THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Multilingual Abstractions: Abstract Syntax Trees and Universal Dependencies

PRASANTH KOLACHINA

UNIVERSITY OF GOTHENBURG

Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2019

Multilingual Abstractions: Abstract Syntax Trees and Universal Dependencies
PRASANTH KOLACHINA

ISBN 978-91-7833-509-1
Technical Report 174D
Department of Computer Science & Engineering

Division of Functional Programming
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden
Telephone +46 (0)31-772 1000

Printed at Chalmers reproservice
Gothenburg, Sweden 2019.

To my family
Mom, Dad and Sudheer

Abstract

This thesis studies the connections between parsing friendly representations and interlingua grammars developed for multilingual language generation. Parsing friendly representations refer to dependency tree representations that can be used for robust, accurate and scalable analysis of natural language text. Shared multilingual abstractions are central to both these representations. Universal Dependencies (UD) is a framework to develop cross-lingual representations, using dependency trees for multlingual representations. Similarly, Grammatical Framework (GF) is a framework for interlingual grammars, used to derive abstract syntax trees (ASTs) corresponding to sentences. The first half of this thesis explores the connections between the representations behind these two multilingual abstractions. The first study presents a conversion method from abstract syntax trees (ASTs) to dependency trees and present the mapping between the two abstractions – GF and UD – by applying the conversion from ASTs to UD. Experiments show that there is a lot of similarity behind these two abstractions and our method is used to bootstrap parallel UD treebanks for 31 languages. In the second study, we study the inverse problem i.e. converting UD trees to ASTs. This is motivated with the goal of helping GF-based interlingual translation by using dependency parsers as a robust front end instead of the parser used in GF.

The second half of this thesis focuses on the topic of data augmentation for parsing – specifically using grammar-based backends for aiding in dependency parsing. We propose a generic method to generate synthetic UD treebanks using interlingua grammars and the methods developed in the first half. Results show that these synthetic treebanks are an alternative to develop parsing models, especially for under-resourced languages without much resources. This study is followed up by another study on out-of-vocabulary words (OOVs) – a more focused problem in parsing. OOVs pose an interesting problem in parser development and the method we present in this paper is a generic simplification that can act as a drop-in replacement for any symbolic parser. Our idea of replacing unknown words with known, similar words results in small but significant improvements in experiments using two parsers and for a range of 7 languages.

Keywords

Natural Language Processing, Grammatical Framework, Universal Dependencies, multilinguality, abstract syntax trees, dependency trees, multilingual generation, multilingual parsers

Acknowledgments

In the time spent working on this, I was also trying to narrow down that one domino that made me pursue this crazy job. I did find it in due time – a stroll on a winter evening with a colleague on the streets of Hyderabad. I told him the crazy idea I had in the months prior working as an research intern, to pursue a Ph.D. By the end of the day, he convinced me I was both necessarily and sufficiently crazy to go for one! This was followed by a conversation with a mentor from those days, who cautioned me about a long tunnel, along with what I thought was some sage advice.

That evening and an year later, I was at Chalmers. In all my time at Chalmers – working with Aarne – I did find myself asking at times if the tale about the tunnel was true. There were certainly times when it seemed true but the ensuing time was also filled with learning moments. Aarne gave me the space and time to address each part I thought I lacked in order to pursue my research interests. For that I will always thank him, more so because he let me address them on my own terms. Thanks are also due to Richard Johansson and Krasimir Angelov – my co-supervisors – who have at all times during this journey supported me. Richard, did at times indulge me listening patiently to what seemed and still seem like crazy ideas to me, and always helped me refine those ideas and aspects of my research that are not often overtly realized, atleast immediately. Krasimir was more hands-on helping me make sense of the craziness involved by sharing his own experiences. The Grammar Technology group – Herbert, Peter Ljunglof, Prasad KVS, Inari, John, Normunds, Gregoire, Thomas Hallgren and Koen Classen (when he consented to being part of the group) and David – was something I could always count on for insightful conversations and at times, for procrastination working on interesting problems. Agneta Nilsson, Mary Sheeran who have been in my committee and Devdatt Dubhashi who acted in the role of my examiner have also supported me throughout the years.

But the journey is not all about research, and those of us here know the role teaching plays in the process. I had taught before coming to Chalmers – when asked to – but never imagined myself liking it, much less, enjoy it. I did discover those aspects of teaching over the years while working with Dag Wedelin on the problem solving course. That only got better working with other people involved in the course – Birgit Grohe, Simon R., Dan R., Victor, Mikael amongst others – and one I think of as a valuable experience. If the idea of Ph.D. seemed crazy to me those years ago, I admit the idea of pursuing an academic career seems equally crazy now. That said if I do pursue one – it will not be due to my problem solving skills – it will surely be due to my experience in the course on problem solving. Thank you for that Dag and everyone who worked in the course the last five years including the students.

I would like to say thanks to Joakim Nivre who shared his insights on my work

and has also graciously agreed to be a part of my defense, in addition to hosting my research visit at Uppsala. The Computational Linguistics group at Uppsala – Miryam, Amir, Ali, Yan, Marie, Fabienne, Aaron – are nothing short of fantastic and an excellent presence to have in close proximity. Thanks also to Filip Ginter who was the discussion leader for my licentiate and Lilja Øvrelid and Marco Kuhlmann who have all accepted to be on the committee for my defense. I met Marco while working on `ud2gf` and his insights have been very helpful in improving my understanding of this work.

Just as all work makes Jack a dull boy, my time at Chalmers was enriched abundantly by time spent with amazing people outside the group. Olof and Mikael with whom I did have conversations on everything that is around – from machine learning and natural language processing to Sweden, from technical to social and perhaps at times even religious – and Alirad were the best colleagues one could ask for. These hangouts were further improved when I spent my time with the larger group of CLT in Gothenburg – Luis, Ildiko, Mehdi, Nina, Markus – who constantly reminded me that it was okay to sign-out from work. There are other things I need to acknowledge as part of this journey – Sweden being the primary one. The who, why and what of that is impossible to precisely quantify – neither the *who* or *what* can be enumerated here in entirety – and is perhaps best left unspecified while relishing all that did happen. I also found great company in the online world – Science Twitter – which made me feel welcome.

Finally, I heard over the years the cliff-climbing-cliche of life and I admit this never felt like one. Most times, it felt as though I had jumped off one – after all climbing gives you a choice to stop at any point but jumping never does – only to realize I had a lot of fantastic support underneath it all. Sudheer – the colleague and brother who encouraged me to start this journey – and Lilla have always and continue to give me sage advice when I need it. Behind him was my mother who despite not knowing why I was doing this, always let me know that things would be okay when I most needed to hear it. The journey might not have started because of them, but it definitely would not have come this far if not for them.

The work presented in this thesis has been funded by the Swedish Research Council as part of the REMU project — *Reliable Multilingual Digital Communication: Methods and Applications* (grant number 2012-5746).

List of Publications

Appended publications

This thesis is based on the following publications:

[I] Prasanth Kolachina and Aarne Ranta "From Abstract Syntax to Universal Dependencies"
Linguistic Issues in Language Technology 13(3), 2016.

[II] Aarne Ranta and Prasanth Kolachina "From Universal Dependencies to Abstract Syntax"
Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017), pp. 107–116.

[III] Prasanth Kolachina and Aarne Ranta "Bootstrapping UD treebanks for Delexicalized Parsing"
Under submission.

[IV] Prasanth Kolachina and Martin Riedl and Chris Biemann
"Replacing OOV Words For Dependency Parsing With Distributional Semantics"
Proceedings of the 21st Nordic Conference on Computational Linguistics (NoDaLiDa), 2017, pp. 11–19.

Other publications

[V] Aarne Ranta and Prasanth Kolachina and Thomas Hallgren "Cross-Lingual Syntax: Relating Grammatical Framework with Universal Dependencies" *Proceedings of the 21st Nordic Conference on Computational Linguistics (NoDaLiDa), System Demos, 2017*.

[VI] Prasanth Kolachina and Aarne Ranta "GF Wide-coverage English-Finnish MT system for WMT 2015"
Proceedings of the Tenth Workshop on Statistical Machine Translation, 2015, pp. 141–144.

[VII] Ramona Enache and Inari Listenmaa and Prasanth Kolachina "Handling non-compositionality in multilingual CNLs" *Fourth Workshop on Controlled Natural Language (CNL 2014), 2014, pp. 147–154*.

Research Contribution

- In Paper I (Chapter 2), the concrete and extended variant of the configurations were made by the author. Necessary changes to the translation algorithm were also made by the author in addition to the complete configuration specification from GF-RGL to Universal Dependencies. The manuscript was jointly written, where the authors contribution was around 75% in the manuscript.
- In Paper II (Chapter 3), the authors contribution was to the experiments described in the manuscript. Subsequent modifications to the implementation include k-best parsing, probabilistic disambiguation and unlexicalized variant of `ud2gf`. Subsequent modifications to the configurations include extensions from UDv1 to UDv2 and extensions to the grammar to improve coverage of the translation method. These changes were made after the publication.
- In Paper III (Chapter 4), the author designed the experimental setup, made the experiments and wrote most of the paper.
- In Paper IV (Chapter 5), the author designed the experimental setup, made the experiments and subsequent data analysis and wrote around 50% of the paper.

Contents

Chapter 1

Introduction

Structured representations such as parse trees have been central in Natural Language Processing (NLP) and Computational Linguistics (CL), often used as intermediate representations in downstream applications like machine translation (MT), question answering (QA) and document summarization. The underlying abstractions used to derive these structures have changed radically in the last three decades — expert-based models have been replaced by models learnt from examples using statistical and machine learning techniques. This paradigm shift has resulted in corpus creation efforts becoming akin to a primitive exercise towards creating basic linguistic resources for a language. In other words, tagged corpora (Francis Nelson and Kučera, 1979) have replaced expert-based automata (Beesley and Karttunen, 2003) and treebanks (Marcus et al., 1994, Abeillé et al., 2000, Böhmová et al., 2003) have replaced hand-crafted grammars (XTAG, 2001, Copestake and Flickinger, 2000, Rayner et al., 2000) – all developed to induce more accurate and robust abstractions (Charniak, 1996).[1]

Most of these were independent efforts in the last three decades, each an attempt to devise optimal representations (Johnson, 1998) suitable for the language and the task in question. These efforts were successful in creating both robust and scalable models for understanding text. Central to this success in web-scale parsing are light-weight representations used to compute shallow meaning in a sentence. These representations range from simple part-of-speech tagged sentences to tree- or graph- like dependency structures that mark grammatical functions in a sentence, e.g. the subject and object in the given sentence. The robustness of these representations and their scalability is derived from efficient algorithms that assign a *plausible representation* to the input devoid of notions like grammaticality and well-formedness of the text. These algorithms coupled with surging interest in multilinguality in NLP highlighted the need for a harmonious representation suitable for a wide range of languages. A **shared** intermediate representation that is useful for applications by abstracting language-specific variations can be seen as a parsimonious representation for the application. This is what set the stage for Universal Dependencies – a framework for cross-linguistically consistent syntactic annotation of text in a wide variety of languages. This framework uses dependency trees and directed acyclic graphs as the primary descriptions in more than 70 languages. This effort in-turn led to many advances in multilingual parsing – producing *universal parsers* and fostering research in cross-lingual parsing even when examples in that language are not available.

[1] This is now a well-accepted fact. Induced abstractions have been shown to be more machine-friendly.

But what about generating language? While above efforts have aided in the *understanding* phase of NLP i.e. in analyzing text, progress in *generation* has been largely driven by enormous advances in language modeling and data-driven techniques. These techniques have shown excellent results in a wide variety of applications – reaching "human-parity" in MT, generating human-like text (Radford et al., 2018) – indirectly contributing to the current focus on monolingual text generation. But what if one is interested in simultaneous multilingual generation? This is particularly of interest when generation originates from an abstract representations of meaning like semantic dependency structures (Abstract Meaning Representations), logical formulae or other formal structures, that need to be simultaneously translated to text in many languages – a sub-task in language generation referred in literature as surface realization. Efforts towards generation using dependency structures have recently started but however, are focussed on monolingual generation. It is not difficult to imagine why and how simultaneous multilingual generation is useful. Translations presented in more than one language serve as explanation aids, question answering systems that provide answers in multiple languages and multilingual summarization systems have been of interest to the community.

So, why has multilingual generation not seen much progress in recent years? One reason is that NLU applications rarely build abstract intermediate representations useful for generation purposes. Second, multilingual generation has been largely put in the domain of producer NLP – tasks that require faithful and grammatical rendering of meaning in languages – leading to efforts in focused domains like instruction manuals, official documents etc. Grammatical Framework is one such framework, originally created for generating multilingual documents as its central aim; past efforts have shown multilingual generation to be one of its core strengths. The primary descriptions here are abstract syntax trees (ASTs) different from the above described dependency structures used to parse documents from the web and sometimes, the web itself.

At this point it should not be difficult to foresee where this is headed – *universal models* for language understanding and generation. Related ideas have been proposed by Vauquois in the context of machine translation over 60 years ago. And indeed these ideas have seen a resurgence in recent years with attempts in MT shifting focus from translation for language pairs to multilingual MT. And while Vauquois himself may not have foreseen this, his idea of uniform models for analysis and synthesis have been shown to be feasible and widely embraced by the field. What has remained elusive in his architecture is a precise form of the interlingua that is expressive enough for both analysis and synthesis of general purpose text. But perhaps one single abstraction for both phases is an impossible task — what if it is replaced with two abstractions? One abstraction to derive representations amenable to analysis tasks and another abstraction that derives representations amenable to synthesis tasks.

This is indeed the focus and aim of the current thesis. Ongoing work in UDs have shown them to be useful representations for the purpose of NLU in a range of applications from question answering to natural language inference. But what does the "bridge" between these two abstractions look like? What are the potential applications of such a "bridge"?

1.1 Research Questions

The following research questions are discussed in this thesis:

(1) How can dependency trees be derived from abstract syntax trees (ASTs) defined by an interlingual grammar? Can this process be *reversible*, i.e. can ASTs be similarly derived for an input dependency tree? Once defined, what are the characteristics of these functions?

(2) The functions are operationalized for two independent multilingual descriptions of language, namely Universal Dependencies (UD) that use dependency trees as primary descriptions and the Resource Grammar Library of Grammatical Framework (GF-RGL / RGL) using ASTs as primary descriptions. The goal here is to both quantitatively and qualitatively understand and assess the **sharedness** in the respective frameworks while leveraging the artifacts of the respective frameworks in NLP applications.

(3) Finally, we attempt to ask the question about the appropriate role of grammars vs machine induced abstractions. Human effort involved in grammar engineering to design or add a grammar for a new language is different from the effort involved in annotating treebanks – both in terms of the sub-tasks involved in each and the time involved. Grammars as expert-designed abstractions have long been considered to be appropriate for restricted domains in NLP – this thesis looks at potential applications of such abstractions by generating synthetic treebanks used to induce dependency parsing models.

1.2 Multilingual Grammars and Representations

Interlingual grammars are one of the multilingual abstractions at the core of this thesis. An interlingual grammar consists of two parts: an abstract syntax that is **shared** across languages and a set of concrete syntaxes defined for each language separately. The abstract syntax defines a set of categories and functions, where functions correspond to rules that specify what parts are combined together. These functions abstract away from language-specific details like word order and what the parts look like: these things are specified in the concrete syntax. Figure 1.1 illustrates an interlingual grammar, a small fragment of the larger Resource Grammar used in Grammatical Framework (Ranta, 2009a, 2004b).[2]
The primary descriptions derived from these abstractions for an input text are **abstract syntax trees** (ASTs), once again a representation that is shared across the languages. The AST combined with the concrete syntax is used to derive an auxiliary representation – **concrete syntax trees** – that is language-specific and reminiscent of constituency trees and phrase-structure trees in syntax literature. The algorithm used to derive concrete syntax trees from an AST is deterministic, a **linearization**[3] into a bracketed string. Figure 1.2 shows the abstract syntax tree and the concrete syntax trees for the input sentence *the black cat sees us today* and its Swedish translation *den svarta katten ser oss idag*.

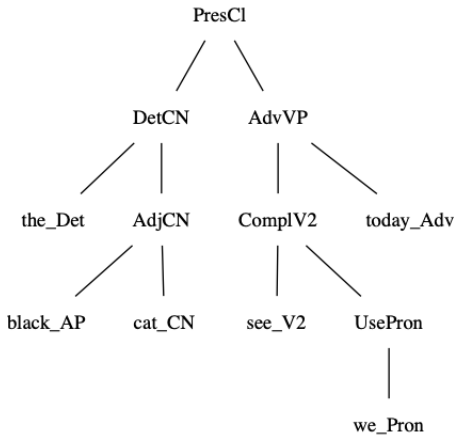
One of the central areas in computer science where these representations have have been studied and applied is compiler development. Programming language compilers use ASTs as an intermediate representation, sharing the representation across several

[2] Any function with a definition written as $f : C_1 \rightarrow C_2 \rightarrow ...C_n \rightarrow C$; can be rewritten as a context-free rule f. $C ::= C_1C_2...C_n$. The former is a notation used across this thesis.

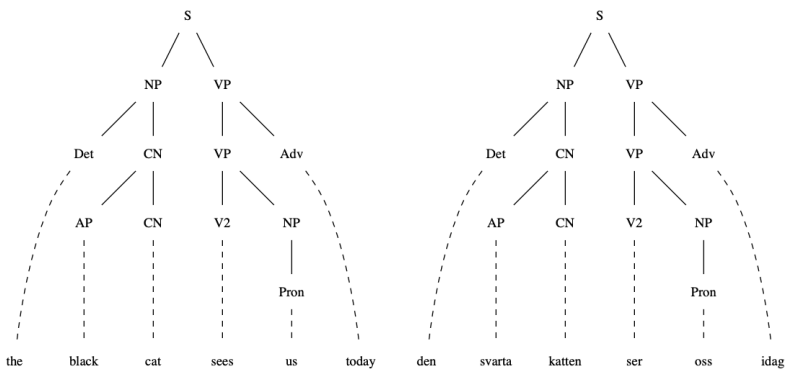
[3] Linearization is the reverse process of parsing i.e. to generate or recover the input sentence from an abstract syntax tree.

```
cat
    S    ;                      -- sentence
    NP   ;                      -- noun phrase
    VP   ;                      -- verb phrase
    AP   ;                      -- adjectival phrase
    CN   ;                      -- common noun
    Det  ;                      -- determiner
    V2   ;                      -- transitive verb
    Pron ;                      -- pronoun
    Adv  ;                      -- adverbial modifier

fun
    PresCl  : NP -> VP -> S ;   -- predication: (the cat)(sees us)
    CompAP  : AP -> VP ;        -- copula:
    ComplV2 : V2 -> NP -> VP ;  -- complementation: (sees)(a cat)
    DetCN   : Det -> CN -> NP ; -- determination: (the)(cat)
    AdvVP   : VP -> Adv -> VP ; -- modification: (see)(today)
    AdjCN   : AP -> CN -> CN ;  -- adjectival modification: (black)(cat)
    UsePron : Pron -> NP ;      -- use pronoun as noun phrase: (us)

    see_V2    : V2   ;          -- see/sees
    the_Det   : Det  ;          -- the
    black_AP  : AP   ;          -- black
    cat_CN    : CN   ;          -- cat/cats
    we_Pron   : Pron ;          -- we/us
    today_Adv : Adv  ;          -- today
```

Figure 1.1: An abstract syntax for a fragment of GF-RGL.



(a) Example of an abstract syntax tree



(b) Example of concrete syntax trees in English and Swedish

```
(S (NP (Det the)(CN (AP black)(CN cat)))(VP (VP (V2 sees)(NP (Pron us)))(Adv today)))
(S (NP (Det den)(CN (AP svarta)(CN katten)))(VP (VP (V2 ser)(NP (Pron oss)))(Adv idag)))
```

(c) Concrete syntax trees as bracketed linearization

Figure 1.2: Primary and auxiliary descriptions derived from an interlingual grammar

source and target languages[4] , only changing the first step of parsing and the last step of code generation. This intermediate representation is used in semantic analysis, for example to type-check programs and make sure that the code is not erroneous. But these abstractions can be described as a combination of two well-studied aspects in linguistic and computational grammars that have been discussed in CL literature – **multi-stratal** abstractions and **synchronous grammars**.

The first of these characterizations referred to here as multi-stratal abstractions can be understood by following the contributions of Curry (1961). Curry himself never uses the words "multi-stratal" or "abstractions", he only proposes that a logic system describing language has two distinct aspects – a **phenogrammatical** and **tectogrammatical** aspect.[5] The phenogrammatical aspect describes how a linguistic phenomenon is realized in the sentence, while the tectogrammatical description refers to a higher level of abstraction – the underlying structure (Muskens, 2010). The idea of abstraction above language-specific details like word order in grammars [6] has long been appealing in computational grammar formalisms like Head-Phrase Structure Grammar (HPSG), Lexical Functional Grammar (LFG) and Tree Adjoining Grammars (TAGs). For example, Vijay-Shanker and Weir (1995) describe a variant of Tree Adjoining Grammars, TAG(ID/LP) that factorizes word order (referred to as *linear order*) information away from tree information (also called *immediate domination* relations). Similar distinction to tectogrammatical and phenogrammatical in CL/NLP literature is made using the terms **deep** and **surface** syntax.

Now the above discussion is too abstract for developing multilingual NLP applications. In NLP, crosslingual or multilingual equivalence is characterized using an corpus of translations, aligned at sentence level and referred to as parallel corpora and multilingual corpora.[7] In other words, multilingual corpora are assumed to be *implicitly equivalent* – irrespective of whether they are lexically or semantically or pragmatically equivalent. The corpora are used to induce **parallel abstractions** (without an abstract syntax) in the form of synchronous context-free grammars (or **syntax directed transducers** (Lewis and Stearns, 1968, Aho and Ullman, 1969a) as they were originally called), used widely in MT. Algorithms to induce basic alignment units in parallel corpora (Vaswani et al., 2012) and different variants of synchronous grammars, for example, Inversion Transduction grammars (Wu, 1997), Hierarchical grammars (Chiang, 2005, 2007), synchronous TAGs (Shieber, 2014) and other abstractions (Nederhof and Vogler, 2012) have been proposed and shown to scale to large parallel corpora (Zhang et al., 2008, Pauls et al., 2010). It is trivial to see how the concrete syntax trees in Figure 1.2 can be modeled using a synchronous CFG, without defining an explicit abstract syntax.[8]

But these abstractions mostly work with two languages, there has been very limited

[4] In Compiler theory, the source language refers to a high-level programming language and the target language refers to a low-level system code. Unlike in NLP, the set of source languages do not overlap with the set of target languages.

[5] The reader is cautioned against drawing similarities between the proposal of Curry (1961) and that of Sgall et al. (1986) referred to as "multi-stratal" in literature on dependency grammars (de Marneffe and Nivre, 2019).

[6] Curry himself outlines this for Categorial Grammars of Lambek (1968).

[7] The reader should be aware of the terms "bitext" and "multi-texts" used as synonyms for parallel and multilingual corpora following Melamed and Wang (2004).

[8] The difference between a parallel and an interlingual abstraction as defined here, is the explicit definition of an abstract syntax corresponding to the multilingual abstraction. Designing an abstract syntax for a given synchronous grammar is not always straight-forward, especially when the grammars of the source and the target languages follow different annotation schemes.

work on multilingual abstractions that work with more than two languages, notably the formalism of **multi-text grammars** proposed in Melamed and Wang (2004), Melamed et al. (2004) is an generalization of synchronous CFGs to work for arbitrary number of languages. More recently, Neubig et al. (2015) propose an analogous framework to Hierarchical grammars that work for more than one target languages. A shortcoming of using these formalisms as multilingual abstractions is they do not scale well with increasing number of target languages.

Grammatical Framework (GF) is a framework to implement interlingual grammars (Ranta, 2004b). The abstract syntax corresponds to the tectogrammatical and the concrete syntax to the phenogrammatical description in Curry's terminology. The concrete syntax of a language in GF has the same expressivity as Parallel Multiple Context-Free Grammars (PMCFG) as shown in Ljunglöf (2004). Parsing in GF is polynomial in sentence length as shown in Angelov (2011). Angelov (2011) also defines a probabilistic variant of GF grammars by defining a distribution on the abstract syntax – this makes the probability information usable for disambiguation across as many languages as there are concrete syntaxes defined. An additional property of GF grammars relevant to the current discussion is that they are **reversible**, i.e. the same grammars can be used for both parsing and generation of text. Also worth mentioning is that the abstract syntax in GF as a stand-alone description is similar to a context-free grammar.[9]

An orthogonal representation to the ones discussed above is a **dependency structure**, that has its origins in descriptive linguistics. One common feature of these representations is that the structure is comprised of asymmetric relations between words in a sentence. But before these representations are defined, the notion of *abstraction* for these representations should be clarified.
Computational frameworks and linguistic theories have been studied for dependency analysis in linguistics, with varying inherent assumptions about the adequacy of dependency analysis for languages (Sgall et al., 1986, Debusmann, 2000). Parsing algorithms for these representations using grammars have also been developed by encoding dependency grammars as variants of context-free grammars and using the CYK or Earley parsing algorithms either in their original or in a modified form. However, it was the development of efficient data-driven methods for parsing into these representations combined with increased emphasis on robustness that made these representations mainstream in NLP/CL. The statistical models used in these data-driven methods do not induce an explicit grammar – instead the linguistic regularities learnt from annotated corpora are implicit in the model behavior. Hence, abstraction in the context of dependency syntax can be any one of the following: (a) a *coherent* description of how all linguistic phenomena are analyzed in the language(s). (b) an *encoding* in the form of a formal grammar, induced using the linguistic examples in the annotated corpora. (c) a statistical model without an explicit grammar, induced only for parsing new text. In the remainder of the discussion here descriptions of how languages are analyzed, otherwise called **annotation guidelines**, will be used as the underlying abstraction behind these representations. The guidelines are designed by linguistic and computational experts and in turn are used by human annotators to create treebanks. A full discussion about the landscape of dependency syntax and parsing is well beyond the scope of this thesis, the reader is recommended to Tesnière (2015), Kübler et al. (2009) for an interesting discussion. The rest of the discussion is also limited to the

[9] The set of categories in the abstract syntax can be partitioned into set of terminals and non-terminals by introducing variants for ambiguous categories C_{term} and $C_{nonterm}$.

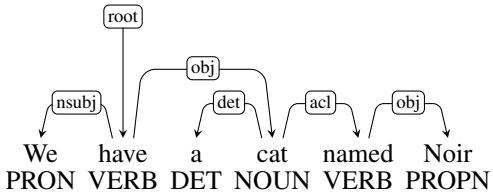


Figure 1.3: Example dependency tree following the UDv2 annotation

extent of design choices made in the Universal Dependencies (UD) framework. The framework has its roots in multiple independent efforts towards developing a consistent multilingual annotation scheme (McDonald et al., 2013, de Marneffe et al., 2014, Rosa et al., 2014) and has undergone a revision to the original scheme – UDv1 and UDv2.

The primary descriptions used in UD are a class of dependency graphs that can always be separated into a **basic dependency tree** and set of edges grouped together as **enhanced dependencies**. The edges are directed from a **head** to a **dependent** (also referred as **modifier** in some literature). The nodes in these graphs correspond to *words* in the sentence and the edges are labelled using grammatical relations. This set of relations (called the core *label set*) used contains 41 labels – ranging from labels marking the subject, object and predicate of a basic clause to loosely defined relations like `list`, `goeswith` used to analyze ungrammatical text. These were revised to 37 relations in UDv2. At the node level, words are tagged with part-of-speech tags and morphological features in the form of an attribute value vector. Figure 1.3 illustrates a prototypical UD structure.

With this, the question of what is language-independent (i.e. abstract) and what is language-dependent (i.e. concrete) in this abstraction can already be answered at least *partially*. The abstract description includes the part-of-speech tags and grammatical labels from the core label set in addition to the choice of direction for different edges[10] The labels and the morphological feature descriptions can be extended for marking language-specific details, for example, the label `obl:tmod` is used to optionally mark temporal expressions in the sentence.

It is worth noting that the fuzziness about the boundary between abstract and concrete is a feature of the framework – UD classifies all information into **core** and **optional** – *universal* relations and feature value descriptions are defined and all languages are encouraged to use them. However, if the realization of a construction in a particular language is different, subtypes of the universal relations can be used to label the distinct realization in that language. This indirectly introduces the possibility that label subtypes can be exploited to selectedly label abstract information in only a few languages.[11] Another distinct feature of the UD scheme is to select semantic words as heads (content-head choice) as opposed to syntactic words (functional head choice). This is motivated by cross-lingual reasons, this choice allows minimal changes to the tree structure across a wide range of languages. For example, not all languages realize determiners using a word – like Swedish where the distinction between definite and indefinite nouns is realized using different word forms of the noun – as such,

[10] Direction here is implied in the *vertical* sense i.e. in terms of the tree structure, and not in the *horizontal* sense i.e. in terms of word order in the sentence.

[11] There are 240 subtypes to the 37 labels defined in the UDv2 scheme, 158 of which are only found in one language.

marking the determiner as the dependent of the noun *when present* is more consistent cross-linguistically. A similar reasoning can also be applied for copula verbs (*the cat is black*), Russian for example does not always use a copula verb.

1.3 Abstract Syntax trees and Dependency trees

As already mentioned, the first research question addressed in this thesis is to derive dependency structures from interlingua grammars. A **dependency configuration** corresponding to the grammar is defined as a specification that defines for each function an **anchor**, using the keyword `head`. In the case of syntactic interlingua, the anchor corresponds to the syntactic head and the relations correspond to grammatical roles and in the case of semantic interlingua, the relations correspond to thematic relations for a semantic head. Example of configurations for the grammar in Figure 1.1 is shown in Figure 1.4. Each line in the configuration starts with the name of the function in the grammar, followed by an assignment of labels to each argument of the function. The labels correspond either to the anchor or to one of the labels defined in the annotation scheme. The anchor for unary functions (e.g. `CompAP`, `UsePron`) is the degenerate case and may be omitted.

```
PresCl   nsubj  head     -- NP -> VP -> S
ComplV2  head   obj      -- V2 -> NP -> VP
DetCN    det    head     -- Det -> CN -> NP
AdvVP    head   advmod   -- VP -> Adv -> VP
AdjCN    amod   head     -- AP -> CN -> CN
```

Figure 1.4: Dependency configurations for the grammar fragment from RGL. Also shown as comments are the rules in the grammar.

The configuration specifies one anchor among the arguments of each function in the grammar. The rest of the arguments can be assigned a default label `dep`. The resulting dependency trees obtained using this configuration are unlabelled i.e. trees with directed edges always labelled using `dep` from the anchor to the head of the arguments. The configuration shown specifies grammatical roles for the arguments with respect to the anchor and the arguments of a function. In this example, the subject and the object of a clause are marked using the labels `nsubj` and `obj`. Similarly, determiners in noun phrases like “the”, “some” are marked using the label `det` and adjectives when used are marked using `amod`. Adverbial modifiers that work with verb phrases are assigned the label `advmod`.[12] The use of an external configuration provides flexibility in the choice of heads and the specific labels used in the target annotation scheme, which can either be subject to revisions or multiple target schemes. We use this configuration as a starting point for deriving dependency trees from ASTs (**ast2dep**) and to build ASTs corresponding to a dependency tree (**dep2ast**).
It is worth mentioning that defining the dependency configuration classifies the functions in the grammar into two classes: **exo-centric** and **endo-centric**. Endo-centric functions are recursive rules where the category of the anchor marked `head` is the same as the value category. Exo-centric functions are all functions that are not endo-centric.

[12] This is revisited later in Section 1.4.1.

In the grammar shown in Figure 1.1, the functions `AdvVP` and `AdjCN` are endo-centric and all other functions are exo-centric.

1.3.1 ast2dep: From Abstract Syntax to Dependency trees

Algorithm

The algorithm to derive a dependency tree for a given abstract syntax tree (**ast2dep**) is a deterministic many-to-one mapping by design, hence the dependency tree is a lossy representation of the AST. The algorithm works in two steps: a recursive labelling step that is a breadth-first traversal over the AST followed by tree derivation step. The labelling procedure marks each argument of a function in the AST according to the configuration. Given this annotated abstract syntax tree T for the word sequence S, the dependency tree is derived as follows:

1) For each word w in the sentence, find the function f_w forming its smallest spanning subtree in the AST. The smallest spanning subtree of a word is the subtree whose top node is the function whose linearization generates that word.

2) Trace the path up from f_w towards the root until a label l is annotated. From the node immediately above l, follow the spine – the unlabelled path of edges – down to another leaf y. y is the head of w with label l.

Figure 1.5 shows the parse tree for the English sentence *the black cat sees us today* and its Swedish translation *den svarta katten ser oss idag*. The nodes in the parse tree are decorated with the abstract functions and the edges with the dependency labels. Arrows are added to the edges, to indicate the direction of the edges in the resulting dependency tree. Each path in this representation – when collapsed into a single edge – matches the edge in the resulting dependency tree. Part-of-speech tags corresponding to the words are obtained using a **category configuration**, a lookup table mapping the lexical categories to their respective tags (shown in Figure 1.6).

An intermediate representation **abstract dependency tree** (ADT) is defined as a directed **unordered** dependency tree defined on the lexical functions (0-argument functions in a grammar) in the AST with labels on the edges, shown in Figure 1.7. Note that the order of the nodes in this dependency tree does not reflect the surface order of the words in the sentence, nodes are shown here in pre-order traversal. The ADT is not explicitly constructed by the algorithm – it is however a useful multilingual abstraction over the dependency trees defined by an interlingual grammar.

Completeness of configurations

The labelling step using the configurations defined above would be sufficient if each word in the tree had a corresponding lexical function in the grammar. This is not always the case since words can be introduced only in the concrete syntax specific to a language – these words are called **syncategorematic** – in which case, the labelling step assigns a default label `dep` to each of them. For example, the copula verb "is" in the sentence *the cat is black* is a syncategorematic word, introduced only in the concrete syntax of English corresponding to the function `CompAP` without a corresponding category in the abstract syntax. Similarly, the concrete syntax of Swedish introduces the translation equivalent of the copula "är". In order to label these syncategorematic

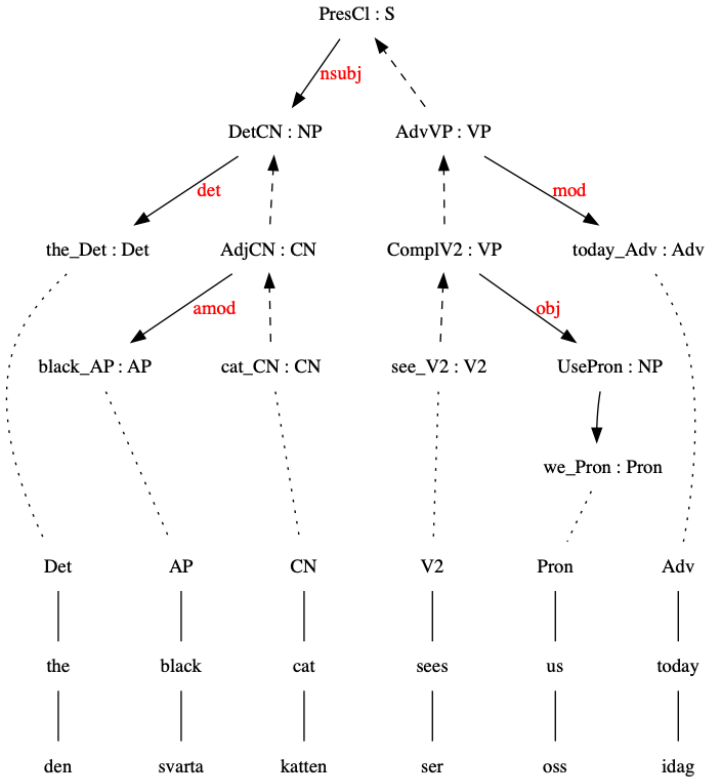


Figure 1.5: Parse tree decorated with abstract syntax functions and dependency labels

```
Det     DET
AP      ADJ
CN      NOUN
V2      VERB
Pron    PRON
Adv     ADV
```

Figure 1.6: Category configuration mapping the categories to part-of-speech tags

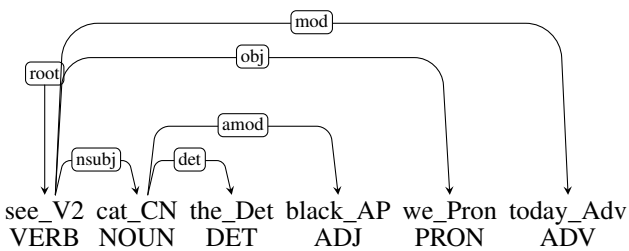


Figure 1.7: An example of a abstract dependency tree

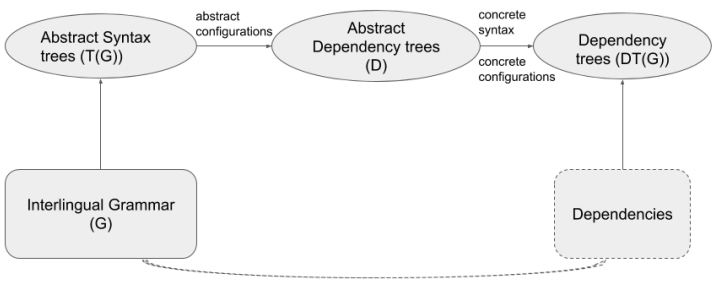


Figure 1.8: Setup of `ast2dep` and intermediate abstractions defined

words, the configurations are extended with **concrete configurations**, defined separately for each language. Shown below are the concrete configurations corresponding to the copula verb in both English and Swedish.

```
CompAP  head  {"is"}  cop  head   -- English
CompAP  head  {"är"}  cop  head   -- Swedish
```

Each rule in the concrete configuration specifies a relabelling operation. The relabelling operation in this instance, renames the label on the edge from the head (*black* in this example) to the copula verb "is" as `cop`.

The configurations defined previously (shown in Figure 1.4) are hereafter referred to as **abstract configurations**. Configurations refer to the union of both abstract configurations and concrete configurations when available for a language. The combination of abstract and concrete configurations are sufficient to derive a fully labelled dependency tree corresponding to an AST. The **domain** of dependency trees at this point is restricted by the set of ASTs defined by the grammar.[13] The setup of `ast2dep` is summarized in Figure 1.8 – in order to derive dependency trees for a new language, both the concrete syntax and the concrete configurations defined for that language are necessary.

1.3.2 dep2ast: From Dependencies to Abstract Syntax trees

The derivation from ASTs to dependency trees in `ast2dep` is deterministic, because it is the linearization of an AST to an annotated string representing the dependency tree. On the other hand, `dep2ast` is a non-deterministic search with a one-to-many relation. By definition, **dep2ast** accepts a dependency tree, builds an abstract dependency tree and returns the set of ASTs that can be translated back to the original dependency tree using `ast2dep`.

[13] A different formulation of this is the tree language of dependency trees generated using `ast2dep` depends on the tree language of the interlingual grammar.

Algorithm

The algorithm works in two steps: lexical annotation (`dep2adt`) followed by syntactic annotation (`adt2ast`).
The **lexical annotation** step builds an **unordered dependency tree** from an input dependency structure[14] where each node is labelled by a label, lemma, POS tag and morphological features. After the tree is built, each lemma is replaced with a lexical function of category C using a lexicon and the category configuration. The resulting data structure is an abstract dependency tree.
The **syntactic annotation** step annotates the ADT recursively with applications of syntactic combination functions. At each step, endo-centric functions are applied (when available) before exo-centric functions are applied using the ASTs corresponding to the sub-trees in the ADT. The algorithm is a depth-first postorder traversal, completed when all nodes in the ADT are covered, with the final result being the ASTs built in the root node of the ADT.

