Alignment*), English

dependency structures converted from LFG structures (see Section 5.4.3 *Conversion of English Dependency Structures*), and Polish sentences enriched with morphosyntactic information (lemmata, parts of speech, morphological features, named-entity tags),[23] the projection module outputs 4,706,688 initially weighted multi-digraphs.

For storing these multi-digraphs, a column-based format similar to the CoNLL format was designed. An example of a multi-digraph encoded in this format is given in Figure 5.21. Similarly as in the CoNLL format, there is a blank line between multi-digraphs. Information about a dependency relation which corresponds to an arc in a multi-digraph is stored in a single line, in 11 columns. The first column and the second column encode indices of the dependent node (*idD*) and the governor node (*idG*) respectively. The surface form (*token*), the lemma form (*lemma*), the part of speech tag (*pos*) and possible morphological features (*morph*) of the dependent are stored in the third, the fourth, the fifth and the sixth column respectively. The seventh column contains the (initial) weight of the arc (*weight*). Information encoded in the last four columns – the score of an edge linking dependents in a bipartite graph (*sD*), the score of an edge linking governors (*sG*), the projected grammatical function (*gf*) and the frequency of projecting the current labelled arc (*freq*) – builds a quadruple labelling the projected arc. The projected multi-digraphs encoded in the column-based format constitute an input to the induction module inferring final dependency trees.

The induction procedure described in Section 5.2 *Weighted Induction* starts with recalculation of initial weights of arcs in the projected multi-digraphs based on arc probabilities estimated with the EM-inspired selection algorithm. From the entire set of initially weighted multi-digraphs (4,706,688), 4,615,698 sets of k-best maximum spanning dependency trees were extracted, for $k = 10$. Sets of arcs in these k-best MSDTs constitute training data for estimation of the probability distribution over arcs. The probability distribution over arcs was estimated in 10 iterations of the EM selection algorithm. Then, initial weights of arcs in the projected multi-digraphs are recalculated based on the estimated probability distribution. The main idea behind the recalculation is to reward arcs with the probability greater than zero and by assigning them higher weights (see Equation (5.8), p. 110), and to penalise other arcs by assigning them lower weights (see Equation (5.9), p. 111). The procedure of recalculating initial arc weights is outlined in Section 5.2.3 *Recalculation of Arc Weights in Projected Multi-Digraphs*.

After recalculating weights of arcs in the projected multi-digraphs, the final maximum spanning dependency trees are selected. Since the final dependency structures are labelled with English grammatical functions, we treat them as unlabelled dependency structures. Arcs in these unlabelled dependency trees are then assigned Polish dependency labels derived from projected English grammatical functions and morphosyntactic

[23] Polish tokens are annotated with lemmata, part of speech tags and morphological features using the *Pantera* tagger. The named entities are recognised in Polish sentences using the statistical named entity recognition tool *Nerf* (Savary and Waszczuk, 2012).

idD	*idG*	*token*	*lemma*	*pos*	*morph*	*weight*	*sD*	*sG*	*gf*	*freq*
1	0	Chyba	chyba	qub	_	10	3	1	PRED	1
1	2	Chyba	chyba	qub	_	3	0	3	SPEC_POSS	1
1	2	Chyba	chyba	qub	_	2	2	0	SUBJ	1
1	3	Chyba	chyba	qub	_	2	2	0	SUBJ	1
1	4	Chyba	chyba	qub	_	2	2	0	SUBJ	1
1	5	Chyba	chyba	qub	_	2	2	0	SUBJ	1
2	0	jest	być	fin	sg.ter.imperf	1	0	1	PRED	1
2	1	jest	być	fin	sg.ter.imperf	3	0	3	SUBJ	1
2	1	jest	być	fin	sg.ter.imperf	3	0	3	INTERP	1
2	1	jest	być	fin	sg.ter.imperf	2	2	0	SPEC_POSS	1
2	1	jest	być	fin	sg.ter.imperf	24	3	3	OBJ	1
2	3	jest	być	fin	sg.ter.imperf	2	2	0	SPEC_POSS	1
2	3	jest	być	fin	sg.ter.imperf	3	3	0	OBJ	1
2	4	jest	być	fin	sg.ter.imperf	2	2	0	SPEC_POSS	1
2	4	jest	być	fin	sg.ter.imperf	3	3	0	OBJ	1
2	5	jest	być	fin	sg.ter.imperf	2	2	0	SPEC_POSS	1
2	5	jest	być	fin	sg.ter.imperf	3	3	0	OBJ	1
3	0	w	w	prep	loc.nwok	1	0	1	PRED	1
3	1	w	w	prep	loc.nwok	24	3	3	INTERP	1
3	1	w	w	prep	loc.nwok	3	0	3	OBJ	1
3	1	w	w	prep	loc.nwok	3	0	3	SUBJ	1
3	2	w	w	prep	loc.nwok	3	0	3	SPEC_POSS	1
3	4	w	w	prep	loc.nwok	3	3	0	INTERP	1
3	5	w	w	prep	loc.nwok	3	3	0	INTERP	1
4	0	samochodzie	samochód	subst	sg.loc.m3	1	0	1	PRED	1
4	1	samochodzie	samochód	subst	sg.loc.m3	3	0	3	OBJ	1
4	1	samochodzie	samochód	subst	sg.loc.m3	3	0	3	SUBJ	1
4	1	samochodzie	samochód	subst	sg.loc.m3	3	0	3	INTERP	1
4	2	samochodzie	samochód	subst	sg.loc.m3	3	0	3	SPEC_POSS	1
5	0	!	!	interp	_	1	0	1	PRED	1
5	1	!	!	interp	_	3	0	3	OBJ	1
5	1	!	!	interp	_	3	0	3	SUBJ	1
5	1	!	!	interp	_	3	0	3	INTERP	1
5	2	!	!	interp	_	3	0	3	SPEC_POSS	1

FIGURE 5.21: A weighted projected multi-digraph of the sentence *Chyba jest w samochodzie!* (Eng. 'I think he is in a car!) encoded in the column-based format. Explanation: *idD* – the index of the dependent node, *idG* – the index of the governor node, *token* – the surface form of the dependent, *lemma* – the lemma of the dependent, *pos* – the part of speech of the dependent, *morph* – morphological features of the dependent, *weight* – the score assigned to the dependency relation, *sD* – the score of the edge linking dependents in the bipartite alignment graph, *sG* – the score of the edge linking governors in the bipartite alignment graph, *gf* – the projected English grammatical function, *freq* – the frequency of projecting the current arc.

features of related Polish tokens (see Section 5.3 *Rule-based Adaptation of Polish Dependency Structures*). The entire induction procedure outputs a bank of 3,958,556 labelled dependency structures on which a Polish dependency parser may be trained.

5.5.1 Preliminary Experiment

Since there is no Polish-English parallel corpus annotated with gold-standard dependency structures on the Polish side, we cannot directly evaluate the induced dependency

structures. Instead, we perform an extrinsic evaluation, similarly as in Section 4.6 *Experiments and Results*. We train a dependency parser on the induced dependency trees and see to what extent these trees affect performance of the parser.

Performance of the dependency parser is evaluated against the set of 822 test trees taken from the bank of converted dependency structures. The same test set was used to evaluate Polish dependency parsers trained on the converted dependency structures. Parsing performance of these 'conversion-based' parsers[24] reported in Section 4.6 *Experiments and Results* constitutes the point of reference (an upper bound) for comparison with performance of 'induction-based' dependency parsers.

The conversion-based parsers were trained using two dependency parsing systems – *MaltParser* (Nivre et al., 2006a) and the *Mate* dependency parser (Bohnet, 2010). In order to compare results, we should use the same parsing systems. However, it turned out that it is technically impossible to train *MaltParser* on the entire bank of induced dependency structures using available machines because of limited memory.[25] Therefore, we only use the *Mate* system (Bohnet, 2010) in our evaluation experiments.

Even though the *Mate* parser is capable of learning a dependency parsing model on the entire induced treebank, the learning process requires a huge amount of time (e.g., one iteration of the *Mate* training on nearly 1 million trees takes almost 38 hours on a machine with up to 30G of memory and 8 cores). Therefore, we conducted some preliminary experiments on a smaller collection of about one million unlabelled induced trees. In these experiments, we tested different parameters of the *Mate* system in order to identify the best setting for learning dependency models for Polish. Results of the preliminary experiments indicate an approximate performance of Polish parsers and thereby the quality of the induced dependency structures.

In the preliminary experiment, we used 1,108,434 projected multi-digraphs originating from the following parts of the bitext collection: CORDIS (74,327 multi-digraphs), DGT-TM (311,954 multi-digraphs), EMEA (105,232 multi-digraphs), Europarl (274,389 multi-digraphs), and a part of EUR-Lex (342,532 multi-digraphs). Arcs of these projected multi-digraphs are assigned initial weights.

As mentioned in Section 5.2.2 *Feature Representations of Arcs*, different features can be used to represent arcs (e.g., surface forms, lemmata or part of speech tags of related tokens, and grammatical functions). Table 5.3 outlines the evaluation results of the *Mate* parser trained on the limited set of unlabelled dependency trees. These trees were acquired with the weighted induction method based on various feature representations of

[24]Since we compare parsing performance of dependency parsers trained on conversion-based dependency structures with performance of parsers trained on projection-based dependency structures, we refer to the dependency parsers trained on converted dependency structures as 'conversion-based parsers' and to dependency parsers trained on dependency structures acquired with the weighted induction method as 'induction-based parsers'.

[25]Machines that we have at our disposal have 32G memory at most. They are insufficient to train *MaltParser* on the entire induced dependency bank.

arcs. Results show that the best feature representation of arcs consists of lemmata of related tokens and grammatical functions.

Feature Representation	**Input**	**UAS**
lemma_{gov}, lemma_{dep}, gf	924,733	76.4
token_{gov}, token_{dep}, gf	939,722	75.2
pos_{gov}, pos_{dep}, gf	1,005,114	75.6
lemma_{gov}, lemma_{dep}, pos_{gov}, pos_{dep}, gf	934,143	75.2
lemma_{gov}, lemma_{dep}, token_{gov}, token_{dep}, pos_{gov}, pos_{dep}, gf	939,855	75.5

TABLE 5.3: Performance of the *Mate* parser trained on Polish unlabelled dependency structures obtained with the weighted induction method based on various feature representations of arcs. The induction-based parsers are trained in one iteration with the heap size of 50 million and the threshold of the non-projective approximation of 0.2. The number of training trees is given in the second column (Input).

From the limited set of projected multi-digraphs, 924,733 unlabelled dependency trees (17.1 tokens per sentence on average) are acquired with the weighted induction method. In the procedure of recalculating arc weights, the parameter k (i.e., the number of the best MSDTs used to estimate the probability distribution over arcs) was set to 10 and the feature representation of arcs was built of lemmata of related tokens and the grammatical function labelling the arc between these tokens. The total number of induced trees differs from the number of initially weighted projected multi-digraphs. This is due to our decision of training dependency parsers solely on properly built dependency trees, e.g., trees with only one dependent of the ROOT node. Improper structures are not taken into account in the dependency parser training. The *Mate* parser model is trained on this set of 924,733 induced dependency trees.

The point of reference for the induction-based parser is a baseline *Mate* parser trained on 1,000,797 unlabelled maximum spanning dependency trees (16.7 tokens per sentence on average). The baseline trees were extracted from the limited set of 1,108,434 projected multi-digraphs with initial weights on arcs using the k-best MSDT selection algorithm (see Appendix D), for $k = 1$. Hence, the baseline parser was trained on MSDTs extracted from the initially weighted multi-digraphs, while the induction-based parser was trained on MSDTs extracted with the weighted induction method.

Both parsers used the same training setting: the heap size of 50 million features and the non-projective approximation threshold of 0.3. The models were learnt in 10 iterations. The results show that the induction-based parser outperforms the baseline parser, but the difference is insignificant. The induction-based parser obtains 71.7% UAS, while the baseline parser obtains 71.1% UAS when tested against the conversion-based test trees.

We also tested how the *Mate* parser performs when it is trained in a number of iterations smaller than 10. We found out that parsing performance decreases with the increasing

number of iterations used to train the parser. The parser trained in one iteration performs better than the parser trained in a number of iterations. The induction-based parser trained in one iteration achieves 76.4% UAS, while the baseline parser obtains 74% UAS. Hence, if the parsers are trained in one iteration, the induction-based parser significantly outperforms the baseline parser. The decrease in parsing performance may be due to noise which is learnt in successive iterations. Therefore, we limit the number of *Mate* iterations to one in further experiments.

Similarly as in Section 4.6.2 *Experiment 2* – Mate *Parser*, we conducted an experiment with different approximation thresholds. The *Mate* parser applies the Non-Projective Approximation Algorithm by McDonald and Pereira (2006) to rearrange arcs in a predicted dependency tree if it results in an enhancement of the tree score. The rearrangement threshold introduced in Bohnet (2009) determines the minimal enhancement of the tree score. The threshold of 0.3 was proven to be the best option for training the *Mate* parser on the conversion-based Polish dependency structures. However, it is worth recalling that these dependency structures were converted from the constituent trees which do not encode linguistic phenomena resulting in crossing edges. Single non-projective arcs result from the post-conversion rearrangement or the manual correction, but generally they are sparsely represented in the conversion-based dependency trees. The entire converted treebank contains only 125 crossing edges (i.e., about 0.15% of all arcs in the treebank trees).

We tested different non-projective approximation thresholds. The test results are displayed in Figure 5.22. The diagram can be interpreted as follows. The x-axis indicates threshold values. The blue y-axis on the left indicates the percentage of non-projective arcs in the test dependency structures parsed with the baseline parser (marked with the blue line with $\times$ points) and the induction-based parser (marked with the blue line with $\bullet$ points). The orange y-axis determines parsing performance in terms of the unlabelled attachment score (UAS). Parsing performance of the baseline parser trained with different approximation thresholds is marked with the orange line with $\times$ points, while performance of the induction-based parser is given with the orange line with $\bullet$ points.

Figure 5.22 shows that the *Mate* parser trained on trees acquired with the weighted induction method outperforms the baseline parser. The best induction-based parser obtains UAS of 76.4%, while the best baseline parser obtains UAS of 75.2%. The threshold of 0.1 is the best choice for training the baseline parser and the threshold of 0.2 is most suitable for training the induction-based parser. For higher threshold values, parser performance decreases rapidly.

With an increase of the threshold, the number of non-projective arcs increases radically. However, there are fewer non-projective arcs in the test dependency trees parsed with induction-based parsers than in the test trees predicted by baseline parsers. This could suggest that the weighted induction method reduced the number of incorrect

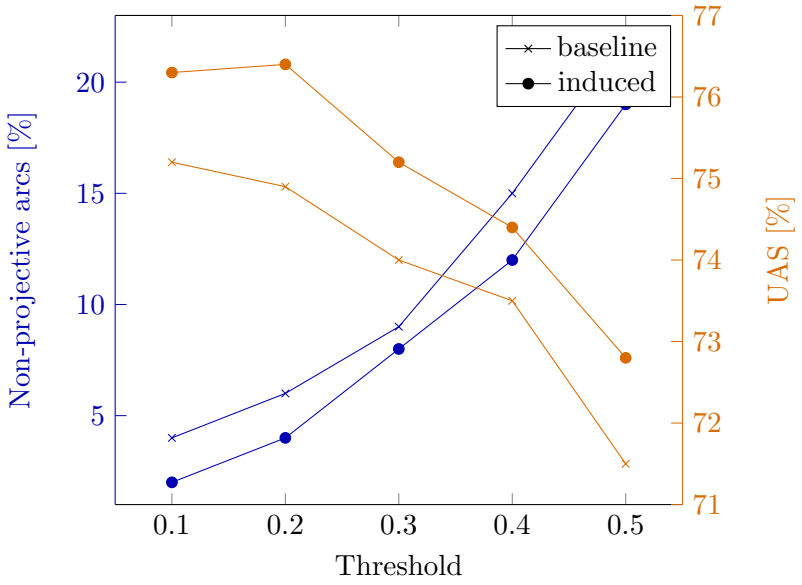


FIGURE 5.22: The impact of different non-projective approximation thresholds (x-axis) on the number of non-projective arcs in parsed test trees (blue y-axis on the left) and on parsing performance in terms of UAS (orange y-axis on the right). *Baseline* parsers are trained on the dependency structures extracted from the initially weighted multi-digraphs. *Induction-based* parsers are trained on the dependency structures acquired with the weighted induction method. Numbers of non-projective arcs in baseline parse trees and induced parse trees are marked with × points and • points respectively. Similarly, UAS scores obtained with baseline parsers and induction-based parsers are marked with × points and • points respectively.

non-projective arcs in training dependency trees and, thus, a parser trained on them performs better. In order to prove this, we tested whether parsers rearrange arcs in dependency trees into correct non-projective arcs. We found out that the baseline parser with the threshold of 0.2 rearranged correctly only 3 of all 514 rearranged arcs (the best baseline parser with the threshold of 0.1 rearranged correctly 2 of all 332 rearranged arcs). The best induction-based parser rearranged correctly only 6 of all 365 rearranged arcs. Since only few arcs were rearranged properly, it may suggest that training data induced with the weighted method are too noisy. Nevertheless, there is a noticeable improvement in relation to the number of incorrectly rearranged baseline arcs.

The model of the *Mate* parser was trained with the passive-aggressive perceptron algorithm implemented as the hash kernel. The hash kernel uses a scoring function $F(x, y) = \vec{w} * \overline{\phi}(x, y)$, where x is a sentence, y is the corresponding dependency structure, $\vec{w}$ is the weight vector and $\overline{\phi}(x, y)$ is a numeric feature representation. The parsing problem is to find a parse tree y_p of the sentence x that maximises the scoring function $\text{argmax}_y F(x, y)$. The learning process consists in fitting the function F so that the predicted parse tree y contains as minimal number of errors as possible. The learning algorithm implemented as the hash kernel extracts features for each training instance (without storing them), maps the features to indices of the weight vector using the hash

Vector Size	Nonzero Values	UAS
50,000,000	49,150,506	76.4
100,000,000	86,941,894	75.9
200,000,000	127,658,581	76.3
500,000,000	166,895,804	75.9

TABLE 5.4: Performance of parsers trained with the *Mate* parsing system on Polish unlabelled dependency structures acquired with the weighted induction procedure. The induction-based parsers are trained in one iteration on 924,734 induced trees (17.1 tokens per sentence on average). Explanation: *Vector Size* – the size of the weight vector; *Nonzero Values* – the number of nonzero values in the feature vectors.

function and calculates the weight arrays. If the hash function maps more than one feature to the same weight, a collision occurs. A large weight vector reduces the risk of collisions. We found out that the accuracy of the *Mate* parser does not change significantly for different sizes of the weight vector (see Table 5.4). However, for vector sizes greater than 100 million values, there are multiple zero weights and hence many collisions.

5.5.2 Experiments on the Entire Set of Induced Trees

In this section we report evaluation results of parsers trained on the entire set of dependency trees acquired with the weighted induction method. The evaluation results are presented in Table 5.5. The first parser (called *induced*) was trained on 3,958,556 induced dependency trees. These trees are treated as unlabelled since their arcs are still assigned English grammatical functions not corresponding to Polish labels. The second parser (denoted *labelled*) was trained on the same set of induced trees with Polish labels assigned to dependency arcs. The third parser (*modified*) was trained on the same set of induced trees which were labelled and modified with the predefined rules. We also outline some tests on filtering unreliable dependency trees. The *filtered* parsers are trained on the refined set of labelled, modified and filtered trees acquired with the weighted induction method.

We evaluate parsing performance against the test set of 822 dependency structures taken from the converted Polish dependency bank (Manual Test). We also provide a version of test trees with automatically generated part of speech tags and morphological features (Automatic Test). Furthermore, we employ the additional test set of 100 manually annotated complex sentences (Additional Test).

The *induced* dependency parser trained on unlabelled trees obtained with the weighted induction method achieves 73.7% UAS when evaluated against the *Manual Test* trees and 72.8% UAS when evaluated against the *Automatic Test* trees. Hence, the *Mate*

Model	Training Data	Filtering (in %)		Manual Test		Automatic Test		Additional Test	
		non-proj	*dep*	UAS	LAS	UAS	LAS	UAS	LAS
induced	3958556	–	–	73.7	–	72.8	–	63.5	–
labelled	3958556	–	–	74.6	69.4	74.0	68.1	63.7	58.3
modified	3958556	–	–	85.1	79.2	84.0	77.3	74.3	68.5
filtered	3548347	50	30	85.0	79.1	83.9	77.2	74.5	69.2
filtered	3036020	30	30	85.2	79.3	83.6	77.0	74.4	68.5
filtered	2352940	30	10	86.0	80.5	84.7	78.3	76.1	**70.3**
default	7405	–	–	**92.7**	**87.2**	88.4	81.0	76.0	69.5
automatic	7405	–	–	91.2	85.6	**90.8**	**84.7**	**76.6**	70.1

TABLE 5.5: Performance of parsers trained with the *Mate* parsing system on the Polish dependency structures acquired with the weighted induction method (*induced*), induced and labelled (*labelled*), labelled and modified (*modified*), and labelled, modified and filtered (*filtered*). Settings of model training: one iteration, the heap size of 100 million features, the threshold of the non-projective approximation of 0.2. The *default* and *automatic* models are trained on 7405 conversion-based dependency structures with manually and automatically annotated tokens respectively (the results of these parsers are repeated from Table 4.1 – *Mate* default and *Mate* automatic). Validation data sets: *Manual Test* – the set of 822 conversion-based test trees; *Automatic Test* – the set of 822 conversion-based test trees with automatically assigned morphosyntactic annotations; *Additional Test* – the set of 100 sentences manually annotated with dependency trees.

parser manages to parse test data with manually or automatically annotated tokens almost equally well since there is a difference of about 1 percentage point.

The *Mate* parser used in our experiments is designed to predict labelled dependency structures. The induced dependency trees employed as training data in the previous experiment were labelled with possibly noisy English grammatical functions. We treated them therefore as unlabelled dependency trees and we only pointed out the UAS scores. Now, we report on performance of the *Mate* parser trained on automatically induced trees the arcs of which are labelled using the rule-based method presented in Section 5.3 *Rule-based Adaptation of Polish Dependency Structures*. There is no significant change in terms of UAS if performance of parsers trained on labelled induced trees (*labelled*) is considered (see the second row in Table 5.5). In the case of the *labelled* model, the difference between performance of the parser evaluated against the Manual Test trees (74.6% UAS) and against the Automatic Test trees (74% UAS) is even slighter than for the *induced* parser.

