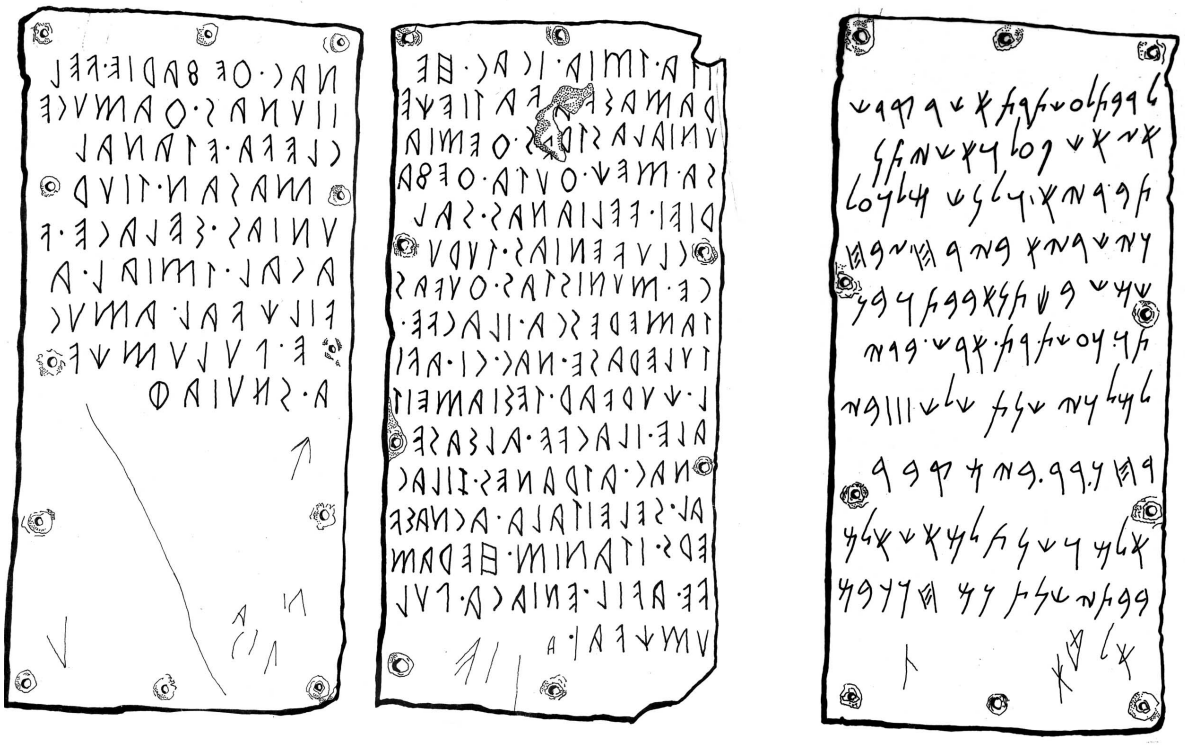
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Syntax-based Concept Alignment for Machine Translation

Master's thesis in Computer Science

Arianna Masciolini

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Syntax-based Concept Alignment for Machine Translation

Arianna Masciolini

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Syntax-based Concept Alignment for Machine Translation
Arianna Masciolini

Supervisor: Aarne Ranta, Department of Computer Science and Engineering
Examiner: Carl-Johan Seger, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Reproduction of the Pyrgi Tablets, a bilingual Etruscan-Phoenician dedicatory text. Courtesy of the Museo Nazionale Etrusco di Villa Giulia.

Typeset in LaTeX
Gothenburg, Sweden 2021

Syntax-based Concept Alignment for Machine Translation
Arianna Masciolini
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis presents a syntax-based approach to Concept Alignment (CA), the task of finding semantical correspondences between parts of multilingual parallel texts, with a focus on Machine Translation (MT). Two variants of CA are taken into account: Concept Extraction (CE), whose aim is to identify new concepts my means of mere linguistic comparison, and Concept Propagation (CP), which consists in looking for the translation equivalents of a set of known concepts in a new language. As opposed to standard statistical alignment methods, our approach allows to simultaneously align individual words and multiword expressions (even discontinuous). Since phrase-level alignments are useful to correctly translate idiomatic expressions, this can be beneficial for grammar-based translation pipelines, such as those based on Grammatical Framework (GF), which we use to put our system to the test. This is made possible by the fact that the alignments extracted by our CA model are not correspondences between strings, but rather between grammatical objects. Another advantage of our system with respects to the solutions adopted in statistical MT is that, being essentially rule-based, it performs consistently well even on extremely small amounts of data. Our system does, however, rely on the quality of the analyses of the parallel corpora it is applied to. In order to mitigate the consequences of the lack of robustness of existing GF and, in general, constituency parsers, alignment is performed on the Universal Dependency (UD) trees generated by a neural dependency parser. The resulting concepts are then used, exploiting the similarities between UD and GF, as a starting point for automatically generating a GF lexicon to be used in translation. The tangible fruit of this work is a Haskell library, accompanied by a number of executables offering a user-friendly interface to perform both variants of CA, extraction and propagation, evaluate their results and use them in MT experiments.