Restrictions of `dep2ast`

The algorithm described above works only when an ADT can be built for every input dependency tree using abstract configurations (such as shown in Figure 1.4). From the discussion on `ast2dep`, we already know that this is not always possible: frequently due to the presence of syncategorematic words. In order to address this, we extend the configuration with what are called **helper categories** and **helper functions**. These helper categories are used to postulate an abstract syntax category for each syncategorematic word and the helper function uses these helper categories. Table 1.1 shows these extensions required to handle the `CompAP` function which introduces the copula. In this instance, only the definition of the helper category Cop- is language-specific. The helper function that uses this category is shared across the three languages, but is only applied if the definition of the helper category is available for the language in the first place. Figure 1.9 shows the abstract dependency tree (ADT) that is the output of the lexical annotation phase in `dep2ast`. These are different from the ADTs in `ast2dep` – henceforth called **quasi-abstract dependency tree** – ADTs that use helper categories not defined in the grammar.

helper category	Cop-	AUX	lemma=be	English
	Cop-	AUX	lemma=vara	Swedish
	Cop-	AUX	lemma=olla	Finnish
helper function	`CompAP-`	Cop- $\rightarrow$ AP $\rightarrow$ VP	`cop head`	quasi-interlingua
function definition	`CompAP-`	λ cop,ap $\rightarrow$ `CompAP` ap		quasi-interlingua

Table 1.1: Configurations added for handling copulas in 3 languages. Function definitions shown are syntactic sugar to the syntax of `dep2ast`

The abstract syntax of an interlingual grammar (G) is characterized using a 3-tuple (C,F,S) where C corresponds to the categories, F corresponds to the functions and S is the start symbol in the grammar (Angelov, 2011). Using a similar characterization, the extended configurations for the same grammar can be written as (EC,EF,S) coupled with $defs$ where EC is the union of the categories defined in the grammar C and the helper categories defined in the configurations. Similarly, EF correspond to the union

[14] The input can be a dependency tree or a dependency graph as defined in UD.

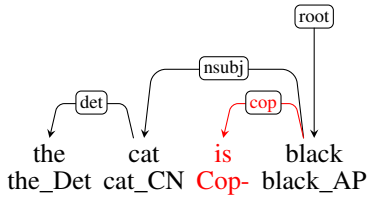


Figure 1.9: ADT for the sentence NP *the cat is black* using helper categories. Also an illustration of a quasi-ADT.

of the functions defined in the grammar F and the helper functions defined in the configurations. The function definitions ($defs$) define how helper functions can be eliminated by applying the function to result a valid AST defined by the grammar. The function definitions are checked for **type consistency** i.e. that the category of the tree from applying helper functions matches the type of the expression provided in the definition of the helper functions. This type consistency verification is an approximation for checking that the grammar defined by the configurations generates the same set of ASTs as the original grammar.[15]

The extended configurations define an approximate grammar for a given interlingual grammar, where all syncategorematic words are eliminated using the helper categories. The helper functions and definitions are called **quasi-interlingua** to emphasize the multilinguality i.e. these are **shared** similarly to the functions defined in the abstract syntax, unlike the definition of helper categories. A similar distinction is drawn in Croft et al. (2017) between **constructions** which are language-independent and **strategies** which can be language-dependent, however strategies refer to only multilingual phenomena e.g. the use of copula is a strategy.

It is interesting to note that the concrete configurations used in `ast2dep` are not equivalent to the helper functions defined in `dep2ast`. In the case of `ast2dep`, the configurations are clearly factored into language-independent (abstract) and language-dependent (concrete) configurations. However, in `dep2ast` the set of helper functions are used to handle both strategies and language-specific cases. The functions in the grammar that trigger both the concrete configurations and helper functions are however the same: the treatment of these however differs based on the direction of the translation between ASTs and dependency trees.

Ambiguity and Spurious ambiguity

This basic algorithm is non-deterministic, though the ambiguity at this point is primarily of one of two types:

1) functional ambiguity: when the abstract syntax has two or more functions with the same configuration, then the **under-specification** in the underlying dependency representation triggers ambiguity in the ASTs

2) structural ambiguity: when the input dependency tree contains more than one endocentric configuration or cyclic exocentric configurations, the ambiguity is triggered because the order in which these functions are applied results in ambiguous ASTs.

[15] In other words, both these grammars have the same **strong generative capacity**.

Ambiguity should be defined in the context of `dep2ast` which is different from ambiguities encountered using the concrete syntax and a parser. A sentence is ambiguous for a grammar G if the parser returns multiple ASTs corresponding to the sentence. `dep2ast` returns the ASTs as generated by the parser, but also returns additional spurious ASTs – an artifact of using the ADT instead of the sentence to build the AST. Factoring the word order information from the data structures in the search keeps the configurations multilingual, however, they also introduce ambiguity which is not always available in the concrete syntax of the language.

Now, to define functional ambiguity, consider the case of two functions in the grammar that share the same configurations.

```
fun1 : C1 -> C2 -> C ;  head  mod
fun2 : C1 -> C2 -> C ;  head  mod
```

The two functions in the grammar correspond to different linguistic phenomena and may hence have different semantics, in which case, this is a genuine case of ambiguity with respect to the abstract syntax: i.e. that the mapping to two ASTs is valid. However, these are more a by-product of under-specification in the dependency scheme, rather than genuine ambiguity. For example, the phrases *two levels* and *level two* are indistinguishable with respect to their dependency trees in UD (*two* is marked as a dependent of *level(s)* using `nummod`) and hence the ASTs resulting from the fragment corresponding to *level two* can be linearized back to both *level two* and incorrectly *two levels*. In order to address functional ambiguity, we specify morphological constraints on top of the configurations to remove spurious ambiguity. In cases when morphological constraints are inadequate for disambiguation, multiple ASTs are returned.[16]

A more frequent case of ambiguity is the case of structural ambiguity: i.e. when an endocentric function can be applied in different orders to cover the same dependency subtree. In the example *men, women, children*, the ADT is indistinguishable from the ADT corresponding to *men, children, women*. Similarly, the phrase *big black cat* is ambiguous without the concrete syntax: the order in which the adjectival modification is carried out can result in ASTs that can be linearized to both *big black cat* and *black big cat*. However, the phrase *grande famille française* (big French family in French) is also ambiguous but both the ASTs are linearized to *grande famille française*. In order to address structural ambiguity, we define a a **normalized** ADT. A normalized ADT is defined as an ordered dependency tree in which children are ordered based on the distance with respect to its parent in the tree. In the above examples, *black* is placed closer to *cat* than *big* in the ADT while both *grande* and *française* are equally close. The lexical annotation step in `dep2ast` builds a normalized ADT in addition to building the ADT and the syntactic annotation step uses a left associativity property in the case of endocentric functions. This is an approximation in the syntactic annotation step – one that eliminates both spurious ambiguity and the need for a generalized function application step, while keeping the algorithm simple.

[16] This can also happen if variations of existing functions are introduced in the abstract syntax to model other linguistic universals, e.g. focus is optionally marked in the abstract syntax using different set of functions.

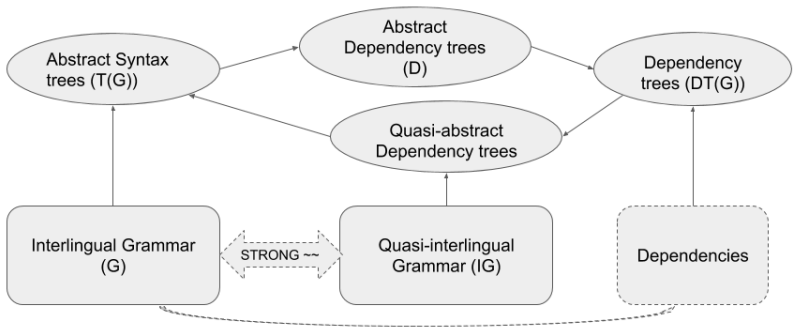


Figure 1.10: Abstractions and representations underlying `ast2dep` and `dep2ast`

1.3.3 Expressivity and Limitations of `ast2dep` and `dep2ast`

The transducers for both `ast2dep` and `dep2ast` in the form presented here are limited in the space of possible dependency structures that can be *covered*. Figure 1.10 shows the primary, auxiliary and intermediate representations derived for an interlingual grammar. In order to better understand this, consider an example function defined in the abstract syntax as `fTernary: C1 -> C2 -> C3 -> C`. The function takes 3 arguments of distinct categories (C1, C2 and C3) and builds a result of category C. If we assume that each of these arguments can be represented by a node in the dependency tree (HC1, HC2, HC3), the number of dependency trees that can be generated are 9: 3 trees of height one and 6 trees of height two. These are visualized in Figure 1.11.[17] `ast2dep` using an abstract configuration defined for the function `fTernary` can only generate the 3 dependency trees of height one. This is also the case for `dep2ast`, where the AST can be recovered from the one-level trees. The implicit assumption defined by the configurations is that each function in the abstract syntax has a unique head and that arguments to the function can be assigned dependency labels limits the expressivity of the transducers.

When operationalizing these transducers for the GF-RGL and the UD scheme, these limitations are relaxed by extending dependency configurations to generate other dependency trees as defined by UD. The extensions necessary to cover the target scheme do not necessitate generalized transducers, relevant extensions in the context of Universal Dependencies are discussed next.

1.4 GF-RGL and Universal Dependencies

The implementation of interlingual grammars central to this thesis is the Resource Grammar Library (RGL / GF-RGL), which consists of concrete syntaxes for over 30 languages for a shared abstract syntax (Ranta, 2009b). These grammars are expert-

[17] Note that word order here is not reflected again since we are talking about ADTs.

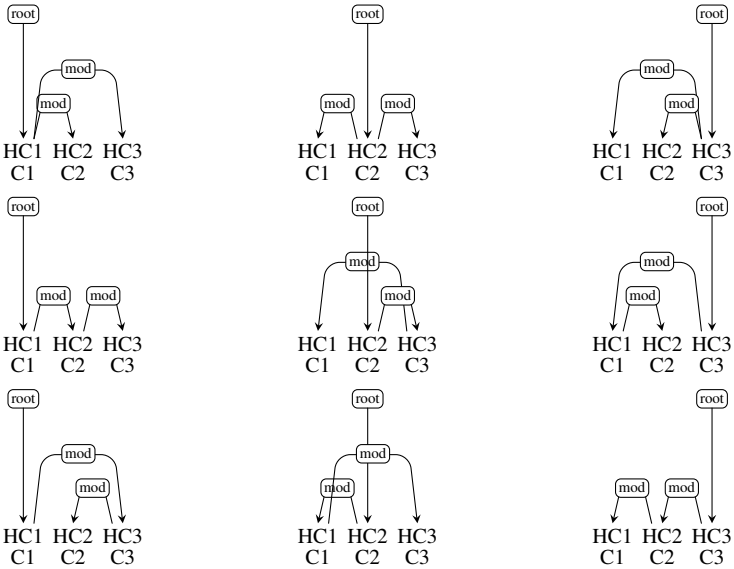


Figure 1.11: Possible dependency trees for 3 nodes. The first row corresponds to one-level trees and are covered by the configurations of `ast2dep` and `dep2ast`.

designed and designed to be formal specifications of the morphology and syntax for languages in the library. This collection of grammars is the primary linguistic artifact central to GF – the RGL is used as a software library of syntax (Ranta, 2009a) to develop **application grammars** used in *multilingual* applications (Ranta et al., 2010, Dannells et al., 2013, Angelov et al., 2014). The design of the RGL i.e. the abstract syntax of RGL has remained stable for well over 12 years. Concrete syntaxes for new languages is ongoing work (Lange, 2017, Papadopoulou, 2013, Paikens and Gruzitis, 2012) and more recently Listenmaa (2019) proposed methods to find errors in these libraries. More recently, there has also been work on connecting the RGL with external linguistic resources like FrameNet (Dannells and Gruzitis, 2014, Gruzitis and Dannélls, 2015) and WordNet (Angelov and Lobanov, 2016, Virk et al., 2014). Bernardy and Chatzikyriakidis (2017) have shown how the RGL can helpful in Natural Language Inference (NLI) applications using formal semantics. Parsing in GF is the inverse of the pure linearization rules specified in the concrete syntax of a language (Angelov, 2009). In the current thesis, reversibility of the GF grammars can be used to linearize both sentences and dependency trees into multiple languages from ASTs.

The UD framework on the other hand develops annotated data a.k.a treebanks, the current version[18] contains treebanks for over 70 languages. The treebanks are directly used to train dependency parsers – universal, multilingual and cross-lingual – by applying machine learning and statistical methods to induce models for parsing text. Applications built using UD include semantic parsing using logical forms (Reddy et al., 2016).

The guidelines specifying the UD schema used in annotation are an encoding of the underlying UD grammar, however the encoding is descriptive and not done

[18] The latest version at the time of writing this is v2.3. The next revision UDv2.4 is expected to contain 83 languages.

property	UD	GF
primitive descriptors	dependency trees	abstract syntax trees
linguistic resources	treebanks	grammars
parser coverage	robust	brittle
parser speed	fast	slow
disambiguation	context-sensitive	context-free
multilingual parsing	easy	difficult
semantics	loose	compositional
generation	non-deterministic	accurate
multilingual generation	difficult	easy
new language	low-level work	high-level work

Table 1.2: Complementary properties of GF and UD strengths above the dividing line.

using a formal grammar. This grammar can be contrasted against the interlingual grammars available in GF, using specification at different linguistic levels, in this case both morphology and syntax. Table 1.2 shows a high-level comparison about the current state of GF and UD excluding ongoing work. For example, context-sensitive disambiguation models defined on abstract syntax in GF has been studied and shown to work, though the widely used disambiguation model in GF is still context-free. Since the two crosslingual efforts have largely been independent, there are differences in ways of thinking when handling some linguistic phenomena. One alternative in these cases is to redesign the RGL to match the target UD scheme.[19] Another alternative is to extend the dependency configuration, thus retaining the algorithms used in `ast2dep` and `dep2ast` while also improving the correspondence between the two multilingual descriptions of language.

1.4.1 gf2ud: Extensions to ast2dep

The deterministic nature of the mapping between the ASTs and UD trees in **gf2ud** means that the abstraction levels between the RGL and UD are similar i.e. both the grammar and the annotation scheme makes the same distinctions. This is not necessarily always the case, which can be illustrated using the example of modifiers.

AdvVP	VP → Adv → VP	head advmod/obl/advcl
AdvS	S → Adv → S	head advcl
AdvAP	AP → Adv → AP	head advmod/obl/advcl
AdvCN	CN → Adv → CN	head nmod
AdvNP	NP → Adv → NP	head nmod

Table 1.3: Functions in RGL for modification. All the functions are recursive rules with endo-centric configurations

Adverbial modifiers in GF are grouped into two classes, functional modifiers and content modifiers. Functional modifiers like AdN, AdA and AdV are used to modifiy numeral expressions, adjectives and verbs respectively. Content modifiers are grouped

[19] UD has undergone one revision from UDv1 to UDv2 in the last 3 years. The current version UDv2 is expected to be stable for next few years.

under a single category Adv, used to modify noun phrases (NPs), verb phrases (VPs) and sentences (Ss). All these categories are characterized by recursive functions that combine the modifiers with different categories. UDv2 on the other hand uses four different labels – `advmod`, `obl`, `nmod`, `advcl` – to map the head of the modifier to its respective label.[20] Functions that use the functional modifiers have a straightforward labelling, the `advmod` label. Table 1.3 shows the different functions that use the content modifier Adv, it can be seen that the mapping in the case of `AdvVP` and `AdvAP` is ambiguous. The precise mapping relies on the internal structure of the modifier, when the Adv contains a VP modifier, it is mapped to a `advcl`, `obl` when it contains a prepositional phrase and `advmod` in all other cases.

In order to induce these finer distinctions, **non-local configurations** are defined, that specify additional context in which certain configurations are applied. Shown below is the handling for the function `AdvVP`, similar extensions are necessary for handling the function `AdvAP`. The first three mappings are applied only when the internal structure of the Adv matches the specified tree-patterns.

```
AdvVP ? (GerundAdv ?)  head advcl  -- GerundAdv : VP -> Adv
AdvVP ? (PrepNP ?  ?)  head obl    -- PrepNP : Prep -> NP -> Adv
AdvVP ? (PrepCN ?  ?)  head obl    -- PrepCN : Prep -> CN -> Adv
AdvVP                  head advmod -- AdvVP  : VP -> Adv -> VP
```

Similarly, the non-local configurations are also defined on the concrete syntax to handle mismatches specific to a language.

Coordination poses a unique problem in `gf2ud` and `ud2gf`, for multiple reasons. Figure 1.12 shows two fragments of GF grammars for coordination of noun phrases (NPs). The first fragment – illustrating how coordination is implemented in GF-RGL – defines a `ConsNP` function that takes two arguments: a NP and ListNP and prepends the NP to the existing list of NPs. ListNP is the category in the grammar used to model a list of NPs of arbitrary length. Similar List* categories are defined and used for other categories in the grammars. The second fragment defines an analogous `AppendNP` function that appends the NP to the existing list of NPs, to the end of the list. In other words, the RGL fragment uses left-branching to build ASTs of ListNP category and the alternative uses right-branching. Both fragments cover the same set of utterances, and the choice of one branching over another was a grammar engineering choice and not motivated for linguistic reasons. In the right-branching AST, local configurations result in a flat dependency tree with the first item of the list as head and all other items attached as dependents using the `conj` label. In the left-branching AST defined by the RGL, similar local configurations result in a chain of head and `conj` edges. This is shown in Figure 1.13.

The dependency tree corresponding to the left-branching AST derived using the implementation in RGL generates non-projective edges (i.e. crossing edges in the dependency tree). In comparison the right-branching AST generates a flat dependency tree – with the first NP as the head and edges to all other NPs marked `conj` – without any crossing edges. This is the annotation used in UDv1 for marking co-ordination.

1.4.2 ud2gf: Extensions to dep2ast

The `dep2ast` method presented in Section 1.3 abstracts away from certain practical details encountered in addressing Universal Dependencies. The extensions in **ud2gf** described below are primarily designed to address them.

[20] In UDv1, these are mapped to three labels, `advmod`, `nmod` and `advcl`.

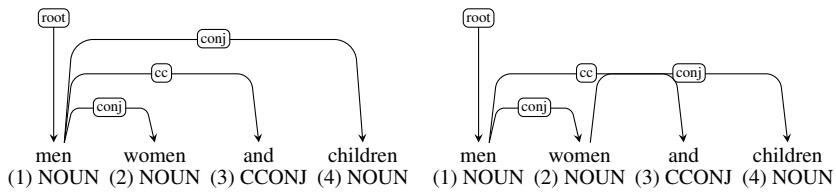
```
BaseNP : NP -> NP -> ListNP ;          BaseNP    head  conj
ConsNP : NP -> ListNP -> ListNP ;      ConsNP    head  conj
ConjNP : Conj -> ListNP -> NP ;        ConjNP    cc    head
```

(a) Co-ordination of noun phrases (NPs) as implemented in GF-RGL and corresponding UD configuration.

```
BaseNP   : NP -> NP -> ListNP ;        BaseNP    head  conj
AppendNP : ListNP -> NP -> ListNP ;    AppendNP  head  conj
ConjNP   : Conj -> ListNP -> NP ;      ConjNP    cc    head
```

(b) An alternative implementation for co-ordination NPs and corresponding UD configuration.

Figure 1.12: Co-ordination in GF.



(a) UD tree for right-branching AST.

(b) UD tree for left-branching AST.

Figure 1.13: Two possible dependency trees for NPs using different grammars for co-ordination.

(1) **Robustness** using Backup grammars: The first extension is designed to simultaneously address two problems with `dep2ast` – incomplete coverage in grammars and recovery from parser errors.

(2) **Temporary categories**: Extensions to the expressivity of `dep2ast` are required to handle specific linguistic constructions in UD. One such extension is the definition and use of temporary categories to the configurations which behave like aliases to existing categories in the grammar. The result is still a restricted transducer, but one that can correctly translate specific constructions in UD like coordination and prepositional verbs.

(3) **De-lexicalized** `ud2gf`: The problem of `dep2ast` in the form presented in Section 1.3 is a constrained search problem, i.e. the availability of a high-precision grammar is a pre-requisite in this setup for completeness. This pre-requisite can be relaxed, especially if we are interested in the frequent use-case of parsing with an incomplete lexicon of a language in GF-RGL or parsing languages not covered by GF-RGL.

Robustness in `ud2gf`

In Section 1.3, we glossed over the possibility in the search algorithm that syntactic annotation at an intermediate step does not result in any functions matching the

dependency subtree. There are two possible reasons why this happens[21]: either the grammar is missing a function (syntactic rule) to handle a specific dependency label or the dependency tree does not match the configuration (head and label information) corresponding to the relevant function in the grammar. The first issue of incomplete coverage happens in the case of ungrammatical sentences where dependency parsers correctly analyse the input, for example the phrase *the every man* in English. Indeed this is one of the strengths of UD that needs to be imported to GF. Possible interpretations for the example are the ASTs for *the man* and *every man* to which *every* and *the* are added as modifiers respectively. The same also can happen because the set of dependency structures generated by combination of the grammar and `gf2ud` is smaller than the set of dependency structures in the UD annotation scheme. The second case is possible due to errors in dependency parsing or annotation, where either the label is wrongly suggested by the parser or the complete edge is wrong in the dependency tree. In order for `ud2gf` to work as a robust front-end for parsing in GF, it is necessary to address both these problems – which we do by extending the grammar with a **backup** grammar. The backup grammar consists of a set of backup functions that are used to recover from both scenarios. The backup grammar defines for every category in the grammar, a pair of functions and a top-level function that can combine an arbitrary number of trees of the `Backup` category (shown in Figure 1.14).

```
BackupC    : C -> Backups -> C ; -- wrap tree of type C with Backups
CBackup    : C -> Backup ;       -- build Backup from tree of type C
NilBackup  : Backups ;           -- No Backups
ConsBackup : Backup -> Backups -> Backups ;  -- chain of Backup
```

Figure 1.14: The two types of elementary functions defined in the backup grammar. The BackupC functions are defined for functions higher in the GF category taxonomy and CBackup functions are defined for every category in the grammar.

Consequently `ud2gf` adds another step- after each dependency subtree is translated into an AST (T), ASTs corresponding to any uncovered subtrees are converted to backups and added as adverbial modifiers to the interpreted AST T.

Temporary categories

The limited expressivity of the transducer discussed in Section 1.3 poses a systematic problem when working with Universal Dependencies[22] due to the annotation of particular linguistic constructions that are common in text. This can be illustrated using two examples in English – although there are other instances of this issue in the RGL.

The first example is a UD tree for the phrase *men, women and children*, Figure 1.15 shows the different dependency trees defined by the annotation scheme of UDv1 and UDv2. The annotation of the entire phrase is a one-level dependency tree in UDv1 while in UDv2 the resulting dependency tree is of depth 2. Converting the one-level dependency tree used to annotated entire coordination in UDv1 is straightforward

[21] The third possibility is plausible but easier to handle. The language-specific UD annotation defines subtype of a core label and needs to be added as an ambiguous configuration to one of the existing functions.

[22] This is more systematic in UDv2 in comparison to UDv1 due to the changes in how coordination is annotated in UDv2.

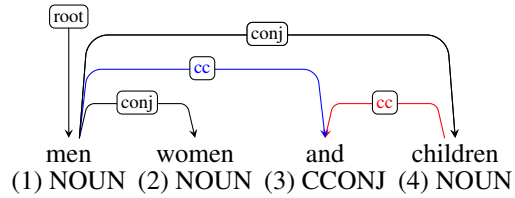


Figure 1.15: NP coordination in phrase *men, women and children* per UD guidelines. The red edge is the treatment of the conjunction per UDv2 while the blue edge is per UDv1 annotation. The part-of-speech tag of the conjunction in UDv1 would be CONJ instead of CCONJ.

```
4.a  root ConjCN 3 (ConsCN 4 (BaseCN (UseN men_N) 2)) [CN] {1,2,3,4}
4.b  root ConsCN 4 (BaseCN (UseN men_N) 2) [ListCN] {1,2,4}
4.c  root BaseCN (UseN men_N) 2 [ListCN] {1,2}
4.d  root UseN men_N [CN] {1}
4.e  root men_N [N] {1}
  1      conj UseN women_N [CN] {2}
  2      cc   and_Conj [Conj] {3}
  3      conj UseN children_N [CN] {4}
```

Figure 1.16: Iterative function application to translate the NP coordination in UDv1. Shown in red and blue is the order in which the ASTs are refined iteratively. The details have been simplified by positing lexical functions for the plural forms.

(iterations in the syntactic annotation phase are shown in Figure 1.16). However, the same does not work for the two-level dependency tree because there is no function that can be applied to a single noun and conjunction in the grammar – the result being that the conjunction "and" is covered using backup functions. The solution to this is to introduce **category aliases** in the configurations and build helper functions that use these categories to build the corresponding ASTs in a compositional way. Table 1.4 shows the relevant additions to the configurations required to handle a single conjunction – similar treatment for all conjunctions is added to the configurations.

category alias	`cat`	CN[and] = CN		
helper function	and[Conj2CN]	CN → CN[and] → CN	head conj	
	and[ConjCN]	ListCN → CN[and] → CN	head conj	quasi-interlingua
function definitions	and[Conj2CN]	λ init,last → ConjCN and_Conj (BaseCN init last)		
	and[ConjCN]	λ init,last → ConjCN and_Conj (ConsCN last init)		
helper function	ENand[CN]	Conj → CN → CN[and]	cc head cc.lemma=and	
	SVand[CN]	Conj → CN → CN[and]	cc head cc.lemma=och	
	FIand[CN]	Conj → CN → CN[and]	cc head cc.lemma=ja	language-specific
function definitions	ENand[CN]	λ conj,last → last		
	SVand[CN]	λ conj,last → last		
	FIand[CN]	λ conj,last → last		

Table 1.4: Configurations added for handling conjunctions in UDv2.

Lexicalized vs De-lexicalized `ud2gf`

The second extension addresses a practical design choice in the pipeline: whether one can assume the availability of multilingual lexicon for all languages as a prerequisite for `ud2gf`. We address this by introducing two flavors of `ud2gf` – a lexicalized variant and a de-lexicalized variant. The lexicalized `ud2gf` assumes the availability of a GF lexicon (monolingual or multilingual), which is a reasonable assumption when working with precision grammars. The de-lexicalized `ud2gf` on the other hand, dynamically builds lexical functions using the lemma and category configurations in the lexical annotation step. This variant is more suitable in a wide-coverage parsing setup, which in turn is used to extend the core RGL in our experiments with treebanks of UD.

A different way to perceive the lexicalized and de-lexicalized variants is to say that the lexicalized variant is a constrained version of the transducer- where additional configurations are required to accurately match non-compositional phrases in the grammar. For example, in the lexicalized `ud2gf` setup, configurations corresponding to phrasal verbs and multi-word expressions are required to match the correct phrasal functions in the dictionary. These are semi-automatically generated from the dictionary for the language and use a combination of helper functions and category aliases. This assumption that a large scale interlingual dictionary is available does not scale well with languages. In the de-lexicalized `ud2gf` setup, functions that build these phrasal verbs for different patterns are introduced to the grammar. This does lead to some over-generation in the de-lexicalized grammar, but also simplifies the handling of configurations when compared against configurations derived from a dictionary in the lexicalized case.

1.4.3 Applications

The implementations of `gf2ud` and `ud2gf` have multiple applications, a few of which are listed below.

[I] The transducer `gf2ud` is useful to bootstrap parallel treebanks according to the annotation scheme of UD.

[II] The transducer `ud2gf` combined with a dependency parser serves as a robust front end instead of the parser used in GF.

[III] The configurations defined in `gf2ud` are also a formal encoding of the UD annotation scheme, which is useful as a qualitative assessment of the cross-lingual annotation scheme.

[IV] The combination of `ud2gf` and concrete syntax for a language has a similar setup to the task of **surface realization** – from a UD tree without the order and words, generate the sentence (Mille et al., 2017, 2018).

[V] Furthermore, the transducer `ud2gf` makes UD treebanks accessible to train statistical models useful for disambiguation of ASTs.

The above applications have been studied in the current thesis, and are visually summarized in Figure 1.17.

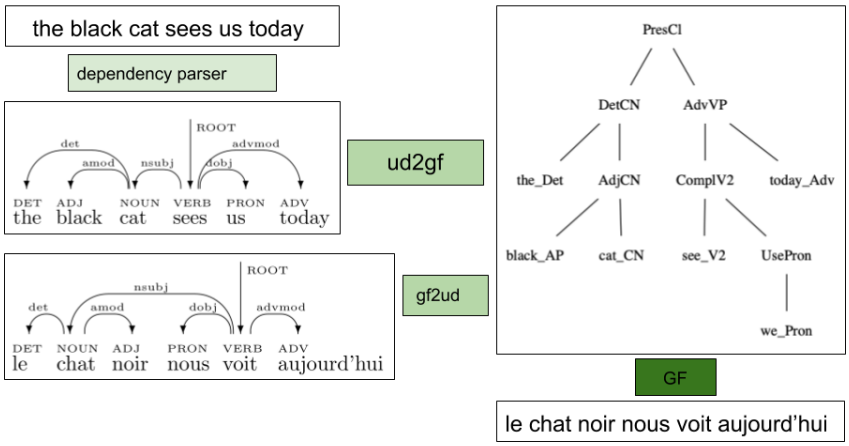


Figure 1.17: Pipeline of `gf2ud` and `ud2gf`.

1.5 Related Work

The dependency configuration central to both `ast2dep` and `dep2ast` are similar to the **head rules** of Collins (1996) used to introduce selectional preferences in context-free grammars for parsing.[23] Translations between representations – from phrase-structure trees to dependency structures – have been widely used in both NLP and CL, and de Marneffe et al. (2006), de Marneffe and Manning (2008) can be considered prototypical approaches in this line of work. The approach of de Marneffe et al. (2006) consists of two steps: (a) dependency extraction and (b) dependency typing. The dependency extraction step uses head rules to identify mark heads for each constituent and construct an unlabelled dependency tree. This is followed by typing, where labels are assigned to edges using pattern matching rules on the phrase-structure tree. The pattern matching is done at every node in the tree and the pattern with the most specific label is marked as the label. This is similar to the algorithm `ast2dep` and `gf2ud`, however, the dependency typing and the head marking happen before the dependencies are extracted. The pattern matching rules necessary for inducing the labels in the tree are defined on the abstract syntax, instead of the phrase-structure tree.

There is however a significant difference in `ast2dep` and `gf2ud` when compared against transducers for translating between representations. Transducers designed between representations rely on heuristic rules, especially to recover from parser errors. Both the head rules and the typing rules are thus defined on plausible tree patterns – which makes them robust but also *non-reversible*. The domain of dependency trees is not well-specified, instead the dependency scheme is precisely specified. Grammars designed in computational syntactic formalisms like HPSG and LFG Kaplan and Bresnan (1982) have been used to induce dependency representations from the primary descriptions used in the respective frameworks(Meurer, 2017, Przepiórkowski and Patejuk, 2018). More broadly, formalisms that use any degree of lexicalization can

[23] Selectional preferences are lexical constraints on plausible sentences in a language like it is more likely to say *I eat pizza with a knife* compared to *I eat a pizza with sauce*.

generate dependency trees as auxiliary descriptions.[24] This is illustrated in both Tree Adjoining Grammars (Joshi and Schabes, 1997) where these auxiliary descriptions are called derivation trees and Combinatory Categorial Grammars Hockenmaier and Steedman (2007). These frameworks constrain plausible dependency trees by defining a domain restricted by the grammar, but also restricts the dependency scheme as defined by the grammar. This is a necessary feature if one is interested especially in translation in the other direction i.e. converting dependency structures to the primary descriptions of the formalism. There has been a long line of work related to parsing in these frameworks where dependency trees have been used to improve the robustness of the parsing pipeline. However, the tight coupling between the original grammar and dependency representations necessitates a redesign of the grammar to change the dependency scheme.

The transducers described in this thesis should be seen as a combination of these two approaches, the external configuration provides a flexibility to modify the target annotation scheme without making changes to the grammar. But the configuration used in `ast2dep` and `gf2ud` is also a *specification* of the grammar and is used in `dep2ast` and `ud2gf` also. Using a dependency parser as a robust front end in `dep2ast` and `ud2gf` is also related to parsing approaches using supertagging (Vaswani et al., 2016, Dridan, 2013, Clark, 2002, Bangalore and Joshi, 1999) where the input is pre-processed using a tagger to assign complex descriptions, significantly simplifying the task of parsing (Bangalore, 1997). The interlingual grammars however are not lexicalized and as such `dep2ast` uses a dependency parser instead of a plain tagger.

1.6 Results

The experiments reported in this thesis use both intrinsic and extrinsic evaluation to validate the methods. The intrinsic evaluation in the experiments with `gf2ud` use statistics about labelled edges in the bootstrapped parallel UD treebanks. Similarly, the intrinsic evaluation in the experiments with `ud2gf` report statistics about the expected tree quality in terms of **coverage** and **interpretability** – interpretability corresponds to the fraction of cases where backups are used to built the AST. In cases where backups are used in building the AST, there is the fraction of the tree that doesnot contain any backups and hence is interepreted in the grammar and there are nodes which trigger the backup functions in the translation algorithm. We measure these two seperately in our evaluation. The results from these intrinsic evaluation for `gf2ud` is reported in Table 1.5 and for `ud2gf` in Table 1.6.

Extrinsic evaluation is the evaluation of the transducers in an external application – in the current thesis, we evaluate `gf2ud` on the task of bootstrapping de-lexicalized UD treebanks. We use synthetic UD treebanks bootstrapped from the GF-RGL grammar and train dependency parsers to evaluate the usefulness of data generated using interlingual abstractions when compared to human annotations. Figure 1.18 shows the learning curves of a de-lexicalized parser for English trained on synthetic data generated using interlingual grammars.

[24] Lexicalized grammars are grammars in which complex descriptions are defined for each word (or lexical item because the definition of *word* itself is complex and varies from language to language). These descriptions can be trees, higher-order functions or attribute-value matrices (AVMs). The degree of lexicalization in a formalism can range from **strong** – where the lexicon is the only and entire grammar – to **weak** in which the lexicon constitutes only a large part of the grammar.

Language	UD treebank		GF treebank
	RGL	Wide-coverage	
Afrikaans	81.57	-	81.73
Amharic	73.60	-	75.29
Bulgarian	76.60	80.53	88.13
Catalan	82.18	76.13	83.10
Chinese	84.89	77.33	82.46
Danish	80.49	-	87.45
Dutch	83.62	68.10	84.23
English	84.12	81.17	93.13
Estonian	82.38	79.10	86.52
Finnish	81.84	74.09	91.27
French	82.81	79.61	93.47
German	83.09	77.47	95.27
Greek	83.09	-	88.13
Hindi	72.63	69.34	84.18
Italian	81.79	77.52	90.47
Japanese	71.39	71.09	84.94
Latvian	72.11	-	89.10
Maltese	83.92	-	90.29
Mongolian	72.22	-	87.34
Nepali	82.91	-	83.19
Norwegian	80.37	-	86.36
Persian	83.87	-	84.40
Punjabi	72.37	-	82.82
Polish	83.05	-	87.38
Romanian	69.58	-	86.75
Russian	87.30	-	87.92
Sindhi	68.21	-	84.59
Spanish	81.41	79.15	91.28
Swedish	82.56	83.05	93.89
Thai	83.72	73.12	85.29
Urdu	72.86	-	84.67

Table 1.5: Percentage of completeness in the bootstrapped dependency treebanks. The Amharic grammar (underlined) is incomplete, i.e. does not implement all RGL functions. The same table appears in Kolachina and Ranta (2016)

language	#trees	#confs	%int'd tree	%int'd nodes
English	2077	59	72	94
Finnish	648	49	68	92
Finnish*	648	0	61	79
Swedish	1219	62	70	91
Swedish*	1219	0	65	76

Table 1.6: Coverage of nodes in each test set (`L-ud-test.conllu`). L* (Swedish*, Finnish*) is with language-independent configurations only. #conf's is the number of language-specific configurations. %int'd trees and %int'd nodes are the percentages of interpreted trees and nodes, respectively. These results are improvements from Ranta and Kolachina (2017).

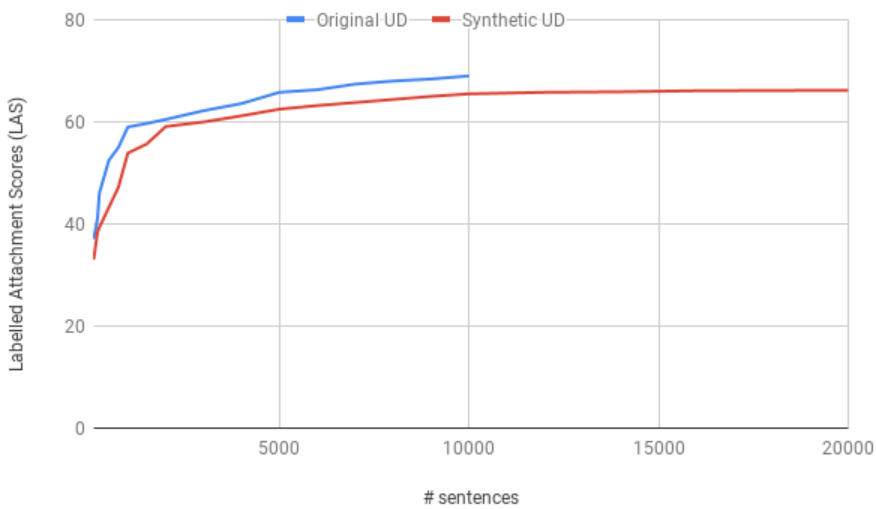


Figure 1.18: Learning curves for English comparing synthetic treebanks vs real treebank. Models trained on synthetic data take approximately twice the size of the real examples to achieve comparable accuracies.

1.7 Summary of the studies

The individual studies themselves are structured as follows:

1) Paper I is a first part of the study to explore the relation between abstract syntax trees (AST) and dependency structures in the UD scheme. This paper extensively studies the connection between the two in addition to proposing a generalized transducer to translate ASTs to dependency trees.

2) Paper II is the second part of the study: where the inverse problem i.e. translating dependency trees to abstract syntax trees is studied. The algorithm presented here is a restricted version of the generalized method discussed in Section 1.4. The algorithm works on a large fragment of structures found in the UD treebanks.

3) Papers III and IV both cover the topic of dependency parsing, in related with Universal Dependencies as the target scheme.

4) While Paper III focuses on the task of de-lexicalized parsing using synthetic UD treebanks, paper IV focuses on addressing one specific short-coming of dependency parsers: parsing sentences with out-of-vocabulary words.

Paper IV is a stand-alone study on symbolic dependency parsers and ways to improve them, but also explores initial steps towards improving robustness of dependency parsers and parsers in general. The work does set the tone for Paper III- which shares the objective of improving robustness of parsers, albeit for GF grammars using a completely different approach. The rest of this chapter summarizes the individual contributions of each of these studies.

1.7.1 Paper I: gf2ud

From Abstract Syntax to Universal Dependencies

This paper presents a conversion method from abstract syntax trees to dependency trees. This is done in two steps: by proposing a general algorithm that builds dependency trees for a given interlingual grammar in GF (`ast2dep`), and applying this algorithm to convert GF-RGL trees to Universal Dependencies (UD). One of the aims of the study is to precisely describe the correspondence between the two multilingual abstractions, namely GF-RGL grammars and UD. The correspondence between GF-RGL and UD turns out to be good, and the relatively few discrepancies give rise to interesting questions about universality.

The conversion also has applications: (a) to bootstrap parallel UD treebanks from GF treebanks; (b) it defines a formal encoding of the annotation guidelines of UD, in terms of functions in the GF-RGL grammar. (c) it makes information from UD treebanks available for the construction of ASTs (d) it gives a method to check the consistency of manually annotated UD trees with respect to the annotation schemes ;

The conversion is tested and evaluated by bootstrapping two small treebanks for 31 languages, as well as comparing a GF version of the English Penn treebank with the UD version. In the first case, the bootstrapped treebanks are evaluated in terms of % of labelled edges, while the Penn treebank is compared against a UDv1 treebank obtained using the Stanford dependency converter on the Penn treebank. Furthermore, the work in this paper serves as a pre-cursor to the work in Paper II and Paper III, and leaves some unexplored directions for future research.