Table 5.5 also outlines the evaluation results of the parser trained on labelled induced dependency structures which were modified with some correction rules (*modified*). The parser obtains 85.1% UAS and 79.2% LAS when evaluated against the Manual Test trees and 84% UAS and 77.3% LAS when evaluated against the Automatic Test trees. Similarly as in the previous case, the *modified* parser analyses automatically annotated

sentences nearly as well as manually annotated sentences. The parser trained on the modified dependency structures considerably outperforms the parsers trained on the labelled induced dependency structures. This indicates that some manually designed correction rules significantly improve the quality of the labelled induced dependency structures. However, the parser trained on the modified trees is still behind the conversion-based parser when evaluated against conversion-based test trees (Manual Test and Automatic Test).

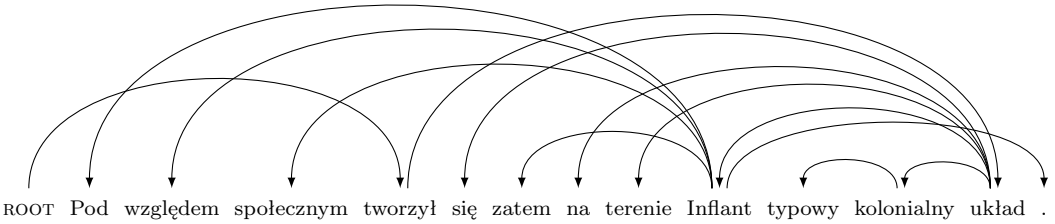


FIGURE 5.23: An incorrect dependency structure of the Polish sentence *Pod względem społecznym tworzył się zatem na terenie Inflant typowy kolonialny układ.* (Eng. 'Socially, a typical colonial system was thus created in the area of Livonia.'; the English corresponding sentence from the parallel corpus: 'Socially, this created a typically colonial system.') with long distances between related nodes.

We observe that incorrect dependency trees are mainly characterised by long distance between related lexical nodes and thus a high number of non-projective arcs (see Figure 5.23 for an example of an incorrect dependency structure acquired with the weighted induction method). Filtering is one of the most common optimisation techniques in projection-based approaches. After automatic induction of trees, some of them were filtered out. Two filtering criteria were used: percentage of non-projective arcs and percentage of arcs labelled with the default function *dep*. The *filtered* parsers were trained on reduced sets of trees. Our results show that filtering of possibly incorrect trees does not significantly improve parsing performance. The best results are achieved when the parser was trained on trees with fewer than 30% of non-projective arcs and with fewer than 10% of *dep*-labelled arcs – 86% UAS and 80.5% LAS when tested against the Manual Test trees and 84.7% UAS and 78.3% LAS when tested against Automatic Test trees.

The reported results are obtained by the *Mate* parser trained on induced trees and evaluated against conversion-based trees. In this parsing scenario, the induction-based parsers (i.e., *induced*, *labelled*, *modified* and *filtered*) are outperformed by the conversion-based parsers. In a more realistic scenario, parsing models are tested against the set of 100 complex trees (*Additional Test* in Table 5.5). Parsing results are generally worse than those reported above. However, the induction-based parsers approach the upper bound (i.e., performance of the conversion-based parser) or even outperform the conversion-based parsers. The *Mate* parser with the *filtered* model achieves 70.3% LAS and thus slightly outperforms the *Mate* default parser that achieves 70.1% LAS.

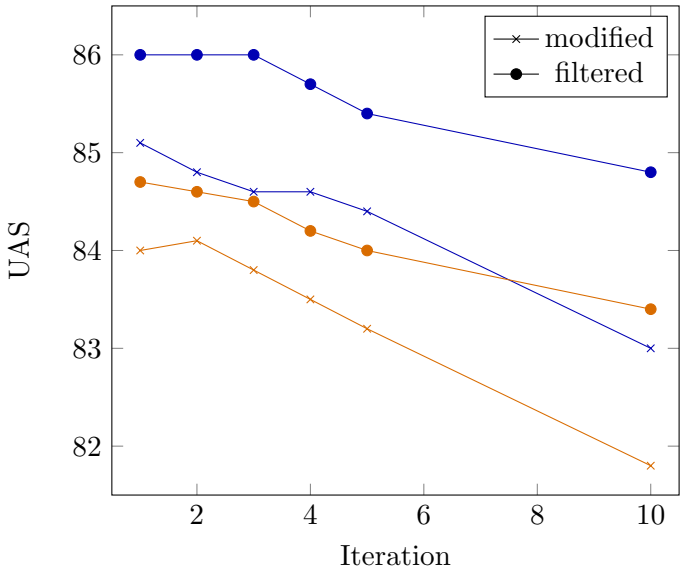


FIGURE 5.24: The impact of the number of training iterations (x-axis) of the *Mate* parser on parsing performance (y-axis) in terms of UAS. Explanation: *modified* parsers are trained on the automatically induced, labelled and corrected dependency structures, *filtered* parsers are trained on the reliable dependency structures obtained with the weighted induction method, labelled, corrected and filtered; blue lines indicate the results of evaluation against the Manual Test trees and orange lines indicate the results of evaluation against the Automatic Test trees.

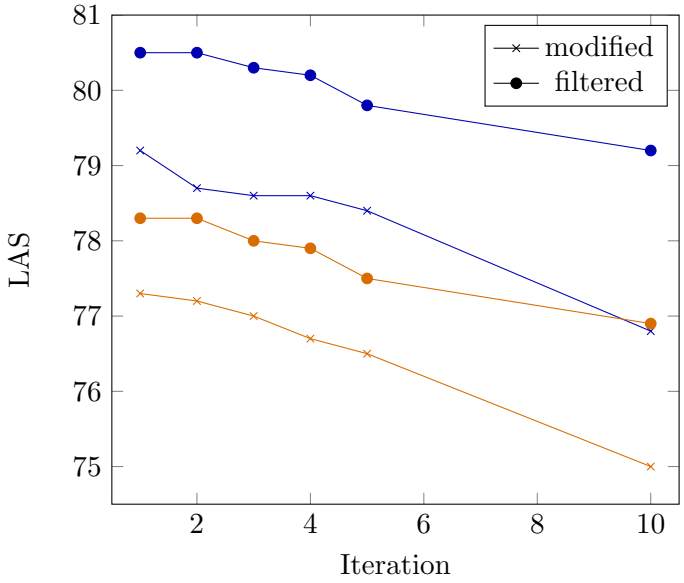


FIGURE 5.25: The impact of the number of training iterations (x-axis) of the *Mate* parser on parsing performance (y-axis) in terms of LAS. Explanation: *modified* parsers are trained on the automatically induced, labelled and corrected dependency structures, *filtered* parsers are trained on the reliable dependency structures acquired with the weighted induction method, labelled, corrected and filtered; blue lines indicate the results of evaluation against the Manual Test trees and orange lines indicate the results of evaluation against the Automatic Test trees.

Similarly as in the case of preliminary parsers trained on the limited set of unlabelled induced dependency trees, the number of *Mate* iterations has a negative impact on parsing performance (see Figures 5.24 and 5.25). Performance of the *Mate* parser trained on the entire set of modified induced dependency structures decreases from 85.1% UAS and 79.2% LAS when the parsing model is trained in one iteration to 83% UAS and 76.8% LAS when the parsing model is trained in ten iterations. A relatively small decrease in parsing performance is also noticed when the parser is trained on filtered dependency trees. Performance of the *filtered* parser decreases from 86% UAS and 80.5% LAS when the parsing model is trained in one iteration to 84.8% UAS and 79.2% LAS when the parsing model is trained in ten iterations.

5.5.3 Evaluation of Individual Relation Labels

In this section we evaluate individual relation labels which are assigned to arcs of trees predicted by the *labelled* parser trained on labelled induced trees and by the *modified* parser trained on labelled induced trees which are modified with correction rules. Table 5.6 presents the results of evaluation of individual dependency labels against the Manual Test trees and the Automatic Test trees.

The results indicate that induction-based parsers perform better on manually annotated sentences than on sentences with automatic morphosyntactic annotations. However, the difference is insignificant. The *labelled* parser labels dependency trees predicted for manually annotated sentences with the average F-score of 0.71 and trees predicted for automatically annotated sentences with the average F-score of 0.69. The *modified* parser achieves the average labelling accuracy of 0.79 and 0.78 in terms of F-score for predicted trees with manually and automatically annotated tokens respectively.

Most of dependency types profit from the application of correction rules. There are three dependency relations – *mwe*, *abbrev_punct* and *cond* – with the F-score close to 0 in the test parse trees predicted by the *labelled* parser. The average F-score of these labels increases to 0.67 in the test trees predicted by the *modified* parser. An improvement of nearly 10 percentage points is also achieved in the case of *comp*, *subj*, *refl*, *item*, *aglt* and *aux* labels. There are also some other dependency types which profit from the application of correction rules – *adjunct*, *punct*, *pred*, *conjunct*, *obj*, *obj_th*, *neg*, *comp_fin* and *pd*. For other dependency types, an improvement is marginal (e.g., *comp_inf*) or there is no slightest improvement (e.g., *app*, *comp_ag*, *coord_punct* and *pre_coord*). Three of dependency labels with no improvement – *app*, *coord_punct* and *pre_coord* – were only assigned to wrong dependency relations. Finally, there are two dependency types – *coord* and *ne* – F-scores of which decrease after application of additional correction rules. To explain why there is no enhancement in these cases, a manual error analysis was performed.

Dependency Relation		Manual Test						Automatic Test					
		Labelled			Modified			Labelled			Modified		
Label	**Frequency**	precision	recall	f-score	precision	recall	f-score	precision	recall	f-score	precision	recall	f-score
adjunct	2185	0.67	0.72	0.70	0.74	0.79	0.76	0.64	0.70	0.67	0.71	0.77	0.74
comp	1311	0.83	0.68	0.75	0.89	0.87	0.88	0.81	0.66	0.73	0.89	0.86	0.88
punct	1159	0.64	0.68	0.66	0.70	0.73	0.71	0.65	0.69	0.67	0.69	0.73	0.71
pred	770	0.85	0.87	0.86	0.92	0.94	0.93	0.85	0.87	0.86	0.92	0.94	0.93
subj	608	0.78	0.74	0.76	0.86	0.84	0.85	0.76	0.71	0.73	0.84	0.77	0.81
conjunct	455	0.78	0.64	0.71	0.86	0.65	0.74	0.75	0.63	0.69	0.83	0.64	0.72
obj	419	0.84	0.73	0.78	0.87	0.79	0.83	0.81	0.67	0.73	0.81	0.72	0.76
obj_th	197	0.71	0.56	0.63	0.74	0.67	0.70	0.71	0.51	0.59	0.73	0.64	0.68
refl	166	0.84	0.58	0.69	0.90	0.93	0.91	0.83	0.58	0.68	0.89	0.92	0.90
ne	120	0.67	0.44	0.53	0.72	0.40	0.51	0.59	0.56	0.57	0.61	0.49	0.55
neg	119	0.94	0.87	0.91	0.96	0.96	0.96	0.95	0.88	0.92	0.96	0.96	0.96
comp_inf	112	0.93	0.84	0.88	0.97	0.83	0.89	0.92	0.83	0.87	0.97	0.82	0.89
comp_fin	101	0.76	0.61	0.68	0.79	0.64	0.71	0.76	0.61	0.68	0.81	0.66	0.73
pd	101	0.73	0.58	0.65	0.72	0.65	0.69	0.75	0.57	0.65	0.73	0.60	0.66
mwe	86	0.23	0.05	0.08	0.74	0.62	0.67	0.23	0.05	0.08	0.73	0.60	0.66
item	65	0.60	0.60	0.60	0.77	0.77	0.77	0.60	0.60	0.60	0.75	0.75	0.75
complm	60	0.77	0.77	0.77	0.74	0.85	0.79	0.75	0.77	0.76	0.74	0.85	0.79
aglt	59	0.97	0.64	0.77	0.97	0.97	0.97	0.97	0.66	0.79	0.95	0.93	0.94
aux	51	0.84	0.63	0.72	0.84	0.82	0.83	0.84	0.63	0.72	0.83	0.84	0.83
app	43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
abbrev_punct	29	0.00	0.00	0.00	0.79	0.79	0.79	0.00	0.00	0.00	0.67	0.41	0.51
coord	26	0.62	0.81	0.70	0.61	0.73	0.66	0.64	0.81	0.71	0.58	0.73	0.64
coord_punct	26	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
cond	11	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	1.00	1.00	1.00
comp_ag	9	1.00	0.67	0.80	1.00	0.67	0.80	1.00	0.67	0.80	1.00	0.67	0.80
pre_coord	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
average:		0.73	0.69	0.71	0.80	0.79	0.79	0.72	0.68	0.69	0.78	0.77	0.78

TABLE 5.6: Parsing performance in terms of accuracy of individual dependency types assigned by the *Mate* parser trained on labelled dependency structures (*labelled*), and on labelled and modified dependency structures (*modified*). Validation test sets: *Manual Test* – the set of 822 conversion-based dependency structures with manual morphosyntactic annotations of tokens; *Automatic Test* – the set of 822 conversion-based dependency structures with automatic morphosyntactic annotations of tokens; average – weighted arithmetic mean.

We start with an analysis of dependency types whose precision, recall and F-scores are equal to 0, i.e., *app*, *coord_punct* and *pre_coord*. Apposition relations are not explicitly encoded in English dependency structures. Since most of them are labelled with the English grammatical function ADJUNCT, its equivalent Polish function *adjunct* often labels apposition relations in induced trees instead of the desirable function *app*. Furthermore, there are multiple examples where the apposition relation is annotated reversely as shown in Figure 5.26. Hence, additional correction rules are required to cover reversely annotated apposition relations.

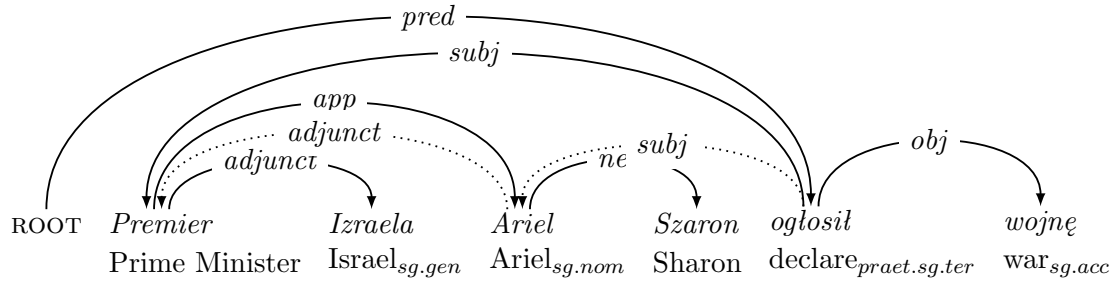


FIGURE 5.26: Discrepancies in annotating apposition relations in the gold standard tree (marked with solid lines) and the induced dependency tree (marked with dotted lines) of the Polish sentence *Premier Izraela Ariel Szaron ogłosił wojnę*. (Eng. 'Israel Prime Minister, Ariel Sharon declared the war.'). In the gold standard tree, the token *Premier* governs the token *Ariel* and the relation is labelled with the *app* function. In the induced dependency tree, in turn, the relation is reversed and the token *Ariel* governs the token *Premier*.

Several *coord_punct* relations connect the ROOT node and a punctuation mark (comma) coordinating two sentences in the test dependency structures. In all these cases, the induction-based parsers annotate the sentence predicate of one of coordinated clauses as the governor of the entire sentence. Moreover, these parsers assign the *coord_punct* function to only one relation in the entire set of test trees and this assignment is wrong. We suppose that constructions with two sentences coordinated with a punctuation mark may be too sparsely represented or incorrectly annotated in training data induced with the weighted method. Therefore, the parsers cannot learn to annotate them properly. Since there is only one *pre_coord* relation in the test corpus, the evaluation of this dependency type is not reliable.

There are two dependency types – *coord* and *ne* – for which F-scores slightly decrease after application of correction rules. In the case of these functions, precision increases (or slightly decreases), but recall significantly decreases. It means that labels *coord* and *ne* assigned by the parser are relevant to some extent, but the number of relevant labels *coord* and *ne* assigned by the parser is rather low. For example, the *labelled* parser assigns the *ne* function to 79 relations, but only 53 of these assignments are correct (hence, precision of 0.67%). The *modified* parser, in turn, assigns the *ne* function to 67 relations and 48 of these assignments are correct (hence, precision of 0.72%). In

the conversion-based test trees, there are 120 relations labelled with the *ne* function. Since the number of correct *ne* labels assigned by the *modified* parser is lower then the number of *ne* labels assigned by the *labelled* parser, i.e., 48 vs. 53, recall is lower as well. The *ne* function is mostly replaced by the *adjunct* function in the test parse trees.

We also consider the default function *dep* which labels dependency relations that could not be labelled or modified with rules. The *dep* function is assigned by the *labelled* parser to 510 relations (more than 6% of all 8289 relations) in the test trees predicted for manually annotated sentences and to 496 relations (almost 6% of all relations) in the test trees predicted for automatically annotated sentences. The number of dependency relations labelled with the *dep* function significantly decreases when sentences are parsed with the *modified* parser, i.e., the *dep* function is assigned to 141 relations (1.7% of all relations) in the test trees predicted for manually annotated sentences and to 175 relations (2.1% of all relations) in the test trees predicted for automatically annotated sentences. This indicates that the correction rules reduce the number of unrecognised dependency relations.

5.6 Annotation Projection: Related Work

The cross-lingual projection method has been successfully applied to various levels of linguistic analysis and corresponding NLP tasks. It was first applied by Yarowsky and his colleagues (Yarowsky et al., 2001; Yarowsky and Ngai, 2001) to induce part of speech taggers, noun phrase chunkers, named entity recognisers and morphological analysers for French, Chinese, Czech, Spanish and possibly for other less researched languages. Even if projected annotations are noisy and incomplete, the employed training methods overcome the weakness of noisy projections and allow for bootstrapping robust NLP tools. For example, the French part of speech tagger trained on noisy data achieves the tag accuracy of 98%.

Several studies that followed the work by Yarowsky et al. (2001) improve the method of projecting part of speech tags and widen the scope of relating languages, e.g., Cucerzan and Yarowsky (2002), Ozdowska (2006), Das and Petrov (2011). The annotation projection method has also been applied to other NLP tasks, such as word sense disambiguation (Diab and Resnik, 2002), verb classification (Merlo et al., 2002), argument identification (Bouma et al., 2008), acquisition of idiomatic expressions (Villeda Moirón and Tiedemann, 2006), acquisition of synonyms (van der Plas and Tiedemann, 2006), temporal analysis (Spreyer and Frank, 2008), semantic role labelling (Padó and Lapata, 2005, 2009), or relation extraction (Kim et al., 2010).

An important area of applying annotation projection is dependency tree projection and parser induction. Experiments with dependency projection were pioneered by Hwa et al.

(2002, 2005). Based on the *Direct Correspondence Assumption* (DCA),[26] Hwa and her colleagues assume that dependencies in one language directly map to dependencies in another language. They project dependency trees from English to Spanish and Chinese and train dependency parsers on projected trees. However, the *Direct Correspondence Assumption*, which underlies the annotation projection approach by Hwa et al. (2005), is an idealisation. First, since a translation may be non-literal, lexical items in a source sentence and its non-literal translation do not correspond to each other (see Figure 5.27).

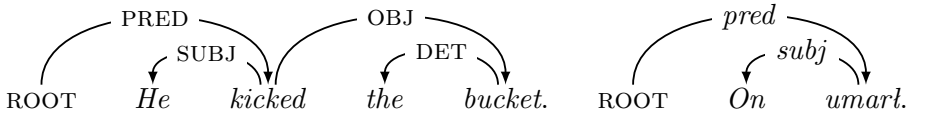


FIGURE 5.27: Translational divergences between the English sentence *He kicked the bucket.* and its Polish translation *On umarł.* (Eng. 'He died.').

Second, lexical items in a source sentence and its literal translation may not correspond to each other because of cross-lingual variability, e.g., in using function words (see Figure 5.28).

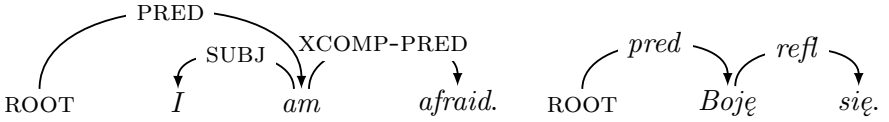


FIGURE 5.28: Cross-lingual variability between the English sentence *I am afraid.* and its Polish translation *Boję się.*

Third, even if lexical items of a source sentence and its target translation perfectly correspond with each other, the projection process may be disturbed by erroneous automatic word alignment and errors in automatically annotated source dependency structures. The disturbed projection results in dependency structures which do not meet properties of a correct dependency tree. For example, a projected dependency structure may be disturbed in terms of the *spanning property* or the *connectedness property* by imperfect or incomplete automatic word alignment.

Translational divergences and true mismatches of dependency structures between languages may radically impair the quality of induced dependency structures. In order to improve the quality of projection-based dependency structures, Hwa et al. (2005) apply some hand-crafted correction rules that locally transform projected structures. The correction rules increase accuracy of unlabelled structures projected via automatic word alignment from 33.9 to 65.7 of unlabelled dependency F-score[27] for Spanish and from

[26] The definition of *Direct Correspondence Assumption* (DCA) given by Hwa et al. (2002, 2005):

Direct Correspondence Assumption: Given a pair of sentences E and F that are (literal) translations of each other with syntactic structures $Tree_E$ and $Tree_F$, if nodes x_E and y_E of $Tree_E$ are aligned with nodes x_F and y_F of $Tree_F$ respectively, and if syntactic relationship $R(x_E, y_E)$ holds in $Tree_E$, then $R(x_F, y_F)$ holds in $Tree_F$.

[27] The unlabelled dependency F-score is not defined in the article by Hwa et al. (2005).

26.3 to 52.4 of unlabelled dependency F-score for Chinese. Moreover, Hwa et al. (2005) identify unreliable target structures based on the number of unaligned and over-aligned words and they filter them out. The filtering procedure reduces the number of projected trees but increases performance of a parser trained on filtered data. In English-Spanish scenario, if the number of sentences is reduced from 98,000 to 20,000 sentences, parsing performance increases from 67.3 to 72.1 of unlabelled F-score. In English-Chinese scenario, a radical loss of data, from 240,000 to 50,000 sentences, causes an increase in parsing performance from 44.3 to 53.9 of unlabelled F-score.

To cope with the limitations of DCA, various methods of dependency projection have been proposed. Some of them apply aggressive filtering (Mareček, 2011; Jiang and Liu, 2009). Other use projection-based smoothing techniques in learning target parsing models (Ganchev et al., 2009). Training with smoothing (i.e., constraint driven learning) prevents a model from overfitting that could occur if the model was trained on noisy data. Some experiments were also conducted on projecting only reliable relations and training a dependency parser on partial dependency structures (Jiang and Liu, 2010; Spreyer, 2011; Täckström, 2013).