Keywords: computational linguistic, machine translation, concept alignment, syntax, dependency parsing, Universal Dependencies, Grammatical Framework

Acknowledgements

I would like to start by thanking my supervisor Aarne Ranta, not only for giving me an opportunity to work on a topic that puts together so many of my interests, but also for the constant dialogue we had throughout the project.

For their time and valuable feedback, I am also grateful to my Examiner Carl-Johan Seger, my opponent Tarik Ala Hadi and Marie-Philippe Gill. With her I had an informal but prolonged exchange of views on our respective thesis projects.

I also want to express my gratitude to the friends who helped me with the translations necessary for one of this thesis' experiments: Michael, Haidar and both Claudios.

Furthermore, I want to thank the Museo Nazionale Etrusco di Villa Giulia for letting me use a reproduction of the Pyrgi Tablets as the cover image for this report. That of the Tablets is an early example of parallel text that, for a series of coincidences, kept coming back to my mind during the six months I dedicated to this thesis. In particular, I am thankful to Alessia Argento for taking the time to look for the image based on my vague description.

There is also a long list of people I would like to mention despite not being directly involved in the project. First of all, I have to thank Claudio once again, both for encouraging me and for putting up with living together during such a busy time (we would say "*per avermi supportato e sopportato*"). I also want to thank my family for supporting my sudden decision to pursue my studies in Sweden. Finally, the list would easily grow too long if I mentioned all the close and distant friends that made my life enjoyable during this time, but I cannot avoid mentioning Herbert, with whom I share a love for languages and Computer Science, and Quelli della B1, the irreplaceable group of people I went through my Bachelor's degree in Perugia with, that still continues being such an important part of my daily life and thoughts.

Arianna Masciolini, Gothenburg, February 2021

Contents

List of Figures

List of Tables

1
Introduction

Concept Alignment (CA) consists in finding semantical correspondences between parts of multilingual parallel texts. Such task, often preliminary to further linguistic analysis, is routinely performed by learners of classical languages when working with a translation alongside the original text. It can - and usually does happen simultaneously - at different degrees of abstraction, ranging from word to sentence level. While in some cases the student identifies new concepts thanks to language comparison itself (*concept extraction*), there are instances where a set of concepts is already known and the objective becomes finding the corresponding expressions in a certain language (*concept propagation*).

Another task that involves CA is natural language translation: the human translator, almost subconsciously, first identifies concepts in the source text and only then looks for ways to render them in the target language. It is then natural to wonder whether it is possible to make use of CA in Machine Translation (MT) as well. The hypothesis motivating this project, whose objective is to develop and test strategies for automating CA, is that it can serve as a step of a compositional MT pipeline. In such case, its users could easily be provided with a way to verify the correctness of the results by examining - at any level - the ways a concept is expressed in different languages instead of having to compare the entire text in the source language to its automatically translated counterparts. From this perspective, CA can help develop a more easily interpretable, and therefore more reliable MT system.

While MT, and in particular Statistical Machine Translation (SMT), does make use of some alignment models, traditional solutions focus on aligning words or, at most, sequences of words, thus generally putting raw strings of text in different languages in relation with each other. This makes it hard to operate at multiple levels of abstraction simultaneously. Nevertheless, there are several reasons why a system able to do that is desirable. First of all, choosing the abstraction level to operate at is not trivial, to the point that one could argue that, even within the particular context of translation, correspondences at different levels of abstraction may be more or less useful according to the specific pair of sentences they occur in. As an example, let us take the following two English-Italian pairs of sentences:

1. "*May I have a piece of cake?*" and "*Potrei avere un pezzo di torta?*"
2. "*Finding useful correspondences isn't exactly a piece of cake*" and "*Trovare corrispondenze utili non è proprio scontato*"

In the first case, correspondences on the word level, like "*piece*"-"*pezzo*" and "*cake*"-"*torta*" are definitely relevant, but in the second sentence pair "*piece of cake*" is used idiomatically and it would probably be more useful to just put the whole phrase in

relation with "*scontato*".