1.7.2 Paper II: ud2gf

From Universal Dependencies to Abstract Syntax

This study is a continuation of Paper I that describes the relation between ASTs and dependency trees. This paper attempts to invert the mapping: take dependency trees from standard UD treebanks and reconstruct AST trees from them. The primary aim of this method is to help GF-based interlingual translation by providing a robust, efficient front end – as a substitute for the exact parsing used in GF. However, since UD trees are based on natural (as opposed to generated) data and built manually or by machine learning (as opposed to rules), the conversion is not trivial.

As I mentioned above, this work uses both insights and artifacts (mainly the dependency configurations) from Study I as a starting point. The study provides a stand-alone description of a basic algorithm, essentially focussed on inverting the conversion i.e. `dep2ast`. This method enables covering around 70% of nodes, and the rest can be covered by approximative backup strategies. Analyzing the reasons of the incompleteness reveals structures missing in GF grammars, but also some problems in UD treebanks.

Extensions to the core algorithm and improvements of the results presented in this study have already been described in Section 1.4.2.

1.7.3 Paper III: Bootstrapping UD treebanks

Bootstrapping UD treebanks for Delexicalized Parsing

Standard approaches to treebanking traditionally employ a waterfall model (Sommerville, 2010), where annotation guidelines guide the annotation process and insights from the annotation process in turn lead to subsequent changes in the annotation guidelines. This process remains a very expensive step in creating linguistic resources for a target language, necessitates both linguistic expertise and manual effort to develop the annotations and is subject to inconsistencies in the annotation due to human errors. In this paper, we propose an alternative approach to treebanking – one that requires writing grammars. This approach is motivated specifically in the context of Universal Dependencies, an effort to develop uniform and cross-lingually consistent treebanks across multiple languages.

We show here that a bootstrapping approach to treebanking via interlingual grammars is plausible and useful in a process where grammar engineering and treebanking are jointly pursued when creating resources for the target language. We demonstrate the usefulness of synthetic treebanks in the task of delexicalized parsing. Our experiments reveal that simple models for treebank generation are cheaper than human annotated treebanks, especially in the lower ends of the learning curves for delexicalized parsing, which is relevant in particular in the context of low-resource languages.

1.7.4 Paper IV: OOV words in Dependency parsing

Replacing OOV Words For Dependency Parsing with Distributional Semantics

Lexical information is an important feature in dependency parsers – both in the case of stand-alone parsing given a tagged sequence and in pipeline systems where other components like part-of-speech (POS) taggers rely on forms of the lexical items. However, there is no such information available for out-of-vocabulary (OOV) words,

which causes many classification errors. In this study, we propose a method to address this shortcoming: replacing OOV words with known, in-vocabulary words that are *similar* according to different notions of similarity. Specifically, we study in detail two such notions: semantic and morphological similarity. The replacement candidates are obtained using distributional similar words computed from a large background corpus, as well as morphologically similar according to common suffixes.

Extensive experiments are done to cover different design parameters: using both transition-based and graph-based symbolic dependency parsers; count-based and dense neural vector-based semantic models for distributional similarity and a set of typologically diverse languages for each of the two similarity heuristics. We show performance differences both for count-based and dense neural vector-based semantic models using the proposed technique. Further, we discuss the interplay of POS and lexical information for dependency parsing and provide a detailed analysis and a discussion of results: while we observe significant improvements for count-based methods, neural vectors do not increase the overall accuracy.

1.8 Conclusions and Future work

The thesis studies the interrelatedness between different types of multilingual abstractions. On one side are interlingual *reversible* grammars built using Grammatical Framework, designed to be used in applications focussed on multilingual generation and semantics. On the other hand, Universal Dependencies are parallel abstractions rather than interlingual, designed to be used in applications focused on light-weight syntax. The primary representations in GF and UD are abstract syntax trees (ASTs) and dependency trees respectively.

The aim of the thesis is to bridge these two representations, with the practical goal to exchange linguistic artifacts between these two frameworks. Algorithms proposed in this work address the translation of ASTs into dependency trees and the reverse direction i.e. translating dependency trees into ASTs. These algorithms are useful for both GF and UD – ease of multilingual parsing in UD is imported to GF, while generation with GF is in turn useful to bootstrap UD resources for new languages. Both these applications have been developed in this thesis – dependency parsers built using UD resources have been used as a robust front end to parsing in GF. At the same time, dependency parsers trained on UD treebanks generated using GF have also been presented. This work shows how grammar engineering can be useful to bootstrap treebanks as opposed to annotating data by hand, especially for languages with low resources.

In the later part of the thesis, we also propose a method to address the well known problem of out-of-vocabulary words in dependency parsers. Using a distributional thesaurus as an additional source of information, we show that parsing can be improved using a simple technique – replacing unknown words semantically similar known words. Empirical results using multiple parsers across 7 languages show overall improvements and improvements specific for out-of-vocabulary words that are statistically significant.

1.8.1 Open Problems and Future directions

From ASTs to Graphs

An alternate representation to Universal Dependencies that captures the semantics in language in Abstract Meaning Representation (AMR) (Banarescu et al., 2013). Much of the ongoing work in AMR is focussed on English and is missing a multilingual component. The representations are richer that what is typically used in tasks focussed on syntax – from trees to graphs and more generally a richer class of graphs. GF grammars have been shown to work in controlled domains as precise models of the semantics – thus it is interesting to look at connections between GF and the representations used in AMR. These connections have been explored both in the context of surface realization Gruzitis et al. (2017) and multilingual summarization (Gruzitis and Barzdins, 2016). However, what remains to be seen is whether GF can help in adding multilinguality to AMR. In the best case, the interlingua grammars of GF may serve as under-specified representations in parallel graph treebanks that can accelerate development of multilingual graph bank resources.

Data Augmentation for Dependency Parsing

The work on data augmentation presented in this thesis is a preliminary study in the bigger topic of *data augmentation* methods for dependency parsing. There are multiple directions left unexplored in this study – the use of grammar-based abstractions as adversarial generators for *breaking* existing parsers as well as *building* new parsing models using augmented data. Another clear direction forward with this line of work is to study how synthetic data can be combined with real human-annotated data in dependency parsing.

Predicting Consistency in Universal Dependencies

The configurations developed in `gf2ud` and `ud2gf` are a formalization of the guidelines provided to human annotators. Xia (2001) described how annotation guidelines of a treebank are a *grammar* – one of a radically different form that the formal grammars used in this thesis. The encoded transductions on the abstract syntax for both `gf2ud` and `ud2gf` raise an obvious question – how much of the annotated treebanks are consistent across languages or within a language across treebanks. Previous work on error detection in treebanks has been done in monolingual cases. Using the RGL – specifically the concrete syntaxes for various languages – the question of how much of the treebanks can be generated can be an indicator of how consistent these treebanks are with respect to the interlingual grammar at hand.

Concept alignment: Inducing Interlingual grammars

Most of the work in this thesis has been focused on multilingual parsing and generation. The obvious direction ahead is to put these two together i.e. to induce interlingua grammars for a given multi-text. This particular question opens up questions that have been avoided previously in NLP – extending sequence alignment methods to multiple sequences and designing better abstractions for more than two languages. Current research with deep neural machine translation has been encouraging to this direction of working with more than two languages – however, interlingua grammars promise an explainable abstraction that is more *compact* than deep neural models.

Chapter 2

Paper I: gf2ud

From Abstract Syntax to Universal Dependencies

Prasanth Kolachina, Aarne Ranta

***Linguistic Issues in Language Technology 13(3), 2016*.**

Abstract

Abstract syntax is a semantic tree representation that lies between parse trees and logical forms. It abstracts away from word order and lexical items, but contains enough information to generate both surface strings and logical forms. Abstract syntax is commonly used in compilers as an intermediate between source and target languages. Grammatical Framework (GF) is a grammar formalism that generalizes the idea to natural languages, to capture cross-lingual generalizations and perform interlingual translation. As one of the main results, the GF Resource Grammar Library (GF-RGL) has implemented a shared abstract syntax for over 30 languages. Each language has its own set of concrete syntax rules (morphology and syntax), by which it can be generated from the abstract syntax and parsed into it.

This paper presents a conversion method from abstract syntax trees to dependency trees. The method is applied for converting GF-RGL trees to Universal Dependencies (UD), which uses a common set of labels for different languages. The correspondence between GF-RGL and UD turns out to be good, and the relatively few discrepancies give rise to interesting questions about universality. The conversion also has potential for practical applications: (1) it makes the GF parser usable as a rule-based dependency parser; (2) it enables bootstrapping UD treebanks from GF treebanks; (3) it defines formal criteria to assess the informal annotation schemes of UD; (4) it gives a method to check the consistency of manually annotated UD trees with respect to the annotation schemes; (5) it makes information from UD treebanks available for the construction and ranking of GF trees, which can improve GF applications such as machine translation. The conversion is tested and evaluated by bootstrapping two small treebanks for 31 languages, as well as comparing a GF version of the English Penn treebank with the UD version.

2.1 Introduction

Computational syntax can work on different levels of abstraction. The lowest level normally used when processing written text is strings of tokens ("words"). But it is often useful to work with more abstract structures: part-of-speech (POS) tagged lemma sequences, phrase structure trees, dependency trees, or some kind of logical forms.

Raising the level of abstraction often gives new ways to relate different languages to each other. Thus tagged lemmas enable the separation of surface strings from word senses, which can be useful, for instance, in factored machine translation. Logical forms ideally ignore all language-related features, and express just the pure propositional meaning. But what about syntax trees? Traditional phrase structure trees preserve surface words and constituent order and are hence language-dependent. Dependency trees are often a bit more abstract, treating the word order as irrelevant and lemmatizing the words. Sharing the POS tags and dependency labels between languages increases this potential to abstract over languages.

Universal Dependencies (UD, de Marneffe et al. (2014)) is a recent approach to dependency parsing that tries to maximize the sharing of structures between languages. UD has a set of dependency labels and POS tags that are designed to fit many languages, and a series of annotation manuals that guide treebank builders to use the labels and tags in a uniform way. The expected gain is that efforts can be shared among languages. For instance, searching for semantic roles in sentences can be defined uniformly for different languages, and parsers for new languages can be bootstrapped by using treebanks for other languages.

As suggested by Nivre (2015), UD can be seen as a modern approach to **universal grammar**. The originally mediaeval idea of a universal grammar has many times been rejected by linguists, often with good reasons. But much of it can be saved if we think of it as an abstraction: on a proper level of abstraction, languages have much in common, so why not try to find out what is common? Universality should be seen as a working hypothesis rather than an *a priori* truth.

In UD, the working hypothesis is that languages have a common set of parts of speech (nouns, verbs, etc) as well as grammatical functions (subject, modifier, etc). Some languages don't have all of these features, and individual languages may have features that are not universal. The annotation manuals for treebank builders have recommendations that maximize the use of common features, but give room to diversity. This approach has proved successful, and as a result, UD has presented treebanks for over 30 languages.

An older but nonetheless computational approach to universal grammar is Curry's notion of **tectogrammatical** structure (Curry, 1961). The tectogrammatical representations are function applications[1]. They are trees that describe **pure constituency**: what the constituents are and how they are put together, but ignoring what word strings are ultimately used and what their linear order is. To give an example, subject-verb-object predication could be presented by a tectogrammatical function `Pred`,

```
Pred : TV -> NP -> NP -> S
```

that takes a transitive verb and two noun phrases as its arguments and produces a sentence. The constituent order (SVO, SOV, etc) is specified separately for each language in their **phenogrammatical** rules. These rules may look as follows:

[1] Also known as lambda terms or LISP terms or, as in Curry's original work, terms of combinatory logic.

```
Pred verb subj obj = subj ++ verb ++ obj
Pred verb subj obj = subj ++ obj ++ verb
```

for SVO and SOV, respectively (with ++ marking concatenation).

Curry's tectogrammar inspired the Prague school of dependency parsing (Böhmová et al., 2003). The grammars, however, are different, since the Prague school is based on the roles of words (similar to what is traditionally called "grammatical functions"), whereas Curry's tectogrammatical functions are functions that combine words. Grammatical Framework (GF, Ranta (2004b, 2011)) is a grammar formalism that is more directly based on Curry's architecture. GF grammars are similar to grammars used in compiler construction, where tectogrammar is called **abstract syntax** and phenogrammar is called **concrete syntax** or **linearization** (McCarthy, 1962, Appel, 1998). [2]

The most comprehensive multilingual grammar in GF is the **Resource Grammar Library**, GF-RGL (Ranta, 2009b), which by the time of writing has concrete syntaxes for over 30 languages, ranging from European through Finno-Ugric and Semitic to East Asian languages.[3] When the UD approach appeared, it became immediately interesting to see how it relates to GF-RGL. The formal correspondence was obvious: once we have a GF abstract syntax tree, we can easily derive a dependency tree. For instance, a rule of the form

```
Pred verb subj obj
```

gives rise to a dependency tree where the first argument produces the head, the second argument produces a dependent with label `subj` and the third argument a dependent with label `obj`. As we will show more formally in Section 2.2.2, a simple recursive function can convert abstract trees to dependency trees in this way. However, there are details that remain to be worked out:

- How to convert GF-RGL to an independently given dependency scheme, such as UD?
- Is GF-RGL complete, in the sense of covering all UD structures?
- Can GF-RGL give any new insights for developing UD further?

The purpose of this paper is to answer these questions. While doing so, we will often discuss the differences in analyses between GF-RGL and UD, and in many cases argue for the GF-RGL decisions. But in a bigger picture, we have been surprised to see how much the approaches have in common. That so similar structures of "universal grammar" have been found in two independent ways can be seen as confirming evidence for both of them.

Our work is also expected to have practical uses: bootstrapping UD treebanks from GF; using UD treebanks to help GF parsing (in particular, statistical disambiguation); assessing UD annotation schemes and treebanks from a formal perspective. Our conversion moreover makes the GF-RGL parser usable as a rule-based UD parser, although a rather slow one. Perhaps more interestingly, generation from UD trees becomes possible (including translations to other languages), because GF grammars are reversible (and generation is fast, as opposed to parsing).

[2] As noted by Dowty (1979), also Montague grammar (Montague, 1974) can be seen as having Curry's architecture, although Montague only used it for English.

[3] The current status of GF-RGL can be seen in `http://www.grammaticalframework.org/lib/doc/synopsis.html` which also gives access to the source code.

The structure of the paper is as follows: Section 2.2 prepares the discussion with a concise introduction to GF and a mathematical definition of the correspondence between abstract syntax trees and dependency trees. Section 2.3 gives an overview of GF-RGL and UD; parts of it can be skipped by the reader who already knows the approaches, and many of the details are given in an Appendix. Section 2.4 goes through the great majority of structures, where GF-RGL and UD are similar enough to allow a simple, local (i.e. compositional) and language-independent conversion of trees. Section 2.5 covers the remaining structures, where non-local or language-dependent conversions are needed. Section 2.6 presents an evaluation with three different treebanks. Section 2.7 concludes.

2.2 Grammars and trees

2.2.1 Abstract and concrete syntax

A GF grammar consists of an abstract syntax and a set of concrete syntaxes. Figure 2.1 shows a set of abstract syntax rules, which is a small fragment of the GF Resource Grammar Library, but representative in the sense that it covers some of the most fundamental syntactic structures. The rule set includes a lexicon that is large enough to cover our running example, the English sentence

the black cat sees us

and its French equivalent

le chat noir nous voit (word to word: "the cat black us sees")

An abstract syntax has two kinds of rules:

- `cat` rules defining **categories**, here `S`, `NP`, `VP`, etc.
- `fun` rules defining **functions**, here `PredVP`, `ComplTV`, etc.

All rules in Figure 2.1 are equipped with comments (starting `--`) that explain the categories and functions.

Categories are the basic building blocks of **types**, which have the form

$$C_1 \to \ldots \to C_n \to C$$

where $n \geq 0$ and $C_1, \ldots, C_n, C$ are categories. Each such type is a **function type**, where $C_1, \ldots, C_n$ are the **argument types** and C is the **value type**. The limiting case $n = 0$ gives types of **constant functions**, which typically correspond to **lexical items**, such as `we_Pron` in Figure 2.1. Types with $n > 1$ typically correspond to **syntactic combinations**, such as `PredVP`, combines an NP with a VP to an S. Types with $n = 1$ are typically **coercions**, such as `UsePron`, which lifts a pronoun into an NP.

A concrete syntax has two kinds of rules, parallel to `cat` and `fun` rules:

- for each category, a `lincat` rule defining its **linearization type**
- for each function, a `lin` rule defining its **linearization**, which is a function that combines the linearizations of the arguments into an object of the linearization type of the value type

To define a concrete syntax for Figure 2.1, we can start by uniformly using `Str` as linearization type:

```
cat
  S ;     -- sentence
  NP ;    -- noun phrase
  VP ;    -- verb phrase
  TV ;    -- transitive verb
  AP ;    -- adjectival phrase
  CN ;    -- common noun
  Det ;   -- determiner
  Pron ;  -- personal pronoun

fun
  PredVP  : NP -> VP -> S ;   -- predication: (the cat)(sees us)
  ComplTV : TV -> NP -> VP ;  -- complementation: (see)(us)
  DetCN   : Det -> CN -> NP ; -- determination: (the)(cat)
  AdjCN   : AP -> CN -> CN ;  -- adjectival modification: (black)(cat)
  UsePron : Pron -> NP ;      -- use pronoun as noun phrase: (us)

  we_Pron : Pron ;  -- we/us
  see_TV  : TV ;    -- see/sees
  the_Det : Det ;   -- the
  black_AP: AP ;    -- black
  cat_CN  : CN ;    -- cat/cats
```

Figure 2.1: An abstract syntax for a fragment of GF-RGL.

```
lincat S, NP, VP, TV, AP, CN, Det = Str
```

All linearizations are then defined as strings and their concatenations (denoted by ++). Thus a tree formed by the function `PredVP` is in both English and French linearized by

```
lin PredVP np vp = np ++ vp
```

concatenating the linearization of the `NP` argument with the linearizations of the `VP` argument. But usually the rules are different. Thus lexical items have rules such as

```
lin black_AP = "black"  -- English
lin black_AP = "noir"   -- French
```

More interestingly, linearization rules can also vary the word order:

```
lin ComplTV tv np = tv ++ np  -- English
lin ComplTV tv np = np ++ tv  -- French

lin AdjCN ap cn   = ap ++ cn  -- English
lin AdjCN ap cn   = cn ++ ap  -- French
```

The concrete syntax rules shown above use only strings and their concatenation. Such rules have an expressive power similar to **synchronous context-free grammars** (Aho and Ullman, 1969b), the main difference being that synchronous grammars don't make the abstract syntax explicit. Synchronous context-free grammars are sufficient for changing the lexical items and their order, which is what we need in our running example. However, the above rules are not correct, because they don't

deal with morphology (case and agreement) nor with variable word order (clitic vs. non-clitic objects in French). To deal with natural languages in full scale while sharing the abstract syntax, we need a bit more expressive power. We will return to this in Section 2.2.3.

2.2.2 Trees and their conversions

Figure 2.2 summarizes the different kinds of trees that we will speak about, by showing different representations of one and the same example.

- **Abstract syntax trees** (a) are trees where the nodes and leaves are abstract syntax functions.
- **Parse trees** (b,c), also known as **concrete syntax trees** or **phrase structure trees**, are trees where the nodes are categories and the leaves are words (strings).
- **Dependency trees** (d,e) are trees where the nodes are words and the edges are marked by **dependency labels**; the order of words is significant.
- **Abstract dependency trees** (f) are trees where the nodes are constant functions and the edges are marked by dependency labels; the order of nodes is not significant.

The tree visualizations are generated by GF software.

The abstract syntax tree (Figure 2.2 (a)) is a non-redundant representation from which all the others can be derived. Parse trees are derived as follows: given an abstract syntax tree T,

[I] Linearize it to a word sequence S.
[II] Link each word in S to its smallest spanning subtree in T.
[III] Replace each function in the nodes of T by its value category.

The **smallest spanning subtree** of a word is the subtree whose top node is the function whose linearization generates that word.

To convert an abstract tree to a dependency tree, we specify, for each abstract syntax function, its **dependency configuration**: which of the arguments is the head, and how the other arguments are labelled. The default dependency configuration used in GF says that the head is the first argument, and the other arguments have labels `dep1`, `dep2`, and so on. This default can be overridden by an explicit configuration. In Figure 2.2, we assume the following configurations to produce the standard UD labels:

```
PredVP  nsubj head
ComplTV head  dobj
DetCN   det   head
AdjCN   amod  head
```

A similar configuration can be used for mapping GF categories to UD part of speech tags. The default is the category symbol itself.

Given an abstract syntax tree T of a word sequence S, a dependency tree is derived as follows:

[I] For each word w in S, find the function f_w forming its smallest spanning subtree in T.
[II] Link each word w in S with either
  (a) the head argument of f_w, if w is not the head
  (b) the head of whole f_w, if w is itself the head

Given a concrete syntax and a dependency configuration, an abstract syntax tree is thus a non-redundant and faithful representation for all information about a sentence.

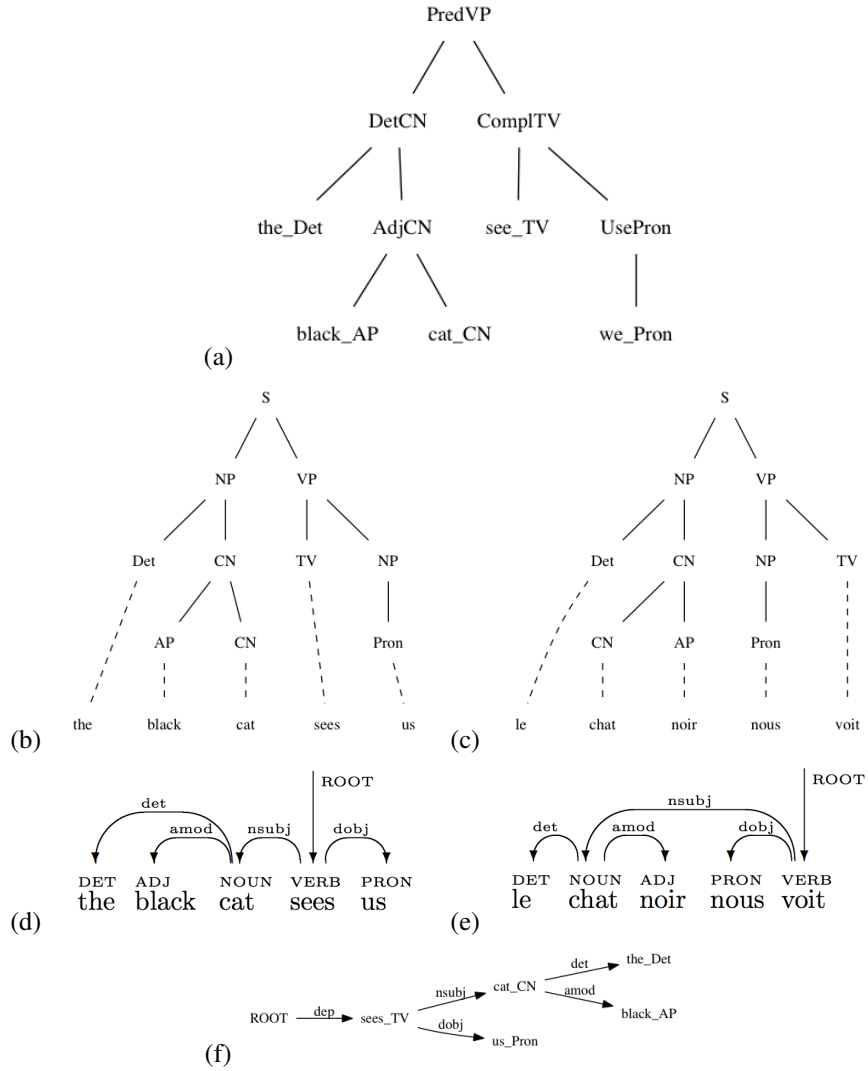


Figure 2.2: Trees for the sentence *the black cat sees us* and its French translation: (a) abstract syntax tree; (b,c) parse trees; (d,e) dependency trees; (f) abstract dependency tree with unordered word senses.

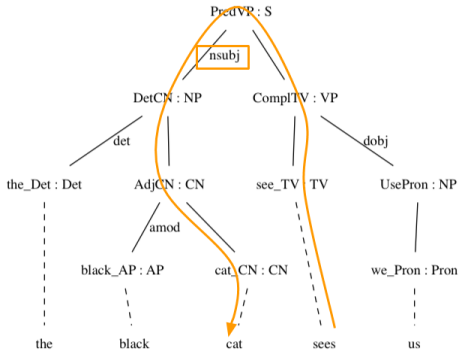


Figure 2.3: Dependency tree derivation from a decorated parse tree. The word *cat* has *sees* as its head and *nsubj* as its label. The path from the top to the word *sees* is a spine.

In contrast to this, parse trees and dependency trees are **lossy representations**. For dependency trees, this is easy to see: many functions can have the same dependency configuration, and if we only see the labels attached to the arguments, we cannot know what the dominating function is. For parse trees, it can likewise happen that a context-free grammar rule encodes different ways of putting together its constituents. For instance, the flat predication rule

$$\mathrm{S} \rightarrow \mathrm{NP\ TV\ NP}$$

can, in a free word order language, match both SVO and OVS sequences.

For the reason of missing information, dependency trees and parse trees cannot in general be derived from each other. This is why the existing algorithms use uncertain heuristics such as head percolation (Collins, 1996). Using abstract syntax trees as the master representation solves this problem.

An alternative way to derive dependency trees is to use parse trees decorated with abstract syntax functions and dependency labels. This gives a natural way to explain the conversions and will therefore be used later in this paper. In a decorated parse tree, we mark the dependency labels at each branching point of the tree, as shown in Figure 2.3, but omit the "head" labels. To find the labelled arc for each word,

[I] Follow edges up from the word until a label is reached: this is the label of the word.

[II] From the dominating node, follow the (unique) path of unlabelled edges down to another word: this is the head of the word. A head path in a tree is called a **spine**.

[III] If no label is encountered on the way upwards, the word is itself the head of the sentence.

Figure 2.3 shows the path corresponding to the labelled arc of the word *cat*.

2.2.3 Abstracting from morphological variation

If we swap the subject and the object in the example sentence of Figure 2.2 and linearize with the concatenation rules of concrete syntax in Section 2.2.1, we get *us*

sees the black cat in English and *nous le chat noir voit* in French. Both sentences have a subject-verb agreement error. The English sentence also has a wrong case of the pronoun, and the French sentence has a wrong word order, since the object can appear before the verb only if it is a clitic pronoun. To solve these problems, we need to introduce morphological variation in the grammar. Since morphological variation is language-dependent, we introduce it in concrete syntax and not in the abstract syntax. This forces us to go beyond the context-free linearization rules of Section 2.2.1.

Let us consider verb inflection first, restricted to present tense indicative forms for simplicity. In English, we need two forms: the third person singular and "other". We define, in the English concrete syntax, a **parameter type** of verb forms, which has two elements:

```
param VForm = SgP3 | Other
```

The category TV (and also VP) has as its linearization type a **table** (similar to an inflection table), which produces a string as a function of a verb form:

```
lincat VP, TV = VForm => Str
```

Thus the verb *see* is linearized as follows:

```
lin TV = table {SgP3 => "sees" ; Other => "see"}
```

For noun phrases, we need the parameter of case, which is nominative or accusative.

```
param Case = Nom | Acc
```

But we also need to account for the subject-verb agreement: the fact that a noun phrase can determine the form of a verb. This we can do by using a **record type** as the linearization type of NP:

```
lincat NP = {s : Case => Str ; a : VForm} ;
```

A record of this type has two fields: the field `s`, which is a case-dependent string, and the field `a` (agreement feature), which is a verb form. An example is the linearization of `we_Pron`:

```
lin we_Pron =
  {s = table {Nom => "we" ; Acc => "us"} ; a = Other}
```

Putting everything together, we obtain the linearization rule for predication:

```
lin PredVP np vp = np.s ! Nom ++ vp ! np.a
```

This rule uses the `s` field of the NP (by the projection operator `.`), from which it takes the `Nom` form (by the selection operator `!`). The result is concatenated to the verb form selected for the value for the `a` field of the NP.

In French, we need different parameter types: for instance, verbs have six forms and not just two. We also need some parameters not present in English, such as the gender of adjectives, nouns, and determiners. The most interesting parameter for the example at hand is, however, a boolean that states if an NP is a clitic, to determine its position when used as object:

```
lincat NP =
  {s : Case => Str ; a : VForm ; isClitic : Bool}
```

Now we can write a complementation rule that inspects the cliticity feature of the object to decide if the verb or the object comes first:

```
lin Compl tv np = table {vf =>
  case np.isClitic of {
    True  => np.s ! Acc ++ tv ! vf ;
    False => tv ! vf ++ np.s ! Acc
    }
  }
```

The generalization of linearization from strings to tables and records leads us from context-free grammars to **multiple context free grammars** (MCFG) (Seki et al., 1991). As shown in Ljunglöf (2004), GF is actually equivalent to PMCFG (Parallel MCFG), which is MCFG with reduplication. [4]

An MCFG is a grammar over tuples of strings rather than just strings. In addition to inflection and word order variations, MCFG enables **discontinuous constituents**. Thus for instance in VSO languages (such as classical Arabic) verb phrases are records with separate fields for the verb and the complement:

```
lincat VP = {verb : Str ; compl : Str}
```

(ignoring all morphological parameters). The VSO order is realized in the predication rule, which puts the subject noun phrase between the verb and the complement:

```
lin PredVP np vp = vp.verb ++ np ++ vp.compl
```

2.2.4 Abstracting from syncategorematic words

Designing a GF grammar involves finding a level of abstraction that makes sense for all languages to be addressed. For instance, morphological distinctions present in one language but not the others should be ignored in the abstract syntax. Some of these questions can be subtle. The **copula** of adjectival predication is an example of a common kind of questions encountered in the RGL-UD mapping task.

Let us extend the abstract syntax of Figure 2.1 with a rule converting adjectival phrases to verb phrases:

```
fun CompAP : AP -> VP
```

The English linearization rule (ignoring morphology, which is irrelevant for this question) introduces the copula *is* prefixed to the AP:

```
lin CompAP ap = "is" ++ AP
```

The copula has thus no abstract syntax of its own. The cross-linguistic justification of this treatment is that many languages (Arabic, Russian) don't need copulas, and that

[4] Reduplication is used only in few places in the RGL: Chinese yes/no questions and some semantic constructions such as the intensification of adjectives in Swedish.

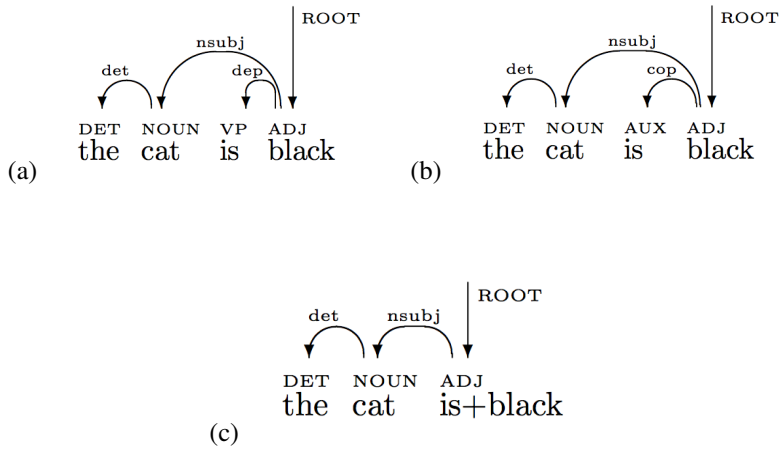


Figure 2.4: Dependency trees for *the cat is black* (a) with a syncategorematic copula (b) with a categorized copula. Also shown in (c) is a collapsed variant of (a)

they should therefore not be introduced in the abstract syntax. Words with no abstract syntax category attached are called **syncategorematic**.

The conversion defined in Section 2.2.2 generates no dependency labels for syncategorematic words. Thus the tree assigned to *the cat is black* in this grammar has no label for the word *is*. A default dummy label "dep" can then be used, as in Figure 2.4 (a). This tree moreover uses VP as the part of speech tag, since this is the category of the smallest spanning subtree of the copula.

However, the UD annotation manual for English says that the copula should have the label "cop" with the word *black* as its head (Figure 2.4 (b)) and AUX as its POS tag. The general principle in dependency parsing is that all words have labels connecting them to a head (except for the head of the whole sentence). According to this principle, *there are no syncategorematic words*.

The principle that all words are categorized makes sense in the dependency parsing context, where the trees are trees of *words* and not of phrases, let alone abstract functions. But from the "universal" point of view, this results in trees that may be unnecessarily different in different languages, because syncategorematic words are not universal. They can also be argued to be irrelevant for the semantic structure. Thus dependency parsers are sometimes supplemented by "flattening" or "collapsing" functions that remove semantically irrelevant words (de Marneffe and Manning, 2008, Ruppert et al., 2015). The "collapsing" process achieves two characteristics: a collapsed tree no longer contains all the words in a sentence and the dependency labels and heads are semantically coherent (but may be syntactically heterogeneous).

When dependency trees are derived from abstract syntax, the situation is the opposite. What we get first, by straightforward dependency configurations, is a kind of "collapsed" trees (Figure 2.4 (c)). The semantically irrelevant words are not provided with a dependency label of their own using dependency configurations defined on abstract syntax. To obtain these labels, we must extend the dependency configurations with rules defined on concrete syntax. Such rules are language-dependent, not universal.

One could question whether not just stop at the collapsed trees, since they are the truly universal ones. However, since we are interested in converting GF trees to complete UD trees, we have chosen to extend our conversion algorithm with a "decollapsing" phase, by extending the abstract (language-independent) dependency configurations with language-dependent ones (Section 2.5). The resulting dependency tree at the end of this "decollapsing" phase is a connected tree where all words have labels connecting them to a head, as done in the UD scheme. The extended conversion algorithm is presented in Section 2.5.7.

An alternative to these language-dependent configurations is to rewrite the grammar. Rewriting the grammars facilitates constructing complete UD trees using dependency configurations defined only on the abstract syntax. For the example at hand, this is simple: just introduce a category `Cop` of copulas, with one element `be_Cop`, and change the `CompAP` rule:

```
cat Cop
fun be_Cop : Cop
fun CompAP : Cop -> AP -> VP
```

In languages with zero copulas, the linearization of `be_Cop` is the empty string.

The line that we follow in this paper, however, is to keep GF-RGL as it is and instead make the dependency configurations more elaborate. This choice has several advantages:

- We can automatically get collapsed trees. These can be post-processed later to add labels for syncategorematic words, but still be useful if the end goal is only to extract relations between the content words in the sentence.
- We don't optimize for the particular dependency annotation scheme of UD and can easily change the configurations.
- Since GF-RGL was from the start designed to be multilingual, we get evidence to assess the universality of the current UD.

2.3 An overview of GF-RGL and UD

This section provides readers with an overview of GF-RGL and the UD annotation project. Readers familiar with GF can skip Section 2.3.1 and readers familiar with UD can skip Section 2.3.2.

2.3.1 Overview of RGL

The first applications of GF were small grammars built for translation systems on specific application domains, such as geographic facts (Dymetman et al., 2000), mathematics (Hallgren and Ranta, 2000), and simple health-care dialogues (Khegai, 2006). The abstract syntax in these applications encoded the semantic structures that were to be preserved in translation. But the idea soon emerged to generalize the abstract syntax idea from domain semantics to domain-independent syntactic structures, such as NP-VP predication. This led to the development of the GF Resource Grammar Library (RGL), whose first version was inspired by the syntactic structures used in CLE (Core Language Engine) (Rayner et al., 2000). The RGL was first intended to be a library that would help domain grammarians by giving them reusable functions for surface syntax and morphology (Ranta, 2009a). But it was soon also seen as a

linguistic experiment, to find out how comprehensive a common abstract syntax could be and how many languages it could apply to. This experiment had no commitment to any theory of linguistic universals, but the idea was just to try and see how far one can get. Nevertheless, the sharing of tree structures was more far reaching than parallel grammar development in systems like CLE, LinGO Matrix (Bender and Flickinger, 2005), and ParGram (Butt et al., 2002).

The first experiments on a handful of not too similar languages (English, Finnish, French, Russian) were encouraging. Gradual modifications led to a stable abstract syntax, which was reported in Ranta (2009b) and was at the time implemented for 14 languages. In late 2015, the same abstract syntax has concrete syntaxes for 30 languages, and 5 to 10 more are under construction. More than 50 persons have contributed to the GF-RGL. Its source code and documentation are available on the GF web page.[5]

The GF Resource Grammar Library has an abstract syntax with 86 categories and 356 functions. Of these functions, 140 are functions that don't take arguments (mostly structural words such as determiners), 85 are one-argument coercions, and 131 are syntactic combinations with more than one argument (the only class that needs dependency configurations). This abstract syntax is the **core RGL**, which is specified as the minimum to be implemented for a language to count as a "complete" resource grammar. This notion of completeness is of course purely formal, and does not mean that the whole language is analysable by the grammar. But it does mean that the standard GF applications, such as controlled language grammars (Ranta et al., 2012, Dannélls et al., 2012, Kaljurand and Kuhn, 2013) are directly portable to that language.

In addition to the core RGL, the library has **language-specific extensions**, which need not be shared by all languages. These extensions are grouped hierarchically, so that for instance Romance languages have a set of common extensions, on top of which Catalan, French, Italian, and Spanish have their own extensions. These modules typically make a 10% addition to the core RGL code. The extension modules enable grammarians to experiment with individual languages without bothering about the universality of all constructs. There is after all no reason why all grammatical constructs should be universal: it is good enough if a core subset is.

Another, recent addition to the core RGL is a set of categories and functions that enable wide coverage parsing and translation (Angelov et al., 2014). To maintain the interlingual translation architecture, these extensions are shared by all languages (15 at the time of writing). But they are generally less precise than the original RGL functions. In particular, there is a category of **chunks**, which is used as a robust back-up to syntactically complete parsing.

The present paper focuses on the core RGL, showing how the main syntactic structures of GF are mapped into UD. This is sufficient for two small treebanks, which are covered for 31 languages. Dealing with a larger treebank, covering 15 languages, also includes the robust back-up rules. But this part is less stable and more ad hoc, and we will report the result for the two parts separately.

Figure 2.18 in the Appendix shows the hierarchy of categories in the core RGL. The same picture appears in (Ranta, 2011), and Ranta (2009b) provides a systematic linguistic discussion of the categories and functions. In this paper, we will present the RGL from another perspective, showing how the functions are decorated by UD labels to convert them to dependency trees. Table 2.11 in the Appendix lists the RGL categories with explanations, examples, and corresponding UD POS tags.

[5] http://www.grammaticalframework.org/lib/doc/synopsis.html

Let us walk through an example, which shows many of the main functions in use. Figure 2.5 shows an abstract syntax tree for the sentence *my two brothers and I would not have bought that red car* and its RGL equivalents in 29 other languages, artificially constructed to show as many structures as possible. The tree is decorated with UD dependency labels.

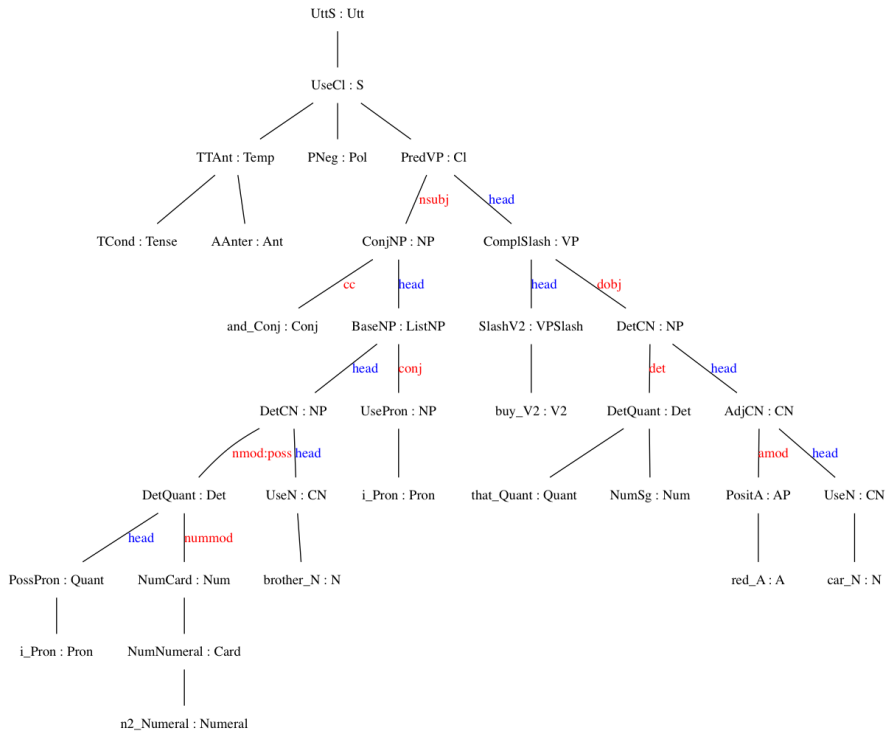


Figure 2.5: Dependency-decorated abstract syntax tree for *my two brothers and I would not have bought that red car*.