Mareček (2011) proposes a dependency projection approach based on combinations of different sets of word alignment links. As some of alignment link types may be more preferred in projection than others, Mareček (2011) uses them to sort target words aligned with a particular source token. Lists of preferences are used to project dependency relations. This approach does not apply language-specific post-correction rules, but as a result of radical filtering more than 87% of sentences are filtered out. This article inspired us to use a combination of different word alignments. In contrast to Mareček (2011), differently consolidated alignment links are used in our approach to weight projected dependencies, and not to designate the priority of target tokens corresponding to a source token.

Jiang and Liu (2009) propose a dynamic programming procedure for searching for projected Chinese trees that are most consistent with source English dependency trees, i.e., Chinese trees that have the highest confidence score estimated as a product of their edge scores. However, only induced trees with the confidence score greater then or equal to 0.35 and the number of words per sentence greater than 6 and less than 100 are included in the set of training trees. Since only 500,000 Chinese trees out of 5.6 million projected trees[28] fulfil these constraints, the filtering procedure leads to an enormous data loss. The MST parser (McDonald et al., 2005a) trained on selected dependency trees achieves performance of 53.28%[29] when evaluated against a part of the Penn Chinese Treebank (CTB). After adaptation to the CTB annotation schema, parsing performance increases to 87.34%.

[28] Projection was conducted on the set of 5.6 million LDC Chinese-English sentence pairs. According to this, the supposed output seems to contain 5.6 million projected graphs. However, only 500.000 Chinese dependency trees met predefined criteria (e.g., 'projecting confidence') and were selected.

[29] Authors do not mention how performance of the parser is measured.

Ganchev et al. (2009) present a *posterior regularisation* approach to learn generative and discriminative models for dependency parsing. In this approach, arcs generated by a supervised English parser and projected via aligned parallel corpus are used as constraints to regularise training of a target language parser. In order to avoid training models on target trees containing errors caused by partial or incorrect word alignment or inaccurate source language parses, training is conducted on a constrained group of arcs rather than entire projected target trees. The basic constraint ensures the expected proportion of conserved edges[30] in a sentence pair. In the performed experiment, training is conducted on these target trees for which the expected proportion of conserved edges in a sentence pair is at least 90%. In order to deal with structural divergences between languages and differences in annotation schemes used in treebanks, Ganchev et al. (2009) also introduce some language-specific constraints. Conserved edge-proportion constraint and language-specific constraints provide weak supervision for learning parsing models for the target language. These constraints are difficult to put into the model structure or to use as priors on the model parameters. Hence, models are estimated by constraining the posterior distribution over possible target trees in terms of projected arcs and language-specific rules. Experiments outlined in Ganchev et al. (2009) consist in projecting annotations from English to Bulgarian and Spanish. Results show that the approach outperforms not only unsupervised methods but also supervised parsers trained on a small amount of data if some language-specific constraints are applied. A discriminative transfer model trained for Bulgarian without additional language-specific rules achieves the attachment accuracy of 66.9%, while introducing 2 or 7 rules increases the accuracy to 77.5% or 78.3% respectively. This constraint-driven learning, which aims to project posterior constraints in order to improve estimation of parsing model parameters, significantly diverges from our approach to inducing well-formed target trees. However, we mention it to maintain a comprehensive overview of all dominating annotation projection approaches.

Smith and Eisner (2009) propose an approach to estimate a joint generative model for annotation projection based on *quasi-synchronous grammar* (QG, Smith and Eisner, 2006). Employing a bitext, the model is induced based on the source and target side syntactic structures, wherein the source structures are treated as fixed and observed. Conditioned on QG features,[31] the model learns syntactic relations that may occur in both languages and correspondences between them in order to predict target language sentences. In experiments with projecting English dependency structures into German and Spanish, the highest quality trees in terms of dependency accuracy are achieved with the presented projection procedure based on QG features and conditional EM training

[30] According to the definition given by Ganchev (2010, pp. 87ff.), an English edge $p \rightarrow c$ is conserved if word p aligns to word p' in the second language, c aligns to c' in the second language and p' is the governor of c'.

[31] The QG model applies both monolingual QG features of words (e.g., part of speech tags of related tokens) and dependencies (similar as features of an arc-factored dependency parsing, e.g., arc direction), and bilingual QG features (i.e., alignment features that consider different alignment scenarios, e.g., two target tokens are aligned with one source token, two target tokens are aligned with two source tokens which are in grandparent–grandchild relation, a target token is aligned with the NULL token)

– 68.5% for German and 64.8% for Spanish. Other projection procedures presented for comparison purposes perform slightly worse.

Apart from approaches employing aggressive filtering or smoothing techniques, there are also some proposals for training dependency parsers on partial dependency projections (Jiang and Liu, 2010; Spreyer, 2011). Training supervised parsing models on partial dependency projections is one of topics explored in the PhD thesis by Spreyer (2011). According to her results, a parsing model may be trained on dependency tree fragments projected with the highest probability. Parsers trained on incomplete trees may achieve performance as high as parsers trained on entire trees. These methods differ from our approach since we aim to induce well-formed dependency trees and not dependency tree fragments. On the other hand, we focus on induction of data that could be used repeatedly for training dependency parsers, similarly as Spreyer (2011).

There is also research on the direct transfer of delexicalised parsers between languages (Zeman and Resnik, 2008; McDonald et al., 2011; Søgaard, 2011). A source parser is trained on delexicalised structures in which dependency relations connect part of speech tags instead of tokens. Since the parser predicts dependency structures based solely on part of speech tags of input tokens, it may parse target sentences represented as strings of part of speech tags. Results by McDonald et al. (2011) indicate that the directly transferred delexicalised parser outperforms state-of-the-art unsupervised models. Furthermore, part of speech tags provide enough information for unlabelled dependency parsing and the knowledge of lexical items is redundant. In the experiment reported by McDonald et al. (2011), the delexicalised English *MaltParser* obtains 82.5% UAS, in comparison to the English *MaltParser* with all features that achieves 89.3% UAS.

Besides, McDonald et al. (2011) propose an approach to adapting directly transferred parsers to a target language using constraint driven learning algorithm. The procedure starts with training a lexicalised parser on target structures which are predicted by a directly transferred delexicalised parser. Then, any English tree predicted with the English lexicalised parser is compared with k-best parse trees predicted by the target lexicalised parser, and the parse whose dependencies are related most closely to English dependencies (in terms of word alignment links) is selected. Selected parses contribute to update the parameter vector of the new target parser, so that it could predict parses that align with corresponding English trees as well as possible. There is one aspect in which our approach is similar to the approach by McDonald et al. (2011) – the idea of employing k-best trees. McDonald et al. (2011) use a list of k-best trees to select a tree that best corresponds to an English tree in terms of alignment constraints. We use sets of k-best trees to recalculate weights on arcs in projected multi-digraphs. In this context, it is also worth mentioning the work by Hall (2007) who employs the k-best MST algorithm in dependency parsing.

Several studies have also been conducted on multi-source cross-lingual transfer. In these approaches, the cross-lingual transfer of dependency parsing models is improved by connecting model parameters from multiple source languages (McDonald et al., 2011; Søgaard, 2011) or by *selective sharing* parameters based on topological features of each language (Naseem et al., 2012; Täckström et al., 2013).

As our work concerns the dependency projection from English to Polish, it is worth noting that some experiments with this language pair have already been conducted, e.g., Ozdowska (2006) or Wróblewska and Frank (2009). In experiments presented by Ozdowska (2006), dependencies are directly projected from English or French to Polish using intersection of unidirectional word alignments. The dependency projection via intersection links seems to increase precision and decrease recall. Ozdowska reports on precision of projected labelled/unlabelled dependencies[32] but not on recall. The results are not significantly affected by the choice of the source language or divergences between involved pair of languages.

Wróblewska and Frank (2009) adapt annotation projection to the framework of *Lexical Functional Grammar* (Bresnan, 2001; Dalrymple, 2001) and induce a bank of Polish f-structures projected from English. Their framework refers to Hwa et al. (2005) since they combine direct projection with language-specific post-correction rules that significantly improve the quality of induced f-structures. Since no additional NLP tools are applied, induced f-structures encode predicate-argument structures of annotated sentences and do not encode morphological features (e.g., case, gender, number) which may not directly result from equivalent English f-structures. Therefore, the induced f-structures may be referred to as PRED-only f-structures since they only contain PRED attributes with their semantic form values and arguments of these predicates with their lower level f-structure values. The directly projected labelled f-structures when evaluated against a set of manually annotated Polish f-structures achieve F-score of 0.50, and 0.63 if corrected with hand-crafted rules.

5.7 Partial Conclusions

This chapter presented a novel weighted induction method of obtaining Polish dependency structures. The weighted induction procedure consists of two main steps: projection of dependency relations and induction of well-formed dependency trees. The projection step resembles cross-lingual dependency projection pioneered by Hwa et al. (2005). However, it is not required in our approach that projection results in dependency trees as in Hwa et al. (2005) or partial dependency structures as in Jiang and Liu (2010) or

[32]Ozdowska achieves precision of 83%/62% when projecting French unlabelled/labelled dependency relations to Polish, and 82%/67% when unlabelled/labelled dependencies are projected from English to Polish.

Spreyer (2011). Instead, all possible dependency relations are projected and they constitute initially weighted multi-digraphs. Previous approaches do not need any further steps after projection of dependency relations since projected trees (or tree fragments) are considered to be the final data for parser training. In our approach, the projected multi-digraphs may contain noisy arcs that should not be used in parser training. We thus proposed a method of recalculating initial weights of arcs in the projected multi-digraphs. The final maximum spanning dependency trees were selected from the projected multi-digraphs with recalculated arc weights. The induced MSDTs are thereby well-formed and presumably more appropriate than directly projected trees. The induced maximum spanning dependency trees were employed to train a dependency parser for Polish.

Similarly as in the evaluation of the conversion-based Polish dependency treebank described in the previous chapter, we performed an extrinsic evaluation and trained dependency parsers on the MSDTs acquired with the weighted induction method. The results showed that weighted induction performs significantly better than baseline induction. Furthermore, performance of an induction-based parser may even be slightly higher than performance of the conversion-based dependency parsers when evaluated against the complex test trees. While our experiment considered the Polish-English language pair, the weighted induction method may be applied to obtain dependency structures for other resource-poor languages which do not have any annotated data but have a reasonable number of sentences which are parallel with their translations in a resource-rich language. The weighted induction method was tested on the task of obtaining dependency structures, but it may also apply to other projection tasks, e.g., semantic role labelling or word sense disambiguation.

Our goal was to create a bank of high quality dependency trees that could be used several times to train dependency parsers. Therefore, our approach differs from constraint-driven learning approaches (Ganchev et al., 2009; Smith and Eisner, 2009) which apply projected information to constrain estimation of dependency parsing models. Our method also differs from approaches to transferring delexicalised parsers between languages (Zeman and Resnik, 2008; McDonald et al., 2011; Søgaard, 2011; Naseem et al., 2012; Täckström et al., 2013).

Chapter 6

Conclusion

The rapid progress in dependency parsing has brought about the creation of efficient and very accurate data-driven systems. These multilingual systems developed mainly in shared tasks on multilingual dependency parsing are language-independent and can be employed for almost every natural language. Since most of these systems apply supervised machine learning techniques, they require high-quality training data. Shared tasks on multilingual dependency parsing have also brought about the dissemination of treebanks of participating languages. These treebanks constitute excellent resources for further development and improvement of data-driven parsing systems.

However, there were only 19 languages participating in the CoNLL shared tasks[1] while there are currently more than 7000 languages spoken worldwide (Lewis et al., 2013). Since dependency parsers are very useful tools for many sophisticated NLP tasks, it should be possible to train dependency parsers for every language. As manually annotated dependency treebanks exist for only a few languages, the availability of training data is still the main bottleneck for supervised approaches. In this dissertation, we have therefore explored two methods of annotating data automatically.

6.1 Summary

Chapter 1 introduced the topic and main assumptions of the dissertation.

Chapter 2 presented concepts of a dependency structure and data-driven dependency parsing. The notion of a dependency structure that originates from dependency formalisms has been adopted for the purpose of dependency parsing. A dependency structure that satisfies all well-formedness properties mentioned in this chapter constitutes a

[1]The following languages participated in the shared task at CoNLL 2006: Arabic, Bulgarian, Chinese, Czech, Danish, Dutch, German, Japanese, Portuguese, Slovene, Spain, Swedish and Turkish. The following languages participated in the shared task at CoNLL 2007: Arabic, Basque, Catalan, Chinese, Czech, English, Greek, Hungarian, Italian and Turkish.

basis for dependency parsing. Dependency parsing is understood as a process of automatic allocation of a dependency structure to an input sentence by a dependency parser. Two main parsing methods based on supervised statistical machine learning were presented in detail in this chapter: transition-based dependency parsing and graph-based dependency parsing.

The main contributions of this dissertation are included in chapters 3–5. In chapter 3, we described a dependency annotation schema which was designed for annotation of Polish sentences with dependency structures. The content of this chapter is based to a large extent on Wróblewska (2012). The annotation schema is adjusted to the linguistic characteristics of Polish and covers primary syntactic phenomena in this language. We tried to maintain a reasonable number of individual dependency relation types. There are thus 28 dependency relation types defined for Polish including arguments, syntactically, morphologically or semantically motivated non-arguments, and relations used to annotate coordinating constructions. Individual dependency relations were characterised and illustrated by examples.

Chapter 4 explored the method of constituency-to-dependency conversion and its adaptation for acquiring a treebank of valid Polish dependency structures. To a large extent the content of this chapter is based on Wróblewska and Woliński (2012) and Wróblewska (2012). A publicly available Polish treebank – *Składnica* – with manually disambiguated constituent trees enabled the adaptation of the conversion method to Polish. The main idea behind the conversion was to cover all language-specific syntactic phenomena encoded in Polish constituent trees and to annotate them with correctly chosen dependencies. Explicitly marked heads of constituents made it relatively straightforward to convert phrase structure trees into unlabelled dependency structures. Despite this, there were some phrase structures in which not only one but several elements were marked as syntactic heads. For these multi-headed constituents, some head-selection heuristics were designed in order to find an unequivocal governor for any token.

Even if the conversion was a relatively straightforward process, some rearrangements of dependency structures were necessary so that converted trees could meet annotation principles defined by the dependency annotation schema in Chapter 3 *Polish Dependency Annotation Schema*. Rearrangement rules were designed for some particular linguistic constructions (discontinuous constituents, passive constructions, subordinate clauses, incorporated conjunctions and clauses with correlative pronouns). Except for rearrangement rules, we also designed a set of complex labelling rules that assign appropriate labels to converted dependency relations.

The entirely automatic conversion process resulted in a bank of 8227 labelled dependency trees. These trees were used in empirical experiments on training and evaluation of Polish dependency parsers. In order to check whether it is possible to train a dependency parser for Polish on a part of the treebank (7405 trees), we applied two dependency

parsing systems: the transition-based *MaltParser* and the graph-based *Mate* parser. Parsing performance was evaluated against the set of 822 converted dependency trees with manually or automatically annotated tokens, and the additional test set of 100 complex trees. The *MaltParser* achieved 90.5% UAS and 85.4% LAS when evaluated against the test trees with manually annotated tokens, and 85.3% UAS and 78.4% LAS in the more realistic evaluation scenario where tokens were automatically annotated with the *Pantera* tagger. The *Mate* parser outperformed *MaltParser* and achieved 92.7% UAS and 87.2% LAS when evaluated against the manually annotated test trees, and 88.4% UAS and 81% LAS in the automatic evaluation scenario. Although we did not investigate the impact of automatic tokenisation and part of speech tagging explicitly, the results suggest that improvements in underlying language processing should enhance the quality of dependency parsing.

As the evaluation experiments showed, it is possible to train a Polish dependency parser on a relatively small set of constituency-to-dependency converted trees. The best Polish parser trained with the *Mate* parsing system obtained quite high results: 92.7% UAS and 87.2% LAS. However, there was a precondition that the *Mate* parser was trained and evaluated on trees with manually annotated tokens. If the parser was evaluated against trees with automatically annotated tokens, its performance decreased even by 6 percentage points. On the other hand, if the *Mate* parser was trained on the converted trees with automatically annotated tokens, it parsed almost equally well sentences with manually and automatically annotated tokens.

It is important to note that different parts of the conversion-based treebank were employed in training and in evaluation of the parsers. Apart from the fact that training and test data come from the same source, the converted trees have relatively simple structures. Therefore, the presented parsing models were also evaluated against an additional validation set of 100 complex dependency structures. In this realistic scenario of parsing complex sentences, the *Mate* parser trained on the converted trees with automatic morphosyntactic annotations of tokens beat other dependency models reaching 76.6% UAS and 70.1% LAS. However, these results are significantly lower than the results reported above.

In chapter 5, we proposed a different way of obtaining dependency structures which is based on a novel weighted induction method. A preliminary version of this method was presented in Wróblewska and Przepiórkowski (2012). The novelty of the method consists in involving a weighting factor in two successive processes of projecting English dependency relations and acquiring unlabelled dependency trees from projected multi-digraphs.

Weighted projection was inspired by cross-lingual dependency projection pioneered by Hwa et al. (2005). Using a parallel English-Polish corpus, the English side was automatically annotated with a parser underpinned by the English LFG grammar and resulting

annotations were transferred to equivalent Polish sentences via an extended set of word alignment links. Instead of projecting relations via links of a single automatic word alignment, dependency relations were projected via bipartite alignment graphs combining links from different automatic word alignments with some additional edges. Projected arcs were then initially weighted based on the certainty of bipartite edges.

Weighted induction consists in extracting maximum spanning dependency trees from multi-digraphs containing all projected arcs with recalculated weights. Arc weights were recalculated based on the probability distribution over reliable arcs. The set of reliable arcs consisted of arcs from k-best MSDTs selected from initially weighted projected multi-digraphs. The probability distribution was estimated with the EM-inspired selection algorithm.

The empirical part of this chapter outlined experiments on training dependency parsers on dependency structures induced from EM-scored multi-digraphs and labelled using a set of predefined labelling rules. The *Mate* dependency parser trained on dependency structures acquired with the weighted induction method achieved 85.1% UAS and 79.2% LAS when tested against the set of 822 conversion-based test trees with manually annotated tokens, and 84% UAS and 77.3% LAS when tested against the same set of test trees with automatically annotated tokens. If additional correction rules were applied, parsing performance increased up to 85.1% UAS and 79.2% LAS. The *Mate* parser trained on corrected and filtered induced trees achieved 86% UAS and 80.5% LAS when tested against the test trees with manually annotated tokens. In the realistic parsing scenario, when the *Mate* parser with the *filtered* model was evaluated against the set of 100 complex trees, the parser achieved significantly lower results – 76.1% UAS and 70.3% LAS – than the results reported above. Nevertheless, these results were close to the results achieved by the conversion-based dependency parser when evaluated agains the complex test trees.

6.2 Comparison of Conversion-based and Projection-based Approaches

In the course of this dissertation, we have tried to answer two research questions posed in Introduction:

1. Is it possible to gather dependency trees automatically (or with a minimal human involvement)?

2. Is it possible to train a good quality supervised dependency parser on automatically or semi-automatically induced training data?

We can provide affirmative answers to both questions since we succeeded in gathering dependency structures automatically with conversion-based or induction-based methods, and in training dependency parsing models on these structures. Now, we make a comparative analysis of conversion-based and induction-based approaches. To compare these approaches, we take into account the following criteria: performance of parsers trained on gathered data, the amount of manual work necessary in a particular method and applicability of presented methods for other languages.

Acquired dependency trees were evaluated extrinsically, i.e., a dependency parser was trained on automatically acquired trees and evaluated against test trees. The quality of parsing indirectly determines the quality of the acquired trees. Dependency parsers, especially those trained on converted Polish dependency structures, achieved parsing performance comparable to dependency parsers trained for other Slavic languages. When tested against 822 converted test trees with manually annotated tokens, the parser trained on 7405 converted dependency structures outperformed the parser trained on more than 2 million automatically induced and filtered trees by about 7 percentage points (both, in terms of LAS and UAS scores). The conversion-based parser also outperformed the parser trained on automatically induced trees when tested against the converted test trees with automatically assigned morphosyntactic annotations. However, in this case, the difference is much smaller – 2.7 pp in terms of LAS and 3.7 pp in terms of UAS. When tested against the additional test set with longer and more complex sentences, both parsers performed significantly worse, but the induction-based parser reached (or even slightly exceeded) performance of the conversion-based parser. There may be several reasons for the decrease in parsing performance, e.g., the complexity of trees in the additional test set, discrepancies in tokenisation and in part of speech tagging of sentences which the converted trees and the additional test trees are structured on.

Considering the complexity of trees, the conversion-based test trees and the additional test trees differ in terms of the average length of sentences – 10.08 tokens per sentence vs. 16.6 tokens per sentence respectively. Trees in these two test sets also differ in the number of non-projective arcs. There is only 0.46% of non-projective arcs in the conversion-based test trees and 2.8% of non-projective arcs in the additional test trees. Parsers evaluated against more complex trees perform worse, but the results are more realistic and constitute the lower bound for performance of Polish parsers.

Morphosyntactic analysis is an important factor that influences parsing quality, next to correctness of training trees. Since manual tagging of tokens in the converted trees is not consistent with tagging performed by a state-of-the-art Polish tagger, the conversion-based parser does not handle automatically annotated input sentences with possible noise as well as it handles correctly annotated sentences. Dependency structures automatically acquired with the weighted induction method are based on automatic part of speech tagging that may be noisy. Hence, the induction-based parser trained on noisy data

performs slightly better than the conversion-based parser when tested against additional test trees with automatically tagged tokens.

Another problem arises from tokenisation discrepancies. The converted dependency trees adapted the manual tokenisation lying at the core of the source constituent trees. Most of additional test sentences, in turn, were automatically tokenised. This tokanisation was not manually modified and may differ from the tokenisation of the converted trees. Tokenisation discrepancies might result in poorer performance of the conversion-based parser when tested against the additional test trees. The tokenisation underlying dependency trees obtained with the weighted induction method was performed automatically and is similar to the tokenisation of additional test trees (provided that the same tokeniser was used). Therefore, the *Mate* parser with the induction-based model (*filtered*) performed slightly better than the *Mate* parser with the conversion-based model. The induction-based *Mate* parser may be thus more appropriate for a realistic and fully automatic parsing scenario.

Both approaches require manual work that consists in the construction of rules. In the conversion-based approach, we manually designed conversion rules (labelling rules, head selection rules and rearrangement rules). These rules cover all linguistic phenomena available in the source constituent trees. In the induction-based approach, we manually designed labelling and correction rules. Labelling rules were defined to assign labels to projected dependency relations. Correction rules were designed to amend errors and divergences, which frequently occur in induced dependency trees. Both labelling and correction rules are very general and cover only the most frequent Polish linguistic phenomena or errors.