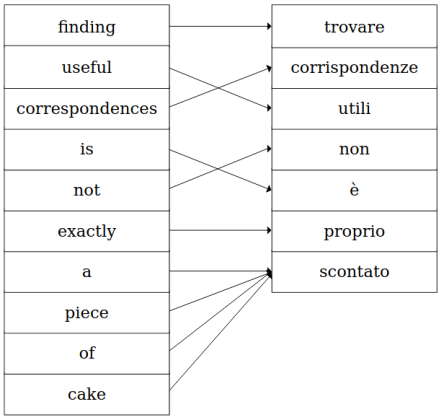


Figure 1.1: An English sentence aligned with its Italian translation. In this case, we are looking for the smallest possible alignments, but it is also possible to find correspondences at a higher level of abstractions, such as "*useful correspondences*" and "*corrispondenze utili*", which puts two noun phrases in relation with each other.

In cases like this, a syntactical comparison between the two sentences can help, if not to extract meaningful correspondences only, which would be the case in the example above, where the complement of the copula "*is*" (resp. "*è*") is a multiword expression in English and a one-word in Italian, at least to obtain a series of alignments at all levels of abstraction from which to select the most relevant at a later stage[1]. The idea is that, in rule-based MT, using aligned phrases (as opposed to aligned words) is often useful for translating idiomatic expressions correctly and, in general, beneficial in terms of fluency of the output sentences.

Furthermore, taking syntax into account allows to easily deal with non-contiguous multiword expressions, another situation which is challenging to deal with by means of traditional statistical approaches. For instance, if we consider the sentence "*Without any linguistic knowledge, it is definitely hard*" and its Italian translation "*Senza conoscenze linguistiche, è decisamente arduo*", a syntactical analysis would make it possible to extract the correspondence "*is hard*"-"*è arduo*", even if in both cases the copula and its complement are separated by an adverb.

With this thesis, we propose a syntax-based approach to both of the above mentioned variants of CA - concept extraction and concept propagation - and put it to the test by integrating it in a prototype domain-specific MT system composed of a neural Universal Dependencies (UD) parser, a rule-based alignment module and a target language generation component based on Grammatical Framework (GF).

[1] an example of a more difficult case could be the alternative translation "Trovare corrispondenze utili non è proprio *un gioco da ragazzi*", roughly corresponding to "*a child's play*" (literally "*a game (played) by children*"). In this case, trying to align the sentence based on a syntactical comparison would yield both "*piece of cake*"-"*gioco da ragazzi*" and the more questionable correspondences "*piece*"-"*gioco*" and "*of cake*-"*da ragazzi*".

Structure of the thesis

This work is structured as follows. Chapter 2 gives a few basic definitions, including a more rigorous one of CA itself, as well as providing the necessary background and contextualizing our project by reviewing a few related works. Chapters 3 and 4 focus respectively on extraction and propagation, presenting both our approach to each task and the corresponding experimental results, while Chapter 5 describes the experiment designed to produce evaluate how well our CA component performs in the context of MT. Finally, Chapter 6 consists of a discussion of the overall results and proposes some ideas for future work.

2

Theory and Technologies

In this chapter, we first review the preliminary notions of Computational Linguistics relevant to this work. We then give a more rigorous definition of the problem at hand and present the basic idea behind our proposed automation approach, comparing it to the existing ones and mentioning the technologies involved.

2.1 Preliminary notions

This first section consists of an overview of the basic notions that are necessary for a full understanding of the approach we propose. First, we discuss the principle of compositionality. After that, we describe and compare the grammar formalisms involved in the project, focussing on the two playing the most prominent roles in this work and comparing them to other, related approaches.

2.1.1 Semantic compositionality

The principle of semantic compositionality states, in its most general form, that the meaning of a complex expression is determined solely by the meanings of its components and by the manner in which these components are combined [40].

A useful variant of this formulation, which refers to natural languages in particular, is the following:

Definition 1 *For every complex expression e in a language L, the meaning of e in L is determined by the structure of e and the meanings of the constituents of e in L.*

Questions of structure and constituency are settled by the syntax of L, while the meanings of simple expressions are given by the lexical semantics of L, meaning that syntax and lexical semantics, together, are sufficient to determine the entire semantics of L [40].