The topmost category in Figure 2.5 is Utt, utterances, which is built from S, sentence; an Utt could also be built from a question or an imperative, as shown in Figure 2.18. The sentence in turn is built from a clause (Cl) by adding temporal and polarity features: conditional anterior negative. The core RGL has 16 tense-polarity combinations for clauses. The clause is built from a noun phrase (NP) and a verb phrase (VP).

The subject noun phrase is a coordination, built from a list of noun phrases (ListNP) with a conjunction (Conj). The list is a recursive structure, with the base case taking two elements. Longer lists are (in English and many other languages) linearized by using commas, for instance, *her mother, my brother and I*.

The first conjunct of the subject is built from a determiner (Det) and a common noun (CN). The determiner is further analysed into the head quantifier (Quant) and a number (Num). The quantifier is a pronoun (Pron) used possessively, and the number is a cardinal numeral. The second conjunct is just a pronoun.

The verb phrase is built from a slash verb phrase (VPSlash), by providing an

object noun phrase; VPSlash is similar to a "slash category" (VP/NP in the notation of Gazdar et al. (1985)). The VPSlash is here just a two-place verb (V2). V2 is a generalization of TV and covers both transitive and prepositional verbs, as well as verbs taking different complement cases in different languages. A verb being transitive is not a multilingual invariant and therefore not maintained in the abstract syntax. The complement NP is built from a Det and a CN that has an AP modifier. The determiner has a dummy number (NumSg) indicating that the quantifier `that_Quant` is used in the singular form.

As we can see from the tree, most of the dependency labels are straightforward to define, since the words ultimately appear as categorized lexical items. But some of them need special attention, in particular the tense and polarity features, which in English are realized syncategorematically as auxiliary verbs (discussed in Section 2.5.4). Also coordination is worth discussion (Section 2.4.5), as it is a perennial question in dependency parsing.

2.3.2 Overview of UD

The UD annotation scheme defines a taxonomy of 40 relations as the core or *universal* dependency label set (shown in Table 2.12 in the Appendix). This taxonomy is further refined (or reduced) to address language-specific extensions, not necessarily shared by all languages. The scheme also defines universal sets of part-of-speech tags and morphological features (shown in Table 2.13 in the Appendix). The part-of-speech tagset includes 17 tags, with extensions to previously proposed Universal Part-of-Speech tagset (Petrov et al., 2012). The project is an effort to **consistently** annotate multilingual corpora with these morphological features, part-of-speech tags and universal labels in the dependency parse tree. Figure 2.6 shows all three layers of UD annotation for a French sentence *Toutefois les filles adorent les desserts au chocolat* (trans. "However girls love chocolate desserts"; example and figure quoted verbatim from Nivre et al. (2016)).

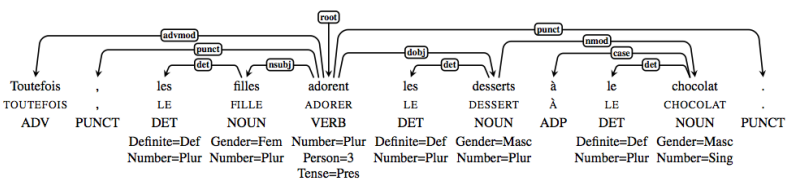


Figure 2.6: UD annotation for a French sentence (lemmas are capitalized). Figure quoted verbatim from Nivre et al. (2016).

Core arguments of clauses are marked as either a subject (`nsubj` in the case of nominal subjects and `csubj` for clausal subjects) or direct/indirect objects depending on their grammatical function. An example is shown in Figure 2.7. In the case of direct objects, distinction between nominal arguments (`dobj`), finite clausal arguments (`ccomp`) and open clausal arguments (`xcomp`) is made. Furthermore, in the case of passive constructions, separate labels are used to mark subjects (`nsubjpass` and `csubjpass`) to indicate transformation of voice. We will look at these constructions separately in Section 2.5.

Non-core dependents of clausal predicates like prepositional phrases attached to

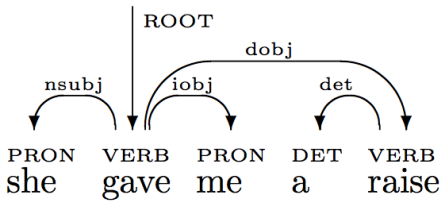


Figure 2.7: UD dependency labels for arguments in clauses

the verb are annotated as nominal dependents using `nmod` label or adverbial modifiers (`advmod` for adverbs and `advcl` for clausal dependents). Other labels used for non-core dependents in a clause are `neg` to mark negation of predicates (and also noun phrases), `vocative` to mark vocative noun phrases and `expl` for expletives. Expletives are nominals that appear with labels for the core arguments in a clause, but have no semantic significance by themselves (for example, there is an expletive in *there is a cat in the house*). More frequently found labels in annotated corpora from this class are `aux` for auxiliary verbs, `auxpass` for auxiliary verbs in passive voice constructions and `cop` for copula verbs. The mapping for auxiliary verbs and copulas is discussed in Section 2.5.

An example of language-specific extensions defined in the UD annotation scheme are labels used to annotate these non-core nominal dependents. In English, the `nmod` label is refined to indicate temporal adverbs `nmod:tmod`, possessive noun modifiers `nmod:poss` and noun phrases used as adverbials `nmod:npmod`. In case of Swedish, the `nmod` is further refined to indicate agents used in passive voice constructions `nmod:agent` and the same possessive noun modifiers as in English. The annotation scheme allows fine-grained extensions to a label, but the choice of annotating the extension is left to the annotators of the language. The guidelines for a specific language provides details on when to annotate these extensions in the language. The mapping we discuss in this paper will ignore these fine-grained extensions, since the goal is to map the core RGL to the core label set. But it is possible to add the fine-grained distinctions to the mapping if we are interested in independently analysing each language. In our own experiments, the extensions to nominal dependents for possessive noun modifiers, and noun phrase adverbials are defined in a separate mapping on the abstract syntax. This mapping is defined only for English. Similarly, the extension in Swedish for agents in passive constructions is equally straight forward to define.

When annotating noun phrases, dependents are typically labelled using either `nummod` for numerical modifiers, `appos` for appositions or a generic `nmod` label for all other nominal modifiers like prepositional phrases. There is a `det` label for determiners and an `amod` label for adjectival modifiers. Relative clauses that modify the noun are marked using the `acl` label (*that we saw* in the NP *the children that we saw*). Prepositions are always marked as modifiers using the `case` label.

Other labels defined in the core label set include `cc` and `conj` for coordination constructions (an example of flat-structure provided for coordinations is shown in Figure 2.9), `compound` for compounding, `mwe` to treat multi-word expressions and a loosely defined `goeswith` label for robust analysis of web and raw texts. Annotation strategies for these specific labels will be clearly explained when we discuss the

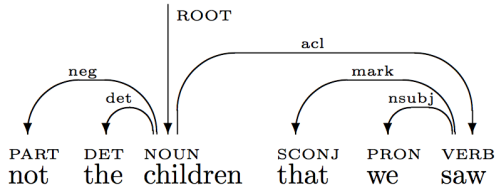


Figure 2.8: UD labels for modifiers in NPs

mapping between the GF-RGL for these phenomena. In this paper, the labels used for defining these loose joining relations will not receive critical attention.

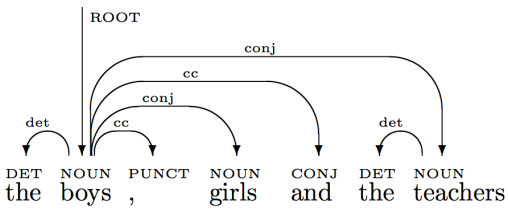


Figure 2.9: UD labels for a coordination of 3 NPs

2.4 Dependency mappings: straightforward cases

In the general case, we define the mapping between functions in the core RGL and the universal UD labels using dependency configurations to UD labels. The mapping to the core UD labels allows bootstrapping treebanks for new languages using the abstract syntax defined in GF-RGL. Additionally, we also define a fine-grained mapping over these functions to derive language-specific UD labels defined in the annotation scheme. Throughout this paper, we will mainly describe the mapping to the core UD labels.

2.4.1 Clausal predicates: predication and complementation

We will start by detailing the mappings for functions in the GF-RGL library used to build declarative clauses. The example shown in Figure 2.5 discussed some of these functions. Table 2.1 shows the complete set of functions used to construct these clauses, the argument types and value type of the resulting phrase and the dependency configuration for each of these functions.

The `PredVP` and `PredSCVP` functions are the main functions responsible for predication. The interpretation of the type signature shown in column 2 of the table is as follows- the `PredVP` function takes a noun phrase (NP) and a verb phrase (VP) and constructs a clause (Cl). Similarly, the `PredSCVP` function takes an embedded clause SC and a verb phrase and constructs a clause. Figure 2.10 shows the parse trees for the example sentences, *John killed him* and *that she came will make news*. We map the noun phrase in the `PredVP` function to the `nsubj` label, and in the case of `PredSCVP`, the embedded clause is mapped to the `csubj` label.

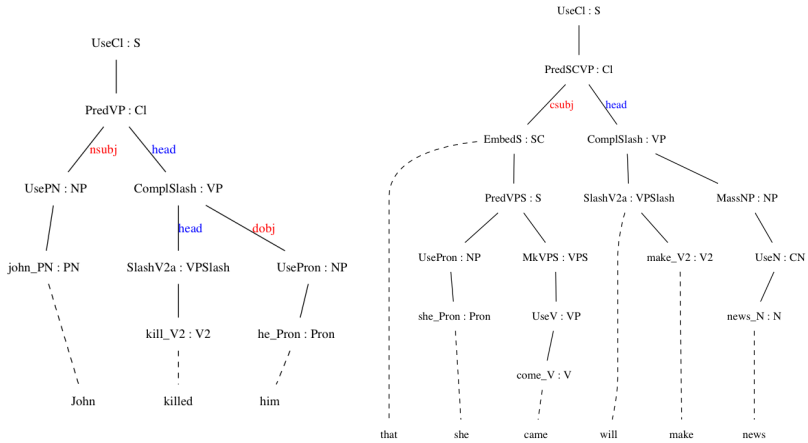


Figure 2.10: Decorated parse trees showing nominal and clausal predication in GF. The parse trees shows the abstract function names, the category of each node and the UD labels.

VP are created using the `ComplSlash` that takes a VPSlash type phrase and NP phrase as a core argument. Alternative complementation functions are `ComplVS` for clausal arguments S (*he says that you want to swim*), `ComplVV` for verb phrase complements (*want to swim*) and `ComplVA` for adjectival phrase complements (*feel bad*). In these cases, we map the NP phrase using a `dobj` label and the clause S using the `ccomp` label. The complement arguments of `ComplVV` and `ComplVA` functions are mapped using the `xcomp` label. In all complementation functions, the arguments with different verb types (VPSlash, VS, VV, VA) are marked as the head of the phrases.

Verbs that take sentential arguments VS in the GF-RGL library are used in the `ComplVS` function to create a VP. The clausal argument (S), is mapped to the `ccomp` label. Alternatively, the RGL also defines a `ComplVQ` function for verbs that takes question clauses as arguments (*do you know who did it*), here also we map the clausal argument to the `ccomp` label.

Similarly, `SlashVP` and `SlashVS` functions are alternate ways to combine an NP with a VPSlash (verb phrase missing an NP), or ClSlash (clause missing an NP). These types of phrases are used to create question clauses and relative clauses. In both these functions, we map the arguments to `nsubj` and `nsubj` and `ccomp` respectively.

Passive voice constructions in GF are created using a type-raising function to create the VP phrase, by dropping one of the mandatory arguments. In the case of transitive verbs and ditransitive verbs, the VPSlash type is converted into a VP type without any additional NPs. The predication functions `PredVP` and `PredSCVP` remain the same since the voice is localized in the sub-tree corresponding to the VP. In order to distinguish the `nsubj` and `nsubjpass` labels in the case of passive voice, we extend our dependency configuration rules defined on the abstract syntax. We describe the mapping for passive constructions in Section 2.5.1.

It is worth mentioning here that the entire mapping of the UD labels is encoded in a declarative fashion, exactly as shown in Table 2.1. Our mappings are the first and the third columns in this table. The other two columns are shown for convenience. The declarative style of specification allows for the conversion algorithm from GF-RGL

PredVP	NP -> VP -> Cl	nsubj head	*John walks*
PredSCVP	SC -> VP -> Cl	csubj head	*that she came will make news*
ComplSlash	VPSlash -> NP -> VP	head dobj	*love it*
ComplVS	VS -> S -> VP	head ccomp	*say that she runs*
ComplVQ	VQ -> QS -> VP	head ccomp	*wonder who runs*
ComplVV	VV -> VP -> VP	head xcomp	*want to sleep*
ComplVA	VA -> AP -> VP	head xcomp	*become red*
SlashVP	NP -> VPSlash -> ClSlash	nsubj head	*(whom) he sees*
SlashVS	NP -> VS -> SSlash -> ClSlash	nsubj head ccomp	*(who) he says that she loves*

Table 2.1: UD mappings for declarative clauses

trees to remain independent of the annotation scheme, allowing one to easily switch between different annotation schemes.

2.4.2 Adverbial modifiers

Adverbial phrases that modify VP phrases are analysed using either `AdVVP` function that takes a VP phrase and a AdV phrase (*always sleep*) or `AdVVPSlash` that modifies a VPSlash instead of a VP phrase. Alternatively, simple adverbs can modify a VP phrase using the `AdvVP` function (*sleep here*). For all these functions, we map the head of the adverbial phrase using the `advmod` label. Table 2.2 lists the functions used to modify VP phrases.

AdVVP	AdV -> VP -> VP	advmod head	*always sleep*
AdVVPSlash	AdV -> VPSlash -> VPSlash	advmod head	*always use (something)*
AdvVP	VP -> Adv -> VP	head advmod	*sleep here*
AdvVPSlash	Adv -> VPSlash -> VPSlash	advmod head	*use (something) here*
AdvS	Adv -> S -> S	advmod head	*then I will go home*
AdAP	AdA -> AP -> AP	advmod head	*very warm*

Table 2.2: UD mappings for adverbial modifiers

2.4.3 Questions and relative clauses

The GF-RGL provides a module for questions combining interrogatives (IP, IComp, IAdv) with verb phrases (VP, VPSlash) and clauses (Cl, ClSlash). For example, the function `QuestIAdv` takes a clause like *John walks* and a pronoun like *why* to construct the question *why does John walk*. Similarly, relative clauses are constructed using one of NP, VP or ClSlash types. Table 2.3 shows the mappings defined for functions used in constructing question and relative clauses.

2.4.4 Noun phrases and modifiers

The GF-RGL provides two primary kinds of functions to create noun phrases. Functions used to generate noun phrases and phrases with adjectival modifiers and functions used to modify these noun phrases using prepositional phrases, appositional NPs and relative clauses. We will present these two groups separately. There are in all two

QuestIAdv	IAdv -> Cl -> QCl	advmod head	*why does John walk*
QuestIComp	IComp -> NP -> QCl	head nsubj	*where is John*
QuestVP	IP -> VP -> QCl	nsubj head	*who walks*
QuestSlash	IP -> ClSlash -> QCl	dobj head	*who does John love*
QuestQVP	IP -> QVP -> QCl	nsubj head	*who buys what he is selling*
RelSlash	RP -> ClSlash -> RCl	mark head	*whom John loves*
RelVP	RP -> VP -> RCl	who loves John	*mark head*

Table 2.3: UD mappings for questions and relative clauses

different categories for nouns defined in the RGL, CN for the basic nouns, N2 for nouns that take a noun complement (*mother of king, list of names*). The primary `DetCN` function takes a determiner and a CN to create a NP (*the boy, this paper*). N2 type nouns take an NP complement phrase to construct a CN. In the `DetCN` function, we map the determiner to the `det` label and make the CN argument head of the NP phrase. In `ComplN2` and `ComplN3` functions, NP arguments are mapped as modifiers of the noun using the `nmod` label (*names* is marked as the child of *list* with `nmod`). Alternative ways to generate NP phrases use `CNIntNP` for phrases like *level 5*, `CNNumNP` for *level five*; in both cases we map the Int/Card argument to the `nummod` label. The `AdjCN` function is used to modify nouns; these nouns still need a Det argument to become noun phrases (*the blue house*). Here, the head of the adjective phrase AP is mapped to the `amod` label. Table 2.4 lists the functions used to construct the basic NP phrases and the respective mappings.

DetCN	Det -> CN -> NP	det head	*the man*
ComplN2	N2 -> NP -> CN	head nmod	*mother of the king*
CNIntNP	CN -> Int -> NP	head nummod	*level 5*
CNNumNP	CN -> Card -> NP	head nummod	*level five*
AdjCN	AP -> CN -> CN	amod head	*big house*
DetQuant	Quant -> Num -> Det	det head	*these five*
DetQuantOrd	Quant -> Num -> Ord -> Det	det head amod	*these five best*

Table 2.4: UD mappings for generating basic noun phrases

The Det type in GF is used for determiners, typically constructed by taking a Quant type and Num type. For the definite article *the*, the analysis would be

```
the:  DetQuant (DefArt) (NumSg)
the:  DetQuant (DefArt) (NumPl)
```

This Det type and the `DetCN` is used to construct determiner phrases like *these five* that can modify a noun or act as noun phrases by themselves. An alternative function to generate the Det type is `DetQuantOrd` using which phrases like *these five best* can be analysed. In both these cases, we map the Quant type as the head of the phrase and the Num type is mapped using the `nummod` label. As such in phrases like *these boys* where there are no numerical modifiers in the phrase, we obtain the same structure as in UD i.e. *these* is given the `det` label and *boys* is the head. However in phrases with numerical modifiers (for example *these five boys*), the numerical modifier *five* is attached to the quantifier *these* and not to the noun *boys*.

In addition to these functions, NPs can be modified using other NPs for apposition, prepositional phrases. The list of functions is shown in Table 2.5. `ApposCN` is used for apposition (*Sam, my brother*) and `PossNP` for possessive nominal modifier constructions (*Marie's book*), `PartNP` for partitive constructions (*glass of wine*). The modifier NP phrases are mapped to `appos`, and `nmod` labels. In the `PossNP`, the noun representing the possessive modifier is assigned the `nmod` (or `nmod:poss` for English) label where as in the `PartNP` function, it is the opposite.

ApposCN	CN -> NP -> NP	head appos	*city Paris*
PossNP	CN -> NP -> CN	head nmod	*Marie's brother*
PartNP	CN -> NP -> CN	head nmod	*glass of wine*
ComparA	A -> NP -> AP	amod head	*warmer than I*
RelCN	CN -> RelS -> CN	head acl	*house that John bought*
RelNP	NP -> RelS -> NP	head acl	*Paris, which is beautiful*

Table 2.5: UD mappings for modifying noun phrases

The `RelCN` and `RelNP` functions are used to modify nouns and noun phrases with relative clauses. In this case, we map the head of the relative clausal predicate using the `acl` label.

2.4.5 Coordination

The RGL defines multiple functions for building coordination constructions by combining lists of phrases (two or more) in the same category using conjunctions. `Base*` functions accepts two phrases of same type and create a tree of type List*. The List* types are combined with conjunctions (marked as Conj) using `Conj*` functions to form trees corresponding to coordination constructions. Additionally, `Cons*` functions recursively add a new phrase to List* phrases. These `Cons*` functions are used when coordinating more than two phrases of a given type. The `Conj*` functions handle both syndetic and asyndetic coordination constructions in all languages. Syndetic coordination is a form of coordination with the help of an explicit coordinating conjunction (*ham and eggs*) while asyndetic coordination allows for conjunctions to be omitted from one or all the conjuncts in the coordination construction (*he came, saw and conquered*).

Let us look at an example of adverbial coordination using the phrase *here, there and everywhere*. The parse tree for this example is shown in Figure 2.11. Adverbials *there* and *everywhere* are first analysed using the `BaseAdv` function. The `ConsAdv` function adds *here* to the list of adverbs from the `BaseAdv` function. This resulting ListAdv phrase is used by the `ConjAdv` function along with the conjunction *and* to form the AST for the coordinated phrase.

The existing UD scheme annotates the first conjunct in the construction as the head, and treats the rest of the conjuncts as modifiers attached directly to the head via the `conj` relation. Similarly, conjunctions are attached to the same head via the `cc` relation. Table 2.6 shows the mapping defined for coordinating any two types in the GF-RGL. Note that when coordinating more than two phrases, the first conjunct always comes from the argument of the `ConsNP` function. The same mapping is defined for the set of these three functions defined for all types in GF-RGL (complex

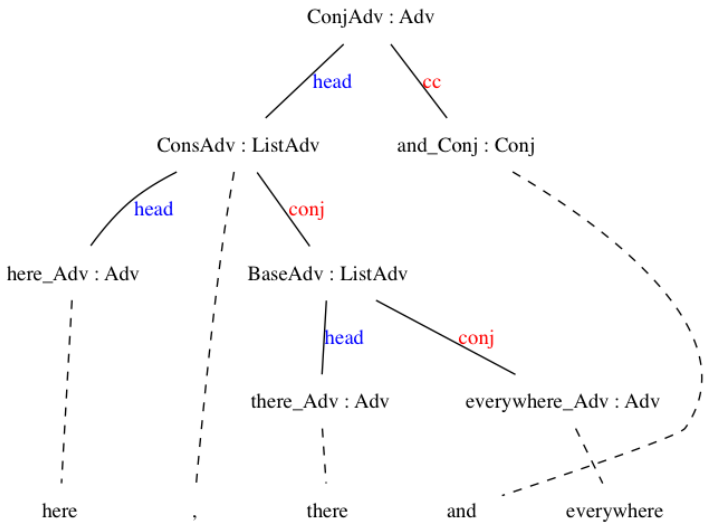


Figure 2.11: Coordination of three adverbials

nouns CN, noun phrases NP, adjectival phrases AP, adverbial phrases AdV, simple clauses S and relative clauses RS).

BaseT	T -> T -> ListT	head conj
ConsT	T -> ListT -> ListT	head conj
ConjT	Conj -> ListT -> T	cc head

Table 2.6: UD mappings for generic type T in GF-RGL. T can be CN, NP, AP, AdV, S and RS

2.5 Dependency mappings: Problematic cases

The mappings described in the previous section are straightforward rules defined over the abstract syntax in GF-RGL. When the equivalent labelled dependency tree for a given phrase can be completely identified from the AST by defining a mapping to dependency labels for the functions over its **immediate** arguments, then such a mapping is sufficient to construct a fully connected dependency tree from the AST.

This is not necessarily the case. Using only rules described in the previous section, the mappings can result in a dependency tree with edges labelled using the `dep` label. The main reason for this is the presence of the syncategorematic words in GF (discussed in Section 2.2.4) due to the abstraction level of GF-RGL intended to maximize sharedness across languages.

For example, GF-RGL defines multiple functions in abstract syntax for existential clauses (*there is a cat*). Existential clauses are analysed using the `ExistNP` or `ExistNPAdv` function. The `ExistNP` function takes a NP and constructs a clause Cl. The dummy pronoun, an expletive *there* and the copula verb *is* are abstracted by the

`ExistNP` function in the AST. Similarly, the `ExistNPAdv` takes a NP and a modifier phrase Adv (for example, *in the bag* to construct the clause. Linearizations of these functions vary across languages depending on whether expletives are necessary and the choice of copula in the language. See Figure 2.12 for the parse trees of the clause in English and Bulgarian. The Bulgarian translation of the existential clause neither contains the expletive or the copula, instead the verb used here is annotated as the main head in current UD annotation. We will discuss existential clauses in Section 2.5.6.

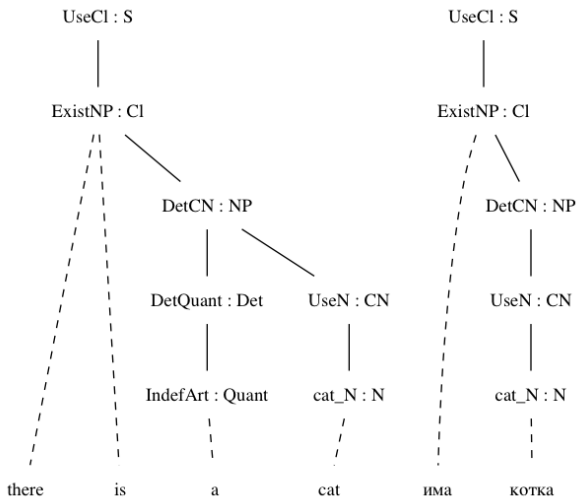


Figure 2.12: Parse trees of the existential clause *there is a cat* and its translation in Bulgarian

We extend the set of mappings defined until now with three additional types of rules:

i) **local abstract rules** are rules on abstract functions addressing their immediate arguments only (all rules defined until this point are of this type)

ii) **non-local abstract rules** allow mappings to be defined on the abstract functions using more context than the immediate arguments of the functions.

iii) **local concrete rules** are defined for each language on the respective linearizations of an abstract function.

iv) **non-local concrete rules** are defined to map the linearization of a function to the UD labels using more contexts like in the case of non-local abstract rules.

	Abstract	**Concrete**
Local	Fun Label+	Fun ReLabel+
Non-local	(Fun args*) Label+	(Fun args*) ReLabel+

Table 2.7: Type of rules in the extended mapping to construct full UD trees

Table 2.7 shows the different types of rules and specification formats used to encode UD information. We previously mentioned that the mapping is encoded in a declarative manner to remain independent of annotation schemes. We briefly explain these new types of rules before explaining how they are used in our conversion. This is discussed in more detail at the end of this section (and Appendix).

The **local abstract rules** discussed previously are encoded using the syntax shown in Section 2.2. `Fun` here refers to the name of a function in the abstract syntax (`PredVP` or `ComplSlash`). One argument of `Fun` is mapped to a label called `head` and the rest are mapped to corresponding UD labels. In the trivial case, where a function takes only one argument, the argument is always mapped to the `head` label.

Non-local abstract rules are an extension of local abstract rules, that are applied only if the arguments of `Fun` in the abstract syntax tree match the values (or patterns) specified in this rule. args* represents the list of arguments of the function `Fun` (`*` corresponds to the Kleene operator in regular expressions) and each item in this list expresses a pattern for the arguments. We explain these in Section 2.5.1. These patterns for arguments encode a context that is outside the scope of what is matched in local abstract rules. Non-local rules can simply be interpreted as multi-level rules in the grammar while local rules are one-level rules in the grammar. We will see these rules in more detail in Section 2.5.1.

Local concrete rules are introduced to address different realizations of an abstract function across multiple languages. The example of existential clauses mentioned above is an instance of this. These mappings are encoded using a list, similar to local abstract rules. In both cases of local and non-local concrete rules, each item in the list represents a relabelling operation of an edge in the dependency tree. This relabelling can be one of three types: relabel an existing edge in the dependency tree with a new label, relabel an edge with a new label after reversing the direction of the edge or add both an edge and a label to the dependency tree. These rules will be discussed in Section 2.5.2 for cases of copula constructions and Section 2.5.3 for verb phrase complements and prepositional verbs.

Non-local concrete rules are again an extension of local concrete rules, where the relabelling operations are applied only if the arguments of the abstract function Fun match the context specified in these rules. These rules are explained in the context of clausal negation and auxiliary verbs in Section 2.5.4.

The terms “local” and “non-local” refer to the scope over which the conversion algorithm matches the specified mappings. The mappings are always specified over functions in the abstract syntax. The terms “abstract” and “concrete” refers to the layer of syntax on which the mappings are applied. Abstract rules are applied on the AST and concrete rules are applied after abstract rules on labelled dependency trees for the specific language. Both sets of local and non-local concrete rules must be defined for each language in the RGL. In the absence of concrete rules, the conversion results in a connected UD tree, where some edges are connected artificially using the `dep` label. When the edges marked with these dummy labels are *collapsed*, the resulting dependency tree structure can be a representation that is closer to the semantics of the sentence. We will show this in our experiments described in Section 2.6.

2.5.1 Passive voice constructions

Passive clauses in GF are constructed using functions that allow a verb to drop one of its obligatory arguments to construct the VP, for example `PassV2` and `PassVPSlash`.

Let us look at the ASTs for the clause *John killed him* and its passive counterpart (*he was killed*), shown in Figure 2.13. The grammatical subject of the passive clause is incorrectly mapped to `nsubj` label using only local abstract rules.

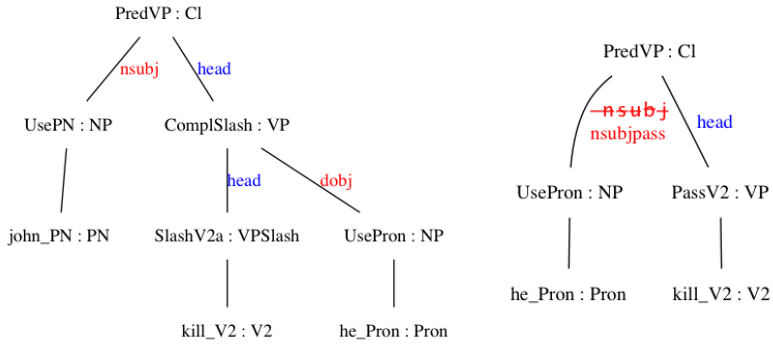


Figure 2.13: Decorated ASTs for the clause *John killed him* and its passive counterpart *He was killed* **using both local and non-local abstract rules**

The `PassV2` function in GF-RGL allows a transitive verb to form a VP phrase without the obligatory NP phrase (`dobj`). The extensions in GF-RGL define a function `PassVPSlash` that can be used to construct passive voice constructions for both transitive and ditransitive verbs. This function can be understood to be a generalization of the `PassV2` defined in the core RGL. However, the top level predication function remains the same (`PredVP`) irrespective of active or passive voice.

The UD annotation scheme distinguishes between subjects of active and passive clauses (NP arguments) using two labels — `nsubj` and `nsubjpass`. In order to make this distinction in our mapping, the rules corresponding to `PredVP` should be enriched with more context. We define the following non-local rules for `PredVP` function for this purpose.

```
(1) (PredVP ? PassV2)       nsubjpass  head
(2) (PredVP ? PassVPSlash) nsubjpass  head
(3)  PredVP                 nsubj      head
```

The context in this example is encoded using abstract function names corresponding to passivization of VP phrases. The first rule `(1)` is **only** applicable in the case when `PassV2` appears in the sub-tree corresponding to the VP phrase i.e. transitive verbs in passive voice constructions. Similarly, the second rule `(2)` addresses di-transitive verbs in passive voice, when `PassVPSlash` appears in the sub-tree of the VP phrase. Finally, the general rule (`(3)`, also the local abstract rule) is applied in all other cases, when none of these two contexts are matched i.e. in active voice constructions. The `?` character represents a meta-variable in the tree. This is interpreted as matching anything i.e. there are no restrictions on the sub-tree corresponding to this argument. In the example of the predication function, there are no restrictions on the sub-tree corresponding to the NP argument. Using these non-local rules, grammatical subjects of passive voice clauses are mapped to the `nsubjpass` label instead of the `nsubj` label.

We also define similar rules for `PredSCVP` function, used in analyses of constructions with clausal subjects. Recall from Section 2.4 that the `PredSCVP` function is

used to construct a clause using an embedded clause and a VP phrase.

The passive voice constructions described above is addressed using a limited non-local context (the daughter nodes). However, in principle it is possible to define contexts corresponding to abstract functions using expressions of arbitrary depth.

Before extending the format of the dependency configurations, we considered the alternative of changing the RGL to resemble UD annotation in this particular case. RGL localizes the voice transformation to the VPs while UD uses a different label for subject arguments that is non-local to the VP, which could be easy to achieve in GF as well. However, if we consider co-ordinated constructions with subject sharing (*he killed his wife and was chased by the police*), the shared subject *he* should be both an `nsubj` and an `nsubjpass`. UD resolves the conflict with its flat structure to coordinations, by choosing the label appropriate for the first conjunct (`nsubj`). However, by localizing the voice to the VP, it is possible to give separate interpretations to the subject.[6]

At this point, it is clear that the addition of non-local rules makes it necessary to introduce a notion of precedence between the local and non-local rules for a specific function. We will formally address this question in Section 2.5.7. For now, it is enough to note that non-local rules should precede local rules, and this is specified in the semantics of dependency configurations.

2.5.2 Copula constructions

Copula constructions in GF-RGL are analysed in two steps: any of the `Comp*` functions are used to convert phrases into copula complements (Comp in GF-RGL) followed by `UseComp` function that converts this complement into a VP. The `UseComp` function also introduces the copula verb into the VP (when necessary). The choice of copula verb and its form is specified in concrete syntax and the abstract syntax tree does not know exactly what the copula verb in the specific language is. This abstraction is desired because the realization of copula constructions varies across languages. For example, the linearization of the `UseComp` in Russian (so called *zero-copula* language) doesnot contain a copula verb in present tense. The linearization rules in English, Finnish and Swedish introduce the verbs *is*, *on* and *är* respectively. In the case of Chinese and Thai, the copula verbs are restricted to only some selected complements. Figure 2.14 shows the decorated parse tree in English for the sentence *John is clever*. Notice the dummy `dep` label for the copula verb *is* obtained using abstract rules.

The UD scheme in the case of copula constructions annotates the copula verb as a child of the complement using the `cop` label. In order to match this structure in our mapping, we introduce a **local concrete rule**, that marks copula verbs using the `cop` label. Shown below is the concrete rule specified for English:

```
UseComp  head {"am", "is", "was",
               "are", "were", "be", "been", "being"} cop head
```

This rule specifies that if any of the different forms of the copula verb in English (*am, is, was, are, were, be, been*) are found under the functions in the spine corresponding to the `head` of the `UseComp` function in the AST, they should be attached to the head of the `UseComp` function using the `cop` label. The expression

[6] With the above local rules for `PredVP`, the AST for this example will yield the same dependency tree as the UD tree, which is just what we wanted.

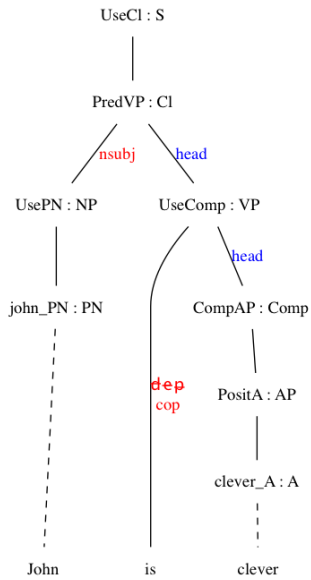


Figure 2.14: Decorated parse tree for a copula construction in English

```
head {"am", "is", ...} cop head
```

specifies a single relabelling operation in the concrete rule.

2.5.3 Verb phrase complements and prepositional verbs

Let us look at one more example of these local concrete rules used in our mapping for verb phrase complements and prepositional verbs. GF-RGL defines a different function for complementation in case the argument is a verb phrase (these verbs are marked as VV in RGL). The `ComplVV` function is used with verb phrase complements. The `ComplSlash` function used for nominal objects additionally is also used for prepositional verbs (*listen to ...*). For these verbs, the required case of the NP argument is specified in the entry corresponding to the verb in the lexicon. This case information is then propagated to the `ComplSlash` function (specifically the function stores this information in a record with label c2).

Figure 2.15 shows the parse tree for the sentence *I like to listen to music in French*. The verb *like* is an example of verb phrase complement and *listen to* is a prepositional verb. Note that in both these functions, the infinitive marker to the verb phrase complement and the case are abstracted from the AST, these are localized in the concrete syntax (seen only in parse tree) of the language. The local abstract rules shown in Table 2.1 map the head of these arguments to `xcomp` and `dobj`, but leave the infinitive marker *to* and preposition *for* unlabelled.

```
ComplSlash  head .c2      case   dobj
ComplVV     head {"to"}   mark   xcomp
```

The local rules above specify the following relabeling operations: for the `ComplSlash` function, the preposition/case in the .c2 record field should be relabelled as a child of

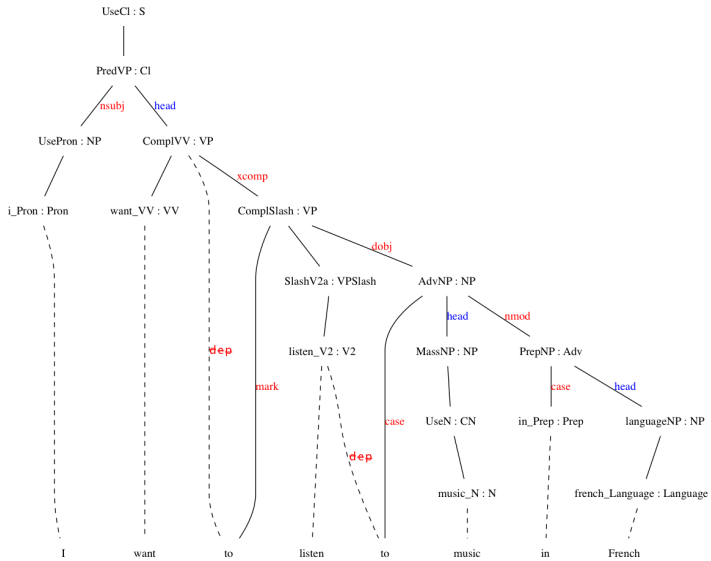


Figure 2.15: Decorated parse tree showing verb phrase complements and prepositional verbs

the direct object (marked `dobj`) using the label `case`. In the `ComplVV` function, the infinitive marker *to* is labelled using the `mark` label as a child of the verb complement.

2.5.4 Auxiliary verbs and verbal negation

Non-local concrete rules address dependencies of words introduced in the concrete syntax where context inside the arguments of an abstract function are necessary to determine the respective UD labels. In order to understand the necessity of these rules, we will look at a concrete example (shown in Figure 2.16). The motivation in this example is to address cases of auxiliary verbs constructed from tense in GF and clausal negation in our mapping.

The decorated parse tree in Figure 2.16 shows the analysis provided by GF for the sentence *John had not slept.* The parse tree for the clause *John sleeps* has the same functions and categories as this sentence. Why is this? The Cl type resulting from the predication (`PredVP`) function represents an untensed clause, in other words a simple proposition. This proposition is transformed to a sentence when `Tense` and `Polarity` arguments are given to the `UseCl` function along with an untensed clause Cl. The `UseCl` function will then result in a sentence of type S. Note the `PNeg` value for the polarity of the clause and the `TPast,AAnter` for the tense in our example. The *not* appears due to the `PNeg` value and the auxiliary verb *had* due to the anteriority of the tense. The tense parameter dictates the choice of auxiliary verbs and the polarity parameter if *not* occurs in the clause. The equivalent UD labels are also shown in the parse tree. It can be seen that the modifiers according to UD are not always in the same sub-tree as the head (*not* modifies *slept*).