Since constituency-to-dependency conversion rules should cover all linguistic phenomena existing in source trees, it might take a lot of manual work to design them. However, in our case, the number of different linguistic phenomena was limited to these available in 8227 relatively simple sentences of the Polish constituent treebank. Hence, a reasonable amount of manual work was required. Similarly, the amount of manual work required to design correction rules modifying automatically induced dependency trees was moderate since rules only cover some particularly frequent errors. Even though manual work was essential in both approaches, conversion-based and induction-based dependency structures were achieved with less manual work than in annotation of thousands of sentences of a complete dependency treebank.

Applicability of presented methods to other languages is the last comparison issue. The conversion-based method is only applicable to languages for which a constituent treebank exists. However, since there are many languages for which neither dependency nor constituency treebanks exist, this method is useless for them. Moreover, even if there is a constituent treebank for a language, the conversion method may not apply directly.

To adopt this method, head-selection rules, language-specific and formalism-specific conversion rules need to be defined anew.

For languages without any treebank, neither dependency nor constituency based, the weighted induction method of acquiring well-formed dependency trees may be an advantageous solution. Since unlabelled dependency trees are induced fully automatically, the weighted induction method may be employed for any language for which a sufficient amount of bitexts (in this language and in a resource-rich language) and a dependency parser for the resource-rich language are available. Manual work is only necessary to design labelling and correction rules modifying automatically induced dependency trees.

Performance of the parser trained on dependency trees acquired with the weighted induction method is mostly slightly below performance of the parser trained on the converted trees when tested on a homogeneous set of rather short sentences from the conversion-based treebank. A test against a small set of long and complex trees showed that the induction-based parser may exceed the conversion-based upper bound. Hence, we may conclude that the induction-based approach to acquiring dependency trees may compete with the conversion-based approach. Furthermore, if more manual work is invested into defining language-specific correction rules or filtering heuristics, the induction approach will provide more appropriate trees and a parser trained on them will thus perform better. Both methods of acquiring training data require an amount of manual work, but only the weighted induction approach is language-independent and may apply to other resource-poor languages.

6.3 Final Remark

In the introduction of this dissertation it was mentioned that Polish did not participate in any of shared tasks on multilingual dependency parsing. However, our doctoral work has contributed to change this state of affairs. There was a shared task on multilingual parsing recently organised at the Workshop on Statistical Parsing of Morphologically Rich Languages in which Polish was represented (Seddah et al., 2013). The Polish dependency data used in this shared task was taken from the conversion-based Polish dependency treebank which was created as part of our work and is described in Chapter 4 *Conversion-based Dependency Bank* of this dissertation.

Appendix A

Labelling Rules Based on Morphosyntactic Properties

Rule 1. Complement of a comparative

Rule: **If** $lemma_{dep}$[1] $\in$ ['niż', 'aniżeli', 'od'] and $word_{gov}$ is marked for comparative degree and $id_{gov} < id_{dep}$ **then** $gf_{dep} := comp$

Example: jaśniejszy$_{adj.comp} \xrightarrow{comp}$ niż słońce (Eng. 'brighter than the sun')

Rule 2. Complement of a number (1)

Rule: **If** $pos_{dep} \in [subst, depr, ign, conj]$ and $pos_{gov} = ign$ and $lemma_{gov}$ is an integer and $id_{gov} < id_{dep}$ and $lemma_{dep} \notin$ ['tysiąc', 'milion', 'miliard', 'bilion'] **then** $gf_{dep} := comp$

Example: 3$_{ign} \xrightarrow{comp}$ drzewa$_{subst}$ (Eng. '3 trees')

Rule 3. Complement of a number (2)

Rule: **If** $lemma_{dep} \in$ ['tysiąc', 'milion', 'miliard', 'bilion'] and $pos_{gov} = ign$ and $lemma_{gov}$ is an integer and $id_{gov} < id_{dep}$ **then** $gf_{dep} := mwe$

Example: 2$_{ign} \xrightarrow{mwe}$ tysiące$_{subst}$ drzew (Eng. '2 thousand trees')

[1] Abbreviations used in labelling rules:
dep – the current node (dependent),
gf – grammatical function,
gov – governor of the current node,
id – position of a token (*dep* or *gov*) in a sentence,
lemma – lemma form of *dep* or *gov*,
case – the grammatical category *case*,
num – the grammatical category *number*,
pos – part of speech of *dep* or *gov*,
word – surface realisation of *dep* or *gov*.
Abbreviations of Polish parts of speech and grammatical functions are clarified in List of Abbreviations and in Chapter 3 *Polish Dependency Annotation Schema* respectively.

Rule 4. Complement of a numeral (1)

Rule: **If** $pos_{dep} \in$ [*subst*, *depr*, *ign*, *conj*] and $pos_{gov} \in$ [*num*, *numcol*] and $id_{gov} < id_{dep}$ and $lemma_{dep} \notin$ [‘tysiąc’, ‘milion’, ‘miliard’, ‘bilion’] **then** $gf_{dep} :=$ *comp*

Example: troje$_{num} \xrightarrow{comp}$ dzieci$_{subst}$ (Eng. ‘three children’)

Rule 5. Complement of a numeral (2)

Rule: **If** $lemma_{dep} \in$ [‘tysiąc’, ‘milion’, ‘miliard’, ‘bilion’] and $pos_{gov} \in$ [*num*, *numcol*] and $id_{gov} < id_{dep}$ **then** $gf_{dep} :=$ *mwe*

Example: dwa$_{num} \xrightarrow{mwe}$ tysiące$_{subst}$ (Eng. ‘two thousand’)

Rule 6. Complement of a numeral (fractions)

Rule: **If** $lemma_{dep} \in$ [‘pierwszy’, ‘drugi’, ‘trzeci’, ‘czwarty’, ‘piąty’, ‘szósty’, ‘siódmy’, ‘ósmy’, ‘dziewiąty’, ‘dziesiąty’, ‘setny’, ‘tysięczny’] and $lemma_{gov} \in$ [‘jeden’, ‘dwa’, ‘trzy’, ‘cztery’, ‘pięć’, ‘sześć’, ‘siedem’, ‘osiem’, ‘dziewięć’, ‘sto’] **then** $gf_{dep} :=$ *mwe*

Example: dwie$_{num} \xrightarrow{mwe}$ trzecie$_{adj}$ (Eng. ‘two-thirds’)

Rule 7. Complement of a preposition (1)

Rule: **If** $lemma_{gov} \in$ [‘niż’, ‘aniżeli’] $gf_{gov} \in$ [OBJ, OBL-COMPAR, *adjunct*, *comp*] and $id_{gov} < id_{dep}$ **then** $gf_{dep} :=$ *comp*

Example: jaśniejszy$_{adj.comp}$ niż $\xrightarrow{comp}$ słońce (Eng. ‘brighter than the sun’)

Rule 8. Complement of a preposition(2)

Rule: **If** $pos_{dep} \in$ [*subst*, *depr*, *ppron12*, *ppron3*, *ger*] and $gf_{dep} =$ OBJ and $lemma_{gov} \in$ [‘jak’, ‘jako’] and $id_{gov} < id_{dep}$ **then** $gf_{dep} :=$ *comp*

Example: mądry jak $\xrightarrow{comp}$ on$_{ppron3}$ (Eng. ‘wise as he’)

Rule 9. Complement of a preposition (3)

Rule: **If** $pos_{gov} =$ *prep* and $id_{gov} < id_{dep}$ and *gov* doesn’t have a complement and *dep* is the best complement **then** $gf_{dep} :=$ *comp*

Example: w $\xrightarrow{comp}$ domu$_{subst}$ (Eng. ‘at home’)

Rule 10. Complement of a preposition (abbreviated third-person pronoun)

Rule: **If** $pos_{gov} =$ *prep* and $pos_{dep} =$ *ppron3* and $word_{dep} =$ ‘ń’ and $id_{gov} < id_{dep}$ **then** $gf_{dep} :=$ *comp*

Example: przeze $\xrightarrow{comp}$ ń$_{ppron3}$ (≈ przez niego, Eng. ‘by him’)

Rule 11. Complement of a preposition (post-prepositional adjective)

Rule: **If** pos_{dep} = *adjp* and pos_{gov} = *prep* and $id_{gov} < id_{dep}$ **then** gf_{dep} := *mwe*

Example: po $\xrightarrow{mwe}$ prostu$_{adjp}$ (Eng. 'simply')

Rule 12. Complement of a subordinator

Rule: **If** pos_{gov} = *comp* and $id_{gov} < id_{dep}$
If $pos_{dep} \in$ [*bedzie*, *fin*, *imps*, *ppas*, *praet*, *pred*, *winien*] **then** gf_{dep} := *comp_fin*
Elif pos_{dep} = *inf* **then** gf_{dep} := *comp_inf*
Elif pos_{dep} = *conj* and coordinated conjuncts $\in$ [*bedzie*, *fin*, *imps*, *ppas*, *praet*, *pred*, *winien*] **then** gf_{dep} := *comp_fin*
Elif pos_{dep} = *conj* and coordinated conjuncts $\in$ [*inf*] **then** gf_{dep} := *comp_inf*
Else gf_{dep} := *dep*

Example: Płacze, choć$_{comp}$ $\xrightarrow{comp\_fin}$ wygrał$_{praet}$ (Eng. 'He is crying although he won')

Rule 13. Modifier of an adverb/adjective

Rule: **If** pos_{dep} = *adv* and $pos_{gov} \in$ [*adj*, *adv*] and id$_{dep} < id_{gov}$ **then** gf_{dep} := *adjunct*

Examples: dużo$_{adv}$ $\xleftarrow{adjunct}$ bardziej$_{adv}$ (Eng. 'much more')

bardzo$_{adv}$ $\xleftarrow{adjunct}$ zły$_{adj}$ (Eng. 'very bad')

Rule 14. Modifier of a substantive (1)

Rule: **If** $pos_{dep} \in$ [*adj*, *pact*, *pcon*, *ppron12*, *ppron3*] and $pos_{gov} \in$ [*subst*, *depr*, *num*, *numcol*, *ign*, *ger*] **then** gf_{dep} := *adjunct*

Example: dobry$_{adj}$ $\xleftarrow{adjunct}$ ojciec$_{subst}$ (Eng. 'a good father')

Rule 15. Modifier of a substantive (2)

Rule: **If** pos_{dep} = *ppas* and $pos_{gov} \in$ [*subst*, *depr*, *num*, *numcol*, *ger*] and *dep* doesn't have any auxiliary dependent **then** gf_{dep} := *adjunct*

Example: przestępstwo$_{subst}$ $\xrightarrow{adjunct}$ popełnione$_{ppas}$ nocą (Eng. 'a crime committed at night')

Rule 16. Modifier of a substantive (3)

Rule: **If** pos_{dep} = *prep* and $pos_{gov} \in$ [*subst*, *depr*] and $id_{gov} < id_{dep}$ **then** gf_{dep} := *adjunct*

Example: plama$_{subst}$ $\xrightarrow{adjunct}$ na$_{prep}$ ścianie (Eng. 'a spot on the wall')

Rule 17. Modifier of a substantive (4)

Rule: **If** pos_{dep} = *ign* and $pos_{gov} \in$ [*subst*, *depr*] **then** gf_{dep} := *adjunct*

Example: emisja$_{subst}$ $\xrightarrow{adjunct}$ CO2$_{ign}$ (Eng. 'emissions of CO_2 (carbon dioxide)')

Rule 18. Modifier of a substantive (5)

Rule: **If** $pos_{gov} \in$ [*subst*, *depr*, *ger*, *ppron12*, *ppron3*] and $id_{gov} < id_{dep}$
If $pos_{dep} \in$ [*subst*, *depr*, *ger*, *ppron12*, *ppron3*] and $case_{dep}$ = 'gen'
then $gf_{dep} :=$ *adjunct*
Elif pos_{dep} = *conj* and parts of speech of conjuncts ∈ [*subst*, *depr*, *ger*, *ppron12*, *ppron3*] and their case = 'gen' **then** $gf_{dep} :=$ *adjunct*

Example: $\text{księga}_{subst} \xrightarrow{adjunct} \text{mądrości}_{subst}$ (Eng. 'a book of wisdom')

Rule 19. Modifier of unknown tokens (ign)

Rule: **If** $pos_{dep} \in$ [*subst*, *depr*, *ger*, *ppron12*, *ppron3*, *ign*, *burk*, *conj*] and pos_{gov} = *ign*
then $gf_{dep} :=$ *adjunct*

Example: $\text{Ispra}_{ign} \xrightarrow{adjunct} \text{Włochy}_{subst}$ (Eng. 'Ispra Italy')

Rule 20. Punctuation mark

Rule: **If** pos_{dep} = *interp* and it is not a coordinating element
If pos_{gov} = *brev* and $lemma_{dep}$ = '.' **then** $gf_{dep} :=$ *abbrev_punct*
Elif $word_{dep} \in$ ['-', 'ć'] and $id_{dep} = 1$ and $if_{gov} \neq 0$ **then** $gf_{dep} :=$ *item*
Else $gf_{dep} :=$ *punct*

Example: $\text{Stój}_{impt} \xrightarrow{punct} !_{interp}$ (Eng. 'Stop!')

Rule 21. Verb dependent – auxiliary verb

Rule: **If** $pos_{dep} \in$ [*bedzie*, *fin*, *impt*, *inf*, *pact*, *pcon*, *praet*] and $lemma_{dep} \in$ ['być', 'zostać, 'zostawać'] and $gf_{dep} \notin$ [COMP, XCOMP] and $pos_{gov} \in$ [*inf*, *ppas*, *praet*, *pred*, *conj*]
then $gf_{dep} :=$ *aux*

Examples: $\text{będę}_{bedzie} \xleftarrow{aux} \text{płakać}_{inf}$ (Eng. 'I will cry')

$\text{zostało}_{praet} \xleftarrow{aux} \text{zrobione}_{ppas}$ (Eng. 'it was done')

Rule 22. Verb dependent – complementiser

Rule: **If** pos_{dep} = *comp* and $lemma_{dep} \in$ ['aby', 'ażeby', 'by', 'iż', 'iżby', 'że', 'żeby'] and $id_{dep} < id_{gov}$
If $pos_{gov} \in$ [*bedzie*, *fin*, *imps*, *impt*, *inf*, *ppas*, *pcon*, *pant*, *praet*, *pred*, *winien*]
then $gf_{dep} :=$ *complm*
Elif $pos_{gov} \in$ [*conj*, *interp*, *adj*] and gf_{dep} = COMP-FORM **then** $gf_{dep} :=$ *complm*
Else $gf_{dep} :=$ *dep*

Example: $\text{Powiedział, że}_{comp} \xleftarrow{complm} \text{przyjdzie}_{fin}$ (Eng. 'He said that he would come')

Rule 23. Verb dependent – conditional clitic

Rule: **If** $pos_{dep} \neq comp$ and $lemma_{dep}$ = 'by'
If $pos_{gov} \in$ [*praet*, *imps*, *winien*] **then** $gf_{dep} := cond$
Elif $pos_{gov} = inf$ and *gov* built complex future form **then** $gf_{dep} := cond$
Elif $pos_{gov} = inf$ and $gf_{gov} \in$ [*comp_inf*, XCOMP] and $id_{dep} < id_{gov}$
then $gf_{dep} := comp$
Else $gf_{dep} := dep$

Example: mógł$_{praet} \xrightarrow{cond}$ by$_{qub}$ (Eng. 'he could')

Rule 24. Verb dependent – imperative marker

Rule: **If** $lemma_{dep} \in$ ['niech', 'niechaj', 'niechże', 'niechajże'] and $pos_{gov} \in$ [*bedzie*, *fin*] **then** $gf_{dep} := imp$

Example: Niech$_{qub} \xleftarrow{imp}$ idzie$_{fin}$ (Eng. 'Let him go')

Rule 25. Verb dependent – mobile inflection

Rule: **If** $pos_{dep} = aglt$ and $lemma_{dep}$ = 'być'
If $pos_{gov} \in$ [*praet*, *winien*] and $num_{dep} = num_{gov}$ **then** $gf_{dep} := aglt$
Elif $pos_{gov} \neq comp$ and $lemma_{gov}$ = 'by' **then** $gf_{dep} := aglt$
Else $gf_{dep} := dep$

Example: zrobili$_{praet} \xrightarrow{aglt}$ śmy$_{aglt}$ (Eng. 'we have done')

Rule 26. Verb dependent – negation marker (1)

Rule: **If** $pos_{dep} = qub$ and $lemma_{dep}$ = 'nie' and $pos_{gov} \in$ [*bedzie*, *fin*, *imps*, *impt*, *inf*, *pact*, *pant*, *pcon*, *ppas*, *praet*, *winien*] and $id_{dep} < id_{gov}$ **then** $gf_{dep} := neg$

Example: nie$_{qub} \xleftarrow{neg}$ powinni$_{winien}$ (Eng. 'they should not')

Rule 27. Verb dependent – negation marker (2)

Rule: **If** $pos_{dep} = qub$ and $lemma_{dep}$ = 'nie' and $pos_{gov} = pred$ **then** $gf_{dep} := neg$

Example: to$_{pred} \xrightarrow{neg}$ nie$_{qub}$ prawda (Eng. 'this is not true')

Rule 28. Verb dependent – question marker

Rule: **If** $pos_{dep} = qub$ and $lemma_{dep}$ = 'czy' and $pos_{gov} \in$ [*bedzie*, *fin*, *imps*, *inf*, *ppas*, *praet*, *pred*, *winien*] and $id_{dep} < id_{gov}$ and *dep* doesn't have dependents labelled with *conjunct* **then** $gf_{dep} := adjunct$

Example: czy$_{qub} \xleftarrow{adjunct}$ płacze$_{praet}$ (Eng. 'is he/she crying')

Rule 29. Verb dependent – reflexive marker

Rule: **If** pos_{dep} = *qub* and $lemma_{dep}$ = 'się' and $pos_{gov} \in$ [*fin*, *ger*, *imps*, *impt*, *inf*, *pact*, *pant*, *pcon*, *praet*] **then** gf_{dep} := *refl*

Example: bojąc$_{pcon} \xrightarrow{refl}$ się$_{qub}$ (Eng. 'fearing')

Rule 30. Verb adjunct

Rule: **If** $pos_{dep} \in$ [*adv*, *prep*, *qub*] and $pos_{gov} \in$ [*fin*, *impt*, *praet*, *winien*, *pred*, *inf*, *imps*, *ppas*, *conj*, *adj*] **then** gf_{dep} := *adjunct*

Example: mówił$_{praet} \xrightarrow{adjunct}$ głośno$_{adv}$ (Eng. 'he spoke loudly')

Rule 31. Subordinated clause

Rule: **If** $gf_{dep} \in$ [COMP, XCOMP, XCOMP-PRED, OBJ, OBJ-TH, OBL, OBL-AG, PFORM, SUBJ] and $pos_{gov} \in$ [*fin*, *impt*, *praet*, *pred*, *bedzie*, *inf*, *ger*, *imps*, *pcon*, *pant*, *pact*, *ppas*, *ign*] and $id_{gov} < id_{dep}$ and *dep* governs a complementiser
If pos_{dep} = *inf* and *gov* doesn't have a dependent labelled with *comp_inf* **then** gf_{dep} := *comp_inf*
Elif $pos_{dep} \in$ [*fin*, *impt*, *praet*, *pred*, *bedzie*, *inf*, *winien*, *imps*, *ppas*] and *gov* doesn't have a dependent labelled with *comp_fin* **then** gf_{dep} := *comp_fin*
Else gf_{dep} := *dep*

Example: Chcę$_{fin} \xrightarrow{comp_inf}$ pomóc$_{inf}$ (Eng. 'I want to help.')

Appendix B

Labelling Rules Based on English Grammatical Functions

Argument Types

Rule 1. COMP, XCOMP

Rule: **If** $gf_{dep} \in$ [COMP, XCOMP]
If $pos_{gov} \in$ [*subst*, *depr*, *ppron12*, *ppron3*]
If $pos_{dep} \in$ [*bedzie*, *fin*, *imps*, *ppas*, *praet*, *pred*, *winien*, *conj*] and *gov* doesn't have a dependent labelled with *comp_fin* and *dep* governs a complementiser **then** $gf_{dep} :=$ *comp_fin*
Elif $pos_{dep} =$ *inf* and *gov* doesn't have a dependent labelled with *comp_inf* **then** $gf_{dep} :=$ *comp_inf*
Else $gf_{dep} :=$ *dep*
Elif $pos_{gov} \in$ [*adv*, *adj*, *qub*] and $pos_{dep} =$ *inf* and *gov* doesn't have a dependent labelled with *comp_inf* **then** $gf_{dep} :=$ *comp_inf*
Else $gf_{dep} :=$ *dep*

Example: fakt$_{subst} \xrightarrow{comp_fin}$ że warunki klimatyczne zmieniają się
(Eng. 'the fact that climatic conditions change'. Note: the underlined token *zmieniają* constitutes the head of the subordinated clause and is a direct dependent of *fakt*.)