According to this definition, then, if L is compositional, the meaning of an expression e in L cannot depend directly on the context in which S is used in or on the intentions of the speaker who uses it, even though it might be the case that the meanings of the constituents of e depend on the context or on the speaker's intentions, thus making the sentence indirectly depend on them [20].

While many artificial languages, such as programming languages, are compositional by construction, the general validity of this principle for natural languages is, despite many arguments in favor, under debate. In this work, we make the assumption that

compositionality also holds for natural languages, or at least that it is a useful notion applying to most natural language sentences[1].

In particular, we adapt the notion of semantic compositionality to translation. The intuitive idea is that the translation of a sentence is composed of translations of each part of the original one. Adjusting Definition 1:

Definition 2 *For every complex expression e in a source language S, the translation of e in a target language T is determined by the structure of e and the translations of the constituents of e in T.*

2.1.2 Constituency grammars

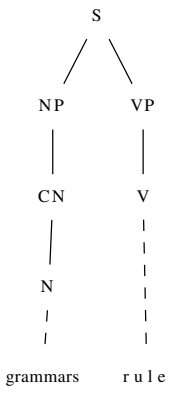
The idea of taking advantage of semantic compositionality is not new in the field of Computational Linguistics. Doing so traditionally involves using some kind of *constituency grammar*.

The notion of constituency or *phrase structure* grammar, introduced by Noam Chomsky in [11], is usually known to computer scientists due to the fact that a class of phrase structure grammars, namely that of *Context-Free Grammars* (CFGs), that can be used to describe programming languages and plays an central role in compiler theory. As a consequence, we find giving a complete formal definition of the latter unnecessary and will restrict ourselves to an informal review of the aspects that are particularly relevant for the specific subject of this thesis.

A constituency grammar consists essentially in a set of *rewrite rules*, such as

```
S → NP VP,
```

which we can read as "S can be *rewritten as* (or *substituted with*) a NP followed by a VP", or as "a sentence S is constituted by a noun phrase NP and a verb phrase VP".




```
S → NP VP
NP → CN
VP → V
CN → N
N → "grammars"
V → "rule"
```

Figure 2.1: Parse tree of the simple sentence "*grammars rule*". On the right, the fragment of the simple but linguistically informed grammar necessary to obtain such tree.

[1] in Section 3.2.3, however, we will see that idiomatic expressions, often used as an arguments against compositionality, are also challenging to deal with in the context of grammar-based CA.

Replacing the term "constituted" with "composed" makes it easy to understand in which way phrase structure grammars are related to the aforementioned principle of compositionality.

In the context of MT, the idea of exploiting the constituency relation and, as such, compositionality, was proposed by Curry in the early 1960s [12] and first put in practice two decades later in the form of an experimental interlingual translation system, not coincidentally named Rosetta, which requires the definition of two distinct logically isomorphic *Montague grammars*[2] - one for the Source Language (SL), one for the Target Language (TL) - and constructs an intermediate representation based on such isomorphism [25].

2.1.2.1 Synchronous grammars

Among constituency grammars, other formalisms that have been employed in more recent MT systems are that of *synchronous CFG*, originally developed for programming language compilation [4] and adapted to natural language translation in several settings [32, 43, 44, 10], and, later on, that of *synchronous TAG* [38], a variation of *TAG* (Tree-Adjoining Grammar) [21]. Both formalisms are meant to characterize correspondences between languages by having as their elements, instead of single rewrite rules, pairs of rules - one for the source and one for the target language. In a synchronous TAG, the constituents of a SL rule may be linked to their counterparts in the corresponding TL rule, and such links may be used to identify concepts in a syntax-based fashion.

2.1.2.2 Grammatical Framework

As the repeated mention of programming language compilers in the above may suggest, it is possible to draw a very close parallel between compiler and MT pipelines. From this perspective, a natural language translation system can consist of:

- a *frontend*, where the SL is analyzed or *parsed* and whose output is an *intermediate representation* most frequently in the form of an *Abstract Syntax Tree* (AST)
- a *backend*, where the AST is *linearized* by means of application of a set of rules, a process usually referred to as *code generation* in the context of programming languages and that we will refer to as Target Language Generation (TLG) or, when the context makes it clear that what the TL is, *Natural Language Generation* (NLG).

[2]Montague grammars are a semantics-oriented development of *categorial grammars*, which are in turn a type of constituency grammars.

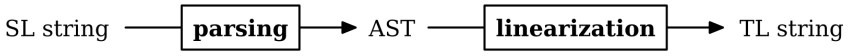


Figure 2.2: Steps of programming language compilation and, in the case of GF, natural language translation. In compilation, the input is usually a high-level programming language and the output is machine code (or code in a lower-level language). In the case of MT, the input and the output are strings in two different natural languages.