In order to make the mapping to UD labels, we need non-local concrete rules, as the specific auxiliary verbs in a language are abstracted by the Tense parameters in the

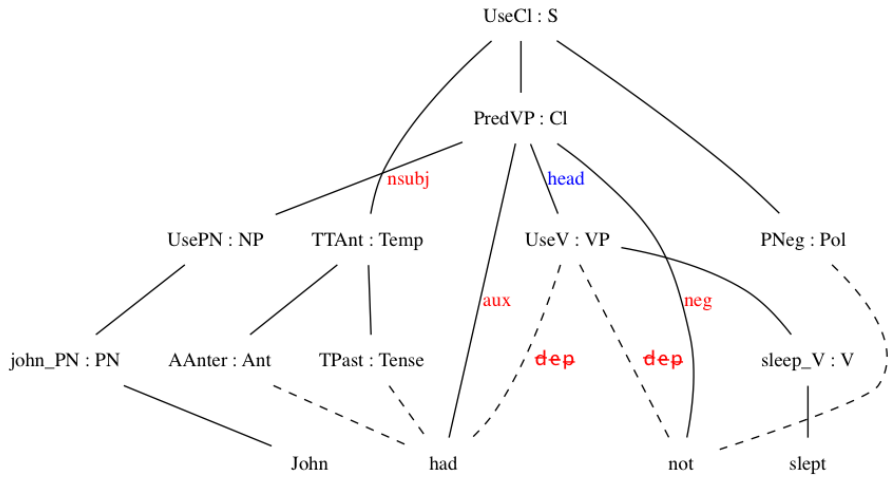


Figure 2.16: Decorated parse tree showing tense and clausal negation with the UD labels

AST and are only available in concrete rules for the language. The non-locality is also required to handle cases of negation.

```
(1) (UseCl ?     PNeg ?)  head {"not"} neg head
(2) (UseCl tense   ?  ?)  head {*}   aux head
```

The non-local rules for auxiliaries and copulas are defined as shown above. The negation modifier (*not*) is introduced into the clause by the `UseCl` function if its polarity argument has the `PNeg` value. The word *not* is introduced under the `UseV` function, which lies on the spine corresponding to the head of the `UseCl` function. The same is also true for auxiliary verbs, they are introduced by the Tense parameter under the `UseV` function. For clausal negation, the edge between the head word of the `UseCl` function and the word *not* is labelled using the `neg` label, with *not* as the child. Similarly, auxiliary verbs introduced by the `Tense` argument to the `UseCl` function, irrespective of what exactly is the auxiliary are always connected to the head using the `aux` label. The `{*}` in the rule specifies that all and any auxiliary verbs can be matched in this relabelling operation.

2.5.5 Multi-word expressions

One of the UD taxons we haven't discussed yet are cases of multi-word expressions and some instances of compounding. Compounding encompasses a wide range of linguistic phenomena, from compound nouns and phrasal verbs to non-compositional multi-word expressions. In the UD scheme, this also includes foreign language phrases and two parts of a single word in poorly written text. The annotation of compound nouns in UD (for example *phone book*, *agent provocateur*) is fairly consistent across languages. However, there is variation in annotations of phrasal verbs. This is not surprising given the variety of "phrasal verb" constructions in different languages.

In the GF-RGL, compounds are handled in two different ways: compositional compounds are analysed using the functions in RGL mentioned previously `CompoundN` and

`CompoundAP` (for example *control systems*, *language independent*). Other compounds i.e. non-compositional compounds are addressed inside the lexicon specific to a language. The analyses provided to these types of compounds is very much dependent on the concrete syntax of the language, whether the translation in the specific language is compositional or not. It is also possible that the translation is simply lexicalized, without any compounding in a languages. For these reasons, the abstract syntax shared for these compounds is only the function name, specific analysis of the compound is localized in the concrete syntax. This is because languages allow for discontinuous compounds, particularly in cases of phrasal verbs and light verb constructions (for example *shut off* in *shut it off*, *shut off the TV*).

We provide three dependency labels for all these compounding phenomena in our mapping, it can either be a `compound`, `mwe` or `goeswith`. However, unlike the local abstract rules for the functions that correspond to compositional compounds, these labels are defined on the concrete syntax of a specific language. Shown below are the local concrete rules used for particle verbs *shut off* and multi-word expressions *according to* and *in front of*.

```
shut_off_V2       head {"off"} compound head
according_to_Prep head {"to"} mwe head
in_front_of_Prep  head {"front", "of"} mwe head
```

The relabelling operation in the case of `shut_off_V2` specifies that the particle *off* be added as a child to *shut* with a `compound` label. Similarly, for `in_front_of_Prep`, edges from *in* to *front* and *of* are relabelled using the `mwe` operation. In all these cases, it is implicitly understood that the first word (*shut*, *according*, *in*) is the head of the function. This is in sync with the current UD annotation scheme in its current form. However, the rule specification format is flexible to allow a different choice of the head, say for example, a semantic head for multi-word expressions. The UD annotation scheme where multi-word expressions are concerned might be revised in the future, and our current mapping scheme allows room for any such changes.

2.5.6 Idiomatic and Semantic (CxG) Constructions

Finally, we discuss semantic constructions that are commonly found in languages and can be translated idiomatically in languages. We discussed at the beginning of this section how GF-RGL defines functions for constructions like existential clauses (*there is a blue house on the hill*) or cleft sentences (*it is money that was owed to him*). There is also a small subset of *semantic* constructions defined in GF-RGL. One example of these semantic constructions is a function that accepts a number, say 10 and renders the semantic linearization of the VP *is 10 years old* in a specific language.

```
has_age_VP n   -- (someone) "is" n "years old"
```

We define the mappings to UD labels over these semantic constructions defined in the RGL. In order to construct the fully connected UD trees, it is necessary to define both abstract rules and concrete rules over these functions. The rules used for the `ExistNP` function are shown below. The expletive pronoun *there* is marked using the `expl` and the copula verb indicating tense has the label `cop`.

```
ExistNP head {"there"}  expl head ; head {*} cop head
```

Notice that in these examples, the concrete mappings specify more than one relabelling operation. This is possible and the `;` is used as a separator between different operations. The conversion algorithm carries out each of these operations in the order specified by the mapping.

2.5.7 Dependency conversion algorithm and specification language

We show the exact **dependency configuration syntax** used to specify the mappings to UD scheme and more generally to any dependency annotation scheme in Figure 2.19 (in Appendix). The figure shows the BNF fragment specifying the syntax of the configuration rules.

In the case of local rules, both abstract and concrete, the key to the mapping is the abstract syntax function name `Fun` that specifies where the mapping should be applied. Similarly for non-local rules, this key is extended from `Fun` to a pattern of expression that matches sub-trees. Each `arg` in the non-local rule is a pattern for the arguments of `Fun`, the number of such patterns being the same as the number of arguments. The `?` character is used to indicate match everything.

For abstract rules, the mapping is a list of labels of the same size as the number of arguments to function. The items in these lists are any of the labels specified according to the annotation scheme or the special label `head` that encodes what is the head corresponding to the abstract function. By definition, there should be one and only one `head` in each mapping specified.

The mapping in the case of concrete rules is a list of relabelling operations. The number of these operations specified by a single rule depends on the number of lexical items introduced by the concrete syntax in the language that are hidden in the AST. There are three possible relabelling operations defined: renaming an existing edge with a new label (retains the already encoded head-modifier relationship), renaming an existing edge after modifying the direction of the edge with a new label or adding a completely new edge that did not exist before. Examples of all three operations have been shown before. Each relabelling operation is specified as

Label Part Label Label

The first `Label` in a relabelling operation is a label specified in the abstract rules, to indicate functions(s) in the AST marked with `Label`. This specifies a spine in the AST used to locate functions under which syncategorematic words can be found. `Part` specifies either a set of words (in the case of copula verbs) or a record label (in the case of prepositional verbs) or a `*` to match all lexical items introduced under a function in the AST. The remaining two `Label` specify the label of the modifier and the label of the head in the AST in that order.

The precedence of the local and non-local rules corresponding to a function `Fun` is defined in the same order as they are specified in the configuration syntax. In other words, non-local rules that must be applied before are specified before the local rules. By choosing to put this in the configuration, we do not modify the conversion algorithm to specially handle non-local rules.

With this dependency configuration syntax, the dependency conversion algorithm is extended to derive the complete UD tree. In Section 2.2, the conversion method given an AST T and a word sequence S to its corresponding dependency tree was described. The algorithm is divided into two separate steps: conversion using the

abstract rules and the concrete rules in that order. Algorithm 1 (in Appendix) describes the extended conversion algorithm we used in this paper.

2.6 Experiments

In our experiments, we focus on evaluating two things: the mapping proposed in this paper to convert GF trees into UD dependency trees and subsequently how much the abstract syntax in GF-RGL and the mapping is useful in bootstrapping treebanks. We carry out experiments to evaluate the correctness of our mapping and to understand the systematic similarities between GF-RGL and the UD annotation scheme.

In the first set of experiments, using mappings defined on the abstract syntax alone (both local and non-local rules), we bootstrap dependency trees for 31 languages from a treebank of ASTs. Analyses of these bootstrapped treebanks give insights about the UD scheme and the GF-RGL. The lack of concrete local and non-local rules for a specific language result in trees that resemble *collapsed* dependency trees (de Marneffe and Manning, 2008, Ruppert et al., 2015) by deleting the dummy `dep` labels. Due to the lack of concrete rules so far, the bootstrapped treebanks for languages other than English do not contain labels for syncategorematic words, and these `dep` labels are collapsed. Figure 2.17 shows examples of bootstrapped and "collapsed" UD trees for Swedish and Bulgarian in the context of this work. When defined, the concrete rules "decollapse" these trees by introducing the right UD labels for syncategorematic words.

2.6.1 UD test treebank

In order to evaluate the mapping described in this paper, we created two treebanks of ASTs. The first treebank was constructed using examples in UD annotation guidelines for English dependency relations [7]. The ASTs from the GF parser (Angelov and Ljunglöf, 2014) were post-edited to correct the ambiguity choices in these examples. The collection of examples from the UD guidelines ensure good coverage of the dependency relations proposed in UD scheme. Similarly, in an attempt to maximize the coverage of functions defined in the GF-RGL, we created a second treebank composed of minimal examples used to document the linguistic usage of abstract functions in writing resource grammars for new languages[8]. The second treebank is relevant for two purposes: the ASTs from the GF-RGL documentation ensure full coverage over the GF interlingua, and also provide minimal examples/test-cases for purposes of UD documentation. These can be used as unit tests to validate the UD annotation efforts. Unit tests are typically used in software engineering to validate the software i.e. if the actual output of the system matches the expected result. Additionally, these examples can be compositionally combined to automatically create a treebank of ASTs which can then be bootstrapped to UD treebanks for languages in GF. We refer to the former treebank as UD treebank and the latter as GF treebanks here, these should not be confused with the treebanks provided in the UD and GF distribution. Using both these treebanks we evaluate the precision of the mapping and make qualitative analyses.

The UD treebank contains a total of 104 ASTs, 66 ASTs that are covered by RGL and 38 ASTs that use the robust back-up rules in the grammar enabled for wide

[7] http://universaldependencies.github.io/docs/en/dep/all.html
[8] http://www.grammaticalframework.org/lib/doc/synopsis.html

coverage parsing and translation. As these examples are hand-picked to document the annotation guidelines for UD treebanks, the treebank does not have a specific genre. We will report the precision of the mapping rules separately for these two sets. One reason to set apart these two sets is the ad-hoc nature of recovering UD labels in case of rules corresponding to chunking that are a small part of the robust back-up rules. Another reason is that these rules are implemented only for 15 of the 31 languages covered by the core RGL. So, the subset of ASTs covered by the RGL are bootstrapped into all 31 languages while the rest of them are bootstrapped into only 15 languages. Many of these examples have been used previously in this paper as examples in Section 2.3, Section 2.4 and Section 2.5.

The GF treebank contains a total of 400 ASTs, that are covered by the core RGL. These examples are used to document the usage of abstract functions while writing concrete rules for a new language. Hence, these examples provide a complete coverage over the RGL-grammar and are minimal examples of different linguistic phenomena that can be tested for cross-lingual consistency in UD annotations. We bootstrap this treebank into all 31 languages in the RGL (these examples have been previously used in this paper in Tables 2.1 and other tables).

2.6.2 Evaluation

	Local	Non-local
Abstract	132	11
Concrete	17	29

Table 2.8: Distribution of rules for English according to their types

The complete mapping described in this paper, contains a total of 185 rules, 143 of which are defined on the abstract syntax. The remaining 46 rules are defined as either concrete local and non-local rules for English. Table 2.8 shows the distribution of the rules in the complete mapping for English. It is necessary to define concrete rules for each language to construct fully labelled UD trees.

Figure 2.17 shows an example of bootstrapped trees for the Swedish, Finnish and Bulgarian translations of the English sentence *John was killed* generated using only the rules defined on the abstract syntax. Also shown is the full UD tree for the English sentence generated using both the abstract and concrete mappings.

Analyses of English dependency trees

The converted UD trees in the UD treebank have a 99% labelled attachment score (LAS) to the gold UD trees. The dissimilarities in English occur only in two cases: modal verbs and noun phrases, specifically noun phrases with `nummod` labels. A 100% LAS score is achieved by adding additional rules to handle modal verbs in our mappings. While our mappings with these additional rules result in perfect matching with the UD trees, we look at these two specific cases in some detail below.

The UD annotation guidelines treat modal verbs in English the same way as auxiliary verbs, with a POS tag AUX and `aux` label. The lexical features of the word indicate its modality feature, but the label assigned is the same as auxiliary verbs. This is expected since modal verbs in English are realized as auxiliaries in English.

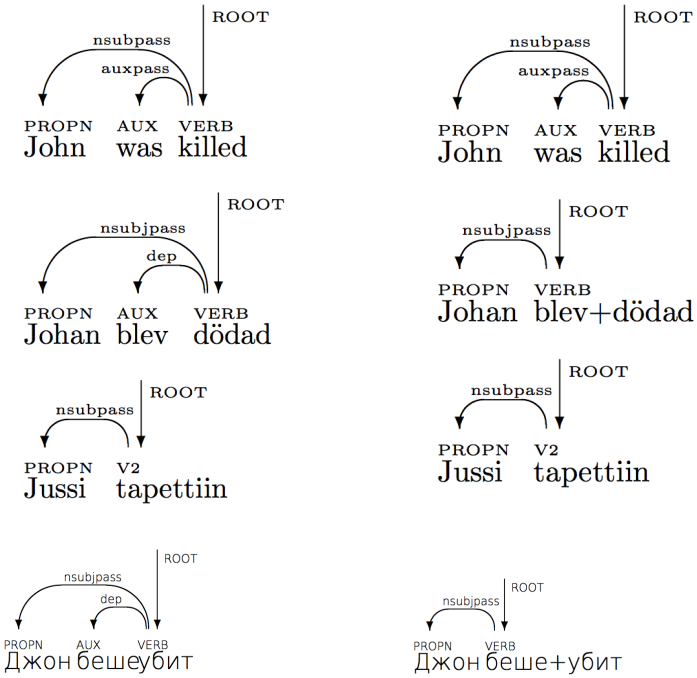


Figure 2.17: Bootstrapped UD trees in Swedish, Finnish and Bulgarian and the complete UD tree in English. The trees on the right are the collapsed variants of the bootstrapped trees.

However, modal verbs in other languages can be realized in inflectional variants of the main verb, for instance, English *would* as conditional forms in French.

Non-auxiliary modal verbs in GF receive a structure that is shared across languages: they are verbs of the category VV, i.e. verbs that take VP complements, similar to verbs like *want* or *like* in *I want to ask you something* and *I like to run*. This analysis of modal verbs from GF-RGL maps the modal verb to the head and the main verb to the `xcomp` label as a child of the modal verb using the mapping defined for the `ComplVV` function (complementation using non-finite arguments). The labels to the arguments of the main verb remain the same though.

It is possible to give modals in English the `aux` label by adding non-local abstract rules for the `ComplVV` function to match only modals:

```
(ComplVV can_VV ?)   aux head
(ComplVV must_VV ?)  aux head
```

However, by defining these mappings on the abstract functions, we would map modal verbs in all the thirty languages to the `aux` label, even in languages where modal verbs are not realized as auxiliary verbs. The UD treebanks in some languages treat the modal verb as the head of the clause, while other languages treat them consistently as auxiliaries (English and Swedish in particular). The problem is that modal verbs are not always realized as auxiliaries across languages. For instance, *can* and *must* in

English translate to *pouvoir* (or *savoir*) and *devoir* in French, which are semantically modal but work otherwise just like normal verbs. In the other direction, *want* is not modal in English, but its equivalent *wollen* in German is. The RGL choice is to have a uniform abstract category VV of VP-complement verbs and to treat the property of being "modal" in the concrete syntax of each language[9].

Alternatively, the mapping of modal verbs can be addressed using concrete rules specific to a language. This allows our conversion method to address the differences in the way modal verbs are treated in different languages in UD. These rules are shown below. These concrete rules relabel the edges of the converted tree in the case of modal verbs to obtain the exact UD tree. What these rules say is that the head of the `ComplVV` function i.e modal verbs in this case (*can* or *must*) are relabelled using the `aux` label while the other argument i.e. the verb is relabelled to be the head of the `ComplVV` function. Defining the mapping for modal verbs using concrete rules for English is the best way to address the different annotations used for modal verbs.

```
(ComplVV can_VV ?)  head {*} aux head; xcomp {*} head head
(ComplVV must_VV ?) head {*} aux head; xcomp {*} head head
```

The other divergence from UD is in the case of noun phrases with numerical modifiers: UD annotation scheme treats all numerical modifiers occurring in NPs as modifiers of the head noun using the `nummod` label. While this is valid in almost all cases, it is possible for the number to require another label than a modifier. For example the phrase *level 3* is treated as a compound in GF, one where both *level* and *3* contribute to the meaning. We retain this analysis by mapping this to the `compound` label, rather than using the `nummod` label. Also, in instances of how noun phrases with numerical heads, as in the case of *these five will come with me*, there is a difference in how the NPs are analysed. As explained previously, we treat the quantifier as the head in the `DetQuant` function, as such these modifiers are attached to the quantifier using the `nummod` label. These can be addressed using non-local abstract rules for the `DetCN` function where the Det argument is an ordinal number.

Analyses of bootstrapped dependency trees

Table 2.9 shows the percentage of labelled edges in the bootstrapped treebanks. We report these numbers by calculating the fraction of edges in the treebank for the language labelled using the `dep` relation. The numbers in turn suggest the reduction in UD annotation efforts by bootstrapping using the GF RGL and wide-coverage abstract syntax. Note here that for these bootstrapping experiments, we only use the rules on abstract syntax (for all languages, English especially). One interesting result is that partial UD treebanks can be obtained even for languages with incomplete grammars. If the linearization rules for all the functions defined by the core RGL are missing in the concrete syntax for a specific language, we say that the RGL grammar for the specific language is incomplete. In Table 2.9, Amharic is one example of a language with an incomplete grammar implementation.

In the case of the UD treebank, it can be seen that on average across all languages, about 80% of the edges are labelled with the UD labels from the RGL bootstrapped treebanks. There is a decrease in this when we go to the wide coverage treebank. This is because of the semantic constructions introduced in the wide coverage grammars,

[9]More concrete syntax distinctions are needed in languages like Finnish, where VV can take its complement in at least five different infinitival forms, and Hindi, where modal verbs interfere with verbalizers.

Language	UD treebank		GF treebank
	RGL	Wide-coverage	
Afrikaans	81.57	-	81.73
Amharic	73.60	-	75.29
Bulgarian	76.60	80.53	88.13
Catalan	82.18	76.13	83.10
Chinese	84.89	77.33	82.46
Danish	80.49	-	87.45
Dutch	83.62	68.10	84.23
English	84.12	81.17	93.13
Estonian	82.38	79.10	86.52
Finnish	81.84	74.09	91.27
French	82.81	79.61	93.47
German	83.09	77.47	95.27
Greek	83.09	-	88.13
Hindi	72.63	69.34	84.18
Italian	81.79	77.52	90.47
Japanese	71.39	71.09	84.94
Latvian	72.11	-	89.10
Maltese	83.92	-	90.29
Mongolian	72.22	-	87.34
Nepali	82.91	-	83.19
Norwegian	80.37	-	86.36
Persian	83.87	-	84.40
Punjabi	72.37	-	82.82
Polish	83.05	-	87.38
Romanian	69.58	-	86.75
Russian	87.30	-	87.92
Sindhi	68.21	-	84.59
Spanish	81.41	79.15	91.28
Swedish	82.56	83.05	93.89
Thai	83.72	73.12	85.29
Urdu	72.86	-	84.67

Table 2.9: Percentage of completeness in the bootstrapped dependency treebanks. The Amharic grammar (underlined) is incomplete, i.e. does not implement all RGL functions.

necessary to achieve the abstractions required for interlingua-based MT. These constructions require concrete mappings in order to construct the dependency trees. In comparison, the GF treebank results in much better scores. The percentage of labelled edges is higher in the GF treebank because syncategorematic words have a much lesser incidence in the unit tests corresponding to functions defined in the GF-RGL.

From the collapsed trees in the bootstrapped treebanks, we primarily learned that the missing labels correspond to auxiliary verbs and copula verbs in most instances. The fact that these are frequent in all languages is clearly the reason for this. But this also suggests that defining a very small fragment of concrete non-local rules will give us huge improvements in connecting the bootstrapped treebanks with the correct UD labels.

A full investigation of the quality of these bootstrapped treebanks is pending due to the lack of manually annotated UD treebanks for this collection of languages.

2.6.3 GF Penn Treebank

Previous work on parsing in GF has resulted in a version of the Penn treebank mapped to the GF-RGL functions (Angelov and Ljunglöf, 2014). The GF-Penn treebank was created by mapping the annotation schema used in Penn treebanks with the abstract functions in the RGL using hand-crafted rules. This treebank is used to train statistical models for disambiguation in the parser and the wide-coverage translator. In cases where a fragment of the sentence is not covered by the grammar, the resulting AST is a tree, except that the missing functions in the grammar are replaced by missing nodes. Hence, the GF-Penn treebank is a GF annotation of the trees in the original Penn treebank where constructions outside the scope of the grammar are marked using a default function indicating incompleteness of the grammar. Nonetheless, the partially complete trees in the GF-Penn treebank are useful in estimating the probability distributions over the grammar.

We performed experiments with converting the GF treebank into the UD annotation scheme using our mapping. The entire treebank has about 5% of missing functions, on average each such function has two arguments. So, 10% of the edges on average are simply assigned the `dep` label because the algorithm can not determine the mapping corresponding to this function. We did not make any changes to the mappings or the conversion procedure to address the case of missing nodes.

The converted treebank is evaluated against a UD version of the Penn treebank, these UD annotations are obtained from the Stanford parser distribution (de Marneffe and Manning, 2008, de Marneffe et al., 2014). We evaluate our converted treebank against this UD treebank by computing labelled attachment scores used in evaluating dependency parser performance. This metric calculates the percentages of edges in the treebank that are attached to their correct head and assigned the correct dependency label. Table 2.10 shows the LAS scores on Sections 22, 23 and 24 of the GF-Penn treebank, against standard partitions of the Penn treebank that are used to report parser accuracies.

Analysis of the converted UD treebank showed three major reasons for failure to map the entire treebank to the UD labels.

i) About 5% of the edges in these sections are wrongly labelled due to divergence in our mapping from UD annotation scheme. Most of these correspond to the treatment of modal verbs (mentioned previously) and a small fraction of these

Section	LAS
WSJ-22	81.34
WSJ-23	83.21
WSJ-24	85.12

Table 2.10: LAS scores to compare our mappings to UDs against Penn UD treebank

edges correspond to numerical modifiers mapped as children of the quantifiers in our mapping.

ii) About 7% of the wrongly labelled edges correspond to proper nouns in the corpus. The UD annotation scheme selects the first token in a name to be the head and all the rest of the tokens in the name are labelled using the `name` label. Contrary to this, GF treats the entire name as a single token and does not assign edges between tokens in a proper noun.

iii) About 7% of the edges are wrong labelled due to the missing functions in the GF treebank. We mentioned previously we did not make any changes to the conversion algorithm or the mappings to address these cases.

2.7 Conclusion

In this paper, we investigate the similarities and differences between GF-RGL and Universal Dependencies (UDs). Our main result is a conversion from GF-RGL abstract syntax trees to UD dependency trees.

The conversion from abstract syntax trees in GF to UD trees is parameterized by a mapping defined on functions in the abstract syntax, to encode the notion of "head" and the grammatical relation of other modifiers with respect to the head. The mapping described in this work has has two levels – rules defined on the abstract syntax and rules defined on the concrete syntax. Rules defined on the abstract syntax are by definition language-independent and "shared". However, rules defined on concrete syntax are defined for each language separately, even if the UD labels for these may be shared across languages. Furthermore, both abstract and concrete rules have two types – "local" and "non-local". In the case of "local" rules, the mapping between function and its arguments and their UD annotations (both the identity of head and the UD labels) is determined using only the function, unlike "non-local" rules where a mapping applies only in certain contexts.

The mapping from GF-RGL to UDs showed that a large part of the GF-RGL and UD annotation scheme is language-independent. About 30 of the 40 UD labels are mapped using the abstract "local" and "non-local" rules in our experiments. These parts of both GF-RGL and UD scheme can be said to be language-independent. The parts of the mapping defined on the concrete syntax interestingly falls into two taxons within the UD annotation scheme (clausal dependents such as `aux`, `auxpass`, `cop`, `mark`, `expl` and compounding labels `compound`, `mwe`).

One application of the conversion detailed in this paper is to automatically create UD treebanks from GF treebanks. Our experiments using a small treebank showed that about 80% of UD treebank annotations can be ported using the abstract syntax in GF-RGL. The remaining annotations can also be derived from GF, provided a small

effort is put into defining concrete rules in the mapping for languages of interest. Other plausible applications and future directions of our work include multilingual consistency checking in the UD annotated treebanks and using these treebanks to estimate probabilistic ranking of GF abstract syntax trees.

Acknowledgements

We would like to thank Krasimir Angelov, Filip Ginter, Richard Johansson, Joakim Nivre and the two anonymous referees for their valuable comments and suggestions. Special thanks to Annie Zaenen for her technical comments in addition to all editorial help. The research was funded by the REMU project (Reliable Multilingual Digital Communication, Swedish Research Council 2012-5746).

2.8 Appendix: GF-RGL and UD Reference

This Appendix gives a listing of GF-RGL categories and UD tags and labels.

2.8.1 GF-RGL categories

Figure 2.18 shows the hierarchy of categories in the core RGL. The same picture appears in (Ranta, 2011), whereas Ranta (2009b) provides a systematic linguistic discussion of the RGL categories and functions. Table 2.11 lists the same categories with explanations, minimal linguistic examples and corresponding UD parts of speech.

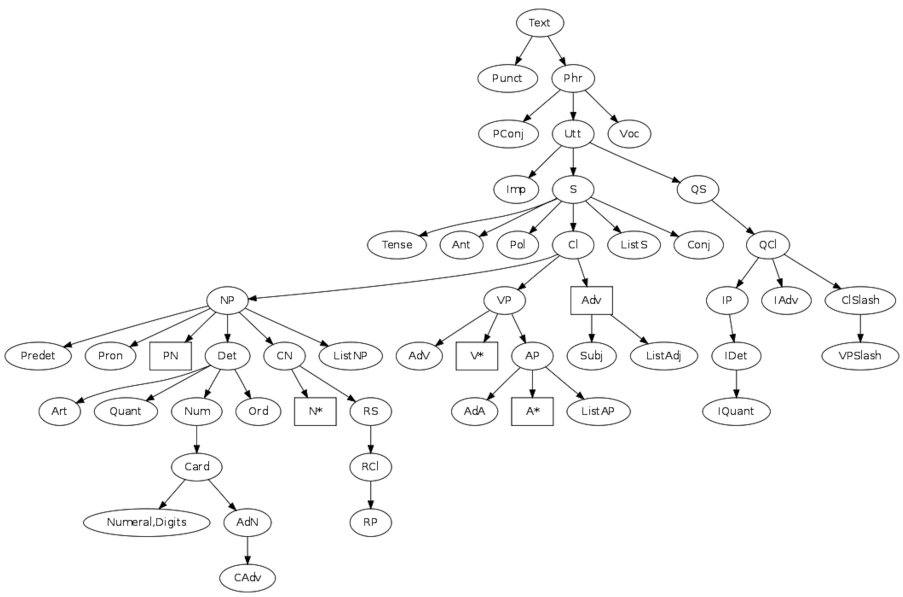


Figure 2.18: The categories of core RGL.

Category	Explanation	Example	UD POS
A	adjective	*old*	ADJ
AP	adjectival phrase	*very warm*	phrasal
Adv	adverb or adverbial phrase	*in the house*	ADV
Ant	anteriority	simultaneous, anterior	syncat
CN	common noun (without determiner)	*red house*	phrasal
Card	cardinal number	*seven*	NUM
Cl	declarative clause, with all tenses	*she looks at this*	phrasal
Comp	complement of copula, such as AP	*very warm*	phrasal
Conj	conjunction	*and*	CONJ
Det	determiner phrase	*those seven*	DET,phrasal
IAdv	interrogative adverb	*why*	ADV
IComp	interrogative complement of copula	*where*	phrasal
IDet	interrogative determiner	*how many*	DET,phrasal
IP	interrogative pronoun	*who*	PRON
Imp	imperative	*look at this*	phrasal
Interj	interjection	*alas*	INTJ
N	common noun	*house*	NOUN
NP	noun phrase (subject or object)	*the red house*	phrasal
Num	number determining element	*seven*	phrasal
Numeral	cardinal or ordinal in words	*five/fifth*	NUM
Ord	ordinal number (used in Det)	*seventh*	NUM
PConj	phrase-beginning conjunction	*therefore*	CONJ
PN	proper name	*Paris*	PROPN
Phr	phrase in a text	*but be quiet please*	phrasal
Pol	polarity	positive, negative	syncat
Predet	predeterminer (prefixed Quant)	*all*	DET
Prep	pre/postposition, or just case	*in*	ADP
Pron	personal pronoun	*she*	PRON
Punct	punctuation mark	*!*	PUNCT
QCl	question clause, with all tenses	*why does she walk*	phrasal
QS	question	*where did she live*	phrasal
Quant	quantifier ('nucleus' of Det)	*this/these*	DET
RCl	relative clause, with all tenses	*in which she lives*	phrasal
RP	relative pronoun	*in which*	PRON
RS	relative clause, tense fixed	*in which she lived*	phrasal
S	declarative sentence	*she lived here*	phrasal
SC	embedded sentence or question	*that it rains*	phrasal
Subj	subjunction	*if*	SCONJ
Temp	temporal and aspectual features	past anterior	syncat
Tense	tense	present, past, future	syncat
Text	text consisting of several phrases	*He is here. Why?*	phrasal
Utt	sentence, question, word...	*be quiet*	phrasal
V	one-place verb	*sleep*	VERB
V2	two-place verb	*love*	VERB
V2V	verb with NP and V complement	*cause*	VERB
V3	three-place verb	*show*	VERB
VA	adjective-complement verb	*look*	VERB
VP	verb phrase	*is very warm*	phrasal
VPSlash	verb phrase missing complement	*give to John*	phrasal
VQ	question-complement verb	*wonder*	VERB
VS	sentence-complement verb	*claim*	VERB
VV	verb-phrase-complement verb	*want*	VERB
Voc	vocative	*my darling*	phrasal

Table 2.11: Main RGL categories and corresponding UD POS tags. The tag "phrasal" means that the category has only complex phrases. "syncat" means that the category is linearized to abstract features, to which words are assigned syncategorematically.

2.8.2 UD tags and labels

Table 2.12 shows the hierarchy of core UD labels used to annotate treebanks in the UD annotation project. Table 2.13 shows the universal part-of-speech tags and the morphological features annotated in the UD treebanks. We recommend the UD Documentation [10] for readers interested in further details about the UD annotation efforts.

Core dependents of clausal predicates		
Nominal dep	*Predicate dep*	
nsubj	csubj	
nsubjpass	csubjpass	
dobj	ccomp	xcomp
iobj		
Non-core dependents of clausal predicates		
Nominal dep	*Predicate dep*	*Modifier word*
nmod	advcl	advmod
		neg
Special clausal dependents		
Nominal dep	*Auxiliary*	*Other*
vocative	aux	mark
discourse	auxpass	punct
expl	cop	
Noun dependents		
Nominal dep	*Predicate dep*	*Modifier word*
nummod	acl	amod
appos		det
nmod		neg
Case-marking, prepositions, possessive		
case		
Coordination		
conj	cc	punct
Compounding and unanalysed		
compound	mwe	goeswith
name	foreign	
Loose joining relations		
list	parataxis	remnant
dislocated		reparandum
Other		
Sentence head	*Unspecified dependency*	
root	dep	

Table 2.12: Dependency labels used in UD, organized as a taxonomy, as shown in Nivre et al. (2016)

[10] http://universaldependencies.org/docs/

Open class words		Closed class words		Other	
ADJ	adjective	ADP	preposition/postposition	PUNCT	punctuation
ADV	adverb	AUX	auxiliary	SYM	symbol
INTJ	interjection	CONJ	coordinating conjunction	X	unspecified POS
NOUN	noun	DET	determiner		
PROPN	proper noun	NUM	numeral		
VERB	verb	PART	particle		
		PRON	pronoun		
		SCONJ	subordinating conjunction		

Lexical	Inflectional	
	(Nominal)	(Verbal)
PronType	Gender	VerbForm
NumType	Animacy	Mood
Poss	Number	Tense
Reflex	Case	Aspect
	Definite	Voice
	Degree	Person
		Negative

Table 2.13: Top table shows the Part-of-speech tags in UD. Table below shows morphological features in UD. Both tables shown verbatim as-in Nivre et al. (2016)

2.9 Appendix: Dependency conversion algorithm and specification language

Figure 2.19 shows the BNF grammar of dependency configurations.

```
Rule     ::= Fun  Label+
           |  Fun  Relabels
           |  Tree Label+
           |  Tree Relabels
Relabels ::= Relabel ; Relabels
           |  Relabel
Relabel  ::= Label Part Label Label
Part     ::= "." Field
           |  "(" Words ")"
           |  "{*}"
Tree     ::= "(" Fun Arg* ")"
Arg      ::= "(" Tree ")"
           |  "?"
Fun      ::= Ident
Label    ::= Ident
Words    ::= QuotedString
           |  QuotedString "," Words
```

Figure 2.19: BNF specification of the dependency configuration syntax. Label+ refers to one or more labels using the syntax of regular expressions.

Shown in Algorithm 1 is the conversion algorithm used to obtain dependency trees from abstract syntax trees.

Data: Abstract syntax tree T, configuration on abstract syntax $C_{abstract}$ and concrete syntax $C_{concrete}$
Result: Dependency tree D
`# Decorate T with labels to get` T_L `and list of concrete configurations`
T_L, concreterelabels ← decorate(T)
`# Convert` T_L `to dependency tree D`
Dpartial ← getdependencies(T_L)
`# Apply concrete mappings to get complete dependency tree`
UD ← relabeledges(Dpartial, concreterelabels)

Function `decorate(`*tree*`)`
    decoratedtree ← tree
    concreterelabels ← Table() `# initialize an empty table`
    foreach *node in AST tree* **do**
        funcontext ← get subtree dominated by node
        funname ← get function name at node
        abstractconfigs ← $C_{abstract}$[funname]
        for *config in abstractconfigs* **do**
            context, labels ← unpack(config)
            if *match(context, funcontext)* **then**
                add labels to children of node in decoratedtree
                break loop
        concreteconfigs ← $C_{concrete}$[funname]
        for *config in concreteconfigs* **do**
            context, relabelops ← unpack(config)
            if *match(context, funcontext)* **then**
                store relabelops for node in table concreterelabels
                break loop
    return decoratedtree, concreterelabels

Function `getdependencies(`*tree*`)`
    D ← DependencyTree() `# initialize an empty dependency tree`
    foreach *word in Linearization(tree)* **do**
        find lowest node parent in tree spanning the word
    foreach *leaf in AST T* **do**
        find UD label by traversing unlabelled edges up the AST
        find head by traversing unlabelled edges from the dominating node
        store the word, parent, head function and UD label in D
    return D

Function `relabeledges(`*deptree, concreterelabels*`)`
    foreach *node in Table concretelabels* **do**
        relabelops ← concreterelabels[fun]
        foreach *relabelop in relabelops* **do**
            (label, words, newlabel, newhead) ← unpack(relabelop)
            for *word in children of node in deptree* **do**
                if *word matches words* **then**
                    assign new UD label newlabel and head newhead in deptree
    return deptree

Algorithm 1: Algorithm for converting ASTs to dependency trees

Data: Context for a node in AST nodecontext and context pattern in configuration configcontext
Result: True or False

remove branches from nodecontext that are not heads from node context
foreach *arg in nodecontext* **do**
    pat ← next(configcontext) `# pattern of next argument`
    if *argmatch(arg, pat)* **then**
        return False;
return True;

Function `argmatch`(*tree, treepattern*)
    if *treepattern is "?"* **then**
        return True `# match everything`
    funname, patargs = unpack(treepattern)
    topnode, args = unpack(tree)
    if *funname does not match name of topnode* **then**
        return False
    else
        if *patargs is Empty* **then**
            return True `# Local rule`
        foreach *argument in args and pat in patargs* **do**
            if *not argmatch(argument, pat)* **then**
                return False
        return True

Algorithm 2: Algorithm used to match contexts in configurations to AST

Chapter 3

Paper II: ud2gf

From Universal Dependencies to Abstract Syntax

Aarne Ranta and Prasanth Kolachina

Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017), pp. 107–116.

Abstract

Abstract syntax is a tectogrammatical tree representation, which can be shared between languages. It is used for programming languages in compilers, and has been adapted to natural languages in GF (Grammatical Framework). Recent work has shown how GF trees can be converted to UD trees, making it possible to generate parallel synthetic treebanks for those 30 languages that are currently covered by GF. This paper attempts to invert the mapping: take UD trees from standard treebanks and reconstruct GF trees from them. Such a conversion is potentially useful in bootstrapping treebanks by translation. It can also help GF-based interlingual translation by providing a robust, efficient front end. However, since UD trees are based on natural (as opposed to generated) data and built manually or by machine learning (as opposed to rules), the conversion is not trivial. This paper will present a basic algorithm, which is essentially based on inverting the GF to UD conversion. This method enables covering around 70% of nodes, and the rest can be covered by approximative back up strategies. Analysing the reasons of the incompleteness reveals structures missing in GF grammars, but also some problems in UD treebanks.

The abstract syntax defines a set of **categories**, such as CN (Common Noun) and AP (Adjectival Phrase), and a set of **functions**, such as ModCN (modification of CN with AP):

```
cat CN ; AP
fun ModCN : AP -> CN -> CN
```

A concrete syntax defines, for each category, a **linearization type**, and for each function, a **linearization function**; these can make use of **parameters**. For English, we need a parameter type Number (singular or plural). We define CN as a **table** (similar to an inflection table), which produces a string as a function Number (`Number=>Str`). As AP is not inflected, it is just a string. Adjectival modification places the AP before the CN, passing the number to the CN head of the construction:

```
param Number = Sg | Pl
lincat CN = Number => Str
lincat AP = Str
lin ModCN ap cn = \\n => ap ++ cn ! n
```

In French, we also need the parameter of gender. An AP depends on both gender and number. A CN has a table on Number like in English, but in addition, an inherent gender. The table and the gender are collected into a **record**. Adjectival modification places the AP after the CN, passing the inherent gender of the CN head to the AP, and the number to both constituents:

```
param Gender = Masc | Fem
lincat CN = {s : Number => Str ;
             g : Gender}
lincat AP = Gender => Number => Str
lin ModCN ap cn = {
  s = \\n => cn ! n ++
              ap ! cn.g ! n ;
  g = cn.g
  }
```

Context-free grammars correspond to a special case of GF where Str is the only linearization type. The use of tables (`P=>T`) and records (`{a : A ; b : B}`) makes GF more expressive than context-free grammars. The distinction between dependent and inherent features, as well as the restriction of tables to finite parameter types, makes GF less expressive than unification grammars. Formally, GF is equivalent to PMCFG (Parallel Multiple Context-Free Grammars) (Seki et al., 1991), as shown in (Ljunglöf, 2004), and has polynomial parsing complexity. The power of PMCFG has shown to be what is needed to share an abstract syntax across languages. In addition to morphological variation and agreement, it permits discontinuous constituents (used heavily e.g. in German) and reduplication (used e.g. in Chinese questions). The GF Resource Grammar Library uses a shared abstract syntax for currently 32 languages (Indo-European, Fenno-Ugric, Semitic and East Asian) written by over 50 contributors.