Rule 2. OBJ, OBJ-TH

Rule: **If** $gf_{dep} \in$ [OBJ, OBJ-TH]
If $pos_{gov} \in$ [*fin*, *imps*, *impt*, *praet*, *pred*, *winien*, *ger*, *pact*, *pant*, *pcon*, *inf*] and $pos_{dep} \in$ [*subst*, *depr*, *ppron12*, *ppron3*, *ger*, *num*, *numcol*, *siebie*, *conj*] or *dep* is an integer
If *gov* doesn't have a dependent labelled with *obj* **then** $gf_{dep} :=$ *obj*,
Else $gf_{dep} :=$ *obj_th*
Elif $pos_{gov} \in$ [*fin*, *imps*, *impt*, *praet*, *pred*, *winien*, *ger*, *pact*, *pant*, *pcon*, *inf*] and $pos_{dep} \in$ [*adv*, *prep*] **then** $gf_{dep} :=$ *comp*
Elif $pos_{gov} \in$ [*num*, *numcol*] and $pos_{dep} \in$ [*subst*, *depr*, *ppron12*, *ppron3*, *ger*, *num*, *siebie*, *siebie*, *brev*] **then** $gf_{dep} :=$ *comp*
Elif $pos_{gov} \in$ [*prep*, *comp*] and *gov* doesn't have a dependent labelled with *comp* and $id_{gov} < id_{dep}$ **then** $gf_{dep} :=$ *comp*
Elif $lemma_{gov} \in$ ['gdy', 'kiedy', 'jak'] and $id_{gov} < id_{dep}$
If $pos_{dep} \in$ [*bedzie*, *fin*, *imps*, *praet*, *pred*, *winien*, *ppas*] **then** $gf_{dep} :=$ *comp_fin*,
Else $gf_{dep} :=$ *comp*
Else $gf_{dep} :=$ *dep*

Example: skutecznie intensyfikować$_{pcon} \xrightarrow{obj}$ aktywność$_{subst}$ wirusa (Eng. 'to effectively intensify the activity of the virus')

Rule 3. OBL

Rule: **If** $gf_{dep} =$ OBL
If $pos_{gov} \in$ [*fin*, *imps*, *impt*, *praet*, *pred*, *winien*, *ger*, *pact*, *pant*, *pcon*, *inf*] and $pos_{dep} \in$ [*subst*, *depr*, *ppron12*, *ppron3*, *ger*, *num*, *numcol*, *siebie*, *conj*] **then** $gf_{dep} :=$ *obj_th*
Elif $pos_{gov} \in$ [*fin*, *imps*, *impt*, *praet*, *pred*, *winien*, *ger*, *pact*, *pant*, *pcon*, *inf*] and $pos_{dep} \in$ [*adv*, *prep*] **then** $gf_{dep} :=$ *comp*
Elif $pos_{gov} =$ *prep* and $pos_{dep} \in$ [*subst*, *depr*, *ppron12*, *ppron3*, *ger*, *num*, *numcol*, *siebie*, *brev*] **then** $gf_{dep} :=$ *comp*
Elif $pos_{gov} =$ *adj* and $pos_{dep} =$ *prep* and $id_{gov} < id_{dep}$ **then** $gf_{dep} :=$ *comp*
Else $gf_{dep} :=$ *dep*

Example: podatny$_{adj} \xrightarrow{comp}$ na$_{prep}$ wymarcie (Eng. 'vulnerable to extinction')

Rule 4. OBL-AG

Rule: **If** gf_{dep} = OBL-AG
If $pos_{gov} \in$ [*ppas*, *ger*, *pact*, *pant*, *pcon*] and pos_{dep} = *prep*
If $lemma_{dep}$ = 'przez' and *gov* doesn't have a dependent labelled with *comp_ag* **then** gf_{dep} := *comp_ag*
Else gf_{dep} := *comp*
Elif $pos_{gov} \in$ [*fin*, *imps*, *impt*, *praet*, *pred*, *winien*, *ger*, *inf*, *adj*] and pos_{dep} = *prep* **then** gf_{dep} := *comp*
Else gf_{dep} := *dep*

Example: wdychane$_{ppas}$ $\xrightarrow{comp\_ag}$ przez$_{prep}$ konsumentów (Eng. 'inhaled by consumers')

Rule 5. OBL-COMPAR

Rule: **If** gf_{dep} = OBL-COMPAR and $pos_{gov} \in$ [*adj*, *adv*] and $pos_{dep} \in$ [*prep*, *conj*]
then gf_{dep} := *comp*

Example: starsza$_{adj}$ $\xrightarrow{comp}$ od$_{praet}$ skały intruzywnej (Eng. 'older than the intrusive rock')

Rule 6. OBL-PART

Rule: **If** $gf_{dep} \in$ [OBL-PART]
If $lemma_{dep}$ = 'z' and $lemma_{gov} \in$ ['jeden', 'dwa', 'każdy', 'niektóry', 'kilka', 'niewiele', 'większość'] and $id_{gov} < id_{dep}$ **then** gf_{dep} := *mwe*
Elif $pos_{gov} \in$ [*fin*, *imps*, *inf*, *ppas*, *praet*, *pred*, *ger*, *pcon*, *pant*, *pact*, *num*, *adj*] and pos_{dep} = *prep* and $id_{gov} < id_{dep}$ **then** gf_{dep} := *comp*
Else gf_{dep} := *dep*

Example: jedno$_{adj}$ $\xrightarrow{comp}$ z$_{prep}$ dzieci (Eng. 'one of the children')

Rule 7. SUBJ

Rule: **If** gf_{dep} = SUBJ

If $pos_{gov} \in$ [*fin*, *impt*, *praet*, *winien*,*ppas*] or (pos_{gov} = *inf* and *gov* is part of a complex future form) and *gov* doesn't have a dependent labelled with *subj*

If $pos_{dep} \in$ [*subst*, *depr*, *ppron12*, *ppron3*, *ger*, *num*, *numcol*] and $num_{dep} = num_{gov}$ **then** $gf_{dep} :=$ *subj*

Elif pos_{dep} = *ign* **then** $gf_{dep} :=$ *subj*

Elif pos_{dep} = *adj* and $case_{dep}$ = 'nom' **then** $gf_{dep} :=$ *subj*

Elif *dep* is a NE **then** gf_{dep} = *subj* **then** $gf_{dep} :=$ *subj*

Elif $pos_{dep} \in$ [*adv*, *prep*] **then** $gf_{dep} :=$ *adjunct*

Else $gf_{dep} :=$ *dep*

Elif *gov* has a dependent labelled with *subj* and $pos_{gov} \in$ [*fin*, *impt*, *praet*, *winien*, *ppas*, *inf*, *ger*, *imps*, *pcon*, *pant*, *pact*]

If $pos_{dep} \in$ [*subst*, *depr*, *ppron12*, *ppron3*, *ger*, *num*, *numcol*, *conj*] or *dep* is an integer

If *gov* doesn't have a dependent labelled with *obj* **then** $gf_{dep} :=$ *obj*,

Else $gf_{dep} :=$ *obj_th*

Elif $pos_{dep} \in$ [*adv*, *prep*] **then** $gf_{dep} :=$ *comp*

Else $gf_{dep} :=$ *dep*

Else $gf_{dep} :=$ *dep*

Example: mózg$_{subst} \xleftarrow{subj}$ przetwarza$_{fin}$ informacje (Eng. 'the brain processes information')

Rule 8. XCOMP-PRED

Rule: **If** gf_{dep} = XCOMP-PRED

If $pos_{gov} \in$ [*prep*, *comp*] and $pos_{dep} \in$ [*subst*, *depr*, *ppron12*, *ppron3*, *ger*, *brev*, *num*, *numcol*, *siebie*, *conj*, *ign*] and $id_{gov} < id_{dep}$ **then** $gf_{dep} :=$ *comp*

Elif $pos_{gov} \in$ [*fin*, *imps*, *impt*, *praet*, *pred*, *winien*, *ger*, *inf*, *ppas*, *pact*, *pcon*]

If $pos_{dep} \in$ [*prep*, *adv*] **then** $gf_{dep} :=$ *comp*

Elif $pos_{dep} \in$ [*subst*, *depr*, *ppron12*, *ppron3*, *ger*, *num*, *numcol*, *siebie*, *ign*]

If $case_{dep} \in$ ['nom', 'acc', 'gen'] and *gov* doesn't have a dependent labelled with *subj* **then** $gf_{dep} :=$ *subj*

Else $gf_{dep} :=$ *dep*

Else $gf_{dep} :=$ *dep*

Else $gf_{dep} :=$ *dep*

Example: przy$_{prep} \xrightarrow{comp}$ niedoborach$_{subst}$ żywności (Eng. 'with food shortages')

Other Grammatical Functions

Rule 9. ADJUNCT

Rule: **If** gf_{dep} = ADJUNCT **then** gf_{dep} := *adjunct*

Example: europejska$_{adj}$ $\xleftarrow{adjunct}$ konferencja$_{subst}$ (Eng. 'European conference')

Rule 10. ADJUNCT-QT

Rule: **If** gf_{dep} = ADJUNCT-QT
If $pos_{dep} \in$ [*fin*, *imps*, *inf*, *ppas*, *praet*] **then** gf_{dep} := *adjunct_qt*
Elif $word_{dep} \in$ ['Zdaniem', 'zdaniem'] **then** gf_{dep} := *adjunct_qt*
Else gf_{dep} := *dep*

Example: Skacz$_{impt}$ $\xrightarrow{adjunct\_qt}$ krzyknął$_{praet}$ (Eng. 'Jump he shouted.')

Rule 11. MOD

Rule: **If** gf_{dep} = MOD **then** gf_{dep} := *adjunct*

Example: dobór$_{subst}$ $\xleftarrow{adjunct}$ zbóż$_{subst}$ (Eng. 'selection of cereals')

Rule 12. NAME-MOD

Rule: **If** gf_{dep} = NAME-MOD
If *dep* is NE and *gov* is NE **then** gf_{dep} := *ne*
Elif $pos_{dep} \in$ [*ign*, *subst*] and *gov* is NE **then** gf_{dep} := *app*
Elif pos_{gov} = *ign* and *dep* is NE **then** gf_{dep} := *ne*
Elif pos_{gov} = *ign* and pos_{dep} = *ign* and $id_{dep} < id_{gov}$ **then** gf_{dep} := *app*
Elif pos_{gov} = *ign* and pos_{dep} = *ign* and $id_{dep} > id_{gov}$ **then** gf_{dep} := *ne*
Elif pos_{gov} = *ign* and pos_{dep} = *subst* and $id_{dep} > id_{gov}$ **then** gf_{dep} := *app*
Elif pos_{gov} = *interj* and $lemma_{gov}$ = 'et' **then** gf_{dep} := *ne*
Elif pos_{gov} = *subst* and $pos_{dep} \in$ ['subst', 'ign'] **then** gf_{dep} := *adjunct*
Else gf_{dep} := *dep*

Example: profesor$_{subst}$ $\xleftarrow{app}$ Sackett$_{ign}$ (Eng. 'Professor Sackett')

Rule 13. PRECOORD

Rule: **If** gf_{dep} = PRECOORD
If $pos_{dep} = conj$ and $pos_{gov} = conj$ and $id_{dep} < id_{gov} - 1$ **then** $gf_{dep} := pre\_coord$
Elif $lemma_{dep}$ = 'jak' and $pos_{gov} = conj$ and $id_{gov} = id_{dep} + 1$
then $gf_{dep} := mwe$
Else $gf_{dep} := dep$

Example: zarówno wytrzymałe jak i lżejsze (Eng. 'both durable and lighter')
$\text{zarówno}_{conj} \xleftarrow{pre\_coord} \text{i}_{conj}$
$\text{jak}_{adv} \xleftarrow{mwe} \text{i}_{conj}$

Rule 14. TOPIC-CLEFT

Rule: **If** gf_{dep} = TOPIC-CLEFT
If $pos_{dep} \in$ [*fin*, *imps*, *ppas*, *praet*, *pred*, *winien*, *conj*] and $pos_{gov} \in$ [*subst*, *depr*, *ppron12*, *ppron3*, *ger*, *num*] and *dep* governs a complementiser
then $gf_{dep} := adjunct$
Elif $pos_{dep} = inf$ and $pos_{gov} \in$ [*subst*, *depr*, *ppron12*, *ppron3*, *ger*, *num*]
then $gf_{dep} := adjunct$
Else $gf_{dep} := dep$

Example: $\text{zachowanie}_{subst} \xrightarrow{adjunct}$ które nam się nie podoba (Eng. 'behaviour that we do not like' Note: the underlined token *podoba* constitutes the head of the relative clause and is a direct dependent of *zachowanie*.)

Appendix C

Correction Rules

Polish-Specific Linguistic Phenomena

Rule 1. Conditional clitic

Rule: If the conditional particle 'by' is not governed by a verb form and it is not tagged as a subordinating conjunction (*comp*) and a token adjacent to the conditional particle on the left side or one of dependents of the conditional particle is realised as a verb form, then the verb form is changed into the governor of the conditional particle. Furthermore, dependents of the conditional particle are annotated as dependents of the verb form (see Figure C.1[1]).

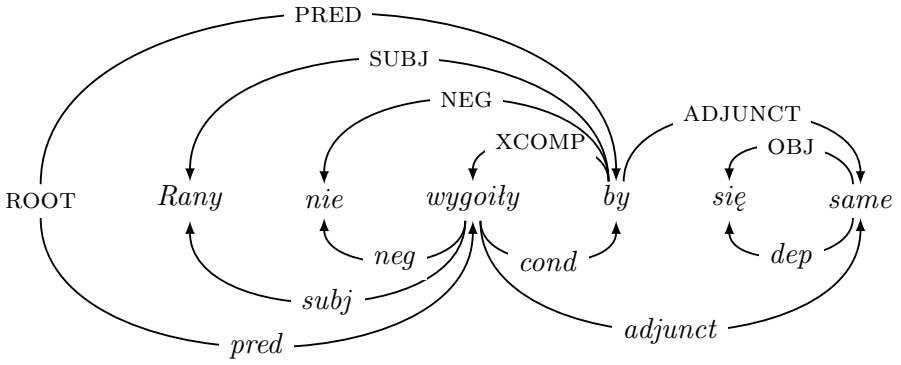


FIGURE C.1: An induced tree (top tree) and a modified tree (bottom tree) of the sentence *Rany nie wygoiłyby się same* (Eng. 'Wounds would not heal by themselves').

[1]The example sentence is annotated with two dependency structures: the tree above the sentence is automatically induced, the bottom tree is modified with correction rules. The same presentation schema is used in the following examples.

Rule 2. Imperative marker

Rule: If an imperative marker ('niech', 'niechaj', 'niechże' or 'niechajże') is not governed by a verb form and it immediately precedes a verb form, then it is annotated as the dependent of the verb form.

Rule 3. Mobile inflection

Rule: If a mobile inflection (lemma: 'być', part of speech: *aglt*) is adjacent to a verb form or a conditional clitic 'by' on the left side, then the left token is annotated as the governor of the mobile inflection. Otherwise, if there is a verb form immediately following the mobile inflection, it is its governor.

Rule 4. Reflexive marker

Rule: If a reflexive marker (lemma: *się*, part of speech: *qub*) is not governed by a verb form and a verb form is close to the reflexive marker on its left side (or possibly on its right side), then the verb form governs the reflexive marker. Furthermore, all dependents of the reflexive marker are rearranged into dependents of the governing verb form.

Divergent Annotation Conventions

Rule 5. Numeral complement

Rule: If a numeral (or a number) does not already govern a noun phrase and it is governed by a noun phrase (NP) and the numeral precedes the governing NP, then the numeral changes into the governor of the NP only if the numeral and the NP morphologically agree. The same applies to a number.

Rule 6. Negation marker

Rule: If the negation marker 'nie' is not already governed by a verb form or it is preceded by its governor, and the negation marker is followed by a verb form, then this verb form is rearranged into the governor of the negation marker. Furthermore, all dependents of the negation marker are annotated as dependents of the governing verb form.

Other Errors

Rule 7. Abbreviation marker

Rule: If an abbreviation marker (a full stop) is immediately preceded by an abbreviation (part of speech: *brev*) and this abbreviation does not already govern the abbreviation marker, then this abbreviation changes into the governor of the abbreviation marker. Furthermore, all dependents of the abbreviation marker are rearranged into dependents of the abbreviation.

Rule 8. Active adjectival participle

Rule: If an active adjectival participle (part of speech: *pact*) is immediately preceded by a substantive and the substantive and the participle have the same governor and they morphologically agree, then the participle is annotated as the dependent of the preceding substantive (see Figure C.2).

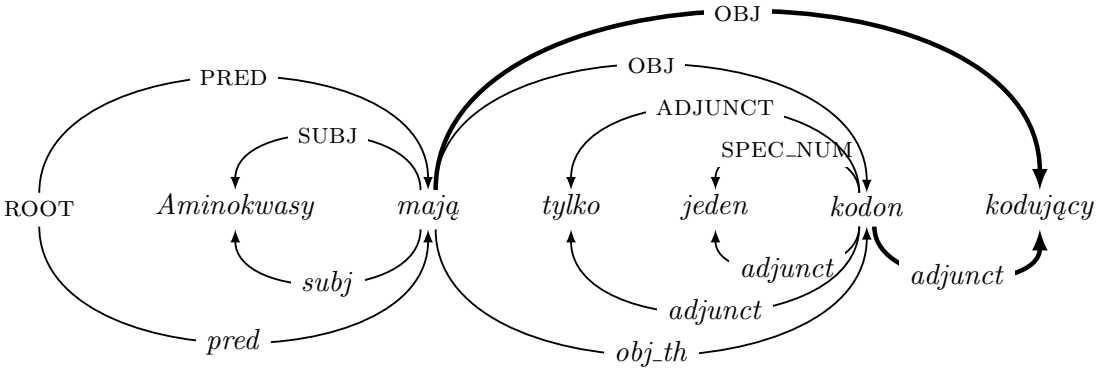


FIGURE C.2: An induced tree (top tree) and a modified tree (bottom tree) of the sentence *Aminokwasy mają tylko jeden kodon kodujący* (Eng. 'Amino acids have only one encoding codon').

Rule 9. Ad-adjectival adjective phrase

Rule: If an ad-adjectival adjective (part of speech: *adja*) is followed by a hyphen and then an adjective (part of speech: *adj*), then the hyphen is the dependent of the ad-adjectival adjective and the adjective is the dependent of the hyphen. A governor of the ad-adjectival phrase is looked for among these governors of participating tokens (the ad-adjectival adjective, the hyphen and the adjective) which do not coincide with considered tokens. If one of them is a substantive or a gerund that morphologically agrees with the adjective, it is selected as the governor of the ad-adjectival adjective (see Figure C.3).

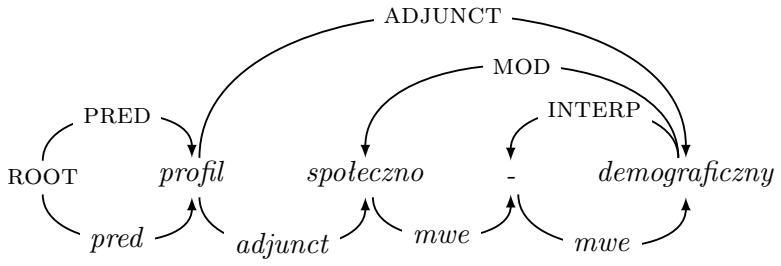


FIGURE C.3: An induced tree (top tree) and a modified tree (bottom tree) of the noun phrase *profil społeczno-demograficzny* (Eng. 'socio-demographic profile').

Rule 10. Apposition

Rule: If a token is not recognised by the tagger (part of speech: *ign*) and it is labelled with SUBJ (or OBJ) and it is immediately preceded by a substantive labelled with the same grammatical function, i.e., SUBJ (or OBJ), and the token and the substantive have the same governor, then the unrecognised token is annotated as the dependent of the preceding substantive and the relation is labelled with *app* (see Figure C.4).

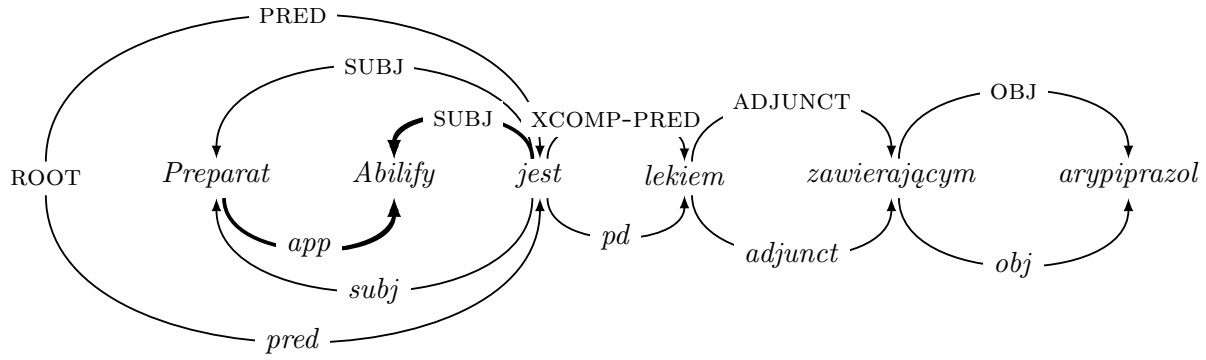


FIGURE C.4: An induced tree (top tree) and a modified tree (bottom tree) of the sentence *Preparat Abilify jest lekiem zawierającym arypiprazol.* (Eng. 'Abilify is a medicine containing apripiprazole.')

Rule 11. Auxiliary verb in complex future constructions

Rule: If a token is recognised as a verb form (parts of speech: *inf* or *praet*) and it is governed by an auxiliary verb form (lemma: 'być', part of speech: *bedzie*), then the verb form governs the auxiliary. Furthermore, possible dependents of the auxiliary are governed by the verb form.

Rule 12. Auxiliary verb in passive constructions

Rule: If a passive adjective participle (part of speech: *ppas*) marked for the nominative case is governed by an auxiliary verb form (possible lemmata: 'być', 'zostać', 'zostawać') which does not function as a predicative verb, then the participle changes into the governor of the auxiliary verb. Furthermore, possible dependents of the auxiliary are governed by the participle.

Rule 13. Comparative phrase (1)

Rule: If the token 'jak'[2] is immediately preceded by a declined form of the adjective 'taki' and both tokens have the same governor, then the token 'jak' is governed by the adjective. Furthermore, all dependents of the adjective with indices greater than the index of 'jak' are rearranged into dependents of 'jak'.

[2]The token 'jak' may be recognised as a preposition by the tagger. Since it is not always the case, the part of speech is not considered by the correction rule.

Rule 14. Comparative phrase (2)

Rule: If the token ‘niż’ is immediately preceded by a token marked for the comparative degree, then the preceding token governs the preposition ‘niż’.

Rule 15. Complementiser

Rule: If a verb form indirectly follows its verbal governor and the relation between them is labelled with COMP, XCOMP, XCOMP-PRED, OBJ or OBL, and the verb form does not already govern a complementiser and a subordinating conjunction (part of speech: *comp*) is found between the verb form and the verbal governor, then the subordinating conjunction is governed by the verb form, if it is realised as ‘aby’, ‘by’, ‘żeby’, ‘że’, ‘iż’, ‘ażeby’ or ‘iżby. Furthermore, all dependents of the complementiser are governed by the verb form. Otherwise, the subordinating conjunction is governed by the verbal governor and the verb form depends on the subordinating conjunction.

Rule 16. Conjunct dependent (1)

Rule: If a conjunct governed by a coordinating element (parts of speech: *conj*, *interp*) precedes this coordinating element, then the dependents of the conjunct with indices greater than the index of the coordinating element are annotated as dependents of the coordinating element.

Rule 17. Conjunct dependent (2)

Rule: If a conjunct governed by a coordinating element (parts of speech: *conj*, *interp*) follows this coordinating element, then the dependents of the conjunct with indices lower than the index of the coordinating element are annotated as dependents of the coordinating element (see Figure C.5).