Grammatical Framework (GF), being a grammar formalism and programming language designed with this parallel in mind, goes a step further in the direction of synchronous CFGs and TAGs. It introduces in fact a clear distinction between the *abstract syntax*, whose aim is to capture the syntactic structures all natural languages taken into account have in common, and the *concrete syntaxes*, specific to each individual language, consisting in their linearization rules [33] [34]. This makes it possible to deal with multiple languages by writing only one grammar, whose components are an abstract syntax and several concrete syntaxes, providing a solid basis for a system able to translate between any pair of for which a concrete syntax is available, in any direction.

With respects to this, GF can be also seen as an *interlingual* MT system where the intermediate representation or *interlingua* is an AST. Interlingua-based systems have the advantage, in terms of efficiency, of making it unnecessary to build $n(n-1)$ translation functions to cover all possible pairs of n languages: having $2n$ is in general enough. In GF in particular, using a multilingual grammar makes it so that translation can be performed, for each pair of languages, in both directions, so that n translation functions (or rahter n concrete syntaxes and an abstract syntax) are already sufficient.

An advantage of GF in particular, in addition, is the availability of a set of *resource grammars* - the Resource Grammar Library (RGL) - for a variety of languages. A resource grammar is essentially a grammar that captures only the syntactic and morphological structures of a language, i.e. a grammar in the traditional sense of the term, and that can be easily extended to construct what is usually referred to as an *application grammar*, i.e. a domain-specific grammar aiming to describe the language in a way that is not only syntactically correct, but also semantically accurate [34].

Extending the RGL in various ways, translation experiments with GF grammars have been conducted in the field of eXplainable Machine Translation (XMT), as the ASTs produced by SL analysis can serve both as to some extent automatically checkable certificates for the correctness of translation - by backlinearization to the SL - and as fully inspectable explanations aimed towards expert users [35].

While the approach described above has proved successful for domain-specific translation, i.e. in cases where the natural languages in question can be reduced to *Constrained Natural Languages* (CNLs), when it comes to open-domain translation, while target language generation remains effective, the results are negatively affected by the lack of robustness of the GF-based parsers available at the time of writing.

2.1.3 Dependency grammars

A class of grammar formalisms alternative to phrase structure, first proposed in [41], is that of *dependency grammars*, the main difference between the two being the relation they are based on. As opposed to constituency, *dependency* is a word-to-word correspondence, meaning that words are simply put in relation with each other via directed links, called in fact *dependencies*. Each dependency is, then, composed of two words: a *head* and a *dependent* that refers to it. For instance, if we try to look at syntactic dependencies in the sentence "*grammars rule*", we could identify the verb "*rule*" as the head and the noun "*grammars*", its subject, as its dependent.

Intuitively, this means that dependency trees are simpler than their phrase structure counterparts. This makes them an easier target for the frontend, possibly ML-based, of a MT system such as the one outlined in the above. Existing *dependency parsers*, such as UDPipe [39] and the Standford parser [9], are often - but not always [6] - neural pipelines trained on dependency treebanks, significantly more robust than their phrase structure counterparts.
On the other hand, there is currently no effective way to use these trees as a starting point for NLG: dependency grammars are an effective way to describe language, but, unlike phrase-structure grammars, they are not *generative*,[3].

One idea is, then, to develop a hybrid system where the frontend is a dependency parser and the backend a grammaticality-preserving GF-based target language generation module. This work goes in this direction, and the connecting link between these two seemingly incompatible stages of the pipeline is, as we will elaborate on in Section 2.2.3.2, a CA component able to identify matching dependency trees and generate the corresponding GF concrete and abstract syntax functions: concepts.
While there are several different dependency-based frameworks, the following section focuses on the one we find most well suited to this purpose, *Universal Dependencies*.

2.1.3.1 Universal Dependencies

Universal Dependencies (UD) is a framework for cross-linguistically consistent grammatical annotation. The UD project aims at developing parallel treebanks for many languages in order to support, among other things, the development of multilingual dependency parsers, such as the aforementioned UDPipe [39]. In order to do so, it specifies an annotation scheme and a standard format for dependency trees consisting in an evolution of (universal) Stanford dependencies [14, 13], Google universal part-of-speech tags [30], and the Interset interlingua for morphosyntactic tagsets [45]. The basic idea behind such standard is to provide a set of Part-Of-Speech (POS) tags, dependency relations and annotation guidelines as language-agnostic as possible - so to facilitate its application in multilingual settings - while allowing language-specific extensions whenever necessary.