Software, grammars, and documentation are available in http://www.grammaticalframework.org

Figure 3.1: GF in a nutshell. The text works out a simple GF grammar of adjectival modification in English and French, showing how the structure can be shared despite differences in word order and agreement.

3.1 Introduction

GF (Grammatical Framework (Ranta, 2011)) is a formalism for multilingual grammars. Similarly to UD (Universal Dependencies, (Nivre et al., 2016)), GF uses shared syntactic descriptions for multiple languages. In GF, this is achieved by using **abstract syntax trees**, similar to the internal representations used in compilers and to Curry's tectogrammatical formulas (Curry, 1961). Given an abstract syntax tree, strings in different languages can be derived mechanically by **linearization functions** written for that language, similar to pretty-printing rules in compilers and to Curry's phenogrammatical rules. The linearization functions of GF are by design reversible to parsers, which convert strings to abstract syntax trees. Figure 3.1 gives a very brief summary of GF to readers unfamiliar with GF.

In UD, the shared descriptions are dependency labels and part of speech tags used in dependency trees. The words in the leaves of UD trees are language-specific, and languages can extend the core tagset and labels to annotate constructions in the

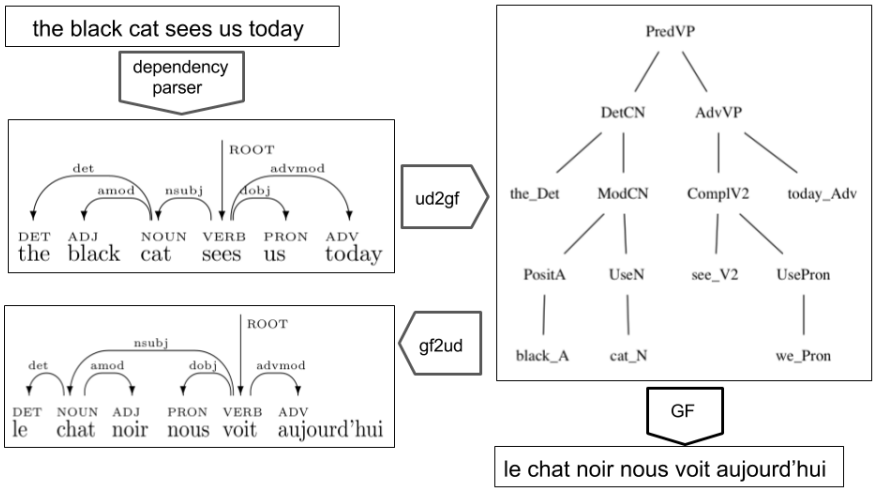


Figure 3.2: Conversions between UD trees, GF trees, and surface strings in English and French.

language. The relation between trees and strings is not defined by grammar rules, but by constructing a set of example trees—a treebank. From a treebank, a parser is typically constructed by machine learning (Nivre, 2006). There is no mechanical way to translate a UD tree from one language to other languages. But such a translation can be approximated in different ways to bootstrap treebanks (Tiedemann and Agic, 2016).

GF's linearization can convert abstract syntax trees to UD trees (Kolachina and Ranta, 2016). This conversion can be used for generating multilingual (and parallel) treebanks from a given set of GF trees. However, to reach the full potential of the GF-UD correspondence, it would also be useful to go to the opposite direction, to convert UD trees to GF trees. Then one could translate standard UD treebanks to new languages. One could also use dependency parsing as a robust front-end to a translator, which uses GF linearization as a grammaticality-preserving backend (Angelov et al., 2014), or to a logical form generator in the style of (Reddy et al., 2016), but where GF trees give an accurate intermediate representation in the style of (Ranta, 2004a). Figure 3.2 shows both of these scenarios, using the term **gf2ud** for the conversion of Kolachina and Ranta (2016) and **ud2gf** for the inverse procedure, which is the topic of this paper.

GF was originally designed for multilingual generation in controlled language scenarios, not for wide-coverage parsing. The GF Resource Grammar Library (Ranta, 2009b) thus does not cover everything in all languages, but just a "semantically complete subset", in the sense that it provides ways to express all kinds of content, but not necessarily all possible ways to express it. It has therefore been interesting to see how much of the syntax in UD treebanks is actually covered, to assess the completeness of the library. In the other direction, some of the difficulties in `ud2gf` mapping suggest that UD does not always annotate syntax in the most logical way, or in a way that is maximally general across languages.

The work reported in this paper is the current status of work in progress. Therefore the results are not conclusive: in particular, we expect to improve the missing coverage

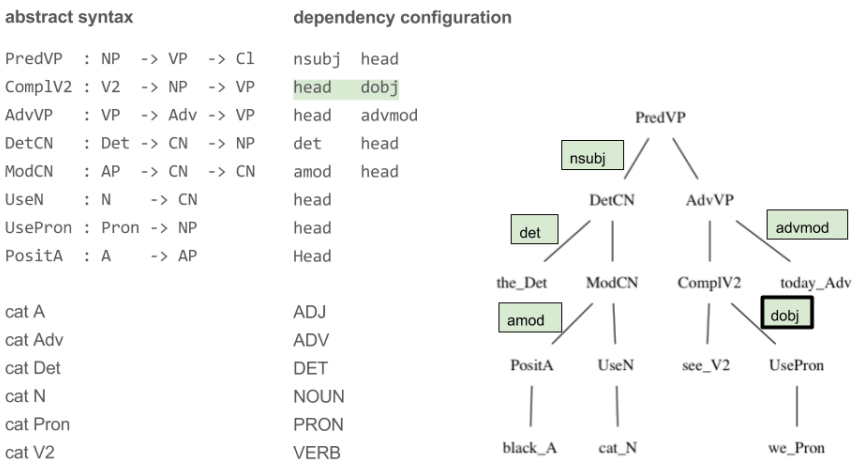


Figure 3.3: Annotating a GF tree with dependency labels. The label `dobj` results from the annotation of the `ComplV2` function. The category annotation (`cat`) are used in Figures 3.2 and 3.4 to map between GF categories and UD POS tags.

in a straightforward way. The most stable part of the work is the annotation algorithm described in Sections 3 an 4. It is based on a general notation for dependency configurations, which can be applied to any GF grammar and to any dependency annotation scheme—not only to the UD scheme. The code for the algorithm and the annotations used in experiments is available open source.[1]

The structure of the paper is as follows: Section 3.2 summarizes the existing `gf2ud` conversion and formulates the problem of inverting it. Section 3.3 describes a baseline bottom-up algorithm for translation from UD trees to GF trees. Section 3.4 presents some refinements to the basic algorithm. Section 3.5 shows a preliminary evaluation with UD treebanks for English, Finnish, and Swedish. Section 3.6 concludes.

3.2 From gf2ud to ud2gf

The relation between UD and GF is defined declaratively by a set of **dependency configurations**. These configurations specify the dependency labels that attach to each subtree in a GF tree. Figure 3.3 shows an abstract syntax specification together with a dependency configuration, as well as a GF tree with corresponding labels attached.

Consider, for example, the second line of the “abstract syntax" part of Figure 3.3, with the symbol `ComplV2`. This symbol is one of the **functions** that are used for building the abstract syntax tree. Such a function takes a number of trees (zero or more) as arguments and combines them to a larger tree. Thus `ComplV2` takes a V2 tree (two-place verb) and an NP tree (noun phrase) to construct a VP tree (verb phrase). Its name hints that it performs complementation, i.e. combines verbs with their complements. Its dependency configuration `head dobj` specifies that the first argument (the verb) will contain the label `head` in UD, whereas the second argument (the noun phrase) will contain the label `dobj` (direct object). When the configuration

[1] https://github.com/GrammaticalFramework/gf-contrib/tree/master/ud2gf

is applied to a tree, the `head` labels are omitted, since they are the default. Notice that the order of arguments in an abstract syntax tree is independent of the order of words in its linearizations. Thus, in Figure 3.2, the object is placed after the verb in English but before the verb in French.

The algorithm for deriving the UD tree from the annotated GF tree is simple:

- for each leaf X (which corresponds to a lexical item)
  - follow the path up towards the root until you encounter a label L
  - from the node immediately above L, follow the **spine** (the unlabelled branches) down to another leaf Y
  - Y is the head of X with label L

It is easy to verify that the UD trees in Figure 3.2 can be obtained in this way, together with the English and French linearization rules that produce the surface words and the word order. In addition to the configurations of functions, we need **category configurations**, which map GF types to UD part of speech (POS) tags.

This algorithm covers what Kolachina and Ranta (2016) call **local abstract configurations**. They are sufficient for most cases of the `gf2ud` conversion, and have the virtue of being compositional and exactly the same for all languages. However, since the syntactic analysis of GF and UD are not exactly the same, and within UD can moreover differ between languages, some **non-local** and **concrete** configurations are needed in addition. We will return to these after showing how the local abstract configurations are used in `ud2gf`.

The path from GF trees to UD trees (**gf2ud**) is deterministic: it is just linearization to an annotated string representing a dependency tree. It defines a relation between GF trees and UD trees: *GF tree t produces UD tree u*. Since the mapping involves loss of information, it is many-to-one. The opposite direction, `ud2gf`, is a nondeterministic search problem: *given a UD tree u, find all GF trees t that can produce u*. The first problem we have to solve is thus

> **Ambiguity**: a UD tree can correspond to many GF trees.

More problems are caused by the fact that GF trees are formally generated by a grammar whereas UD trees have no grammar. Thus a UD tree may lack a corresponding GF tree for many different reasons:

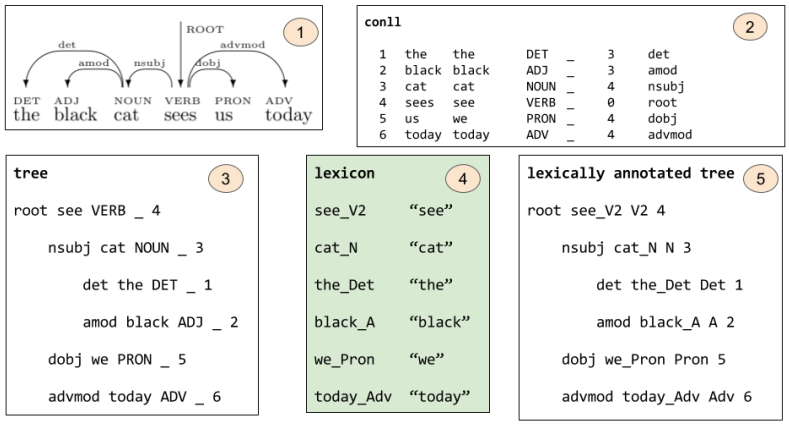
> **Incompleteness**: the GF grammar is incomplete.
> **Noise**: the UD tree has annotation errors.
> **Ungrammaticality**: the original sentence has grammar errors.

Coping with these problems requires **robustness** of the `ud2gf` conversion. The situation is similar to the problems encountered when GF is used for wide-coverage parsing and translation (Angelov et al., 2014). The solution is also similar, as it combines a declarative rule-based approach with disambiguation and a back-up strategy.

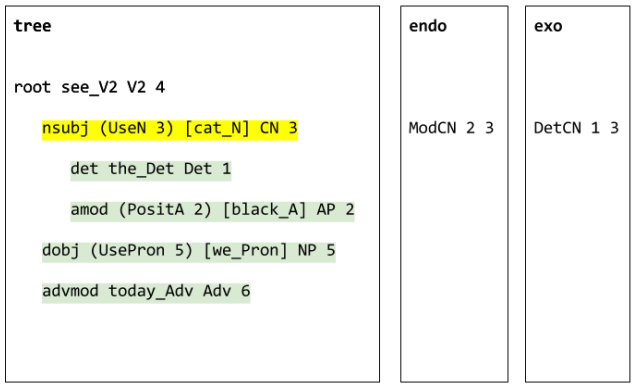
3.3 The ud2gf basic algorithm

The basic algorithm is illustrated in Figure 3.4
Its main data-structure is an **annotated dependency tree**, where each node has the form

Restructuring and lexical annotation



A node annotation by endo- and exocentric functions



The final annotated tree

```
root (PredVP 3 4) [(AdvVP 4 6),(ComplV2 4 5),see_V2] VP 4
   nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

Figure 3.4: Steps in ud2gf

$< L,t,ts,C,p >$ where

- L is a dependency label (always the same as in the original UD tree)
- t is the current GF abstract syntax tree (iteratively changed by the algorithm)
- ts is a list of alternative GF abstract syntax trees (iteratively changed by the algorithm)
- C is the GF category of t (iteratively changed by the algorithm)
- p is the position of the original word in the UD tree (always the same as in the original UD tree)

Examples of such nodes are shown in Figure 3.4, in the tree marked (5) and in all trees below it.

The algorithm works in the following steps, with references to Figure 3.4:

1. **Restructuring**. Convert the CoNLL graph (marked (2) in Figure 3.4) to a tree data-structure (3), where each node is labelled by a dependency label, lemma, POS tag, and word position. This step is simple and completely deterministic, provided that the graph is a well-formed tree; if it isn't, the conversion fails[2].

2. **Lexical annotation**. Preserve the tree structure in (3) but change the structure of nodes to the one described above and shown in (5). This is done by using a GF lexicon (4), and a category configuration, replacing each lemma with a GF abstract syntax function and its POS with a GF category.[3]

3. **Syntactic annotation**. The GF trees t in the initial tree (5) are lexical (0-argument) functions. The syntactic annotation step annotates the tree recursively with applications of syntactic combination functions. Some of them may be **endofunctions** (i.e. **endocentric** functions), in the sense that some of the argument types is the same as the value type. In Figure 3.3, the functions `AdvVP` and `ModCN` are endocentric. All other functions are **exofunctions** (i.e. **exocentric** functions), where none of the argument types is the same as the value type. In the syntactic annotation, it is important to apply endofunctions before exofunctions, because exofunctions could otherwise block later applications of endofunctions.[4] The algorithm is a depth-first postorder traversal: for an annotated tree $T = (N\,T_1 \ldots T_n)$, where $N =< L,t,ts,C,p >$,

- syntax-annotate the subtrees $T_1, \ldots, T_n$
- apply available combination functions to N:
  - if an endofunction $f : C \to C$ applies, replace $< t,ts >$ with $< ((ft), \{t\} \cup ts >$
  - else, if an exofunction $f : C \to C'$ applies, replace $< t,ts,C >$ with $< (ft), \{t\} \cup ts, C' >$

where a function $f : A \to B$ **applies** if $f = (\lambda x)(g \ldots x \ldots)$ where g is an endo- or exocentric function on C and all other argument places than x are filled with GF trees from the subtrees of T. Every subtree can be used at most once.

[2] This has never happened with the standard UD treebanks that we have worked with.

[3] The GF lexicon is obtained from the GF grammar by linearizing each lexical item (i.e. zero-place function) to the form that is used as the lemma in the UD treebank for the language in question.

[4] This is a simplifying assumption: a chain of two or more exofunctions could in theory bring us back to the same category as we started with.

An example of syntactic annotation is shown in the middle part of Figure 3.4. The node for the word *cat* at position 3 (the second line in the tree) has one applicable endofunction, `ModCN` (adjectival modification), and one exofunction, `DetCN` (determination). Hence the application of the endofunction `ModCN` combines the AP in position 2 with the CN in position 3. For brevity, the subtrees that the functions can apply to are marked by the position numbers. [5] Hence the tree

```
DetCN 1 3
```

in the final annotated tree actually expands to

```
DetCN the_Det
  (ModCN (PositA black_A) (UseN cat_N))
```

by following these links. The whole GF tree at the root node expands to the tree shown in Figures 3.2 and 3.3.

3.4 Refinements of the basic algorithm

We noted in Section 2 that `ud2gf` has to deal with ambiguity, incompleteness, noise, and ungrammaticality. The basic algorithm of Section 3 takes none of these aspects into account. But it does contain what is needed for ambiguity: the list ts of previous trees at each node can also be used more generally for storing alternative trees. The "main" tree t is then compared and ranked together with these candidates. Ranking based on tree probabilities in previous GF treebanks, as in (Angelov, 2011), is readily available. But an even more important criterion is the **node coverage** of the tree. This means penalizing heavily those trees that don't cover all nodes in the subtrees.

This leads us to the problem of incompleteness: what happens if the application of all possible candidate functions and trees still does not lead to a tree covering all nodes? An important part of this problem is due to **syncategorematic words**. For instance, the copula in GF is usually introduced as a part of the linearization, and does not have a category or function of its own.[6] To take the simplest possible example, consider the adjectival predication function and its linearization:

```
fun UseAP : AP -> VP
lin UseAP ap = \\agr => be agr ++ ap
```

where the agreement feature of the verb phrase is passed to an auxiliary function `be`, which produces the correct form of the copula when the subject is added. The sentence *the cat is black* has the following tree obtained from UD:

```
root (PredVP 2 4) [UseAP...black_A] S 4
  nsubj (DetCN 1 2) [UseN 2,cat_N] 2
    det the_Det Det 1
  cop "be" String 3 ***
```

[5] If the argument has the same node as the head (like 3 here), the position refers to the next-newest item on the list of trees.

[6] This is in (Kolachina and Ranta, 2016) motivated by cross-lingual considerations: there are languages that don't need copulas. In (Croft et al., 2017), the copula is defined as a **strategy**, which can be language-dependent, in contrast to **constructions**, which are language-independent. This distinction seems to correspond closely to concrete vs. abstract syntax in GF.

The resulting GF tree is correct, but it does not cover node 3 containing the copula.[7] The problem is the same in `gf2ud` (Kolachina and Ranta, 2016), which introduces language-specific concrete annotations to endow syncategorematic words with UD labels. Thus the concrete annotation

```
UseAP head {"is","are","am"} cop head
```

specifies that the words *is,are,am* occurring in a tree linearized from a `UseAP` application have the label `cop` attached to the head.

In `ud2gf`, the treatment of the copula turned out to be simpler than in `gf2ud`. What we need is to postulate an abstract syntax category of copulas and a function that uses the copula. This function has the following type and configuration:

```
UseAP_ : Cop_ -> AP -> VP ; cop head
```

It is used in the basic algorithm in the same way as ordinary functions, but eliminated from the final tree by an explicit definition:

```
UseAP_ cop ap = UseAP ap
```

The copula is captured from the UD tree by applying a category configuration that has a condition about the lemma:[8]

```
Cop_ VERB lemma=be
```

This configuration is used at the lexical annotation phase, so that the last line of the tree for *the cat is black* becomes

```
cop be Cop_ 3
```

Hence the final tree built for the sentence is

```
PredVP (DetCN the_Det (UseN cat_N))
  (UseAP_ be (PositA black_A))
```

which covers the entire UD tree. By applying the explicit definition of `UseAP_`, we obtain the standard GF tree

```
PredVP (DetCN the_Det (UseN cat_N))
  (UseAP (PositA black_A))
```

Many other syncategorematic words—such as negations, tense auxiliaries, infinitive marks—can be treated in a similar way. The eliminated constants are called **helper functions** and **helper categories**, and for clarity suffixed with underscores.

Another type of incomplete coverage is due to missing functions in the grammar, annotation errors, and actual grammar errors in the source text. To deal with these, we have introduced another type of extra functions: **backup functions**. These functions collect the uncovered nodes (marked with `***`) and attach them to their heads as adverbial modifiers. The nodes collected as backups are marked with single asterisks (`*`). In the evaluation statistics, they are counted as **uninterpreted nodes**, meaning that they are not covered with the standard GF grammar. But we have added linearization

[7] We use `***` to mark uncovered nodes; since *be* has no corresponding item in the GF lexicon, its only possible categorization is as a `String` literal.

[8] The simplicity is due to the fact that the trees in the treebank are lemmatized, which means that we need not match with all forms of the copula.

```
I have a change in plans next week .

root have_V2 : V2 2                 I have a change in plans "."
 nsubj i_Pron : Pron 1              [ next week ]
 dobj change_N : N 4
  det IndefArt : Quant 3            minulla on muutos suunnitelmissa "."
  nmod plan_N : N 6                 [ seuraava viikko ]
   case in_Prep : Prep 5
 nmod:tmod Backup week_N : N 8 *    jag har en ändring i planer "."
  amod next_A : A 7 *               [ nästa vecka ]
 punct "." : String 9
```

Figure 3.5: A tree from the UD English training treebank with lexical annotations and backups marked, and the resulting linearizations to English, Finnish, and Swedish.

rules to them, so that they are for instance reproduced in translations. Figure 3.5 gives an example of a UD tree thus annotated, and the corresponding translations to Finnish and Swedish, as well as back to English. What has happened is that the temporal modifier formed from the bare noun phrase *next week* and labelled `nmod:tmod` has not found a matching rule in the configurations. The translations of the resulting backup string are shown in brackets.

3.5 First results

The `ud2gf` algorithm and annotations are tested using the UD treebanks (v1.4)[9]. The training section of the treebank was used to develop the annotations and the results are reported on the test section. We evaluated the performance in terms of coverage and interpretability of the GF trees derived from the translation. The coverage figures show the percentage of dependency nodes (or tokens) covered, and interpreted nodes show the percentage nodes covered in "normal" categories, that is, other than the Backup category. The percentage of interpreted nodes is calculated as the number of nodes in the tree that use a Backup function to cover all its children. Additionally, the GF trees can be translated back into strings using the concrete grammar, allowing for qualitative evaluation of the translations to the original and other languages.[10]

We performed experiments for three languages: English, Swedish and Finnish. Table 3.1 show the scores for the experiments using the gold UD trees. Also shown are the number of trees (i.e. sentences) in the test set for each language. The results show an incomplete coverage, as nodes are not yet completely covered by the available Backup functions. As a second thing, we see the impact of language-specific configurations (mostly defining helper categories for syncategorematic words) on the interpretability of GF trees. For example, in Swedish, just a small number of such categories (26) increases the coverage significantly. Further experiments also showed an average increase of 4-6% points in interpretability scores when out-of-vocabulary words were handled using additional functions based on the part-of-speech tags; in other words,

[9] https://github.com/UniversalDependencies/,retrievedinOctober2016

[10] A quantitative evaluation would also be possible by standard machine translation metrics, but has not been done yet.

language	#trees	#confs	%cov'd	%int'd
English	2077	31	94	72
Finnish	648	12	92	61
Finnish*	648	0	74	55
Swedish	1219	26	91	65
Swedish*	1219	0	75	57

Table 3.1: Coverage of nodes in each test set (`L-ud-test.conllu`). L* (Swedish*, Finnish*) is with language-independent configurations only. #conf's is the number of language-specific configurations. %cov'd and %int'd are the percentages of covered and interpreted nodes, respectively.

rule type	number
GF function (given)	346
GF category (given)	109
backup function	16
function config	128
category config	33
helper function	250
helper category*	26

Table 3.2: Estimating the size of the project: GF abstract syntax (as given in the resource grammar library) and its abstract and concrete configurations. Helper category definitions are the only genuinely language-dependent configurations, as they refer to lemmas.

more than 10% of uninterpreted nodes contained words not included in the available GF lexica.

Table 3.2 shows how much work was needed in the configurations. It shows the number of GF functions (excluding the lexical ones) and language-independent configurations. It reveals that there are many GF functions that are not yet reached by configurations, and which would be likely to increase the interpreted nodes. The helper categories in Table 3.2, such as Copula, typically refer to lemmas. These categories, even though they can be used in language-independent helper rules, become actually usable only if the language-specific configuration gives ways to construct them.

A high number of helper functions were needed to construct tensed verb phrases (VPS) covering all combinations of auxiliary verbs and negations in the three languages. This is not surprising given the different ways in which tenses are realized across languages. The extent to which these helper functions can be shared across languages depends on where the information is annotated in the UD tree and how uniform the annotations are; in English, Swedish, and Finnish, the compound tense systems are similar to each other, whereas negation mechanisms are quite different.

Modal verbs outside the tense system were another major issue in `gf2ud` (Kolachina and Ranta, 2016), but this issue has an easier solution in `ud2gf`. In GF resource grammars, modal verbs are a special case of VP-complement verbs (VV), which also contains non-modal verbs. The complementation function `ComplVV` hence needs two configurations:

```
ComplVV : VV -> VP -> VP ;  head xcomp
```

property	**UD**	**GF**
parser coverage	robust	brittle
parser speed	fast	slow
disambiguation	cont.-sensitive	context-free
semantics	loose	compositional
generation	?	accurate
new language	low-level work	high-level work

Table 3.3: Complementary strengths and weaknesses of GF and UD. UD strengths above the dividing line, GF strengths below.