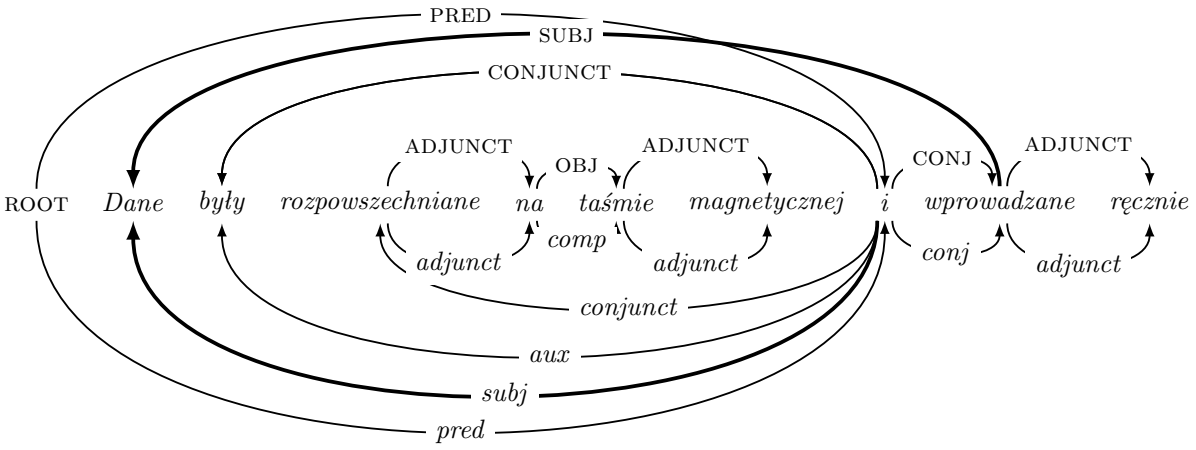


FIGURE C.5: An induced tree (top tree) and a modified tree (bottom tree) of the sentence *Dane były rozpowszechniane na taśmie magnetycznej i wprowadzane ręcznie.* (Eng. ‘The data was distributed on a magnetic tape and inserted manually.’).

Rule 18. Complement of *Mieć*

Rule: If a verb form of 'mieć' (Eng. 'to have') does not have any dependents and it is governed by a substantive, then the verb form is governed by the original governor of the substantive and the substantive, in turn, is governed by the verb form.

Rule 19. Modifier (adverbial) of an adjective/adverb

Rule: If an adverb (lemmata: 'tak', 'bardzo', 'mało', 'dużo', 'coraz') is immediately followed by an adjective or an adverb, and the adverb and the following token have the same governor, then the following token governs the current adverb.

Rule 20. Modifier (adjectival) of a substantive (1)

Rule: If a substantive is governed by an adjective and the substantive and the adjective morphologically agree (in terms of number and case) and the adjective is not already governed by another substantive and the distance between the substantive and the adjective is smaller than three tokens, then the substantive governs the adjective, if the adjective is not governed by a predicative verb form (lemma: 'być', 'zostać' or 'zostawać'). Otherwise, it may be a predicative construction in which the substantive governs the adjective only if the substantive and the adjective either precede or follow the predicative verb form and the substantive is adjacent to the adjective.
Furthermore, if the adjective precedes the substantive, then the dependents of the adjective with indices greater then the index of the substantive are governed by the substantive. Otherwise, if the adjective follows the substantive, then the dependents of the adjective with indices lower than the index of the substantive are governed by the substantive (see Figure C.6).

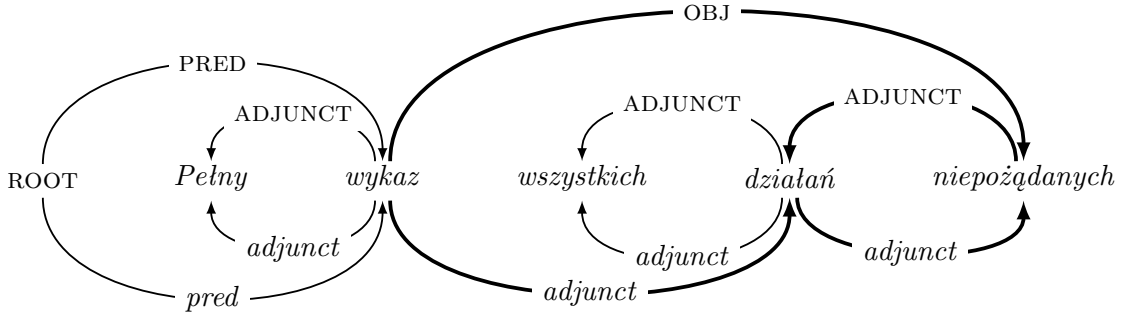


FIGURE C.6: An induced tree (top tree) and a modified tree (bottom tree) of the noun phrase *Pełny wykaz wszystkich działań niepożądanych* (Eng. 'A complete list of all side effects').

Rule 21. Modifier (adjectival) of a substantive (2)

Rule: If an adjective is adjacent to a substantive and the adjective and the substantive have the same governor and they morphologically agree, then the substantive governs

the adjective. Furthermore, if the adjective precedes the substantive, then the dependents of the adjective with indices greater than the index of the substantive are governed by the substantive. Otherwise, if the adjective follows the substantive, then the dependents of the adjective with indices lower than the index of the substantive be governed by the substantive.

Rule 22. Modifier (nominal) of a substantive (1)

Rule: If a substantive is governed by a substantive marked for the genitive case (henceforth, the genitive substantive) and the substantive precedes the genitive substantive and the distance between these substantives is smaller than four tokens, then the substantive governs the genitive substantive, if the governor of the genitive substantive is not represented as a predicative verb form (lemma: 'być', 'zostać' or 'zostawać'). Otherwise, it may be a predicative construction in which the substantive governs the genitive substantive only if both substantives either precede or follow the predicative verb form and the substantive is adjacent to the genitive substantive. Furthermore, if the genitive substantive governs some dependents and indices of these dependents are lower than the the index of the substantive, then these dependents are governed by the substantive.

Rule 23. Modifier (nominal) of a substantive (2)

Rule: If a substantive marked for the genitive case (henceforth, the genitive substantive) and both substantives are adjacent and they have the same governor, then the substantive governs the genitive substantive. Furthermore, if the genitive substantive precedes the substantive, then the dependents of the genitive substantive with indices greater than the index of the substantive are governed by the substantive. Otherwise, if the genitive substantive follows the substantive, then the dependents of the genitive substantive with indices lower than the index of the substantive are governed by the substantive.

Rule 24. Partitive phrase

Rule: If a preposition annotated with PART precedes a token labelled with OBL-PART, and both the preposition and the other token have the same governor, then the other token is annotated as a dependent of the preposition (see Figure C.7).

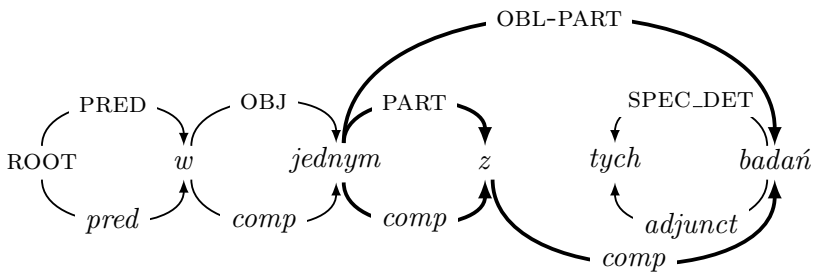


FIGURE C.7: An induced tree (top tree) and a modified tree (bottom tree) of the preposition phrase *w jednym z tych badań* (Eng. 'in one of these studies').

Rule 25. Post-prepositional adjective

Rule: If a post-prepositional adjective (part of speech: *adjp*) is not governed by a preposition, then the closest preposition on the left side is changed into the governor of the post-prepositional adjective.

Rule 26. Predicative construction

Rule: If a verb form of 'być' is governed by an adjective following the verb form or by a noun phrase marked for the instrumental case, then the governing token changes into the dependent of the verb form.

Rule 27. Prepositional complement

Rule: If a preposition does not have any dependent:

- If the preposition and the immediately following token are labelled with the function PART and they have the same governor and the preposition precedes its governor, then the preposition depends on the governor of its original governor, the subsequent token is governed by the preposition and the original governor of the preposition is governed by the following token (see Figure C.8).
- Elif the preposition is labelled with the function PFORM and its governor also governs a token labelled with OBL-AG or OBJ, then the preposition governs the other token (see Figure C.9).
- Elif the preposition is governed by a following noun phrase marked for the case required by the preposition, then the preposition is governed by the governor of the noun phrase and the noun phrase depends on the preposition.
- Else, if the token adjacent to the preposition on the right side fulfils properties of a preposition complement (i.e., it is a noun phrase marked for the case required by the preposition), it is annotated as the preposition complement.

Else, if a preposition complement precedes a preposition, then another complement of the preposition is searched for : if a token adjacent to the preposition on the right fulfils properties of a preposition complement (i.e., it is a noun phrase marked for the case required by the preposition), it is annotated as the preposition complement.
Otherwise, if a preposition has more than one dependent, then the best of them[3] is selected as the preposition complement. Otherwise, a token adjacent to the right side of the preposition is taken into account as the preposition's complement.

[3]The complement of the preposition should be adjacent to the preposition on its right side and should fulfil the case requirement imposed by the preposition.

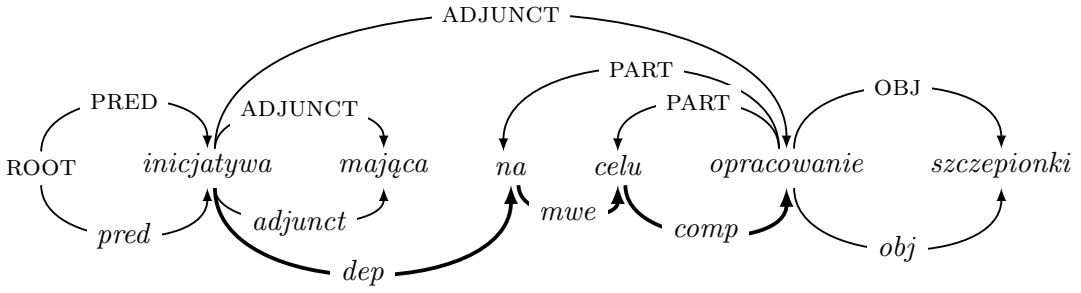


FIGURE C.8: An induced tree (top tree) and a modified tree (bottom tree) of the noun phrase *inicjatywa mająca na celu opracowanie szczepionki* (Eng. 'initiative aimed at developing a vaccine').

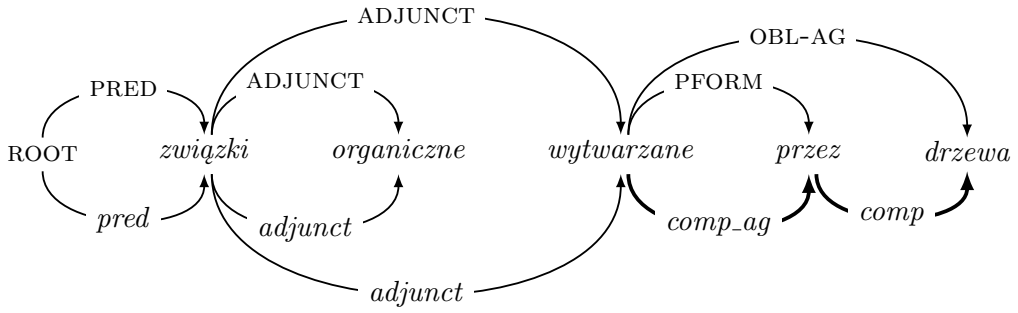


FIGURE C.9: An induced tree (top tree) and a modified tree (bottom tree) of the noun phrase *związki organiczne wytwarzane przez drzewa* (Eng. 'organic compounds produced by trees').

Rule 28. Punctuation dependent

Rule: If a punctuation mark not functioning as a coordinating element governs some dependents, these dependents are annotated as dependents of the governor of the punctuation mark.

Rule 29. Root dependent (1)

Rule: If a token (parts of speech: *adv*, *adjp*, *qub*, *ign* or *interp*) is governed by the ROOT node and the token does not function as a coordinating element and it governs a verb or quasi-verb form (but not a 'być'-form), then this verb form or quasi-verb form is annotated as the head of the token and all other dependents of the token are rearranged into dependents of the verb form (see Figure C.10).

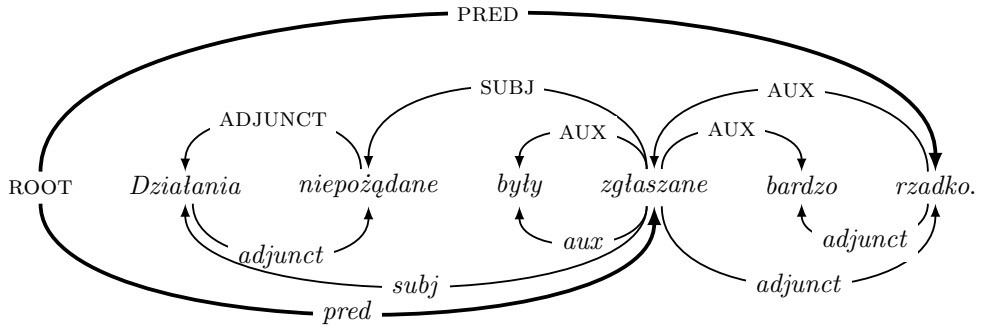


FIGURE C.10: An induced tree (top tree) and a modified tree (bottom tree) of the sentence *Działania niepożądane były zgłaszane bardzo rzadko* (Eng. 'Adverse reactions have been reported very rarely.').

Rule 30. Root dependent (2)

Rule: If a token (parts of speech: *adj*, *depr*, *ger*, *num*, *ppron12*, *ppron3*, *prep* or *subst*) is governed by the ROOT node and the token governs a verb form (parts of speech: *fin* or *praet* but not 'być'-form), then this verb form changes into the governor of the token. Furthermore, if the index of the verb form is lower than the index of the token, the dependents of the token with indices lower than the index of the verb form are rearranged into dependents of the verb form. If the verb index, in turn, is greater than the index of the token, dependents of the token with indices greater than the verb index are rearranged into verb dependents.

Rule 31. Simple question

Rule: If a verb form of 'być' governs two tokens 'co' and 'to', and 'co' precedes 'to', then the token 'to' is annotated as the governor of both the verb form and the token 'co', and the dependents of the verb form and the token 'co' are rearranged into dependents of the token 'to' (see Figure C.11).

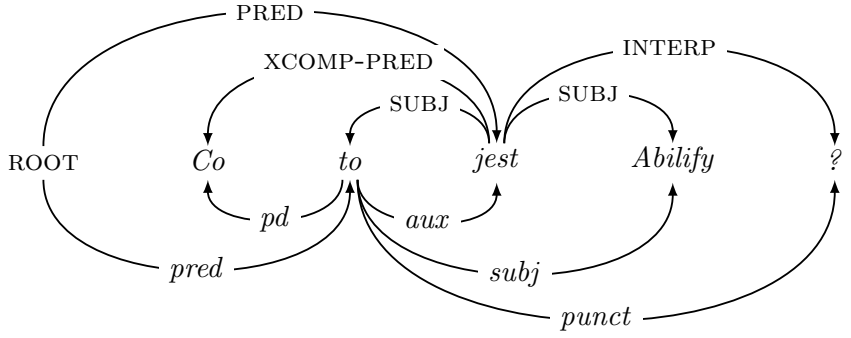


FIGURE C.11: An induced tree (top tree) and a modified tree (bottom tree) of the interrogative question *Co to jest Abilify?* (Eng. 'What is Abilify?').

Appendix D

K-best MST Algorithm

D.1 Pseudocode

best(G, Y, Z) *# 1-best MST algorithm*
$G = (V_G, E_G)$
$E_G := (E_G \cup Y) - Z$
σ_G and τ_G are the incidence functions, and ω_G is the weighting function of G
$B := \emptyset$
$\beta := \emptyset$
$\mathscr{C} := (V_{\mathscr{C}}, E_{\mathscr{C}})$, where $V_{\mathscr{C}} = V_G$ and $E_{\mathscr{C}} = \emptyset$

While there exists an unvisited vertex $v \in V_G$, for $v \neq v_0$ *# collapsing phase*
    Find the best incoming edge b for v, so that $\tau_G(b) = v$ and $\omega_G(b)$ is maximal
    $B := B \cup \{b\}$
    $\beta(v) := b$
    If B contains a cycle C
        $G := G_C$
        Colapse C into a new vertex u in G_C
        Add the new vertex u to $\mathscr{C}$
        For each vertex w in the cycle C, add an edge from u to w to $E_{\mathscr{C}}$
        $B := B - C$

While $\mathscr{C}$ contains non-isolated vertices *# expanding phase*
    Identify in $\mathscr{C}$ a path with vertices $u_0, u_1, ..., u_k$, where u_0 is any non-isolated root of $\mathscr{C}$ and $u_k = \tau(\beta(u_0))$
    For $h \in \{0, ..., k-1\}$
        $\beta(u_{h+1}) := \beta(u_h)$
        Remove from $\mathscr{C}$ the vertex u_h and all edges directed out of u_h
Return $A = \{\beta(v) | v$ is a vertex of G but not the root$\}$

next(G, A, Y, Z) *# next best MST algorithm*
$G = (V_G, E_G)$
$E_G := (E_G \cup Y) - Z$
$B := \emptyset$
$d := +\infty$
σ_G and τ_G are the incidence functions, and ω_G is the weighting function of G

While there exists an unvisited vertex $v \in V_G$, for $v \neq v_0$
  Find the best incoming edge b for v, so that $\tau_G(b) = v$ and $\omega_G(b)$ is maximal
  $B := B \cup \{b\}$
  If $b \in A - Y$
    If there is an edge f' which is second best to b
      $f := f'$
    Else $f :=$ dummy edge $(\omega_G(f) = -\infty)$
    If $\omega_G(b) - \omega_G(f) < d$
      $e := b$
      $d := \omega_G(b) - \omega_G(f)$
  If B contains a cycle C of G, resolve it as in the **best** algorithm
Return the edge e and the difference score d

rank(G, k) *# k-best MSDTs selection*
$G = (V_G, E_G)$
k is a number of the best dependency trees to select
$\mathscr{P} := \emptyset$ *# a list to represent MSTs*
$\mathscr{B} := \emptyset$ *# a priority queue to store k-best MSTs*
$A_1 :=$ **best**$(G, \emptyset, \emptyset)$

If A_1 is a proper dependency tree
  Add A_1 to $\mathscr{B}$
  $e, d :=$ **next**$(G, A_1, \emptyset, \emptyset)$
  Add a tuple $(\omega(A_1) - d, e, A_1, \emptyset, \emptyset)$ to $\mathscr{P}$
  While the number of trees in $\mathscr{B}$ is less than k
    Remove from $\mathscr{P}$ the tuple (w, e, A, Y, Z) for which w is maximal
    If $w = -\infty$; return "all spanning trees of G have been output"
    $Y' := Y \cup \{e\}$; $Z' := Z \cup \{e\}$
    $A_j :=$ **best**(G, Y, Z')
    If A_j is a proper dependency tree
      Add A_j to $\mathscr{B}$
      $e', d' :=$ **next**(G, A, Y', Z)
      Add a tuple $(\omega(A) - d', e', A, Y', Z)$ to $\mathscr{P}$
      $e'', d'' :=$ **next** (G, A_j, Y, Z')
      Add a tuple $(\omega(A_j) - d'', e'', A_j, Y, Z')$ to $\mathscr{P}$
    Else $e', d' :=$ **next**(G, A, Y, Z')
  Return $\mathscr{B}$
Else return None

D.2 Explanation

The k-best MST algorithm by Camerini et al. (1980) finds k-best MSTs in a weighted directed graph $G = (V_G, E_G)$ with the set of vertices $V_G = \{v_0, ..., v_n\}$, where v_0 is the root node, and the set of edges $E_G = \{e_1, ..., e_m\}$. All found k-best MSTs are rooted at the same vertex v_0. The algorithm has the linearithmic time complexity of $O(km \log n)$.

First, the algorithm finds the maximum spanning tree A in the weighted directed graph G using the function **best**(G, Y, Z). The parameter Y denotes a set of arcs from G which are required to be in the solution and are not directed to v_0. Z corresponds to a set of arcs which cannot be part of the solution. The algorithm requires a weighting function $\omega : E_G \to \mathbb{R}$ and two incidence functions, σ and τ. The σ function returns a vertex (governor) out of which a particular edge is directed. The τ function returns a vertex (dependent) into which a particular edge is directed. For example, if we have an edge e between two vertices u and v, the edge direction from u to v is determined by $\sigma(e) = u$ and $\tau(e) = v$.

The function **best** is divided into two phases – the *collapsing phase* and the *expanding phase*. In the collapsing phase, the current graph G is iteratively collapsed and the current best graph B is updated until there is only one unvisited vertex v_0 in V_G. The best incoming edge b found for each unvisited vertex v is added to the current best graph B and is stored as the value of v in the mapping β, which is used to store the best incoming edges for each visited vertex.

It is then checked if there is a cycle in the current best graph B after inserting the edge b into B. If the cycle C is found in B, all nodes of C are collapsed into a new vertex u of the graph G_C. The collapsing procedure is as follows. For each vertex w in the cycle C and for each vertex v in $G - C$ being a parent of w, an edge from v to u is added to G_C. The weight of the new edge is $\omega_G(v, w) - \omega_G(w', w)$ where w' immediately precedes w in C. For each vertex w in C and for each vertex v in $G - C$ being w's child, an edge from u to v is added to G_C. The weight of the new edge is $\omega_G(u, v)$. Then, vertices from C are removed from G_C.

The new vertex u corresponding to C is added to the forest $\mathscr{C}$. The forest $\mathscr{C} = (V_{\mathscr{C}}, E_{\mathscr{C}})$, with $V_{\mathscr{C}} = V_G$ and $E_{\mathscr{C}} = \emptyset$, is used for keeping track of iterative collapsing of G (i.e., it stores a history of cycle collapses). Vertices of the cycle C are attached as children of the new vertex u in $\mathscr{C}$. Arcs of the cycle C are removed from B.

If all vertices of V_G and all newly created vertices are visited, the algorithm proceeds to the *expanding phase*, in which cycles stored in the forest $\mathscr{C}$ are resolved (or *expanded*), β and $\mathscr{C}$ are updated and the maximum spanning tree A of G is recovered from B.

The expanding procedure is as follows. While the forest $\mathscr{C}$ contains non-isolated vertices,[1] a path $u_0, u_1, ..., u_k$ of vertices not present in the original graph G is identified in $\mathscr{C}$. The path starts with u_0 which is a non-isolated ROOT of $\mathscr{C}$, i.e., it does not have any parents, but it has some children. The vertex u_k is a direct or indirect dependent of u_0, i.e., $u_k = \tau(\beta(u_0))$. There are two prerequisites on considered vertices. First, their number may not exceed $l - 1$, for l being the number of leaves in $\mathscr{C}$. Second, they must have at least two children. For each vertex in this path, assuming that the vertex is not a leaf (i.e., it was not originally in V_G), the mapping β for storing best incoming edges and the forest $\mathscr{C}$ for storing a history of cycle collapses are updated. For u_0, $\beta(u_1)$ is assigned the best edge stored in β coming into the node u_0 (i.e., $\beta(u_1) = \beta(u_0)$) and u_0 and all edges directed out of v_0 are removed from $\mathscr{C}$. For u_{h-1}, in turn, $\beta(u_h)$ is assigned $\beta(u_{h-1})$ and u_h and all edges directed out of u_h are removed from $\mathscr{C}$. Hence, $\beta(u_{h-1})$ stores the edge resolving the cycle C. The algorithm outputs the maximum spanning tree A containing vertices from G and the best incoming edges for these vertices stored in β.