In the following, we give a quick overview of the standard format UD uses to store

[3] that does not mean, however, that dependency grammars have not been made use of in MT research, in conjunction with statistical techniques, in order to better capture grammatical generalizations [31].

dependency trees and of the aspects of the annotation scheme that are most relevant to the task at hand. Appendix A provides a more comprehensive, but not completely exhaustive, description of all the POS tags and dependency labels mentioned in the examples appearing in this work based on that in [36]. The reader interested in a full specification of the UD annotation scheme may refer to the official UD documentation[4].

2.1.3.1.1 The CoNLL-U format

The standard plain text format for dependency trees, CoNLL-U, is an extension of CoNNL-X [8] which may contain *comment lines*, starting with `#`, *blank lines* marking sentence boundaries and *word lines* containing the annotation of a single token (generally a word, with some exceptions[5] in 10 tab-separated fields:

1. `ID`: integer[6] representing the position of the word in the sentence, starting from 1
2. `FORM`: inflected from of the word
3. `LEMMA`: lemmatized form of the word
4. `UPOS`: universal POS tag
5. `XPOS`: language-specific POS tag (optional)
6. `FEATS`: list of (universal or language-specific) morphological features (optional)
7. `HEAD`: head of the current word, i.e. either the value of the `ID` of another word in the same sentence or 0 in case the word at hand is a `root`
8. `DEPREL`: universal dependency relation to the `HEAD` (`root` in case the word at hand is itself the `root`)
9. `DEPS`: enhanced dependency graph in the form of a list of `HEAD-DEPREL` pairs (optional)
10. `MISC`: any other annotation (optional)

```
1 this this PRON _ _ 5 nsubj _ _
2 is be AUX _ _ 5 cop _ _
3 a a DET _ _ 5 det _ _
4 dependency dependency NOUN _ _ 5 compound
_ _
5 tree tree NOUN _ _ 0 root _ _
```

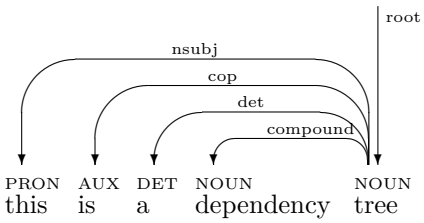


Figure 2.3: A dependency tree in CoNNL-U format alongside its graphical representation. Optional fields in the CoNLL-U file are left blank.

2.1.3.1.2 Universal POS tags

Universal POS tags mark the core part-of-speech categories, such as nouns, verbs, pronouns and determiners.

[4] available at `universaldependencies.org`.

[5] examples of such exceptions are Italian contractions such as "*dello*", which is generally divided into "*del*" + "*lo*".

[6] again, there are exceptions: in the case of Italian contractions and other multiword tokens, ranges may be used. Furthermore, empty nodes are characterized by decimal numbers between 0 and 1.

An important distinction we can make based on POS tags is, as we will see in Section 3.1.2.1, that between *content* and *function* words. *Content* or *open class* words are words with a lexical meaning (nouns, lexical verbs, adjectives, adverbs and interjections). *Function* words, like pronouns and determiners, only have a grammatical meaning. Since they do not readily accept new members[7], they are often referred to also as *closed class* words.

While the majority of Universal POS tags correspond to the grammatical categories of traditional grammars, the UD annotation scheme does have its peculiarities. Most importantly for the following discussion, verbs are divided into lexical verbs, tagged `VERB`, and auxiliaries, tagged `AUX`, thus providing an easy way to distinguish between content verbs and function verbs. A more systematic description of `UPOS` tags is given in A.1.

2.1.3.1.3 Universal dependency relations

Universal dependency relations, largely based on [13], represent, as mentioned in the above, syntactic dependencies between individual pairs of words occurring in the same sentence. In particular, according to the CoNLL-U standard (cf. Section 2.1.3.1.1), each word is assigned a *dependency label* indicating in which way it is liked to its `HEAD`.

In this sense, the only exceptional case, which will be the starting point for our short overview of dependency relations, is that of the `root` label, generally assigned to the main verb of a sentence, ignoring any auxiliaries. For instance, "*smoked*" is the root of the sentence

Example 1 *"Katia has just smoked a cigarette"*

In cases such as the following, where the main verb is a copula, the root is its complement and the verb is linked to it with the label `cop`.