```
ComplVV : VV -> VP -> VP ;  aux  head
```

The first configuration is valid for the cases where the VP complement is marked using the `xcomp` label (e.g. *want to sleep*). The second one covers the cases where the VP complement is treated as the head and the VV is labelled `aux` (e.g. *must sleep*). The choice of which verbs are modal is language-specific. For example, the verb *want* marked as VV in GF is non-nodal in English but translated in Swedish as an auxiliary verb *vilja*. In `gf2ud`, modal verbs need non-local configurations, but in `ud2gf`, we handle them simply by using alternative configurations as shown above.

Another discrepancy across languages was found in the treatment of progressive verb phrases (e.g. *be reading*, Finnish *olla lukemassa*). in English the verb *be* is annotated as a child of the content verb with the `aux` label. In Finnish, however the equivalent verb *olla* is marked as the head and the content verb as the child with the `xcomp` label. This is a case of where the content word is not chosen to be the head, but the choice is more syntax-driven.

3.6 Conclusion

The main rationale of relating UD with GF is their complementary strengths. Generally speaking, UD strengths lie in parsing and GF strengths in generation. UD pipelines are robust and fast at analyzing large texts. GF on the other hand, allows for accurate generation in multiple languages apart from compositional semantics. This suggests pipelines where UD feeds GF.

In this paper, we have done preparatory work for such a pipeline. Most of the work can be done on a language-independent level of abstract syntax configurations. This brings us currently to around 70–75 % coverage of nodes, which applies automatically to new languages. A handful of language-specific configurations (mostly for syncategorematic words) increases the coverage to 90–95%. The configuration notation is generic and can be applied to any GF grammar and dependency scheme.

Future work includes testing the pipeline in applications such as machine translation, abstractive summarization, logical form extraction, and treebank bootstrapping. A more theoretical line of work includes assessing the universality of current UD praxis following the ideas of Croft et al. (2017). In particular, their distinction between **constructions** and **strategies** seems to correspond to what we have implemented with shared vs. language-specific configurations, respectively.

Situations where a shared rule would be possible but the treebanks diverge, such as the treatment of VP-complement verbs and progressives (Section 5), would deserve

closer inspection. Also an analysis of UD Version 2, which became available in the course of the project, would be in place, with the expectation that the differences between languages decrease.

Acknowledgements

We want to thank Joakim Nivre and the anonymous referees for helpful comments on the work. The project has been funded by the REMU project (Reliable Multilingual Digital Communication, Swedish Research Council 2012-5746).

Chapter 4

Paper III: Bootstrapping UD treebanks

Bootstrapping UD treebanks for Delexicalized Parsing

Prasanth Kolachina and Aarne Ranta

Under submission.

Abstract

Standard approaches to treebanking traditionally employ a waterfall model (Sommerville, 2010), where annotation guidelines guide the annotation process and insights from the annotation process in turn lead to subsequent changes in the annotation guidelines. This process remains a very expensive step in creating linguistic resources for a target language, necessitates both linguistic expertise and manual effort to develop the annotations and is subject to inconsistencies in the annotation due to human errors. In this paper, we propose an alternative approach to treebanking—one that requires writing grammars. This approach is motivated specifically in the context of Universal Dependencies, an effort to develop uniform and cross-lingually consistent treebanks across multiple languages.

We show here that a bootstrapping approach to treebanking via interlingual grammars is plausible and useful in a process where grammar engineering and treebanking are jointly pursued when creating resources for the target language. We demonstrate the usefulness of synthetic treebanks in the task of delexicalized parsing. Our experiments reveal that simple models for treebank generation are *cheaper* than human annotated treebanks, especially in the lower ends of the learning curves for delexicalized parsing, which is relevant in particular in the context of low-resource languages.

4.1 Introduction

Treebanking remains a vital step in the process of creating linguistic resources for a language – a practice that was established in the last 2-3 decades (Marcus et al., 1994). The process of treebanking involves training human annotators in order to obtain high-quality annotations. This is a human-intensive and costly process where multiple iterations are performed to refine the quality of the linguistic resource. Grammar engineering is a complementary approach to creating linguistic resources: one that requires a different kind of expertise and incurs comparable costs. These two approaches have remained orthogonal for obvious reasons: treebanks are primarily necessary to induce abstractions in NLU (Natural Language Understanding) models from data, while grammars are themselves abstractions arising from linguistic knowledge. Abstractions induced from data have proven themselves to be useful for robust NLU tasks, while grammars are better at precision tasks involving NLG (Natural Language Generation).

Given the resources required for treebanking, synthetic treebanks have been proposed and used as substitute in cross-lingual parsing for languages where treebanks do not exist. Such treebanks are created using parallel corpora where parse trees in one language are bootstrapped into a target language using alignment information through annotation projection (McDonald et al., 2011, Tiedemann, 2014) or using machine translation systems to bootstrap existing treebanks in one or more source language(s) to the target language (Tiedemann and Agic, 2016, Tyers et al., 2018). More recently, synthetic treebanks are generated for both real and artificial languages using multilingual treebanks by learning feasible parameter combinations (Wang and Eisner, 2016) – Wang and Eisner (2018) show that such treebanks can be useful to select the most similar language to train a parsing model for an unknown language.

At the same time, grammar-based treebanking approaches have been shown to work in monolingual setups—to derive rich linguistic representations defined by explicit grammars (Oepen et al., 2004). These approaches are carried out by parsing raw corpora with a target grammar and using an additional human disambiguation phase. Alternatively, existing treebanks are matched against the target grammar further reducing the human effort in disambiguation: these approaches face a challenge of under-specification in the source treebanks (Angelov, 2011). In the current paper, we propose a hybrid of these two methods: we use abstract syntax grammars as core linguistic abstraction to generate synthetic treebanks for a grammar that can be translated to target representations with high precision.

The question of annotation costs and ways to minimize the dependence on such annotated corpora has been a recurring theme in the field for the last two decades (Ngai and Yarowsky, 2000, Garrette and Baldridge, 2013). This question has also been extensively addressed in the context of dependency treebanks. We revisit this question in context of Universal Dependencies and recent work on the interplay between interlingua grammars and multilingual dependency trees in this scheme (Kolachina and Ranta, 2016, Ranta and Kolachina, 2017, Ranta et al., 2017). The use of interlingua grammars to bootstrap dependency treebanks guarantees two types of consistencies: multilingual treebank consistency and intra-treebank consistency. We study the efficacy of these dependency treebanks using learning curves of a transition-based parser in a *delexicalized parsing* setup. The delexicalized parsing setup allows for generation of parallel UD treebanks in multiple languages with minimal pre-requisites on language-specific knowledge.

Another rationale behind the the current work in the context of cross-lingual

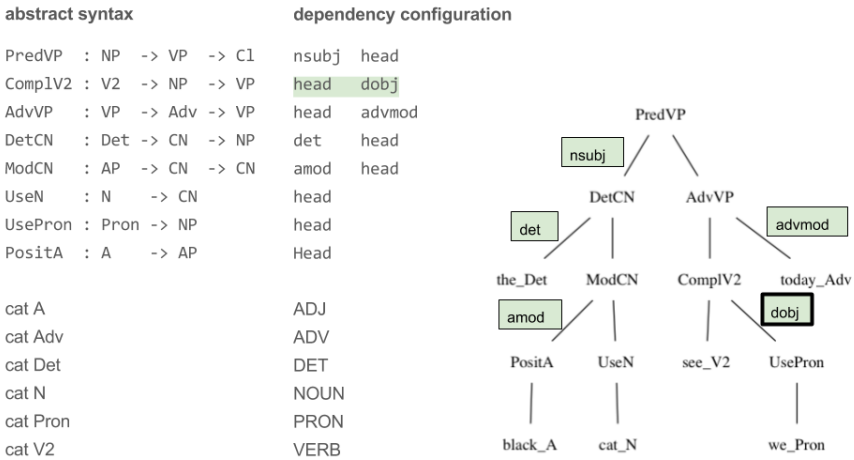


Figure 4.1: Abstract syntax of a GF grammar and its specification for UD scheme. Also shown is an example AST for the sentence *the black cat sees us today*. Any function with a definition written as $f : C_1 \rightarrow C_2 \rightarrow ...C_n \rightarrow C$; can be rewritten as a context-free rule $f.\ C ::= C_1C_2...C_n$.

parsing is while synthetic treebanks offer a "cheap" alternative, the signal for the target language is limited by the quality of the MT system. On the other hand, interlingua grammars provide a high-quality signal about the target language. High-quality using interlingual grammars refers to accurate generation of word-order and morphology – although lexical selection in translation is still a problem. There have not been previous attempts in cross-lingual parsing to our knowledge studying the effect of these.

This paper is structured as follows- Section 4.2 gives the relevant background on interlingua grammars and the algorithm used to generate UD trees given treebank derived from an interlingua grammar. Section 4.3 describes our algorithm to bootstrap treebanks for a given interlingua grammar and parallel UD treebanks from them along with an intrinsic evaluation of these bootstrapped UD treebanks. Section 4.4 shows the parsing setup we use and Section 4.5 details the results of the parsing experiments.

4.2 Grammatical Framework

Grammatical Framework (GF) is a multilingual grammar formalism using abstract syntax trees (ASTs) as primary descriptions (Ranta, 2011). Originating in compilers, AST is a tectogrammatical tree representation that can be shared between languages. A GF grammar consists of two parts – an abstract syntax shared between languages and concrete syntax that is defined for each language. The abstract syntax defines a set of categories and a set of functions, as shown in Figure 4.1. The functions defined in the abstract syntax specify the result of putting subparts of two categories together and the concrete syntax specifies how the subparts are combined i.e. word-order preferences and agreement constraints specific to the language.

A comprehensive implementation of a multilingual grammar in GF is the **Resource Grammar Library**, GF-RGL (Ranta, 2009b), which currently has concrete syntaxes

for over 40 languages, ranging from Indo-European through Finno-Ugric and Semitic to East Asian languages. [1] This implementation contains a full implementation of the morphology of the language, and a set of 284 syntactic constructors that correspond to the core syntax of the language. Also included is a small lexicon of 500 lexical concepts from a set of 19 categories, of which 10 correspond to different sub-categorization frames of verbs, 2 classes of nouns and adjectives. These grammars are *reversible*- i.e. they can be used for parsing and simultaneous multilingual generation into multiple languages. The concrete syntaxes for all the languages define the rules for these syntactic constructors and the lexical concepts. The expressivity of these grammars is equivalent to a PMCFG (Seki et al., 1991), which makes parsing complexity of this formalism polynomial in sentence length. Polynomial parsing with high exponents can still be too slow for many tasks, and it is also brittle if the grammars are designed to not over-generate. But generation using GF grammars has been shown to be both precise and fast, which suggests the idea of combining data-driven parsing with grammar-driven generation. We refer the interested reader to Ljunglöf (2004) for discussion on expressivity of this formalism and Angelov (2011), Angelov and Ljunglöf (2014) for discussion on probabilistic parsing using GF grammars.

4.2.1 gf2ud

Kolachina and Ranta (2016) propose an algorithm to translate ASTs to dependency trees, that takes a specification of the abstract syntax of the GF grammar (referred to as *configurations*, see Figure 4.1) which describes the mapping between the grammar and a target dependency scheme, in this case Universal Dependencies. These configurations can be interpreted as a synchronous grammar over the abstract syntax as source and dependency scheme as target.

The first step in this transducer is a recursive annotation that marks for each function in the AST, one of the arguments as `head` and specifies labels for the other arguments, as specified by the configuration. The algorithm to extract the resulting dependency tree from the *annotated AST* is simple.

- for each leaf X (which corresponds to a lexical item) in the AST
  - trace the path up towards the root until you encounter a label L
  - from the node immediately above L, follow the **spine** (the unlabeled branches) down to another leaf Y
  - Y is the head of X with label L

At the end of these two steps, the resulting data-structure is an *abstract dependency tree* (ADT shown in Figure 4.2). It should be noted that the order of nodes shown in the ADT does not reflect the surface order that is specific to a language. The ADT combined with the concrete syntax of a language and concrete configurations (when necessary) results in the corresponding full UD tree. The concrete configurations are necessary to provide appropriate labels to syncategorematic words like auxiliary verbs and negation particles. Additionally, the category configuration on the abstract syntax can be augmented with a language-specific category configurations to generate the morphological features in the dependency tree with a desired tag set.

Kolachina and Ranta (2016) show that their method can be used to generate partially labeled UD trees for 30 languages when the corresponding concrete syntax is available. They also show that using configurations defined on abstract syntax alone

[1] The current status of GF-RGL can be seen in `http://www.grammaticalframework.org/lib/doc/synopsis.html` which also gives access to the source code.

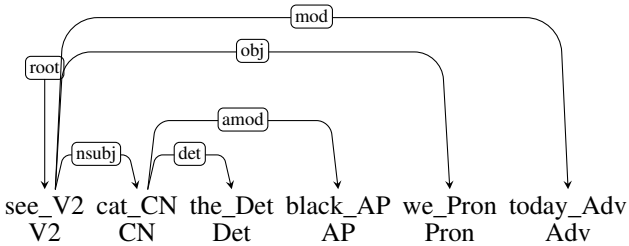


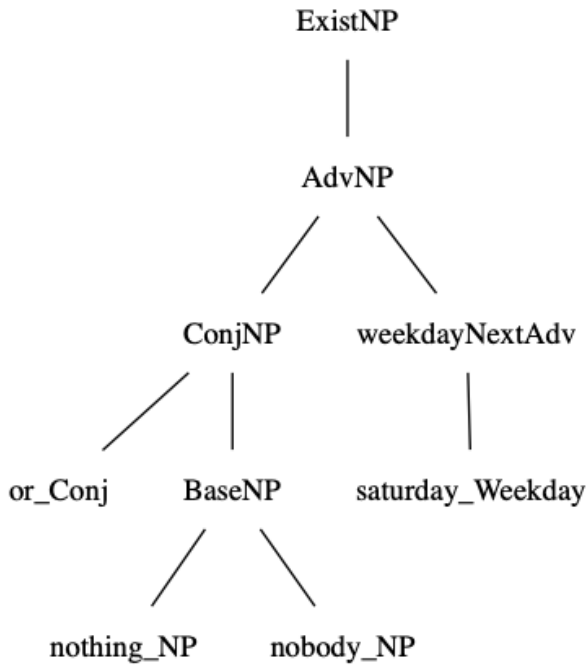
Figure 4.2: ADT for the sentence *the black cat sees us today*. The nodes in the ADT correspond to lexical functions defined in the grammar. Also shown is the UD part-of-speech tag sequence. Note that the order of nodes does not reflect the surface order in any particular language.

and depending on the availability of the concrete syntax, a large fraction (around 75–85% of edges) of the dependency treebanks can be generated automatically. This is done with small treebanks of ASTs – a UD treebank of 104 ASTs and a GF treebank of 400 ASTs. Their results show that parallel UD treebanks can be bootstrapped using ASTs and interlingua grammars, the usefulness of such treebanks however is not addressed in that work. Full UD treebanks can be generated when concrete configurations (those addressing syncategorematic words) are additionally available for the language.

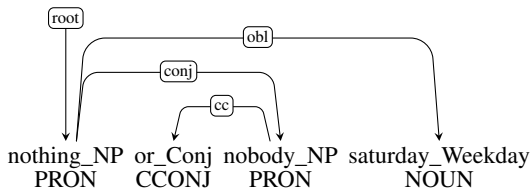
4.3 Bootstrapping AST and UD treebanks

The abstract syntax component of a GF grammar is an algebraic datatype definition, which can also be seen as a context-free grammar (CFG). The disambiguation model defined in GF uses a context-free probability distribution defined on the abstract syntax. The advantage of defining the distribution on the abstract syntax is it allows for transfer of distribution to languages for which data GF treebanks do not exist. The context-free distribution decomposes the probability estimate of a tree as the product of probabilities of the sub-trees and the probability of the function applied to these subtrees. The probabilistic abstract syntax grammar can therefore be defined in terms analogous to a probabilistic CFG (PCFG). The probability distribution over the set of categories in the grammar is also included in the distribution corresponding to the abstract syntax.

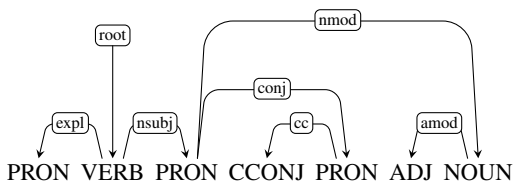
We use this formulation as a starting point and generate ASTs for a given grammar. The ASTs bootstrapped using the probability model defined above are correct in terms of the grammar but do not follow the selectional preferences typically found in language. For this reason, we refer to the bootstrapped treebanks as "synthetic" data. Additionally, while the algorithm used to bootstrap ASTs does not change depending on whether the grammar includes a lexicon or not, it is significantly faster depending the size of the grammar. Stacking `gf2ud` defined using abstract configurations on top of these bootstrapped ASTs results in a treebank of ADTs. Alternatively, the concrete syntax of a language can straightforwardly be used to *linearize* a corpus of the target language. The concrete syntax and the concrete configurations when available are used to generate fully labelled UD treebanks for a target language.Figure 4.3 shows an example of a synthetic AST and delexicalized UD tree bootstrapped using the RGL.



(a) An AST of an existential clause bootstrapped using our model.



(b) ADT corresponding to the above example that has to be delexicalized.



(c) The delexicalized UD tree in both English and Swedish shares the same part-of-speech tag sequence and dependency labels

Figure 4.3: Example of a bootstrapped AST and UD tree and the intermediate ADT.

The bootstrapping algorithm uses a parameter corresponding to the maximum depth of the trees d to be generated. The generative story is as follows:

- Pick a category C using the distribution over categories defined in the probability model.
- Select a function f with the definition $C_1 \rightarrow C_2 ... C_n \rightarrow C$ according to the conditional distribution $P(F|C)$.
- Recursively apply the same step to build subtrees of maximum depth $d-1$, t_1, t_2 ... t_n of categories C_1, C_2 ... C_n respectively.
- Apply the function to all the subtrees and return (f t_1 t_2 .. t_n).

4.3.1 Differences against UDv2

The design of the RGL and corresponding configurations do not contain all of the structures defined in the UD annotation scheme. The missing structures fall into two major categories: labels that depend on the lexical realization in a specific language, and structures that correspond to specific linguistic constructions that are not part of the core RGL syntax. Examples of the first type include multi-word expressions and proper nouns (labeled using `fixed` and `flat` label). In the second class, are ellipsis and paratactic constructions in addition to labels that are used in robust analysis of web text (`orphan`, `goeswith` and `reparandum`). Examples that cover these labels can be generated by re-writing the grammar: however, we found very few instances of these in the treebanks. Finally, another variation in the bootstrapped treebanks is in the case of label subtypes that are optionally defined in a language-specific manner. While the configurations allow for accurate generation of certain labels (e.g. `obl:agent` in the case of passive agents), recovering similar information in other instances is not possible without a significant redesign of the RGL (e.g. `obl:tmod` for temporal modifiers). We address this issue by restricting `gf2ud` to generate only the core labels in UD and ignore subtype labels uniformly across languages.

Table 4.1 shows the entropies of the conditional probability distribution defined as probability of a UD label given the part-of-speech tag of the head. The distributions are estimated on both the synthetic UD treebank and a human annotated UD treebank [2] Also shown in the table are the cross-entropy values between the distribution estimated from the synthetic and the original treebanks.

4.4 UD Parsing

The bootstrapped UD treebanks are used to train delexicalized parsing models. We choose to work with the delexicalized UD treebanks for two reasons: the context-free assumption in the probabilistic model defined on the abstract syntax makes the tree generation decomposable, but selectional preferences are not encoded in the generative model used for bootstrapping the ASTs. Additionally, generating a full UD treebank assumes the availability of a interlingua lexicon – which reduces the portability of this approach to new languages.[3] For both these reasons, we restrict ourselves to strictly delexicalized UD treebanks in our parsing experiments.

[2] The UD treebanks are taken from the v2.3 distribution.

[3] There is ongoing work on developing interlingual lexica from linked data like WordNet (Virk et al., 2014, Angelov and Lobanov, 2016).

Language	H(P_{UD})	H(P_{GF})	Cross-entropy
Afrikaans	39.59	58.34	63.12
Arabic	40.00	42.13	51.38
Basque	44.19	51.19	54.21
Bulgarian	32.09	53.76	61.23
Catalan	44.49	49.37	57.39
Chinese	39.25	42.10	59.76
Danish	44.85	55.28	63.39
Dutch	48.99	49.67	61.27
English	50.52	45.31	58.17
Estonian	39.45	43.82	49.35
Finnish	47.86	41.52	54.39
French	43.41	49.43	53.47
German	41.35	49.35	51.29
Greek	29.48	41.13	49.17
Hindi	32.99	43.18	54.27
Italian	38.55	51.37	59.64
Japanese	27.34	40.18	47.25
Latin	42.07	43.47	49.89
Latvian	49.75	49.91	59.26
Norwegian (bokmal)	40.29	45.97	53.17
Norwegian (nynorsk)	37.29	44.56	56.32
Persian	33.07	47.29	47.16
Polish	23.85	41.27	49.83
Portuguese	40.84	48.73	53.60
Romanian	47.31	52.31	57.12
Russian	39.14	47.92	52.84
Spanish	46.36	52.17	57.73
Swedish	35.36	47.41	51.39
Urdu	33.70	42.14	58.73
Icelandic	N/A	51.26	N/A
Thai	N/A	41.23	N/A

Table 4.1: Entropy values of probability distributions P(label|(head-pos)) for different languages estimated from real (P_{UD}) and bootstrapped (P_{GF}) treebanks. If a language has more than one treebank in the UD distribution, we select one treebank as the primary treebank and use that to estimate the distribution and in the parsing experiments. Languages for which a UD treebank does not exist but is included in GF-RGL are listed towards the bottom of the table.

We are interested in the following three use-cases depending on the size of the training data (N) available for inducing parsing models.

- When N $\leq$ 1K sentences[4] are available for a language. There are around 20 treebanks in the current UD distribution that match this criterion and almost all these treebanks have been manually annotated from scratch. This corresponds to the scenario of under-resourced languages, where either the monolingual corpus for treebank or annotators for treebanking are scarce. This scenario strongly corresponds to our proposed idea of simultaneous grammar engineering and treebanking.

- When 1K $\leq$ N $\leq$ 5K sentences[5] are available for a language. There are around 18 treebanks in the current UD distribution that match this criterion. While one can argue that these languages are not really under-resourced, this setup matches the typical case of training domain-specific parsers either for a particular domain like bio-medical or legal texts.

- The case where treebanks are larger than either of the two previous scenariosN $\geq$ 5K. This setup is interesting to test the limit of how useful are bootstrapped ASTs and UD treebanks to train parsing models.

For each of these use-cases, we train parsing models using data from both human annotated UD treebanks and synthetic treebanks for different sizes of training data. The resulting parsing models are evaluated using labelled attachment scores, obtained by parsing the test set of the UD treebank for the language in question. We experiment with an off-the-shelf transition-based dependency parser that gives good results in the dependency parsing task (Straka and Straková, 2017). In the ideal case the experiments need to be carried out using multiple parsers from both the transition-based and graph-based paradigms, we leave that here for future work.

4.5 Experiments

We ran experiments with 3 languages – English, Swedish and Finnish in this paper. In addition to the availability of a concrete syntax for the language, our approach also requires concrete configurations for the languages (Ranta and Kolachina, 2017) in order to bootstrap full UD trees. Table 4.2 shows statistics about the concrete configurations for the RGL grammar for the languages. The probability distribution defined on the RGL was estimated using the GF-Penn treebank (Marcus et al., 1994, Angelov, 2011) of English. The raises another question – how well does the distribution defined on the abstract syntax of the RGL estimated from monolingual data transfer across other languages. The bootstrapping algorithm was restricted to generate 20K ASTs of depth less than 10.[6]

We use UDPipe (Straka and Straková, 2017) to train parsing models, using comparable settings to the baseline systems provided in the CoNLL18 shared task (Zeman and Hajič, 2018). Gold tokenization and part-of-speech tags are used in both training and testing the parser. This was done to control for differences in tagging performance

[4]This approximately corresponds to 20K tokens.
[5]This approximately corresponds to 20K – 100K tokens.
[6]Trees of depth less than 4 were filtered out in the process.

Language	Abstract	Concrete	Morph-features
English	143	21	57
Swedish	143	25	59
Finnish	143	31	57

Table 4.2: Estimate of the effort required in `gf2ud`. The abstract configurations are the same for all languages, while the concrete functions and morph-features are defined for each language. The first column corresponds to configurations for syntactic constructors in the RGL, and second column corresponds to constructors that use syncategorematic words in the linearization.

across the synthetic and original UD treebanks. The models are trained using the primary treebanks from Universal Dependencies v2.3 distribution.[7] We plot the learning curves for parsing models in Figure 4.4 trained on both the original and synthetic treebank data for each use case outlined in Section 4.4. The learning curves were plotted using the LAS accuracies obtained on the test set for the three languages using models trained on both the original and the synthetic treebanks. It is seen from the learning curves that models trained on the synthetic treebanks do not outperform the models trained using original UD treebanks.

However, the full learning curves shown in Figure 4.4 do not tell the complete story. Figure 4.5 shows the learning curves (visualized using bar plots) for English, Finnish and Swedish in the setup where less than 1K sentences from UD treebanks are used. It is clear from the plots for all the three languages that the synthetic treebanks are sub-optimal when directly compared against real treebanks of the **same size**. However, what is interesting is that parsing models in this range (i.e. N $\leq$ 1K) with synthetic treebanks quickly reach comparable accuracies to using real treebank data, with an approximate effective data coefficient of 2.0. In other words comparable accuracies can be obtained using roughly twice the amount of synthetic data, generated for free by the abstract syntax grammar.

It is interesting to note that the learning curves using the synthetic data for the English parsing models become comparably flat in our setup with less than 5K sentences (shown in Figure 4.6a). Despite the lower improvements with increasing treebank sizes, there is still a consistent improvement in parsing accuracies with the best accuracy of 65.4 LAS using 10K synthetic samples (shown in Figure 4.6b). This pattern is consistent across Swedish and Finnish, which allows us to draw the conclusion that while the effective data co-efficient is smaller, the synthetic treebanks are still useful to improve parsing accuracies.

4.6 Related Work

The current trend in dependency parsing is directed towards using synthetic treebanks in an attempt to cover unknown languages for which resources are minimal or do not exist altogether. Such treebanks rely on various auxiliary resources: parallel corpora (Tiedemann, 2014), multilingual word-embeddings (Xiao and Guo, 2014), MT system for the target language (Tiedemann and Agic, 2016, Tyers et al., 2018) or more minimally, tagged corpora in the target language (Wang and Eisner, 2018).

[7] The notion of primary treebank for a language has been made obsolete in UD v2.3 distribution - with all treebanks being assigned a code. So, we use the term primary in this paper to refer to EWT for English, TDT for Finnish and Talbanken for Swedish.

Tiedemann and Agic (2016) propose a method to generate synthetic treebanks for new languages using machine translation systems to transfer cross-linguistic information from resource-rich language to under-resourced languages. This work builds on top of many previous approaches to cross-lingual parsing using parallel corpora and multilingual word-embeddings. The synthetic treebanks generated in the current work are are different in two ways:

- we assume multilingual abstraction and the concrete syntaxes are available- namely the GF-RGL to generate language-independent samples in the form of ASTs.
- we also assume that a distribution of the target language is not available- and what is available is a distribution on the abstract syntax that generalizes to other languages.

Hence, the resulting treebank is licensed by a grammar, and high-precision cross-linguistic information is specified, but the distribution over the resulting treebank is different from the distribution obtained using the real treebanks. An alternative to the method of bootstrapping UD treebanks is to use `ud2gf` (Ranta and Kolachina, 2017) as a way to translate existing UD treebanks to GF treebanks, that are licensed by a grammar.

The current work also relates to more recent work in data-augmentation for dependency parsing (Sahin and Steedman, 2018) and more generally in NLP (Sennrich et al., 2016). The augmentation methods are designed to address data scarcity by exploiting monolingual corpora or generating synthetic samples in multilingual applications. However, the underlying abstractions used to generate the synthetic data are induced from auxiliary corpora.

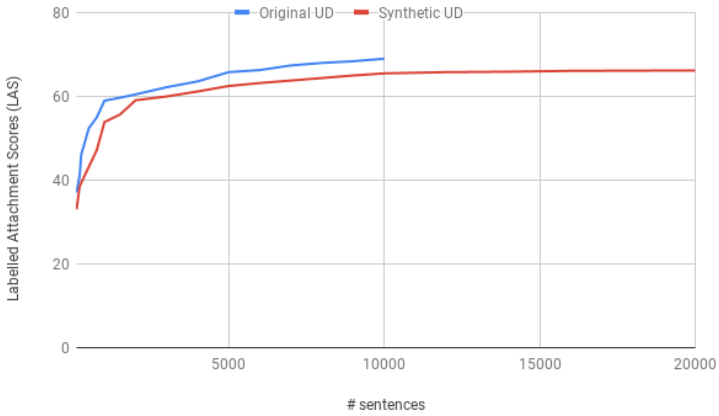
Jonson (2006) show that synthetic corpora generated using a GF grammar can be used to build language models for speech recognition. Experiments in their work show that synthetic in-domain examples generated using the grammar when combined with large out-of-domain data result in significant reduction of word error rate of the speech recognizer. This work falls in line with similar approaches to combine corpus driven approaches with rule-based systems (Bangalore and Johnston, 2004), as a way to combine the statistical information available from corpora with good coverage resulting from rule-based abstractions especially when working with restricted domains. In this paper, we restrict ourselves to utilizing synthetic treebanks for parsing, and leave the discussion on ways to combine synthetic treebanks with real treebanks as future work. This choice is primarily motivated by our interest in grammar-based development of dependency treebanks as opposed to the traditional way of treebanking– by training human annotators.

4.7 Conclusions

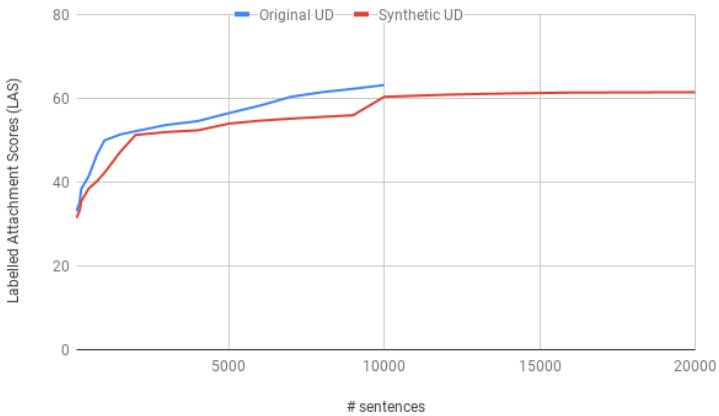
In the current paper, we propose an alternative approach to cross-lingual treebanking—one that recommends grammar engineering. Multilingual abstractions that facilitate bootstrapping of cross-lingual treebanks have been previously explored in the setup of low precision high recall methods. These methods presume the availability of different resources in order to induce the cross-linguistic signal – parallel or multilingual corpora, word embeddings etc. Our approach explores the opposite direction- multilingual grammars of high precision are used to bootstrap parallel treebanks. While these multilingual grammars are not easy to develop, the question of how useful such

grammars are is one that has been largely unexplored in the context of cross-lingual syntactic parsing.

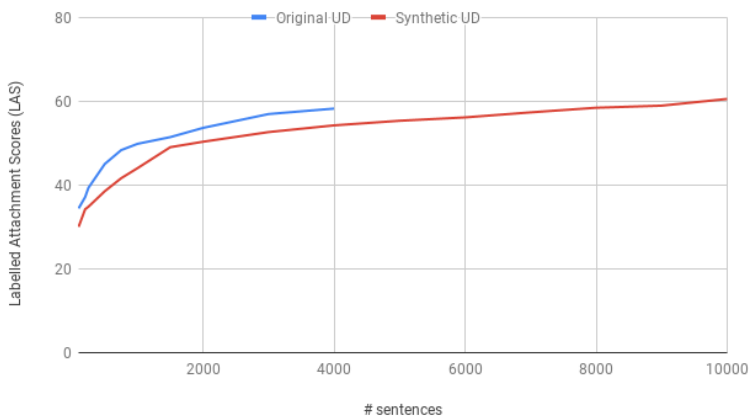
We use a context-free model to generate ASTs that are used to bootstrap parallel UD treebanks in 3 languages. Experiments in delexicalized parsing show that these treebanks are useful in two scenarios– when data in the target language is minimal (<1K sentences) and small (<5K sentences). In the future, we intend to look at ways to generate synthetic treebanks from existing UD treebanks of languages using `ud2gf` (Ranta and Kolachina, 2017), that aims to address the lack of syntactic distributions in our synthetic treebanks. We also did not pursue the obvious direction of combining the real and synthetic treebanks in the current work- we leave this for future work. Another direction that is of interest is to augment existing treebanks with syntactic variations to quantify the need for regular syntactic variants in parser development, such as converting declaratives to questions, varying tense and polarity, adding and removing modifiers, and so on. String-based augmentation (as opposed to precise grammar-based generation) in this direction has already shown promising results (Sahin and Steedman, 2018).



(a) Learning curves for English

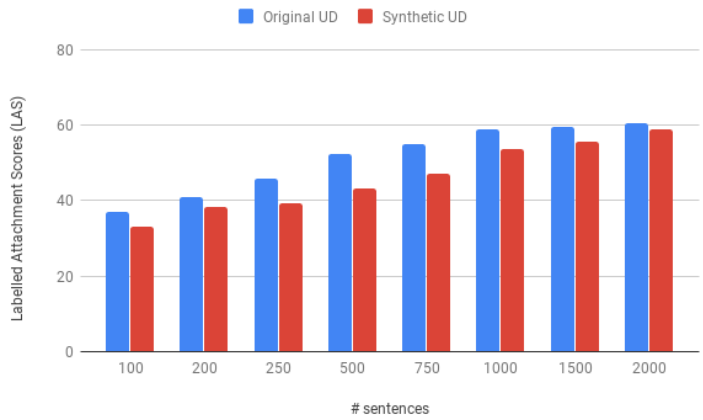


(b) Learning curves for Finnish

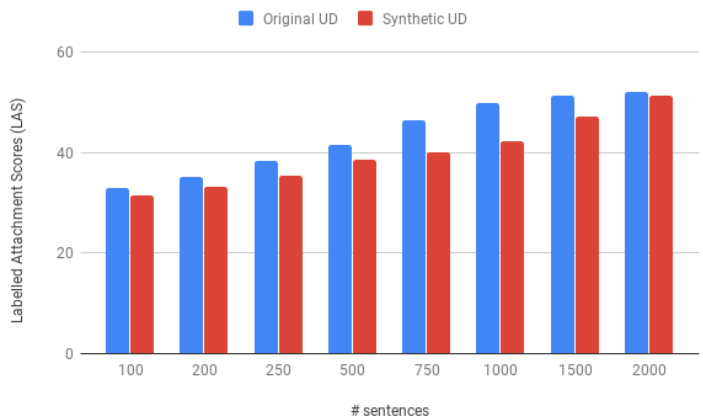


(c) Learning curves for Swedish

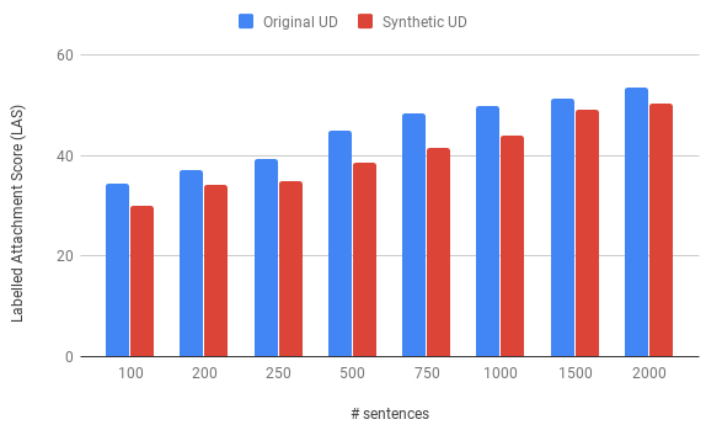
Figure 4.4: Learning curves for parsing models of trained on original UD and synthetic UD treebanks.



(a) Learning curves for English

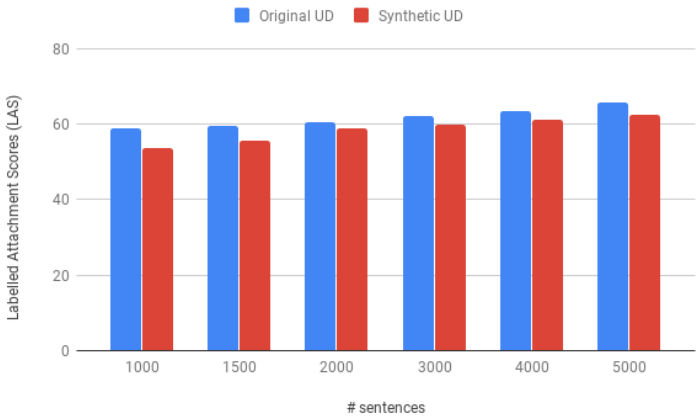


(b) Learning curves for Finnish

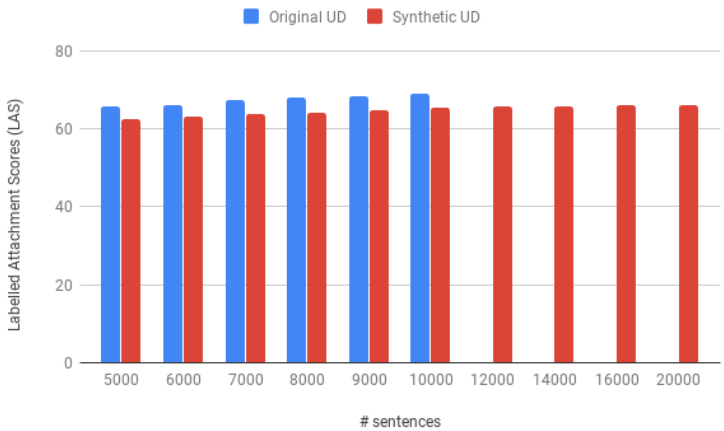


(c) Learning curves for Swedish

Figure 4.5: Learning curves shown using bar plots for parsing models trained on less than 1000 sentences from original UD and 2000 sentences from synthetic UD treebanks.



(a) Learning curves for English with N between 1K and 5K samples



(b) Learning curves for English with N between 5K and 10K samples

Figure 4.6: Learning curves shown using bar plots for parsing models of English

Chapter 5

Paper IV: OOV words in Dependency Parsing

Replacing OOV Words For Dependency Parsing With Distributional Semantics

Kolachina Prasanth and Martin Riedl and Chris Biemann

***Proceedings of the 21st Nordic Conference on Computational Linguistics (NoDaLiDa), pp. 11–19*.**

Abstract

Lexical information is an important feature in syntactic processing like part-of-speech (POS) tagging and dependency parsing. However, there is no such information available for out-of-vocabulary (OOV) words, which causes many classification errors. We propose to replace OOV words with in-vocabulary words that are semantically similar according to distributional similar words computed from a large background corpus, as well as morphologically similar according to common suffixes. We show performance differences both for count-based and dense neural vector-based semantic models. Further, we discuss the interplay of POS and lexical information for dependency parsing and provide a detailed analysis and a discussion of results: while we observe significant improvements for count-based methods, neural vectors do not increase the overall accuracy.

5.1 Introduction

Due to the high expense of creating treebanks, there is a notorious scarcity of training data for dependency parsing. The quality of dependency parsing crucially hinges on the quality of part-of-speech (POS) tagging as a preprocessing step; many dependency parsers also utilize lexicalized information, which is only available for the training vocabulary. Thus errors in dependency parsers often relate to OOV (out of vocabulary, i.e. not seen in the training data) words.

While there has been a considerable amount of work to address the OOV problem with continuous word representations (see Section 5.2), this requires a more complex model and hence, increases training and execution complexity.

In this paper, we present a very simple yet effective way of alleviating the OOV problem to some extent: we use two flavors of distributional similarity, computed on a large background corpus, to replace OOV words in the input with semantically or morphologically similar words that have been seen in the training, and project parse labels back to the original sequence. If we succeed in replacing OOV words with in-vocabulary words of the same syntactic behavior, we expect the tagging and parsing process to be less prone to errors caused by the absence of lexical information.

We show consistent significant improvements both for POS tagging accuracy as well as for Labeled Attachment Scores (LAS) for graph-based semantic similarities. The successful strategies mostly improve POS accuracy on open class words, which results in better dependency parses. Beyond improving POS tagging, the strategy also contributes to parsing accuracy. Through extensive experiments – we show results for seven different languages – we are able to recommend one particular strategy in the conclusion and show the impact of using different similarity sources.

Since our method manipulates the input data rather than the model, it can be used with any existing dependency parser without re-training, which makes it very applicable in existing environments.

5.2 Related Work

While part-of-speech (POS) tags play a major role in detecting syntactic structure, it is well known (Kaplan and Bresnan (1982) inter al.) that lexical information helps for parsing in general and for dependency parsing in particular, see e.g. Wang et al. (2005).

In order to transfer lexical knowledge from the training data to unseen words in the test data, Koo et al. (2008) improve dependency parsing with features based on Brown Clusters (Brown et al., 1992), which are known to be drawing syntactic-semantic distinctions. Bansal et al. (2014) show slight improvements over Koo et al. (2008)'s method by tailoring word embeddings for dependency parsing by inducing them on syntactic contexts, which presupposes the existence of a dependency parser. In more principled fashion, Socher et al. (2013) directly operate on vector representations. Chen et al. (2014) address the lexical gap by generalizing over OOV and other words in a feature role via feature embeddings. Another approach for replacing OOV words by known ones using word embeddings is introduced by Andreas and Klein (2014).

All these approaches, however, require re-training the parser with these additional features and make the model more complex. We present a much simpler setup of replacing OOV words with similar words from the training set, which allows retrofitting any parser with our method.

This work is related to Biemann and Riedl (2013), where OOV performance of fine-grained POS tagging has been improved in a similar fashion. Another similar work to ours is proposed by Huang et al. (2014), who replace OOV named entities with named entities from the same (fine-grained) class for improving Chinese dependency parsing, which largely depends on the quality of the employed NER tagger and is restricted to named entities only. In contrast, we operate on all OOV words, and try to improve prediction on coarse universal POS classes and universal dependencies.

On a related note, examples for a successful application of OOV replacements is demonstrated for Machine Translation (Gangadharaiah et al., 2010, Zhang et al., 2012).

5.3 Methodology

For replacing OOV words we propose three strategies: replace OOV words by most similar ones using distributional semantic methods, replace OOV words with words with the most common suffix and replacing OOV words before or after POS tagging to observe the effect on dependency parsing. The influence of all components is evaluated separately for POS tagging and dependency parsing in Section 5.5.

5.3.1 Semantic Similarities

In order to replace an OOV word by a similar in-vocabulary word, we use models that are based on the distributional hypothesis (Harris, 1951). For showing the impact of different models we use a graph-based approach that uses the left- and right-neighbored word as context, represented by the method proposed by Biemann and Riedl (2013), and is called distributional thesaurus (*DT*). Furthermore, we apply two dense numeric vector-space approaches, using the skip-gram model (*SKG*) and CBOW model of the `word2vec` implementation of Mikolov et al. (2013).

5.3.2 Suffix Source

In addition, we explore replacing OOVs with words from the similarity source that are contained in the training set and share the longest suffix. This might be beneficial as suffixes reflect morphological markers and carry word class information in many languages. The assumption here is that for syntactic dependencies, it is more crucial that the replacement comes from the same word class than its semantic similarity. This also serves as a comparison to gauge the benefits of the similarity source alone. Below, these experiments are marked with *suffix*, whereas the highest-ranked replacement from the similarity sources are marked as *sim*. As a *suffix-only* baseline, we replace OOVs with its most suffix-similar word from the training data, irrespective of its distributional similarity. This serves as a sanity check whether semantic similarities are helpful at all.

5.3.3 Replacement Strategies regarding POS

We explore two different settings for dependency parsing that differ in the use of POS tags:

(1) *oTAG*: POS-tag original sequence, then replace OOV words, retaining original tags for parsing;

(2) *reTAG*: replace OOV word, then POS-tag the new sequence and use the new tags for parsing.

The *oTAG* experiments primarily quantify the sensitivity of the parsing model to word forms, whereas *reTag* assess the potential improvements in the POS tagging.

5.3.4 Replacement Example

As an example, consider the automatically POS-tagged input sentence “We/P went/V to/P the/D aquatic/N park/N" where “aquatic" is an OOV word. Strategy *oTAG sim* replaces “aquatic" with “marine" since it is the most similar in-vocabulary word of “aquatic". Strategy *oTAG suffix* replaces it with “exotic" because of the suffix “tic" and its similarity with “aquatic". The *suffix-only* baseline would replace with “automatic" since it shares the longest suffix of all in-vocabulary words. The *reTAG* strategy would then re-tag the sentence, so the parser will e.g. operate on “We/P went/V to/P the/D marine/ADJ park/N". Table 5.1 shows an example for different similarity-based strategies for English and German[1]. We observe that the *sim* strategy returns semantically similar words that do not necessarily have the same syntactic function as the OOV target.

	sim	**sim&suffix**
English OOV: upgraded		
Suffix-only	paraded	
CBOW	upgrade	downloaded
SKG	upgrade	expanded
DT	expanded	updated
German OOV: Nachtzeit		
Suffix-only	Pachtzeit	
CBOW	tagsüber	Ruhezeit
SKG	tagsüber	Echtzeit
DT	Jahreswende	Zeit

Table 5.1: Here we show replacements for different methods using different strategies.

5.4 Experimental Settings

Here we describe the methods, background corpora used for computing similarities and all further tools used for the experiments. With our experiments, we target to address the following research questions:

- Can syntactic processing benefit from OOV replacement, and if so, under what strategies and conditions?
- Is there a qualitative difference between similarity sources with respect to tagger/parser performance?
- Are there differences in the sensitivity of parsing inference methods to OOV replacement?

[1]Translations: Nachtzeit = night time; tagsüber = during the day; Pachtzeit = length of lease; Ruhezeit = downtime; Echtzeit = real time; Jahreswende = turn of the year

5.4.1 Similarity Computations

We are using two different approaches to determine semantic similarity: a symbolic, graph-based framework for distributional similarity and a neural language model that encodes words in a dense vector space.

Graph-based Semantic Similarity

The computation of a corpus-based distributional thesaurus (marked as *DT* below) is performed following the approach by Biemann and Riedl (2013) as implemented in the JoBimText[2] software. For computing similarities between words from large unlabeled corpora, we extract as word-context the left and right neighboring words, not using language-specific syntactic preprocessing. Words are more similar if they share more of their most salient 1000 context features, where salient context features are ranked by Lexicographer's Mutual Information (LMI), (Evert, 2005). Word similarity in the DT is defined as the count of overlapping salient context features. In addition we prune similar words[3] below a similarity threshold of 5.

In order to use such a DT to replace an OOV word, we look up the most similar terms for the OOV word and choose the highest-ranked word from the training data vocabulary, respectively the most similar word with the longest common suffix.

Neural Semantic Similarity

As an alternative similarity we run `word2vec` with default parameters (marked as *w2v* below) (Mikolov et al., 2013) on our background corpora, obtaining 200-dimensional dense vector embeddings for all words with a corpus frequency larger than 5. We conduct this for both flavors of *w2v*: skipgram, marked as *SKG* below (based on positional windows) and *CBOW* (based on bag of word sentential contexts).

Following the standard approach, we use the cosine between word vectors as a similarity measure: for each OOV, we compare vectors from all words in the training set and pick the word that correspond to the most similar vector as a replacement, respectively the most similar word of those with the longest common suffix.

5.4.2 Corpora for Similarity Computation

As we perform the experiments on various languages, we will compute similarities for each language separately. The English similarities are computed based on 105M sentences from the Leipzig corpora collection (LCC) (Richter et al., 2006). The German (70M) and the Hindi (2M) corpora are extracted from the LCC as well. We compute similarities on 19.7M sentences of Arabic, 259.7M sentences of French and 128.1M sentences of Spanish extracted from web corpora[4] provided by Schäfer and Bildhauer (2013). For the computation of the Swedish similarities we use a 60M-sentence news corpus from Spraakbanken.[5] In summary, all background corpora are in the order of about 1 Gigaword, except the Hindi corpus, which is considerably smaller.

[2] http://www.jobimtext.org

[3] we have tried a few thresholds in preliminary experiments and did not find results to be very sensitive in the range of 2 – 20

[4] http://corporafromtheweb.org/

[5] http://spraakbanken.gu.se

5.4.3 Dependency Parser and POS Tagger

For the dependency parsing we use the implementation of the graph-based dependency parser provided in Mate-tools (Bohnet, 2010, version 3.6) and the transition-based Malt parser (Nivre, 2009, version 1.8.1). Graph-based parsers use global inference to construct the maximum spanning dependency tree for the input sequences. Contrary, the greedy algorithm in the transition-based parser uses local inference to predict the dependency tree. The parsing models for both parsers, Mate-tools and Malt parser, are optimized using cross-validation on the training section of the treebank[6]. We train the dependency parsers using POS tags (from the Mate-tools tagger) predicted using a 5-fold cross-validation. The evaluation of the parser accuracies is carried out using MaltEval. We report labeled attachment score (LAS) for both overall and on OOV token positions.

5.4.4 Treebanks

For training and testing we apply the treebanks (train/dev/test size in tokens in parentheses) from the Universal Dependencies project (Nivre et al., 2016, version 1.2 released November 15th, 2015) for Arabic (226K/28K/28K), English (205K/25K/25K), French (356K/39K/7K), German (270K/12K/16K), Hindi (281K/35K/35K), Spanish (383K/41K/8K), and Swedish (67K/10K/20K). Tagset definitions are available online.[7]

5.5 Results