The function **next**(G, A, Y, Z) outputs an arc e such that replacing e with the next best arc in the maximum spanning tree A will result in the next best MST. The **next** function is similar to **best**. However, since the maximum spanning tree A is already known, there is no need for updating the forest $\mathscr{C}$ and the mapping β nor for the expanding phase. Instead, it is searched for the arc f which is second best to the current best arc b. If there is no second best arc, a dummy edge with the weight $\omega(f) = -\infty$ is returned. Taking into account the next best edges, the function **next** outputs the edge e (corresponding to the edge b in A that should be replaced by f) with the smallest difference score $d = \omega_G(b) - \omega_G(f)$.

A list of k-best maximum spanning trees of the graph G is computed with the ranking algorithm **rank**. This function is slightly modified in relation to the original function **rank** (see Camerini et al., 1980, p. 107). There are some additional conditions that force adding only well-formed maximum spanning trees to the priority queue $\mathscr{B}$. Apart from the priority queue $\mathscr{B}$ for storing k-best maximum spanning dependency trees in the descending order of their weights, the algorithm uses a list $\mathscr{P}$ to store partitions of next best MSDTs of the graph G. A partition of a next best MSDT added to $\mathscr{P}$ is represented as a tuple of the form (w, e, A, Y, Z), where w is the weight of the next best MSDT, e is the edge of A used to identify the next best MSDT, A is the current best MSDT, Y is the set of edges required to be in the solution, and Z is a set of edges which cannot be part of the solution.

The **rank** function starts with selecting the maximum spanning tree. If the selected MST is not a well-formed dependency tree, next best trees are not selected any more from the graph G. If the MST is a well-formed dependency tree, it is added as the first tree

[1] An *isolated* vertex has no outgoing or incoming edges. Hence, a non-isolated vertex has parents or/and children.

to the priority queue $\mathscr{B}$. Subsequently, the algorithm searches for $k-1$ best maximum spanning dependency trees. After identifying the edge e with the function **next**, the next best MST A_j is identified with the function $\textbf{best}(G, Y, Z \cup \{e\})$. If A_j is a well-formed dependency tree, it is added to the priority queue $\mathscr{B}$ of k-best MSDTs. Subsequently, the algorithm searches for two candidate edges to be replaced in the next best MSDT. The first one e' is found with the function $\textbf{next}(G, A, Y \cup \{e\}, Z)$ and the tuple $(\omega(A) - d', e', A, Y \cup \{e\}, Z)$ is added to $\mathscr{P}$. Another edge e'' is found with $\textbf{next}(G, A_j, Y, Z \cup \{e\})$ and the tuple $(\omega(A_j) - d', e', A_j, Y, Z \cup \{e\})$ is added to $\mathscr{P}$. The tuple with the highest w is selected from $\mathscr{P}$ and the algorithm continues selecting next best trees until there are k-best trees in $\mathscr{B}$ or a next tree cannot be found. If A_j is not a well-formed dependency tree, it is not inserted into the priority queue $\mathscr{B}$. Instead, it is searched for another candidate edge e' to be replaced in the next best MSDT. The algorithm returns a list $\mathscr{B}$ with k-best maximum spanning dependency trees.

List of Abbreviations

abbrev_punct	Abbreviation marker
acc	Accusative case
adja	Ad-adjectival adjective (part of speech)
adj	Adjective (part of speech)
adjc	Predicative adjective (part of speech)
adjp	Post-prepositional adjective (part of speech)
adjunct_qt	Quotation adjunct
adv	Adverb (part of speech)
AER	Alignment Error Rate
aglt	Mobile inflection
aglt	Agglutinative *być* (part of speech)
AP	Adjective phrase
app	Apposition
Atr	Attribute in noun phrases in PDT
AtvV	Verbal attribute / complement
aux	Auxiliary verb
BC	Before Christ
bedzie	Future *być* (part of speech)
brev	Abbreviation (part of speech)
CDT	Copenhagen Dependency Treebank
com	Comparative degree
comp	Complement
comp_ag	Agentive complement in passive
comp	Subordinating conjunction (part of speech)
comp_fin	Clausal complement
comp_inf	Infinitival clausal complement
complm	Complementizer
cond	Conditional clitic
conj	Coordinating conjunction (part of speech)
conjunct	Coordinated conjunct
CoNLL	Conference on Computational Natural Language Learning
coord	Coordinating conjunction

coord_punct	Punctuation conjunction
CP	Subordinate clause
CTB	Penn Chinese Treebank
dat	Dative case
DCA	Direct Correspondence Assumption
DDT	Danish Dependency Treebank
DEPBANK	PARC 700 Dependency Bank
EM	Expectation-Maximisation algorithm
f	Feminine gender
fin	Non-past form (part of speech)
gdfa	`Grow-diag-final-and` symmetrisation heuristic
GEN	Grammatical gender
gen	Genitive case
ger	Gerund (part of speech)
GFJP	Gramatyka formalna języka polskiego
HMM	Hidden Markov model
HPSG	Head-driven Phrase Structure Grammar
IBM	International Business Machines Corporation
imp	Imperative marker
imps	Impersonal (part of speech)
impt	Imperative (part of speech)
inf	Infinitive (part of speech)
inst	Instrumental case
interp	Punctuation (part of speech)
IP	Finite clause
item	Enumeration marker
LAS	Labelled attachment score
LFG	Lexical Functional Grammar
loc	Locative case
m	Masculine gender
MST	Maximum/minimum spanning tree
MTT	Meaning-Text Theory
mwe	Multiword expression
ne	Named entity
neg	Negation marker
NLP	Natural language processing
NLTK	Natural Language Toolkit
n	Neuter gender
nom	Nominative case
NP	Noun phrase
NUM	Grammatical number

num	Numeral (part of speech)
obj	Direct object
Obj	Object in PDT
obj_th	Thematically restricted object
pact	Active adjectival participle (part of speech)
pant	Anterior adverbial participle (part of speech)
pcon	Contemporary adverbial participle (part of speech)
pd	Predicative complement
PDT	Prague Dependency Treebank
PERS	Person
pl	Plural number
POS	Part of speech
pos	Positive degree
ppas	Passive adjectival participle (part of speech)
pp	Percentage point
ppron12	Non-third person pronoun (part of speech)
ppron3	Third person pronoun (part of speech)
praet	L-participle (part of speech)
pre_coord	Pre-conjunction
pred	Sentence predicate or nominal predicate
pred	Predicative (part of speech)
prep	Preposition (part of speech)
pri	First person
punct	Punctuation mark
QG	Quasi-synchronous grammar
qub	Particle-adverb (part of speech)
refl	Reflexive marker
Sb	Subject in PDT
sec	Second person
sg	Singular number
siebie	Pronoun *siebie* (part of speech)
subj	Subject
subst	Substantive/Noun (part of speech)
sup	Superlative degree
ter	Third person
UAS	Unlabelled attachment score
voc	Vocative case
VP	Verb phrase
XLE	Xerox Linguistic Environment

Bibliography

Abeillé, A. and Rambow, O., editors (2000). *Tree Adjoining Grammars. Formalisms, Linguistic Analysis and Processing*. CSLI Lecture Notes Series. CSLI Publications, Stanford, CA.

Acedański, S. (2010). A Morphosyntactic Brill Tagger for Inflectional Languages. In *Advances in Natural Language Processing*, volume 6233 of *Lecture Notes in Computer Science*, pages 3–14. Springer-Verlag.

Ágel, V. and Fischer, K. (2010). Dependency Grammar and Valency Theory. In Heine, B. and Narrog, H., editors, *The Oxford Handbook of Linguistic Analysis*, pages 223–255. Oxford University Press, Oxford.

Attardi, G. (2006). Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-X)*, pages 166–170. Association for Computational Linguistics.

Attardi, G. and Ciaramita, M. (2007). Tree Revision Learning for Dependency Parsing. In Sidner, C. L., Schultz, T., Stone, M., and Zhai, C. X., editors, *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 388–395. The Association for Computational Linguistics.

Bang-Jensen, J. and Gutin, G. Z. (2009). *Digraphs. Theory, Algorithms and Applications*. Springer Monographs in Mathematics. Springer-Verlag, London.

Berovic, D., Agic, Z., and Tadić, M. (2012). Croatian Dependency Treebank: Recent Development and Initial Experiments. In Calzolari, N., Choukri, K., Declerck, T., Doğan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation*, LREC'12. European Language Resources Association (ELRA).

Bień, J. S. (1997). Komputerowa weryfikacja formalnej gramatyki Świdzińskiego. *Biuletyn Polskiego Towarzystwa Językoznawczego*, LII:147–164.

Bień, J. S. (2007). Innovative use of parameters in DCG-like logic grammars. In Vetulani, Z., editor, *Proceedings of the 3th Language & Technology Conference*, pages 285–289.

Bień, J. S., Szafran, K., and Woliński, M. (2001). Experimental Parsers of Polish. In Zybatow, G., Junghanns, U., Mehlhorn, G., and Szucsich, L., editors, *Current Issues in Formal Slavic Linguistics*, volume 5 of *Linguistik International*, pages 185–190, Frankfurt am Main. Peter Lang.

Bies, A., Ferguson, M., Katz, K., and MacIntyre, R. (1995). *Bracketing Guidelines for Treebank II Style Penn Treebank Project.*

Bird, S., Loper, E., and Klein, E. (2009). *Natural Language Processing with Python.* O'Reilly Media Inc.

Boguslavsky, I., Chardin, I., Grigorieva, S., Grigoriev, N., Iomdin, L., Kreidlin, L., and Frid, N. (2002). Development of a Dependency Treebank for Russian and its possible Applications in NLP. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation, Las Palmas, Gran Canaria*, pages 852–856.

Böhmová, A., Hajič, J., Jakočová, E., and Hladká, B. (2003). The Prague Dependency Treebank: A Three-Level Annotation Scenario. In Abeillé, A., editor, *Treebanks. Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*, pages 103–128. Kluwer Academic Publishers, Dordrecht/Boston/London.

Bohnet, B. (2003). Mapping Phrase Structures to Dependency Structures in the Case of Free Word Order Languages. In *Proceedings of the First International Conference on Meaning-Text Theory*, MTT 2003, pages 239–249.

Bohnet, B. (2009). Efficient Parsing of Syntactic and Semantic Dependency Structures. In *Proceedings of the 13rd Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 67–72.

Bohnet, B. (2010). Very High Accuracy and Fast Dependency Parsing is not a Contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING 2010, pages 89–97.

Bojar, O. and Hajič, J. (2008). Phrase-Based and Deep Syntactic English-to-Czech Statistical Machine Translation. In *The Third Workshop on Statistical Machine Translation*, pages 143–146.

Bouma, G., Kuhn, J., Schrader, B., and Spreyer, K. (2008). Parallel LFG Grammars on Parallel Corpora: A Base for Practical Triangulation. In *Proceedings of the LFG08 Conference*, pages 169–189.

Brants, S., Dipper, S., Hansen, S., Lezius, W., and Smith, G. (2002). The TIGER treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, pages 24–41.

Bresnan, J. (2001). *Lexical-Functional Syntax.* Blackwell, Oxford.

Brown, P. F., Della Pietra, V. J., Della Pietra, S. A., and Mercer, R. L. (1993). The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19(2):263–311.

Buch-Kromann, M. (2006). *Discontinuous Grammar. A model of human parsing and language acquisition*. Dr.ling.merc. dissertation, Copenhagen Business School (Handslshøjskolen), København.

Buch-Kromann, M., Gylling-Jørgensen, Morten andJelsbech Knudsen, L., Korzen, I., and Høeg Müller, H. (2010). The inventory of linguistic relations used in the Copenhagen Dependency Treebanks. The CDT manual, Center for Research and Innovation in Translation and Translation Technology, Copenhagen Business School.

Buch-Kromann, M. and Korzen, I. (2010). The unified annotation of syntax and discourse in the Copenhagen Dependency Treebanks. In *Proceedings of the Fourth Linguistic Annotation Workshop*, LAW IV '10, pages 127–131. Association for Computational Linguistics.

Buchholz, S. and Marsi, E. (2006). CoNLL-X shared task on Multilingual Dependency Parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164.

Butt, M., Dyvik, H., King, T. H., Masuichi, H., and Christian, R. (2002). The Parallel Grammar Project. In *Proceedings of the COLING 2002 Workshop on Grammar Engineering and Evaluation*, pages 1–7.

Camerini, P. M., Fratta, L., and Maffioli, F. (1980). The K Best Spanning Arborescences of a Network. *Networks*, 10:91–110.

Carreras, X. (2007). Experiments with a Higher-Order Projective Dependency Parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 957–961.

Chanev, A., Simov, K., Osenova, P., and Marinov, S. (2006). Dependency conversion and parsing of the BulTreeBank. In *Proceedings of the LREC workshop Merging and Layering Linguistic Information*, pages 16–23.

Chang, C.-C. and Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm.

Chen, S. F. (1993). Aligning Sentence in Bilingual Corpora Using Lexical Information. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 9–16.

Choi, J. D. and Palmer, M. (2010). Robust Constituency-to-Dependency Conversion for English. In *Proceedings of the 9th International Workshop on Treebanks and Linguistic Theories*, volume 9 of *Nealt Proceedings Series*, pages 55–66. Northern European Association for Language Technology (NEALT).

Chu, Y. J. and Liu, T. H. (1965). On the Shortest Arborescence of a Directed Graph. *Science Sinica*, 14:1396–1400.

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania.

Collins, M. (2003). Head-Driven Statistical Models for Natural Language Parsing. *Computational Linguistics*, 29(4):589–637.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, 7:551–585.

Crouch, D., Dalrymple, M., Kaplan, R., King, T., Maxwell, J., and Newman, P. (2011). *XLE Documentation*. Palo Alto Research Center (PARC), Palo Alto, CA.

Cucerzan, S. and Yarowsky, D. (2002). Bootstrapping a Multilingual Part-of-speech Tagger in One Person-day. In *Proceedings of the 6th Conference on Natural Language Learning*, COLING'02, pages 1–7.

Dalrymple, M. (2001). *Lexical-Functional Grammar. Syntax and Semantics*, volume 34. Academic Press.

Das, D. and Petrov, S. (2011). Unsupervised Part-of-Speech Tagging with Bilingual Graph-Based Projections. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, volume 1 of *HTL'11*, pages 600–609.

de Marneffe, M.-C., MacCartney, B., and Manning, C. D. (2006). Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of the Fifth International Conference of Language Resources and Evaluation*, LREC'06, pages 449–454.

de Marneffe, M.-C. and Manning, C. D. (2008a). *Stanford typed dependencies manual*. Stanford University.

de Marneffe, M.-C. and Manning, C. D. (2008b). The Stanford typed dependencies representation. In *Proceedings of the Workshop on Cross-framework and Cross-domain Parser Evaluation*, COLING 2008, pages 1–8. Association for Computational Linguistics.

Debusmann, R., Duchier, D., and Kruijff, G.-J. M. (2004). Extensible Dependency Grammar: A New Methodology. In *Proceedings of the COLING Workshop on Recent Advances in Dependency Grammar*, pages 78–85.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

Derwojedowa, M. (2011). *Składnia liczebników we współczesnym języku polskim. Zarys opisu zależnościowego*. Wydawnictwo Wydziału Polonistyki UW, Warszawa.

Diab, M. and Resnik, P. (2002). An Unsupervised Method for Word Sense Tagging using Parallel Corpora. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL'02, pages 255–262.

Diestel, R. (2000). *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, electronic edition edition.

Dębowski, Ł. (2009). Valence extraction using EM selection and co-occurrence matrices. *Language Resources and Evaluation*, 43(4):301–327.

Duan, X., Zhao, J., and Xu, B. (2007). Probabilistic Parsing Action Models for Multi-Lingual Dependency Parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 940–946.

Duchier, D. and Debusmann, R. (2001). Topological Dependency Trees: A Constraint-Based Account of Linear Precedence. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics and the 10th European Chapter of the ACL*, pages 180–187. Association for Computational Linguistics.

Džeroski, S., Erjavec, T., Ledinek, N., Pajas, P., Žabokrtsky, Z., and Žele, A. (2006). Towards a Slovene dependency treebank. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, LREC'06, pages 1388–1391.

Edmonds, J. (1967). Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B(4):233–240.

Eisner, J. M. (1996). Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proceedings of the 16th International Conference on Computational Linguistics*, COLING 1996, pages 340–345.

Eisner, J. M. (2000). Bilexical Grammars and their Cubic-Time Parsing Algorithms. In Bund, H. and Nijholt, A., editors, *New Developments in Natural Language Parsing*, volume 16 of *Text, Speech and Language Technology*, pages 29–62. Springer-Verlag.

Falk, Y. (2001). *Lexical-Functional Grammar: An Introduction to Parallel Constraint-Based Syntax*. CSLI Publications, Standford, CA.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9:1871–1874.

Finkel, J., Dingare, S., Manning, C. D., Nissim, M., Alex, B., and Grover, C. (2005). Exploring the Boundaries: Gene and Protein Identification in Biomedical Text. *BMC Bioinformatics*, 6.

Finkel, J. R., Grenager, T., and Manning, C. D. (2007). The Infinite Tree. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 272–279. Association for Computational Linguistics.

Forst, M., Bertomeu, N., Crysmann, B., Fouvry, F., Hansen-Schirra, S., and Kordoni, V. (2004). Towards a Dependency-based Gold Standard for German Parsers - The TiGer Dependency Bank. In *Proceedings of the 5th International Workshop on Linguistically Interpreted Corpora (LINC-04) at COLING*, pages 31–38.

Fraser, A. and Marcu, D. (2005). ISI's Participation in the Romanian-English Alignment Task. In *The ACL 2005 Workshop on Building and Using Parallel Texts*, pages 91–94.

Fraser, A. and Marcu, D. (2007). Measuring Word Alignment Quality for Statistical Machine Translation. *Computational Linguistics*, 33:293–303.

Gale, W. A. and Church, K. W. (1991). A Program for Aligning Sentences in Bilingual Corpora. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 177–184.

Ganchev, K. (2010). *Posterior Regularization for Learning with Side Information and Weak Supervision*. PhD thesis, University of Pennsylvania.

Ganchev, K., Gillenwater, J., and Taskar, B. (2009). Dependency Grammar Induction via Bitext Projection Constraints. In *Proceedings of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, volume 1, pages 369–377.

Gelbukh, A., Torres, S., and Calvo, H. (2005). Transforming a Constituency Treebank into a Dependency Treebank. *Procesamiento del Lenguaje Natural*, 35:145–152.

Gillenwater, J., Ganchev, K., Graça, J., Pereira, F., and Taskar, B. (2010). Sparsity in Dependency Grammar Induction. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 194–199.

Gómez-Rodríguez, C. and Nivre, J. (2010). A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1492–1501. Association for Computational Linguistics.

Graliński, F. (2002). Wstępujący parser języka polskiego na potrzeby systemu POLENG. *Speech and Language Technology. Technologia Mowy i Języka*, 6(III):263–276.

Graliński, F. (2007). *Formalizacja nieciągłości zdań przy zastosowaniu rozszerzonej gramatyki bezkontekstowej*. PhD thesis, Adam Mickiewicz University, Poznań.

Graliński, F., Jassem, K., and Junczys-Dowmunt, M. (2012). PSI-Toolkit: Natural Language Processing Pipeline. *Computational Linguistics – Applications*, 458:27–39.

Hajič, J. (1998). Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. In Hajičová, E., editor, *Issues of Valency and Meaning. Studies in Honour of Jarmila Panevová*, pages 106–132. Karolinum, Charles University Press, Prague, Czech Republic.

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Mart'ı, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., Straňák, P., Surdeanu, M., Xue, N., and Zhang, Y. (2009). The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, CoNLL '09, pages 1–18. Association for Computational Linguistics.

Hajič, J., Hajičová, E., Panevová, J., Sgall, P., Bojar, O., Cinková, S., Fučíková, E., Mikulová, M., Pajas, P., Popelka, J., Semecký, J., Šindlerová, J., Štěpánek, J., Toman, J., Urešová, Z., and Žabokrtský, Z. (2012). Announcing Prague Czech-English Dependency Treebank 2.0. In Calzolari, N., Choukri, K., Declerck, T., Doğan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation*, LREC'12. European Language Resources Association (ELRA).

Hajič, J., Smerž, O., Zemánek, P., and Beška, E. (2004). Prague Arabic Dependency Treebank: Development in Data and Tools. In *NEM-LAR International Conference on Arabic Language Resources and Tools*, pages 110–117. ELDA.

Hall, K. (2007). k-best Spanning Tree Parsing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 392–399.

Harper, M. P. and Helzerman, R. A. (1995). Extensions to Constraint Dependency Parsing for Spoken Language Processing. *Computer Speech and Language*, 9:187–234.

Haverinen, K., Viljanen, T., Laippala, V., Kohonen, S., Ginter, F., and Salakoski, T. (2010). Treebanking Finnish. In Dickinson, M., Müüisep, K., and Passarotti, M., editors, *Proceedings of the Ninth International Workshop on Treebanks and Linguistic Theories*, volume 9 of *Nealt Proceedings Series*, pages 79–90. Northern European Association for Language Technology (NEALT).

Headden, III, W. P., Johnson, M., and McClosky, D. (2009). Improving Unsupervised Dependency Parsing with Richer Contexts and Smoothing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL 2009, pages 101–109. Association for Computational Linguistics.

Hellwig, P. (1986). Dependency Unification Grammar. In *Proceedings of the 11th International Conference on Computational Linguistics*, COLING 1986, pages 195–198.

Hellwig, P. (2003). Dependency Unification Grammar. In Ágel, V., Eichinger, L. M., Eroms, H. W., Hellwig, P., Herringer, H. J., and Lobin, H., editors, *Dependenz und Valenz / Dependency and Valency. Ein internationales Handbuch der zeitgenössischen Forschung / An international Handbook of Contemporary Research (1. Halbband / Volume 1)*, volume 25.1 of *Handbücher zur Sprach- und Kommunikationswissenschaft / Handbooks of Linguistics and Communication Science (HSK)*, pages 593–635. De Gruyter Mouton, Berlin, Boston.

Hudson, R. (1990). *English Word Grammar*. Basil Blackwell, Oxford.

Hwa, R., Resnik, P., Weinberg, A., Cabezas, C., and Kolak, O. (2005). Bootstrapping Parsers via Syntactic Projection across Parallel Texts. *Natural Language Engineering*, 11(3):311–325.

Hwa, R., Resnik, P., Weinberg, A., and Kolak, O. (2002). Evaluating Translational Correspondence using Annotation Projection. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, ACL'02, pages 392–399.