Example 2 *"Katia is a psychologist"*

If a sentence contains is no verbs at all, there is no fixed rule excepts that, in order to avoid unnecessary discrepancies between languages, the root should be a content word.

Other dependency labels commonly found in simple clauses and that will be dwelt on in Chapter 3, all exemplified in Figure 2.4, are:

- `nsubj`, marking the link between a noun, proper noun, pronoun or numeral to the `root` of a sentence or, more in general, to the head of a clause
- `aux`, marking auxiliary verbs other than the copula
- `obj`, `iobj` and `obl` marking the link between a verb and its object, indirect object and other complements respectively
- `advmod`, `amod`, `nummod` and `nmod`, marking links between modifiers and the nouns or verbs they refer to
- `flat`, indicating a flat multiword expression, and `compound`, appearing in compound nouns whenever they are written as two or more separate words.

[7] one exception is the recent introduction of the gender-neutral pronoun "*hen*" in Swedish.

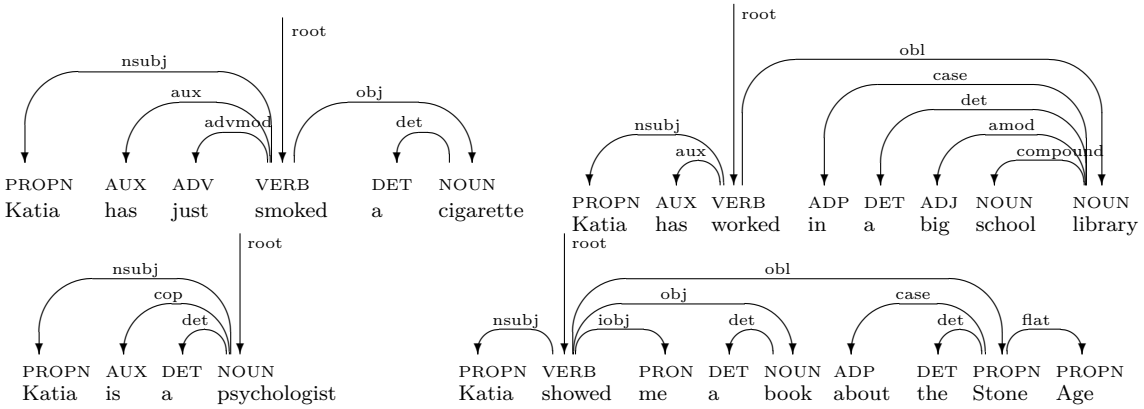


Figure 2.4: UD trees showing the common dependency relations that are discussed in Chapter 3. The three on the left correspond to examples 1 and 2.

A more detailed description of all the UD labels mentioned in this work is given in A.2. However, because the topic will arise in Chapter 3, it is also worth mentioning that CoNNL-U trees can present UD labels followed by a *subtype*, separated by the label itself by a semicolon and used to indicate grammatical relations that are specific to one language or a small group of related languages. A subtype that is commonly used in English is, for instance, `pass`, added to both the (clausal or nominal) syntactic subject and `aux` of a sentence in passive voice.

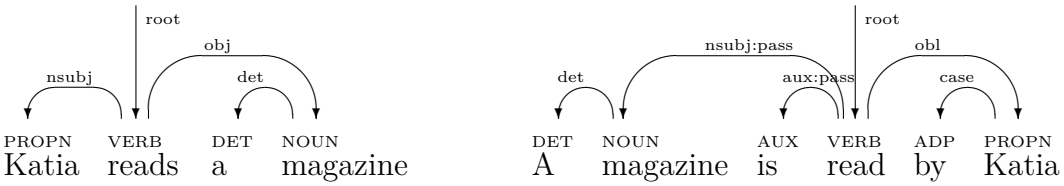


Figure 2.5: The UD trees of an example active sentence and its passive counterpart.

2.1.3.1.4 Relation to GF

The complementarity of constituency and dependency grammars and the multilingual nature of UD make it interesting to use in conjunction with GF, and experiments aimed at exploiting their similarities have already been performed [24, 37]. In the case of hybrid MT pipelines as the one sketched in Section 2.1.3, UD trees need to be at some point converted into GF ASTs, albeit with the disadvantage that, unlike its reverse, the UD-to-GF conversion is a non-deterministic search problem. An algorithm for conversion in this direction is presented in [37], where `gf-ud`, a program for converting GF ASTs into UD trees initially presented in [24], is extended to also be able to do the reverse.