In this section, we report experimental results and compare them to the baseline without OOV replacement. All statistical significance tests are done using McNemar's test. Significant improvements ($p < 0.05$) over the baseline without OOV replacement are marked with an asterisk (*), significant performance drops with a hashmark (#) and the best result per experiment is marked in bold.

5.5.1 Results for POS Tagging

In Table 5.2 we show overall and OOV-only POS tagging accuracies on the respective test set for seven languages using similarities extracted from the DT.

Unsurprisingly, we observe consistent performance drops, mostly significant, for the *suffix-only* baseline. For all languages except German, the *DT*-based replacement strategies result in significant improvements of either overall accuracy, OOV accuracy or both. In most experiments, the *DT suffix* replacement strategy scores slightly higher than the *DT sim* strategy.

Table 5.3 lists POS accuracies for three languages for similarities from the *w2v* neural language model in its *SKG* and *CBOW* flavors using the cosine similarity. In contrast to the *DT* -based replacements, there are no improvements over the baseline, and some performance drops are even significant. Also replacing the cosine similarity

[6]Using Malt Optimizer (Ballesteros and Nivre, 2014) for the Malt parser; for Mate-tools, we tuned the parameter that represents the percentage of non-projective edges in a language, which matches the parameters suggested by Bohnet (2010).

[7]http://universaldependencies.org/

LANG	OOV	baseline		suffix only		DT sim		DT suffix	
	%	all	OOV	all	OOV	all	OOV	all	OOV
Arabic	10.3	**98.53**	**94.01**	97.82#	87.44#	98.49#	93.67#	98.52	93.91
English	8.0	93.43	75.39	93.09#	72.03#	**93.82***	**78.67***	93.61*	76.75
French	5.3	95.47	83.29	95.17#	78.30#	95.68*	86.28*	**95.73***	**86.78***
German	11.5	**91.92**	85.63	90.88#	77.70#	91.84	85.32	**91.92**	**85.68**
Hindi	4.4	95.35	76.41	95.07#	71.27#	95.41	77.57	**95.44***	**78.00***
Spanish	6.9	94.82	79.62	95.00	81.17	95.45*	**86.36***	**95.49***	85.84*
Swedish	14.3	95.34	89.80	94.78#	86.04 #	95.57*	90.88*	**95.82***	**92.40***

Table 5.2: Test set overall OOV rates, POS accuracy in % for baseline, suffix-only baseline, DT similarity and suffix replacement strategies for seven languages.

LANG	SKG				CBOW			
	sim		suffix		sim		suffix	
	all	OOV	all	OOV	all	OOV	all	OOV
Arabic	98.46#	93.39#	98.50#	93.73#	98.48#	93.60#	98.52	93.94
English	93.10#	72.29#	93.57	76.31	93.24#	73.91	93.52	75.70
German	90.99#	77.65#	91.62#	83.61#	91.78	83.92#	91.91	85.43

Table 5.3: Test set POS accuracies for *w2v*-based model's similarity and suffix replacement strategies for three languages.

with the Euclidian distance did not change this observation. The suffix-based strategy seems to work better than the similarity-based strategy also for the *w2v*-based replacement.

It seems that count-based similarities perform better for the replacement. Thus, we did not extend the experiments with *w2v* to other languages.

5.5.2 Results for Dependency Parsing

As a general trend for all languages (see Table 5.4), we observe that the graph-based parser achieves higher LAS scores than the transition-based parser.

However, the optimal replacement strategy depends on the language for both parsers. Only for Swedish (*reTAG DT suffix*) and Spanish (*reTAG DT sim*), the same replacements yield the highest scores both on all words and OOV words for both parsers. Using the modified POS tags (*reTAG*) results in improvements for the transition-based parser for 4 languages and for 5 languages using the graph-based parser. Whereas the results improve only marginal when using the *reTAG* strategy as can be observed from Table 5.4, most improvements are significant.

Using word embeddings for the *reTAG* strategy (see Table 5.5), we again observe performance drops, except for Arabic.

Following the *oTAG* strategy, we observe significant improvements on German and Arabic for the CBOW method. For German the best performance is obtained with the SKG model (74.47*) which is slightly higher than the *suffix only* replacement, which

			oTAG						reTAG					
	baseline		suffix only		DT sim		DT suffix		suffix only		DT sim		DT suffix	
Language	all	OOV	all	OOV	all	OOV	all	OOV	all	OOV	all	OOV	all	OOV
	Graph-based Parser													
Arabic	75.60	56.90	75.61	57.76*	75.74*	58.18*	75.71*	58.31*	74.54#	52.84#	**75.75***	58.18*	75.72*	58.31*
English	79.57	63.64	79.55	63.77	79.64	64.38*	79.54	64.20	79.24#	62.37	**79.95***	**66.17***	79.78*	65.30*
French	77.76	64.59	77.91	65.34	77.61	64.09	77.79	64.84	77.59	64.59	77.59	64.09	**77.97**	**65.84**
German	74.24	68.93	74.43*	**69.66***	74.27	69.14	74.21	69.24	72.26#	63.43#	74.13	68.10	74.22	69.09
Hindi	87.67	72.00	87.76*	72.74	**87.78***	72.80*	87.71	**72.86***	87.49#	70.60	87.67	72.62	87.69	72.74
Spanish	80.02	63.56	80.07	65.28*	80.32*	67.18*	80.30*	66.84*	79.38#	64.59	**80.41***	**68.91***	80.27	68.05*
Swedish	77.13	70.70	77.16	70.87	77.44*	71.07	77.31*	71.03	76.55#	69.12#	77.62*	71.96*	**77.65***	**72.05***
Δ all	0.00	0.00	0.10	0.72	0.10	0.89	0.08	0.93	-0.79	-1.89	0.02	0.95	**0.12**	**1.35**
	Transition-based Parser													
Arabic	72.63	52.81	72.71	53.67	72.79*	53.94*	72.75*	53.91*	71.75#	48.61#	72.77*	53.84*	72.74*	53.84*
English	77.26	61.84	77.15#	61.67	77.16	61.84	77.30	62.41	76.85#	60.14#	77.32	62.33	**77.53***	**63.29***
French	74.25	63.09	74.37	63.84	74.38	64.09	74.24	62.84	74.14	62.34	74.59*	**64.59**	**74.69***	64.09
German	70.29	63.02	70.24	62.97	70.22	62.76	70.29	63.07	67.97#	56.38#	70.21	62.19	70.16	62.34
Hindi	84.08	66.14	83.99#	65.16	**84.16***	**67.24***	84.14*	67.05*	83.78#	63.08#	84.10	66.99	84.14	66.99
Spanish	75.39	57.86	75.52	59.59*	75.67*	59.93*	75.38	59.07	75.19	60.10	**76.10***	**63.90***	75.68	62.52*
Swedish	73.45	66.59	73.48	66.46	73.52	66.66	73.60*	67.02	72.91#	64.61#	74.01*	68.27*	**74.09***	**68.53***
Δ all	0.00	0.00	0.02	0.36	0.11	0.70	0.02	0.53	-0.76	-2.10	0.12	1.01	**0.20**	**1.50**

Table 5.4: LAS scores for the parsing performance on the test sets when replacing OOV words with a DT. Additionally, we present Δ values for all languages.

	oTAG								reTAG							
	similarity				suffix				similarity				suffix			
	SKG		CBOW		SKG		CBOW		SKG		CBOW		SKG		CBOW	
Language	all	OOV	all	OOV	all	OOV	all	OOV	all	OOV	all	OOV	all	OOV	all	OOV
	Graph-based Parser															
Arabic	75.62	58.00*	75.71*	57.97*	75.67	**58.62***	75.73*	58.49*	75.54	57.66*	75.69	57.83*	75.65	58.42*	75.73*	58.49*
English	79.55	63.85	79.57	64.16	79.58	63.99	79.61	64.03	78.86#	59.97#	79.64	64.12	79.38	62.81	79.57	64.03
German	**74.47***	69.55*	74.39	69.29	74.39*	69.35	74.40*	69.24	72.82#	64.26#	73.70#	66.60#	74.06	67.95	74.14	68.41
Δ all	0.08	0.64	0.08	0.83	0.09	0.65	0.11	0.76	-0.73	-2.53	-0.11	-0.10	-0.13	-0.31	0.01	0.49
	Transition-based Parser															
Arabic	72.62	53.67*	72.65	53.60*	**72.88***	**54.80***	72.72	53.67*	72.60	53.46	72.64	53.49*	72.85*	54.53*	72.71	53.63*
English	77.10#	61.49	77.24	62.06	77.17	62.28	77.28	62.46*	76.54#	57.78#	77.22	61.84	77.07	60.58	77.24	62.37
German	70.19	63.07	70.22	63.38	70.17	**63.54**	**70.36**	63.49	68.90#	57.62#	69.48#	60.68#	69.98#	62.09	70.06	62.60
Δ all	-0.09	0.19	0.01	0.98	-0.02	0.46	0.06	0.65	-0.71	-2.94	-0.09	-0.16	-0.28	-0.55	0.06	0.31

Table 5.5: LAS scores for the parsing performance replacing OOV words with `w2v` and Δ values.

achieves high scores in the *oTAG* setting. Whereas for POS tagging the suffix-based DT replacement mostly results in the highest scores, there is no clear recommendation for a replacement strategy for parsing all languages. Looking at the average delta (Δ) values for all languages (see Tables 5.4 and 5.5) in comparison to the baseline, the picture is clearer: here, for both parsers the *reTAG DT suffix* strategy yields the highest improvements and the CBOW and SKG methods only result in consistent improvements for the oTAG strategy. Further average performance gains are observed for the CBOW suffix-based method using the reTAG strategy.

To sum up, we have noted that the *DT*-based strategies seem more advantageous than the *w2v*-strategies across languages. Comparing the different strategies for using *DTs*, we observe an advantage of *reTAG* over *oTAG* and a slight advantage over *suffix* vs. *sim*. Most notably, *DT reTAG suffix* is the only strategy that never resulted in a significant performance drop on all datasets for both parsers and yields the highest average Δ improvement of 1.50. Given its winning performance on the POS evaluation, we recommend to use this strategy.

5.6 Data Analysis

5.6.1 Analysis of POS Accuracy

Since POS quality has a direct influence on parser accuracy, we have analyzed the two *reTag* strategies *suffix* and *sim* for our three similarity sources (*DT*, *SKG*, *CBOW*) in more detail for German and English by comparing them to the *oTAG* baselines. In general, differences are mostly found for open word classes such as ADJ, ADV, NOUN, PROPN and VERB, which naturally have the highest OOV rates in the test data. In both languages, the DT-based strategies supply about 84% of the replacements of the *w2v* strategies.

For German, only the *DT suffix*-based replacements led to a slight overall POS improvement. All similarity sources improved the tagging of NOUN for *suffix*, but not for *sim*. All replacements led to some losses in VERBs, with *SKG* losing the most. Both *w2v* sources lost more on ADJ than the *DT*, which also showed the largest improvements on ADV. In addition, we analyzed the POS classification only for tokens that could be replaced both by the *DT* and the *w2v*-methods. For these tokens, the *SKG* method can not surpass the *oTAG* performance. Furthermore, for *DT* and *CBOW*, the *suffix* strategies achieve slightly lower scores than *sim* (0.18%-0.63%). On the tokens where all methods propose replacements, the *DT* results in better accuracy (86.00%) than *CBOW* (85.82%).

For English, the picture is similar but in general the improvement of the scores is larger: while the *DT sim* led to the largest and the *DT suffix* to the second-largest overall improvements, the *suffix*-based *w2v*-strategies can also improve POS tagging quality, whereas the *sim w2v*-strategies decrease POS accuracy. Here, we see improvements for ADJ for all but the *sim*-based *w2v*-strategies, improvements on NOUN for all but *SKG suffix*, and for all *suffix* strategies for VERB. Inspecting again the words that can be replaced by all replacement strategies we observe the highest accuracy improvement using the *suffix* strategies: here the scores outperform the baseline (78.07%) up to 84.00% using the *DT* and up to 80.90% with *CBOW*.

The largest difference and the decisive factor for English and German happens on the PROPN tag: Whereas *DT sim* and *SKG suffix* only result in small positive changes, all other strategies frequently mis-tag PROPN as NOUN, increasing this error class by a relative 15% – 45%. These are mostly replacements of rare proper names with rare nouns, which are less found in *DT* replacements due to the similarity threshold. Regarding the other languages, we found largest improvements in French for NOUN for the *DT sim* replacement, coupled with losses on PROPN. Both *DT* strategies improved VERB. For Spanish largest improvements were found in ADJ, NOUN and PRON for both *DT* strategies. Small but significant improvements for Hindi were distributed across parts of speech, and for Arabic, no sizeable improvements were observed.

Only for Arabic we observe a general performance drop when replacing OOV words. Inspecting the OOV words, we detect that around 97% of these words have been annotated as X (other). Overall, the test set contains 8.4% of such annotations, whereas X is rarely encountered in our other languages. Since the baseline performance for Arabic POS is very high, there is not much to improve with replacements.

5.6.2 Analysis of Parsing Accuracy by Relation Label

We have conducted a differential analysis comparing LAS F-scores on all our languages between the baseline and the different replacement options, specifically for

understanding the effects of *DT reTAG* strategies. Focusing on frequent dependency labels (average occurrence: 4% – 14%), we gain improvements for the relations `conj`, `amod` and `case` across all test sets. Except for Hindi, the LAS F1 score increases up to 0.6% F1 for `case` relations, which is the relation between preposition (or postpositions) and the head noun of the prepositional phrase. For the `amod` relation that connects modifying adjectives to nouns, we observe a +0.5% – +1% improvement in F-score for all languages except Hindi and French, corresponding largely to the increased POS accuracy for nouns and adjectives.

For English, we found most improvements in the relations `compound` (about +1 F1) and `name` (+0.5 – +5.0 F1) for both parsers, while relations `cop` and `xcomp` were recognized less precisely (-0.2 – -0.9 F1). The graph-based parser also improves largely in `appos` (+3.5 – +4.2 F1) and `nmod:npmod` (+5.2 – +6.5 F1), while the transition-based parser sees improvements in `iobj` (+3.8 – +5.1 F1) and `neg` (+1.0 F1). For German, the `case` relation improves for both parsers with +0.2 – +0.6 F1. The graph-based parser improves on `auxpass` (+1.1 – 1.4 F1) and `conj` (+0.4 – +0.9 F1). Whereas pinpointing systematic differences between the two parsers is hardly possible, we often observe that the graph-based parser seems to perform better on rare relations, whereas the transition-based parser deals better with frequent relations.

As with the overall evaluation, there is no clear trend for the *suffix* vs. the *sim* strategy for single relations, except for graph-based German `dobj` and `iobj`, which stayed the same or performed worse for the *DT suffix reTAG* (0 – -0.9 F1), but improved greatly for *DT sim reTAG* (+0.9 – +2.4 F1).

In summary, OOV replacement seems to benefit dependency parsing mostly on relations that involve open class words, as well as relations that need semantic information for disambiguation, e.g. `case`, `dobj` and `iobj`.

5.7 Discussion

In the following we want to discuss about selecting a recommendation for the OOV replacement and will highlight the differences we observed in our experiments between graph-based and dense-vector-based similarities.

5.7.1 Recommendations for OOV Replacement

Our experiments show that a simple OOV replacement strategy can lead to significant improvements for dependency parsing across typologically different languages. Improvements can be partially attributed to gains in the POS tagging quality especially with the *suffix*-based replacement strategy, and partially attributed to improved use of lexicalized information from semantic similarity.

Overall, the strategy of replacing OOV words first and POS-tagging the sequence on the basis of the replacements (*reTAG*) shows to be more effective than the other way around. While improvements are generally small yet significant, we still believe that OOV replacement is a viable strategy, especially given its simplicity. In learning curve experiments, as exemplified in Figure 5.1, we found the relative effect to be more pronounced for smaller amounts of training, despite having less in-vocabulary material to choose from. Thus, our approach seems especially suited for low-resource languages where labeled training material is notoriously scarce.

The question whether to use *DT suffix* or *DT sim* as replacement strategy for dependency parsing is not easily answered – while *DT suffix* shows the best overall

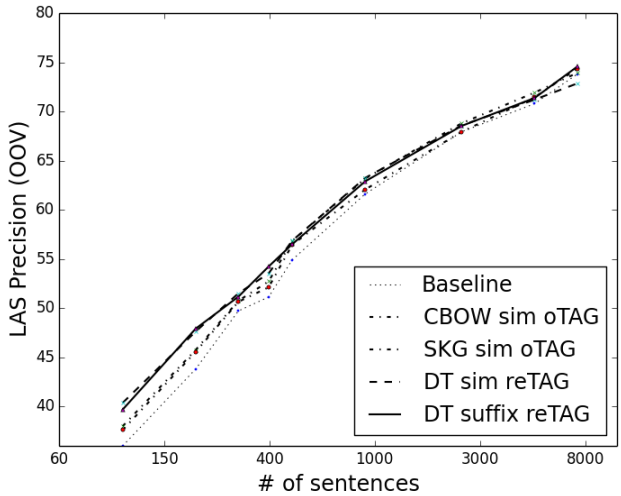


Figure 5.1: Learning curve of LAS for OOV words for English development set.

improvements across the datasets, *DT sim* performs slightly better on Arabic and English graph-based parsing and English POS tagging.

5.7.2 On Differences between Graph-Based and Dense-Vector Similarity

What would be needed to fruitfully utilize the popular neural language model *w2v* as a similarity source, and why does the graph-based *DT* seems to be so much more suited for OOV replacement? From above analysis and from data inspection, we attribute the advantage of *DT* to its capability of NOT returning replacements when it has too low confidence, i.e. no in-vocabulary word is found with a similarity score of 5 or more. In contrast, vector spaces do not provide an interpretable notion of similarity/closeness that can be uniformly applied as a similarity threshold: we have compared cosine similarities of token replacements that lead to improvements, no changes and drops, and found no differences between their average values. A further difference is the structure of the vector space and the *DT* similarity rankings: Whereas the *DT* returns similar words with a frequency bias, i.e. rather frequent words are found in the most similar words per OOV target, the vector space does not have such frequency bias and, since there are more rare than frequent words in language, returns many rare words from the background corpus[8]. This effect can be alleviated to some extent when applying frequency thresholds, but is in turn aggravated when scaling up the background corpus. Thus, a condition that would only take the top-N most similar words from the background collection into account for expansions is also bound to fail for *w2v*. The only reasonable mechanism seems to be a background corpus frequency threshold on the in-vocabulary word. However, even when comparing only on the positions where both *DT* and *w2v* returned replacements, we still find *DT* replacements

[8] we have seen this effect repeatedly and consistently across corpora, languages and parameters

more advantageous. Inspection revealed that while many replacements are the same for the similarity sources, the *DT* replacements more often stay in the same word class (cf. Table 5.1), e.g. regarding conjugative forms of verbs and regarding the distinction between common and proper nouns.

5.8 Conclusion

In this paper, we have shown that syntactic preprocessing, both POS tagging and dependency parsing, can benefit from OOV replacement. We have devised a simple yet effective strategy (*DT suffix reTAG*) to improve the quality of universal dependency parsing by replacing OOV words via semantically similar words that share a suffix, subsequently run the POS tagger and the dependency parser over the altered sequence, and projecting the labels back to the original sequence. In these experiments similar words from a count-based distributional thesaurus are more effective than the dense numeric `w2v` approach.

In future work, we will apply our method for other types of lexicalized parsers, such as constituency grammar and combinatory categorial grammar parsers, as well as examine the influence of OOVs on semantic tasks like semantic role labeling or frame-semantic parsing.

Bibliography

Anne Abeillé, Lionel Clément, and Alexandra Kinyon. Building a Treebank for French. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*, Athens, Greece, May 2000. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2000/pdf/230.pdf.

Alfred Aho and Jeffrey Ullman. Properties of syntax directed translations. *Journal of Computer and System Sciences*, 3(3):319 – 334, 1969a. ISSN 0022-0000. doi: https://doi.org/10.1016/S0022-0000(69)80018-8. URL http://www.sciencedirect.com/science/article/pii/S0022000069800188.

Alfred V. Aho and Jeffrey D. Ullman. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3(1):37–56, 1969b.

Jacob Andreas and Dan Klein. How much do word embeddings encode about syntax? In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 822–827, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-2133. URL https://www.aclweb.org/anthology/P14-2133.

Krasimir Angelov. Incremental Parsing with Parallel Multiple Context-Free Grammars. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 69–76, Athens, Greece, March 2009. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/E09-1009.

Krasimir Angelov. *The Mechanics of the Grammatical Framework*. PhD thesis, Chalmers University of Technology, 2011.

Krasimir Angelov and Peter Ljunglöf. Fast Statistical Parsing with Parallel Multiple Context-Free Grammars. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 368–376, Gothenburg, Sweden, April 2014. Association for Computational Linguistics. doi: 10.3115/v1/E14-1039. URL https://www.aclweb.org/anthology/E14-1039.

Krasimir Angelov and Gleb Lobanov. Predicting Translation Equivalents in Linked WordNets. In *Proceedings of the Sixth Workshop on Hybrid Approaches to Translation (HyTra6)*, pages 26–32, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL https://www.aclweb.org/anthology/W16-4504.

Krasimir Angelov, Björn Bringert, and Aarne Ranta. Speech-Enabled Hybrid Multilingual Translation for Mobile Devices. In *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association*

for Computational Linguistics, pages 41–44, Gothenburg, Sweden, April 2014. Association for Computational Linguistics. doi: 10.3115/v1/E14-2011. URL https://www.aclweb.org/anthology/E14-2011.

Andrew Appel. *Modern Compiler Implementation in ML*. Cambridge University Press, 1998.

Miguel Ballesteros and Joakim Nivre. MaltOptimizer: Fast and effective parser optimization. *Natural Language Engineering*, FirstView:1–27, 2 2014.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract Meaning Representation for Sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W13-2322.

Srinivas Bangalore. *Complexity of Lexical Descriptions and Its Relevance to Partial Parsing*. PhD thesis, University of Pennsylvania, Philadelphia, Pennsylvania, 6 1997.

Srinivas Bangalore and Michael Johnston. Balancing data-driven and rule-based approaches in the context of a Multimodal Conversational System. In *HLT-NAACL 2004: Main Proceedings*, pages 33–40, Boston, Massachusetts, USA, May 2004. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N04-1005.

Srinivas Bangalore and Aravind K. Joshi. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265, 1999. URL https://www.aclweb.org/anthology/J99-2004.

Mohit Bansal, Kevin Gimpel, and Karen Livescu. Tailoring Continuous Word Representations for Dependency Parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 809–815, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-2131. URL https://www.aclweb.org/anthology/P14-2131.

Kenneth R. Beesley and Lauri Karttunen. *Finite State Morphology*. CSLI, 2003.

Emily M. Bender and Dan Flickinger. Rapid Prototyping of Scalable Grammars: Towards Modularity in Extensions to a Language-Independent Core. In *Companion Volume to the Proceedings of Conference including Posters/Demos and tutorial abstracts*, 2005. URL https://www.aclweb.org/anthology/I05-2035.

Jean-Philippe Bernardy and Stergios Chatzikyriakidis. A Type-Theoretical system for the FraCaS test suite: Grammatical Framework meets Coq. In *IWCS 2017 - 12th International Conference on Computational Semantics - Long papers*, 2017. URL https://www.aclweb.org/anthology/W17-6801.

Chris Biemann and Martin Riedl. Text: Now in 2D! A Framework for Lexical Expansion with Contextual Similarity. *Journal of Language Modelling*, 1(1): 55–95, 2013. URL http://jlm.ipipan.waw.pl/index.php/JLM/article/view/60.

Alena Böhmová, Jan Hajič, Eva Hajičová, and Barbora Hladká. The Prague Dependency Treebank. In *Treebanks*, pages 103–127. Springer, 2003.

Bernd Bohnet. Top Accuracy and Fast Dependency Parsing is not a Contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China, August 2010. Coling 2010 Organizing Committee. URL https://www.aclweb.org/anthology/C10-1011.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jenifer C. Lai, and Robert L. Mercer. Class-Based *n*-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–480, 1992. URL https://www.aclweb.org/anthology/J92-4003.

Miriam Butt, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. The Parallel Grammar Project. In *COLING-02: Grammar Engineering and Evaluation*, 2002. URL https://www.aclweb.org/anthology/W02-1503.

Eugune Charniak. Tree-bank Grammars. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1031–1036, Portland, Oregon, August 1996. ISBN 0-262-51091-X. URL http://portal.acm.org/citation.cfm?id=1864519.1864540.

Wenliang Chen, Yue Zhang, and Min Zhang. Feature Embedding for Dependency Parsing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 816–826, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics. URL https://www.aclweb.org/anthology/C14-1078.

David Chiang. A Hierarchical Phrase-Based Model for Statistical Machine Translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 263–270, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219873. URL https://www.aclweb.org/anthology/P05-1033.

David Chiang. Hierarchical Phrase-Based Translation. *Journal of Computational Linguistics*, 33(2):201–228, 2007. doi: 10.1162/coli.2007.33.2.201. URL https://www.aclweb.org/anthology/J07-2003.

Stephen Clark. Supertagging for Combinatory Categorial Grammar. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pages 19–24, Universitá di Venezia, May 2002. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W02-2203.

Michael John Collins. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 184–191, Santa Cruz, California, USA, June 1996. Association for Computational Linguistics. doi: 10.3115/981863.981888. URL https://www.aclweb.org/anthology/P96-1025.

Ann Copestake and Dan Flickinger. An Open Source Grammar Development Environment and Broad-coverage English Grammar Using HPSG. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*, Athens, Greece, May 2000. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2000/pdf/371.pdf.

William Croft, Dawn Nordquist, Katherine Looney, and Michael Regan. Linguistic Typology meets Universal Dependencies. In *Treebanks and Linguistic Theories (TLT-2017)*, pages 63–75, Bloomington IN, January 20–21, 2017.

Haskell B. Curry. Some Logical Aspects of Grammatical Structure. In *Structure of Language and its Mathematical Aspects: Proceedings of the Twelfth Symposium in Applied Mathematics*, pages 56–68. American Mathematical Society, 1961.

Dana Dannells and Normunds Gruzitis. Extracting a bilingual semantic grammar from FrameNet-annotated corpora. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2466–2473, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2014/pdf/1079_Paper.pdf.

Dana Dannélls, Mariana Damova, Ramona Enache, and Milen Chechev. Multilingual Online Generation from Semantic Web Ontologies. In *Proceedings of the 21st International Conference on World Wide Web*, pages 239–242, Lyon, France, 2012. ACM.

Dana Dannells, Aarne Ranta, Ramona Enache, Mariana Damova, and Maria Mateva. Multilingual access to cultural heritage content on the Semantic Web. In *Proceedings of the 7th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 107–115, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W13-2715.

Marie-Catherine de Marneffe and Christopher D. Manning. The Stanford Typed Dependencies Representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, Manchester, UK, August 2008. Coling 2008 Organizing Committee. URL https://www.aclweb.org/anthology/W08-1301.

Marie-Catherine de Marneffe and Joakim Nivre. Dependency grammar. *Annual Review of Linguistics*, 5(1):197–218, 2019. doi: 10.1146/annurev-linguistics-011718-011842. URL https://doi.org/10.1146/annurev-linguistics-011718-011842.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy, May 2006. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2006/pdf/440_pdf.pdf.

Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. Universal Stanford

dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4585–4592, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2014/pdf/1062_Paper.pdf.

Ralph Debusmann. An introduction to dependency grammar. January 2000.

David R. Dowty. *Word Meaning and Montague Grammar*. D. Reidel, Dordrecht, 1979.

Rebecca Dridan. Ubertagging: Joint Segmentation and Supertagging for English. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1201–1212, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D13-1120.

Marc Dymetman, Veronika Lux, and Aarne Ranta. XML and Multilingual Document Authoring: Convergent Trends. In *COLING 2000 Volume 1: The 18th International Conference on Computational Linguistics*, pages 243–249, Saarbrücken, Germany, 2000. URL https://www.aclweb.org/anthology/C00-1036.

Stefan Evert. *The Statistics of Word Cooccurrences: Word Pairs and Collocations*. PhD thesis, Institut für maschinelle Sprachverarbeitung, University of Stuttgart, 2005.

Winthrop Francis Nelson and Henry Kučera. *Manual of Information to accompany a Standard Corpus of present-day edited American English, for use with digital computers*. Brown University, Department of Lingustics, 1979.

Rashmi Gangadharaiah, Ralf D. Brown, and Jaime Carbonell. Monolingual Distributional Profiles for Word Substitution in Machine Translation. In *Coling 2010: Posters*, pages 320–328, Beijing, China, August 2010. Coling 2010 Organizing Committee. URL https://www.aclweb.org/anthology/C10-2037.

Dan Garrette and Jason Baldridge. Learning a Part-of-Speech Tagger from Two Hours of Annotation. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 138–147, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N13-1014.

Gerald Gazdar, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. *Generalized Phrase Structure Grammar*. Basil Blackwell, Oxford, 1985.

Normunds Gruzitis and Guntis Barzdins. The role of CNL and AMR in scalable abstractive summarization for multilingual media monitoring. In *Controlled Natural Language*, volume 9767 of *Lecture Notes in Computer Science*, pages 127–130. Springer, 2016. doi: 10.1007/978-3-319-41498-0. URL http://arxiv.org/abs/1606.05994.

Normunds Gruzitis and Dana Dannélls. A Multilingual FrameNet-based Grammar and Lexicon for Controlled Natural Language. *Language Resources and Evaluation*, 2015. URL http://arxiv.org/abs/1511.03924.

Normunds Gruzitis, Didzis Gosko, and Guntis Barzdins. RIGOTRIO at SemEval-2017 Task 9: Combining machine learning and grammar engineering for AMR parsing and generation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval)*, pages 924–928, Vancouver, Canada, 2017. doi: 10.18653/v1/S17-2159. URL http://www.aclweb.org/anthology/S17-2159.

Thomas Hallgren and Aarne Ranta. An Extensible Proof Text Editor. In *Logic for Programming and Automated Reasoning: 7th International Conference, LPAR 2000 Proceedings*, volume 1955 of *LNCS/LNAI*, pages 70–84. Springer, 2000.

Zellig S. Harris. *Methods in Structural Linguistics*. University of Chicago Press, Chicago, 1951.

Julia Hockenmaier and Mark Steedman. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Journal of Computational Linguistics*, 33(3):355–396, 2007. doi: 10.1162/coli.2007.33.3.355. URL https://www.aclweb.org/anthology/J07-3004.

Hen-Hsen Huang, Huan-Yuan Chen, Chang-Sheng Yu, Hsin-Hsi Chen, Po-Ching Lee, and Chun-Hsun Chen. Sentence Rephrasing for Parsing Sentences with OOV Words. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2859–2862, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2014/pdf/60_Paper.pdf.

Mark Johnson. PCFG Models of Linguistic Tree Representations. *Computational Linguistics*, 24(4):613–632, 1998. URL https://www.aclweb.org/anthology/J98-4004.

Rebecca Jonson. Generating Statistical Language Models from Interpretation Grammars in Dialogue Systems. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006. URL https://www.aclweb.org/anthology/E06-1008.

Aravind K. Joshi and Yves Schabes. Tree-Adjoining Grammars. In *Handbook of Formal Languages*, 1997.

Kaarel Kaljurand and Tobias Kuhn. A Multilingual Semantic Wiki Based on Attempto Controlled English and Grammatical Framework. In *Proceedings of The Semantic Web: Semantics and Big Data: 10th International Conference, ESWC 2013*, pages 427–441. Springer, 2013.

Ronald M. Kaplan and Joan Bresnan. Lexical-Functional Grammar: A Formal System for Grammatical Representation. In *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA, 1982.

Janna Khegai. GF Parallel Resource Grammars and Russian. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 475–482, Sydney, Australia, July 2006. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P06-2062.

Prasanth Kolachina and Aarne Ranta. From Abstract Syntax to Universal Dependencies. *Linguistic Issues in Language Technology*, 13(3), 2016.

Terry Koo, Xavier Carreras, and Michael Collins. Simple Semi-supervised Dependency Parsing. In *Proceedings of ACL-08: HLT*, pages 595–603, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P08-1068.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. Dependency Parsing. *Synthesis Lectures on Human Language Technologies*, 2(1):1–127, 2009. doi: 10.2200/S00169ED1V01Y200901HLT002. URL https://doi.org/10.2200/S00169ED1V01Y200901HLT002.

Joachim Lambek. The Mathematics of Sentence Structure. *Journal of Symbolic Logic*, 33(4):627–628, 1968. doi: 10.2307/2271418.

Herbert Lange. Implementation of a latin grammar in grammatical framework. In *Proceedings of the 2Nd International Conference on Digital Access to Textual Cultural Heritage*, DATeCH2017, pages 97–102, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5265-9. doi: 10.1145/3078081.3078108. URL http://doi.acm.org/10.1145/3078081.3078108.

P. M. Lewis, II and R. E. Stearns. Syntax-directed transduction. *J. ACM*, 15(3): 465–488, July 1968. ISSN 0004-5411. doi: 10.1145/321466.321477. URL http://doi.acm.org/10.1145/321466.321477.

Inari Listenmaa. *Formal Methods for Testing Grammars*. PhD thesis, Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden, October 2019.

Peter Ljunglöf. *The Expressivity and Complexity of Grammatical Framework*. PhD thesis, Department of Computing Science, Chalmers University of Technology and University of Gothenburg, 2004.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyr, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: Annotating Predicate Argument Structure. In *HUMAN LANGUAGE TECHNOLOGY: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, pages 114–119, 1994. URL https://www.aclweb.org/anthology/H94-1020.

John McCarthy. Towards a mathematical science of computation. In *Proceedings of the Information Processing Congress (IFIP) 62*, pages 21–28, Munich, West Germany, August 1962. North-Holland.

Ryan McDonald, Slav Petrov, and Keith Hall. Multi-Source Transfer of Delexicalized Dependency Parsers. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 62–72, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D11-1006.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. Universal Dependency Annotation for Multilingual Parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P13-2017.

I. Dan Melamed and Wei Wang. Statistical Machine Translation by Parsing. *CoRR*, cs.CL/0407005, 2004. URL http://arxiv.org/abs/cs.CL/0407005. An alternate version of Generalized Parsers for Machine Translation.

I. Dan Melamed, Giorgio Satta, and Benjamin Wellington. Generalized Multitext Grammars. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 661–668, Barcelona, Spain, July 2004. doi: 10.3115/1218955.1219039. URL https://www.aclweb.org/anthology/P04-1084.

Paul Meurer. From LFG structures to dependency relations. *Bergen Language and Linguistics Studies*, 8(1), November 2017. doi: 10.15845/bells.v8i1.1341. URL https://bells.uib.no/index.php/bells/article/view/1341.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of First International Conference on Learning Representations, ICLR 2013, Workshop Track*, ICLR 2013, pages 1310–1318, 2013. URL http://arxiv.org/pdf/1301.3781.

Simon Mille, Bernd Bohnet, Leo Wanner, and Anja Belz. Shared Task Proposal: Multilingual Surface Realization Using Universal Dependency Trees. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 120–123, Santiago de Compostela, Spain, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-3517. URL https://www.aclweb.org/anthology/W17-3517.

Simon Mille, Anja Belz, Bernd Bohnet, Yvette Graham, Emily Pitler, and Leo Wanner. The First Multilingual Surface Realisation Shared Task (SR'18): Overview and Evaluation Results. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 1–12, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W18-3601.

Richard Montague. *Formal Philosophy*. Yale University Press, New Haven (Conn.) (etc.), 1974. Collected papers edited by Richmond Thomason.

Reinhard Muskens. New Directions in Type-Theoretic Grammars. *Journal of Logic, Language and Information*, 19(2):129–136, 2010.

Mark-Jan Nederhof and Heiko Vogler. Synchronous Context-Free Tree Grammars. In *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, pages 55–63, Paris, France, September 2012. URL https://www.aclweb.org/anthology/W12-4607.

Graham Neubig, Philip Arthur, and Kevin Duh. Multi-Target Machine Translation with Multi-Synchronous Context-free Grammars. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 293–302, Denver, Colorado, May–June 2015. Association for Computational Linguistics. doi: 10.3115/v1/N15-1033. URL https://www.aclweb.org/anthology/N15-1033.

Grace Ngai and David Yarowsky. Rule Writing or Annotation: Cost-efficient Resource Usage for Base Noun Phrase Chunking. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 117–125, Hong Kong,

October 2000. Association for Computational Linguistics. doi: 10.3115/1075218.1075234. URL https://www.aclweb.org/anthology/P00-1016.

Joakim Nivre. *Inductive Dependency Parsing*. Text, Speech and Language Technology. Springer Netherlands, 2006.

Joakim Nivre. 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*, pages 351–359, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P09-1040.

Joakim Nivre. Towards a Universal Grammar for Natural Language Processing. In *CICLing 2015: Proceedings of Computational Linguistics and Intelligent Text Processing*, volume 9041 of *LNCS*, pages 3–16, Cairo, Egypt, April 2015.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, pages 1659–1666, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL https://www.aclweb.org/anthology/L16-1262.

Stephan Oepen, Dan Flickinger, Kristina Toutanova, and Christopher D. Manning. LinGO Redwoods: A Rich and Dynamic Treebank for HPSG. *Research on Language and Computation*, 2(4):575–596, December 2004. ISSN 1572-8706. doi: 10.1007/s11168-004-7430-4. URL https://doi.org/10.1007/s11168-004-7430-4.

Peteris Paikens and Normunds Gruzitis. An implementation of a Latvian resource grammar in Grammatical Framework. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 1680–1685, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2012/pdf/976_Paper.pdf.

Ioanna Papadopoulou. GF Modern Greek Resource Grammar. In *Proceedings of the Student Research Workshop associated with RANLP 2013*, pages 126–133, Hissar, Bulgaria, September 2013. INCOMA Ltd. Shoumen, BULGARIA. URL https://www.aclweb.org/anthology/R13-2019.

Adam Pauls, Dan Klein, David Chiang, and Kevin Knight. Unsupervised Syntactic Alignment with Inversion Transduction Grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 118–126, Los Angeles, California, June 2010. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N10-1014.

Slav Petrov, Dipanjan Das, and Ryan McDonald. A Universal Part-of-Speech Tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 2089–2096, Istanbul, Turkey, May 2012. European

Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2012/pdf/274_Paper.pdf.

Adam Przepiórkowski and Agnieszka Patejuk. From Lexical Functional Grammar to Enhanced Universal Dependencies. In *Proceedings of the Joint Workshop on Linguistic Annotation, Multiword Expressions and Constructions (LAW-MWE-CxG-2018)*, pages 2–4, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W18-4902.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2018. URL https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

Aarne Ranta. Computational Semantics in Type Theory. *Mathematics and Social Sciences*, 165:31–57, 2004a.

Aarne Ranta. Grammatical Framework: A Type-Theoretical Grammar Formalism. *The Journal of Functional Programming*, 14(2):145–189, 2004b.

Aarne Ranta. Grammars as Software Libraries. In *From Semantics to Computer Science. Essays in Honour of Gilles Kahn*, pages 281–308. Cambridge University Press, 2009a.

Aarne Ranta. The GF Resource Grammar Library. *Linguistic Issues in Language Technology*, 2(2), 2009b.

Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011.

Aarne Ranta and Prasanth Kolachina. From Universal Dependencies to Abstract Syntax. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 107–116, Gothenburg, Sweden, 2017. Association for Computational Linguistics.

Aarne Ranta, Krasimir Angelov, and Thomas Hallgren. Tools for Multilingual Grammar-Based Translation on the Web. In *Proceedings of the ACL 2010 System Demonstrations*, pages 66–71, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P10-4012.

Aarne Ranta, Ramona Enache, and Grégoire Détrez. Controlled Language for Everyday Use: The MOLTO Phrasebook. In *Controlled Natural Language: Second International Workshop, CNL 2010, Revised Papers*, volume 7175 of *LNCS/LNAI*, pages 115–136. Springer, 2012.

Aarne Ranta, Prasanth Kolachina, and Thomas Hallgren. Cross-Lingual Syntax: Relating Grammatical Framework with Universal Dependencies. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 322–325, Gothenburg, Sweden, May 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W17-0247.

Manny Rayner, David Carter, Pierrette Bouillon, Vassilis Digalakis, and Mats Wirén. *The Spoken Language Translator*. Cambridge University Press, Cambridge, 1st edition, 2000.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. Transforming Dependency Structures to Logical Forms for Semantic Parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140, December 2016. doi: 10.1162/tacl_a_00088. URL https://www.aclweb.org/anthology/Q16-1010.

Matthias Richter, Uwe Quasthoff, Erla Hallsteinsdóttir, and Chris Biemann. Exploiting the Leipzig Corpora Collection. In *Proceedings of the IS-LTC 2006*, pages 68–73, Ljubljana, Slovenia, 2006. URL http://wortschatz.uni-leipzig.de/~cbiemann/pub/2006/RichterQuasthoffHallsteinsdottirBiemann-ISLTC.pdf.

Rudolf Rosa, Jan Mašek, David Mareček, Martin Popel, Daniel Zeman, and Zdeněk Žabokrtský. HamleDT 2.0: Thirty Dependency Treebanks Stanfordized. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2334–2341, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2014/pdf/915_Paper.pdf.

Eugen Ruppert, Jonas Klesy, Martin Riedl, and Chris Biemann. Rule-based Dependency Parse Collapsing and Propagation for German and English. In *Proceedings of International Conference of the German Society for Computational Linguistics and Language Technology*, Essen, 2015.

Gozde Gul Sahin and Mark Steedman. Data Augmentation via Dependency Tree Morphing for Low-Resource Languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5004–5009, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D18-1545.

Roland Schäfer and Felix Bildhauer. Web Corpus Construction. *Synthesis Lectures on Human Language Technologies*, 6(4):1–145, 2013. doi: 10.2200/S00508ED1V01Y201305HLT022. URL https://doi.org/10.2200/S00508ED1V01Y201305HLT022.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229, 1991.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving Neural Machine Translation Models with Monolingual Data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1009. URL https://www.aclweb.org/anthology/P16-1009.

Petr Sgall, Eva Hajičová, and Jarmila Panevová. *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Reidel, Dordrecht, 1986.

Stuart M. Shieber. Bimorphisms and synchronous grammars. *Journal of Language Modelling*, 2(1):51–104, 2014. doi: 10.15398/jlm.v2i1.84. URL http://jlm.ipipan.waw.pl/index.php/JLM/article/view/84.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. Parsing with Compositional Vector Grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P13-1045.

Ian Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition, 2010. ISBN 0137035152, 9780137035151.

Milan Straka and Jana Straková. Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-3009. URL https://www.aclweb.org/anthology/K17-3009.

Lucien Tesnière. *Elements of Structural Syntax*. John Benjamins, 2015. URL https://www.jbe-platform.com/content/books/9789027269997.

Jörg Tiedemann. Rediscovering Annotation Projection for Cross-Lingual Parser Induction. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1854–1864, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics. URL https://www.aclweb.org/anthology/C14-1175.

Jörg Tiedemann and Zeljko Agic. Synthetic Treebanking for Cross-Lingual Dependency Parsing. *The Journal of Artificial Intelligence Research (JAIR)*, 55:209–248, 2016. doi: 10.1613/jair.4785.

Francis Tyers, Mariya Sheyanova, Aleksandra Martynova, Pavel Stepachev, and Konstantin Vinogorodskiy. Multi-source synthetic treebank creation for improved cross-lingual dependency parsing. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 144–150, Brussels, Belgium, November 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W18-6017.

Ashish Vaswani, Liang Huang, and David Chiang. Smaller Alignment Models for Better Translations: Unsupervised Word Alignment with the l0-norm. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 311–319, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P12-1033.

Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. Supertagging With LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1027. URL https://www.aclweb.org/anthology/N16-1027.

K Vijay-Shanker and David Weir. The Equivalence Of Four Extensions Of Context-Free Grammars. *Mathematical Systems Theory*, 27, 02 1995. doi: 10.1007/BF01191624.

Shafqat Mumtaz Virk, K.V.S Prasad, Aarne Ranta, and Krasimir Angelov. Developing an interlingual translation lexicon using WordNets and Grammatical Framework. In *Proceedings of the Fifth Workshop on South and Southeast Asian Natural Language Processing*, pages 55–64, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University. doi: 10.3115/v1/W14-5508. URL https://www.aclweb.org/anthology/W14-5508.

Dingquan Wang and Jason Eisner. The Galactic Dependencies Treebanks: Getting More Data by Synthesizing New Languages. *Transactions of the Association for Computational Linguistics*, 4:491–505, December 2016. doi: 10.1162/tacl_a_00113. URL https://www.aclweb.org/anthology/Q16-1035.

Dingquan Wang and Jason Eisner. Synthetic Data Made to Order: The Case of Parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1325–1337, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D18-1163.

Qin Iris Wang, Dale Schuurmans, and Dekang Lin. Strictly Lexical Dependency Parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 152–159, Vancouver, British Columbia, October 2005. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W05-1516.

Dekai Wu. Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora. *Computational Linguistics*, 23(3):377–403, 1997. URL https://www.aclweb.org/anthology/J97-3002.

F. Xia. *Automatic grammar generation from two different perspectives*. PhD thesis, 2001. AAI3031738.

Min Xiao and Yuhong Guo. Distributed Word Representation Learning for Cross-Lingual Dependency Parsing. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 119–129, Ann Arbor, Michigan, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-1613. URL https://www.aclweb.org/anthology/W14-1613.

XTAG. A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS-01-03, IRCS, University of Pennsylvania, 2001.

Daniel Zeman and Jan Hajič. Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. Brussels, Belgium, October 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/K18-2000.

Hao Zhang, Daniel Gildea, and David Chiang. Extracting Synchronous Grammar Rules From Word-Level Alignments in Linear Time. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 1081–1088, Manchester, UK, August 2008. Coling 2008 Organizing Committee. URL https://www.aclweb.org/anthology/C08-1136.

Jiajun Zhang, Feifei Zhai, and Chengqing Zong. Handling Unknown Words in Statistical Machine Translation from a New Perspective. In *Proceedings of the 1st*

Conference on Natural Language Processing and Chinese Computing, NLP&CC '12, pages 176–187, Beijing, China, 2012. ISBN 978-3-642-34455-8. URL http://link.springer.com/chapter/10.1007%2F978-3-642-34456-5_17.