Järvinen, T. and Tapanainen, P. (1998). Towards an implementable dependency grammar. In *Proceedings of the Workshop on Processing of Dependency-Based Grammars (ACL-COLING)*, pages 1–10.

Jassem, K. (2006). *Przetwarzanie tekstów polskich w systemie tłumaczenia automatycznego POLENG*. Wydawnictwo Naukowe UAM, Poznań.

Jiang, W. and Liu, Q. (2009). Automatic Adaptation of Annotation Standards for Dependency Parsing – Using Projected Treebank as Source Corpus. In *Proceedings of the 11th International Conference on Parsing Technologies*, IWPT'09, pages 25–28.

Jiang, W. and Liu, Q. (2010). Dependency Parsing and Projection Based on Word-Pair Classification. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 12–20.

Johansson, R. and Nugues, P. (2007). Extended Constituent-to-Dependency Conversion for English. In Nivre, J., Kaalep, H.-J., Muischnek, K., and Koit, M., editors, *Proceedings of the 16th Nordic Conference of Computational Linguistics*, NODALIDA-2007, pages 105–112.

Joshi, A. K. and Schabes, Y. (1997). Tree-adjoining grammars. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin, New York.

Kahane, S. (2003). The Meaning-Text Theory. In Ágel, V., Eichinger, L. M., Eroms, H. W., Hellwig, P., Herringer, H. J., and Lobin, H., editors, *Dependenz und Valenz / Dependency and Valency. Ein internationales Handbuch der zeitgenössischen Forschung / An International Handbook of Contemporary Research (1. Halbband /*

Volume 1), volume 25.1 of *Handbücher zur Sprach- und Kommunikationswissenschaft / Handbooks of Linguistics and Communication Science (HSK)*, pages 546–570. De Gruyter Mouton, Berlin, Boston.

Kaplan, R. M. and Bresnan, J. (1995). Lexical-Functional Grammar: A Formal System for Grammatical Representation. In Dalrymple, M., Kaplan, R. M., Maxwell III, J. T., and Zaenen, A., editors, *Formal Issues in Lexical-Functional Grammar*, volume 47 of *CSLI Lecture Notes Series*, pages 29–130. CSLI Publications, Stanford.

Kaplan, R. M., Riezler, S., King, T. H., Maxwell III, J. T., Vasserman, A., and Crouch, R. (2004). Speed and Accuracy in Shallow and Deep Stochastic Parsing. In *Proceedings of the Human Language Technology Conference and the 4th Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, HLT-NAACL'04, pages 97–104.

Kim, S., Jeong, M., Lee, J., and Lee, G. G. (2010). A Cross-lingual Annotation Projection Approach for Relation Detection. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING 2010, pages 564–571.

King, T. H., Crouch, R., Riezler, S., Dalrymple, M., and Kaplan, R. M. (2003). The PARC 700 Dependency Bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora*, LINC-03, pages 1–8.

Kiss, T. and Strunk, J. (2006). Unsupervised Multilingual Sentence Boundary Detection. *Computational Linguistics*, 32:485–525.

Klein, D. and Manning, C. D. (2004). Corpus-Based Induction of Syntactic Structure: Models of Dependency and Constituency. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 478–485.

Klemensiewicz, Z. (1968). *Zarys składni polskiej.* PWN, Warszawa.

Koehn, P. (2005). Europarl: A Parallel Corpus for Statistical Machine Translation. In *Proceedings of the 10th Machine Translation Summit Conference*, pages 79–86.

Koehn, P. (2010). *Statistical Machine Translation.* Cambridge University Press.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, ACL'07, pages 177–180.

Koo, T., Globerson, A., Carreras, X., and Collins, M. (2007). Structured Prediction Models via the Matrix-Tree Theorem. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 141–150.

Kromann, M. T. (2003). The Danish Dependency Treebank and the DTAG treebank tool. In *Proceedings of the Second Workshop on Treebanks and Linguistic Theories*, TLT 2003, pages 217–220.

Kruijff, G.-J. M. (2002). Formal and Computational Aspects of Dependency Grammar. Historical development of DG. `http://www.univ-orleans.fr/lifo/membres/duchier/teaching/ESSLLI-2002/esslli-history.pdf`. Notes for the ESSLLI 2002 course on Formal and Computational Aspects of Dependency Grammar.

Kübler, S., Maier, W., Rehbein, I., and Versley, Y. (2008). How to Compare Treebanks. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odjik, J., Piperidis, S., and Tapias, D., editors, *Proceedings of the Sixth International Conference on Language Resources and Evaluation*, LREC'08, pages 2322–2329. European Language Resources Association (ELRA).

Kübler, S., McDonald, R. T., and Nivre, J. (2009). *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Levy, R. and Andrew, G. (2006). Tregex and Tsurgeon: tools for querying and manipulating tree data structures. In *Proceedings of the Fifth International Conference of Language Resources and Evaluation*, LREC'06, pages 2231–2234.

Lewis, M. P., Simons, G. F., and Fenning, C. D., editors (2013). *Ethnologue: Languages of the World*. SIL International, Dallas, Texas, 17th edition.

Maamouri, M., Bies, A., Krouna, S., Tabessi, D., and Gaddeche, F. (2011). Penn Arabic Treebank Guidelines. Technical report, Linguistic Data Consortium, University of Pennsylvania.

Magerman, D. M. (1994). *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Stanford University.

Magerman, D. M. (1995). Statistical Decision-Tree Models for Parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283.

Marciniak, M., Mykowiecka, A., Kupść, A., and Węgiel, M. (2000). Klasyfikacja zjawisk syntaktycznych na potrzeby testowego zbioru wyrażeń języka polskiego. Technical Report 908, Institute of Computer Science, Polish Academy of Sciences, Warszawa.

Marcus, M., Kim, G., Marcinkiewicz, M. A., Macintyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The Penn Treebank: Annotating Predicate Argument Structure. In *ARPA Human Language Technology Workshop*, pages 114–119.

Mareček, D. (2011). Combining Diverse Word-Alignment Symmetrizations Improves Dependency Tree Projection. In *Computational Linguistics and Intelligent Text Processing: 12th International Conference*, volume 6608 of *Lecture Notes in Computer Science*, pages 144–154. Springer-Verlag.

Marinov, S. (2009). *Dependency-Based Syntactic Analysis of Bulgarian*. PhD thesis, University of Gothenburg.

Maxwell III, J. T. and Kaplan, R. M. (1993). The Interface between Phrasal and Functional Constraints. *Computational Linguistics*, 19(4):571–590.

McDonald, R., Crammer, K., and Pereira, F. (2005a). Online Large-Margin Training of Dependency Parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, ACL 2005, pages 91–98.

McDonald, R., Lerman, K., and Pereira, F. (2006). Multilingual Dependency Analysis with a Two-Stage Discriminative Parser. In *Proceedings of the 10th Conference on Computational Natural Lanugage Learning (CoNLL-X)*, pages 216–220.

McDonald, R. and Pereira, F. (2006). Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, EACL 2006, pages 81–88.

McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005b). Non-projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HTL'05, pages 523–530.

McDonald, R., Petrov, S., and Hall, K. B. (2011). Multi-Source Transfer of Delexicalized Dependency Parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP'11, pages 63–72.

Mel'čuk, I. A. (1988). *Dependency Syntax : Theory and Practice*. SUNY Press, Albany.

Mel'čuk, I. A. and Pertsov, N. V. (1987). *Surface syntax of English. A formal model within the meaning-text framework*, volume 13 of *Linguistic and Literary Studies in Eastern Europe*. John Benjamins Publishing Company, Amsterdam.

Menzel, W. and Schröder, I. (1998). Decision Procedures for Dependency Parsing Using Graded Constraints. In *Proceedings of the Workshop on Processing of Dependency-Based Grammars (ACL-COLING)*, pages 78–87.

Merlo, P., Stevenson, S., Tsang, V., and Allaria, G. (2002). A Multilingual Paradigm for Automatic Verb Classification. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL'02, pages 207–214.

Milewska, B. (2003). *Przyimki wtórne we współczesnej polszczyźnie*. Wydawnictwo Uniwersytetu Gdańskiego, Gdańsk.

Moore, R. C. (2002). Fast and Accurate Sentence Alignment of Bilingual Corpora. In *Machine Translation: From Research to Real Users*, volume 2499 of *Lecture Notes in Computer Science*, pages 135–244. Springer-Verlag.

Nakagawa, T. (2007). Multilingual Dependency Parsing using Global Features. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 952–956.

Naseem, T., Barzilay, R., and Globerson, A. (2012). Selective Sharing for Multilingual Dependency Parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, pages 629–637.

Newman, M. E. (2010). *Networks*. Oxford University Press, New York.

Nivre, J. (2006). *Inductive Dependency Parsing*, volume 34 of *Text, Speech and Language Technology*. Springer-Verlag, Dordrecht.

Nivre, J. (2008). Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34:513–553.

Nivre, J. (2009). Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, volume 1, pages 351–359.

Nivre, J., Boguslavsky, I. M., and Iomdin, L. L. (2008). Parsing the SynTagRus treebank of Russian. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 641–648.

Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., and Yuret, D. (2007). The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932.

Nivre, J., Hall, J., and Nilsson, J. (2006a). MaltParser: A Data-Driven Parser-Generator for Dependency Parsing. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, LREC'06, pages 2216–2219.

Nivre, J., Kuhlmann, M., and Hall, J. (2009). An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies*, IWPT '09, pages 73–76. Association for Computational Linguistics.

Nivre, J. and Nilsson, J. (2005). Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, ACL '05, pages 99–106. Association for Computational Linguistics.

Nivre, J., Nilsson, J., and Hall, J. (2006b). Talbanken05: A Swedish Treebank with Phrase Structure and Dependency Annotation. In *Proceedings of the Fifth International Conference of Language Resources and Evaluation*, LREC'06, pages 1392–1395.

Obrębski, T. (2002). *Automatyczna analiza składniowa języka polskiego z wykorzystaniem gramatyki zależnościowej*. PhD thesis, Institute of Computer Science, Polish Academy of Sciences, Warsaw.

Obrębski, T. (2003). MTT-compatible computationally effective surface-syntactic parser. In *Proceedings of First International Conference on Meaning-Text Theory*, pages 259–268.

Och, F. J. and Ney, H. (2003). A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 29(1):19–51.

Øvrelid, L., Kuhn, J., and Spreyer, K. (2009). Improving Data-Driven Dependency Parsing Using Large-Scale LFG Grammars. In *Proceedings of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (Conference Short Papers)*, pages 37–40.

Ozdowska, S. (2006). Projecting POS tags and syntactic dependencies from English and French to Polish in aligned corpora. In *Proceedings of the EACL Workshop on Cross-Language Knowledge Induction*, pages 53–60.

Padó, S. and Lapata, M. (2005). Cross-linguistic Projection of Role-semantic Information. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 859–866.

Padó, S. and Lapata, M. (2009). Cross-lingual Annotation Projection for Semantic Roles. *Journal of Artificial Intelligence Research*, 36:307–340.

Patejuk, A. and Przepiórkowski, A. (2012). Towards an LFG parser for Polish: An exercise in parasitic grammar development. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation*, LREC'12, pages 3849–3852. ELRA.

Pęzik, P., Ogrodniczuk, M., and Przepiórkowski, A. (2011). Parallel and spoken corpora in an open repository of Polish language resources. In *Proceedings of the 5th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, pages 511–515.

Polguére, A. and Mel'čuk, I. A., editors (2009). *Dependency in Linguistic Description*, volume 111 of *Studies in Language Companion Series (SLCS)*. John Benjamins Publishing Company, Amsterdam.

Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago.

Popel, M., Mareček, D., Štěpánek, J., Zeman, D., and Žabokrtský, Z. (2013). Coordination Structures in Dependency Treebanks. In *Proceedings of the 51st Annual Meeting of the Association for Computer Linguistics*, pages 517–527.

Przepiórkowski, A. (2008). *Powierzchowne przetwarzanie języka polskiego*. Problemy Współczesnej Nauki. Teoria i Zastosowania: Inżynieria lingwistyczna. Akademicka Oficyna Wydawnicza Exit, Warszawa.

Przepiórkowski, A., Bańko, M., Górski, R. L., and Lewandowska-Tomaszczyk, B., editors (2012). *Narodowy Korpus Języka Polskiego*. Wydawnictwo Naukowe PWN, Warsaw.

Przepiórkowski, A., Kupść, A., Marciniak, M., and Mykowiecka, A. (2002). *Formalny opis języka polskiego: Teoria i implementacja*. Problemy Współczesnej Nauki. Teoria i Zastosowania: Inżynieria lingwistyczna. Akademicka Oficyna Wydawnicza Exit, Warszawa.

Przepiórkowski, A., Skwarski, F., Hajnicz, E., Patejuk, A., Świdziński, M., and Woliński, M. (2013). Modelowanie własności składniowych czasowników w nowym słowniku walencyjnym języka polskiego. `http://clip.ipipan.waw.pl/Walenty?action=AttachFile&do=view&target=walenty.20130929.1114.pdf`. A draft version of the article submitted for Polonica.

Ramasamy, L. and Žabokrtský, Z. (2011). Tamil Dependency Parsing: Results Using Rule Based and Corpus Based Approaches. In *Proceedings of the 12th International Conference on Intelligent Text Processing and Computational Linguistics*, volume Part I of *CICLing'11*, pages 82–95, Berlin, Heidelberg. Springer-Verlag.

Saloni, Z. (2010). *Czasownik polski*. Wiedza Powszechna, Warszawa.

Saloni, Z. and Świdziński, M. (1989). *Składnia współczesnego języka polskiego*. Wydawnictwo Naukowe PWN, Warszawa.

Saloni, Z. and Świdziński, M. (2011). *Składnia współczesnego języka polskiego*. Wydawnictwo Naukowe PWN, Warszawa.

Savary, A., Chojnacka-Kuraś, M., Wesołek, A., Skowrońska, D., and Śliwiński, P. (2012). Anotacja jednostek nazewniczych. In Przepiórkowski et al. (2012), pages 129–167.

Savary, A. and Waszczuk, J. (2012). Narzędzia do anotacji jednostek nazewniczych. In Przepiórkowski et al. (2012), pages 225–252.

Seddah, D., Tsarfaty, R., Kübler, S., Candito, M., Choi, J. D., Farkas, R., Foster, J., Goenaga, I., Gojenola, K., Goldberg, Y., Green, S., Habash, N., Kuhlmann, M., Maier, W., Nivre, J., Przepiórkowski, A., Roth, R., Seeker, W., Versley, Y., Vincze, V., Woliński, M., Wróblewska, A., and de la Clérgerie, E. V. (2013). Overview of the SPMRL 2013 Shared Task: A Cross-Framework Evaluation of Parsing Morphologically Rich Languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically Rich Languages*, pages 146–182.

Sgall, P., Hajičová, E., and Panevová, J. (1986). *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Dordrecht: Reidel Publishing Company and Prague: Academia.

Sleator, D. D. and Temperley, D. (1993). Parsing English with a Link Grammar. In *Proceedings of the Third International Workshop on Parsing Technologies*, pages 277–292.

Smith, D. A. and Eisner, J. (2006). Quasi-Synchronous Grammars: Alignment by Soft Projection of Syntactic Dependencies. In *Proceedings of the HLT-NAACL Workshop on Statistical Machine Translation*, pages 23–30.

Smith, D. A. and Eisner, J. (2009). Parser Adaptation and Projection with Quasi-Synchronous Grammar Features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 822–831.

Smith, D. A. and Smith, N. A. (2007). Probabilistic Models of Nonprojective Dependency Trees. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 132–140.

Søgaard, A. (2011). Data point selection for cross-language adaptation of dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 682–686.

Spitkovsky, V. I., Alshawi, H., Jurafsky, D., and Manning, C. D. (2010). Viterbi Training Improves Unsupervised Dependency Parsing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 9–17.

Spreyer, K. (2011). *Does It Have To Be Trees? Data-Driven Dependency parsing with Incomplete and Noisy Training Data*. PhD thesis, Universität Potsdam.

Spreyer, K. and Frank, A. (2008). Projection-based Acquisition of a Temporal Labeller. In *Proceedings of the 3rd International Joint Conference on Natural Language Processing*, IJCNLP 2008, pages 489–496.

Steinberger, R., Eisele, A., Klocek, S., Pilos, S., and Schlüter, P. (2012). DGT-TM: A freely Available Translation Memory in 22 Languages. In *Proceedings of the 8th International Conference on Language Resources and Evaluation*, pages 454–459.

Świdziński, M. (1989). A dependency syntax of Polish. In Maxwell, D. and Schubert, K., editors, *Metataxis in practice. Dependency syntax for multilingual machine translation*, pages 69–88. Foris Publications, Dordrecht, Providence.

Świdziński, M. (1992). *Gramatyka formalna języka polskiego*, volume 349 of *Rozprawy Uniwersytetu Warszawskiego*. Wydawnictwa Uniwersytetu Warszawskiego, Warszawa.

Świdziński, M. and Woliński, M. (2010). Towards a Bank of Constituent Parse Trees for Polish. In Sojka, P., Horák, A., Kopeček, I., and Pala, K., editors, *Text, Speech and Dialogue, 13th International Conference, TSD 2010, Brno, Czech Republic*, volume 6231 of *LNAI*, pages 197–204, Heidelberg. Springer-Verlag.

Szpakowicz, S. (1978). *Automatyczna analiza składniowa zdań pisanych*. Ph.D. dissertation, Uniwersytet Warszawski, Warszawa.

Szpakowicz, S., editor (1986). *Formalny opis składniowy zdań polskich*. Wydawnictwa Uniwersytetu Warszawskiego, Warszawa.

Szpakowicz, S. and Świdziński, M. (1990). Formalna definicja równorzędnej grupy nominalnej we wspólczesnej polszczyźnie pisanej. *Studia Gramatyczne*, IX:9–54.

Täckström, O. (2013). *Predicting Linguistic Structure with Incomplete and Cross-Lingual Supervision*. PhD thesis, Uppsala Universitet.

Täckström, O., McDonald, R., and Nivre, J. (2013). Target Language Adaptation of Discriminative Transfer Parsers. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT 2013, pages 1061–1071.

Tapanainen, P. and Järvinen, T. (1997). A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pages 64–71.

Tesnière, L. (1959). *Éléments de syntaxe structurale*. Klincksieck, Paris.

Tiedemann, J. (2011). *Bitext Alignment*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Tiedemann, J. (2012). Parallel Data, Tools and Interfaces in OPUS. In *Proceedings of the 8th International Conference on Language Resources and Evaluation*, pages 2214–2218.

Titov, I. and Henderson, J. (2007). Fast and Robust Multilingual Dependency Parsing with a Generative Latent Variable Model. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 947–951.

Tu, K. (2012). *Unsupervised learning of probabilistic grammars*. PhD thesis, Iowa State University, Ames, Iowa.

Tu, K. and Honavar, V. (2012). Unambiguity Regularization for Unsupervised Learning of Probabilistic grammars. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2012)*, pages 1324–1334. Association for Computational Linguistics.

van der Plas, L. and Tiedemann, J. (2006). Finding Synonyms Using Automatic Word Alignment and Measures of Distributional Similarity. In *Proceedings of the COLING/ACL Main Conference Poster Sessions*, COLING-ACL '06, pages 866–873.

Varga, D., Halászcy, P., Kornai, A., Nagy, V., Németh, L., and Trón, V. (2005). Parallel corpora for medium density languages. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing*, RANLP 2005, pages 590–596.

Vetulani, Z. (2004). *Komunikacja człowieka z maszyną. Komputerowe modelowanie kompetencji językowej.* Problemy Współczesnej Nauki. Teoria i Zastosowania: Informatyka. Akademicka Oficyna Wydawnicza Exit, Warszawa.

Villeda Moirón, B. n. and Tiedemann, J. (2006). Identifying idiomatic expressions using automatic word alignment. In *Proceedings of the EACL 2006 Workshop on Multiword Expressions in a Multilingual Context*, pages 33–40.

Vogel, S., Ney, H., and Tillmann, C. (1996). HMM-Based Word Alignment in Statistical Translation. In *Proceedings of the 16th Conference on Computational Linguistics*, volume 2 of *COLING'96*, pages 836–841.

Wang, W. and Harper, M. P. (2004). A Statistical Constraint Dependency Grammar (CDG) Parser. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 42–49. Association for Computational Linguistics.

Woliński, M. (2004). *Komputerowa weryfikacja gramatyki Świdzińskiego.* Ph.D. dissertation, Institute of Computer Science, Polish Academy of Sciences, Warsaw.

Woliński, M. (2005a). An efficient implementation of a large grammar of Polish. In Vetulani, Z., editor, *Proceedings of the 2nd Language & Technology Conference*, pages 303–347.

Woliński, M. (2005b). An efficient implementation of a large grammar of Polish. *Archives of Control Sciences*, 15(3):251–258.

Woliński, M., Głowińska, K., and Świdziński, M. (2011). A Preliminary Version of Składnica – a Treebank of Polish. In Vetulani, Z., editor, *Proceedings of the 5th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, pages 299–303.

Wróblewska, A. (2012). Polish Dependency Bank. *Linguistic Issues in Language Technology*, 7(1):1–15.

Wróblewska, A. and Frank, A. (2009). Cross-Lingual Projection of LFG F-Structures: Building an F-Structure Bank for Polish. In *Proceedings of the Eighth International Workshop on Treebanks and Linguistic Theories*, TLT 8, pages 209–220.

Wróblewska, A. and Przepiórkowski, A. (2012). Induction of Dependency Structures Based on Weighted Projection. In *Proceedings of the 4th International Conference on Computational Collective Intelligence Technologies and Applications, Part I*, volume 7653 of *Lecture Notes in Artificial Intelligence*, pages 364–374, Berlin. Springer-Verlag.

Wróblewska, A. and Woliński, M. (2012). Preliminary Experiments in Polish Dependency Parsing. In *Security and Intelligent Information Systems: International Joint*

Conference (SIIS 2011), Revised Selected Papers, volume 7053 of *Lecture Notes in Computer Science*, pages 279–292. Springer-Verlag.

Xue, N., Xia, F., Chiou, F.-D., and Palmer, M. (2005). The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.

Yamada, H. and Matsumoto, Y. (2003). Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of 8th International Workshop on Parsing Technologies*, pages 195–206.

Yarowsky, D. and Ngai, G. (2001). Inducing Multilingual POS Taggers and NP Bracketers via Robust Projection across Aligned Corpora. In *Proceedings of the 2nd Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 200–207.

Yarowsky, D., Ngai, G., and Wicentowski, R. (2001). Inducing Multilingual Text Analysis Tools via Robust Projection across Aligned Corpora. In *Proceedings of the First International Conference on Human Language Technology Research*, pages 1–8.

Zeman, D. and Resnik, P. (2008). Cross-Language Parser Adaptation between Related Languages. In *Proceedings of the IJCNLP-08 Workshop on NLP for Less Privileged Languages*, pages 35–42.