2.2 Concept Alignment

In this section, we aim to give an exhaustive description of CA and of the subtasks it consists of. In order to do that, we deem it necessary to start by giving a definition

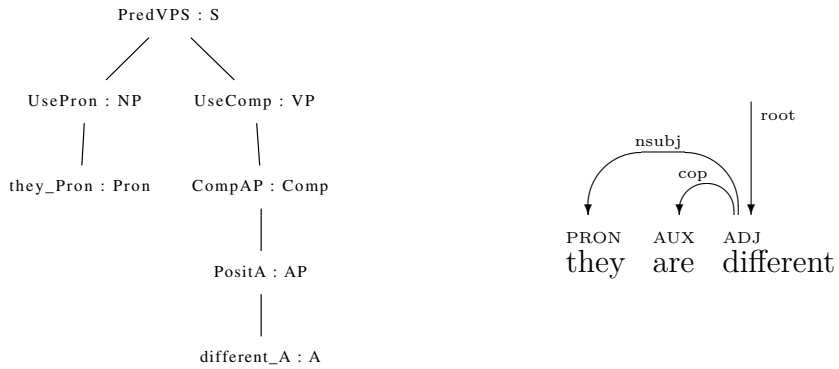


Figure 2.6: A GF AST and its UD counterpart. Note how the copula is implicit in the GF AST and subordinate to its complement in UD.

of what we commonly refer to as *concepts*.

2.2.1 Concepts

Intuitively, concepts are the components of meaning, and therefore, in a multilingual context, the units of translation. If we assume the principle of compositionality to be valid and apply it to translation, these meaning components are the common denominator between an expression in the source language and its translation, which can be generated using them as the starting point.

From the compiler-like perspective described in Section 2.1.2.2, this means that concepts are what the abstract syntax should represent, i.e. that they *are* the "interlingua" of the MT system. This raises the question of how to obtain the abstract syntax needed for GF-based MT starting from a corpus of parallel texts, and it is at this point that language comparison and, as such, CA, come into play.

2.2.2 Alignments

In terms of parallel text analysis, an *alignment* consists in a pair of semantically equivalent (i.e., in this context, sharing the same abstract syntax) concrete expressions, one in the source and one in the target language. Even though, unless otherwise specified, we will keep referring to *language pairs*, with a source and a target language, it is easy to see how this can be generalized to a more-than-bilingual, multidirectional definition. The pair just mentioned simply becomes a tuple of equivalent expressions in different languages. Formally,

Definition 3 *An n-lingual alignment is an n-uple* $\langle e_1, ..., e_n \rangle$ *of semantically equivalent expressions, where each expressions* e_i *is in a different language* L_i*.*

While this general definition does not specify it, expressions are not necessarily represented as strings, but can rather be defined as tree-like structures, and in the present case as dependency trees that can later be replaced by GF ASTs. Matching ASTs can finally be used to generate the rules of a multilingual GF grammar, as we will discuss in Chapter 5.

As mentioned in the introduction, the task of aligning concepts comes in two variants:

1. ***Concept Extraction*** (CE), which consists in identifying new concepts via linguistic comparison and whose output is a set of bilingual alignments
2. ***Concept Propagation*** (CP), i.e. the task of finding the concrete expressions corresponding to a set of known concepts in a particular language. Here, by concepts we do not mean necessarily ASTs, as a suitable approach is to propagate UD tree alignments before obtaining the corresponding abstract representation.

If, as in the case of this project, the aim is to develop a multilingual MT system, these two tasks can be seen as two potentially subsequent steps. A first objective, in fact, can be to extract a set of concepts by comparing two translations of the same text. Once these concepts are in adequate number and of sufficient quality, it becomes possible to provide support for additional languages by simply looking for the concrete expressions that, in the new language, correspond to each of the previously gathered concepts. In other situations, clearly, CE alone can be enough or, if a set of concepts is already known, CP can be applied independently from CE to find their equivalents in a new language.

2.2.3 Approaches to automation

Some form of manual, semi- or fully automated CA is in a sense at the heart of all traditional, even very early, MT systems. In the following section, we review some standard approaches. After discussing their limitations, we conclude the chapter with an overview of the grammar-based approach proposed in this thesis.

2.2.3.1 Existing methods

In the simplest case, *word alignment* consists in finding pairs of individual words that translate to each other and store them in a dictionary. The standard automatic approaches to this task are statistical, and among them the five IBM Models [7] and their numerous variations stand out. The IBM Models are a sequence of increasingly complex alignment models [42]:

- Model 1 is based exclusively on lexical translation probabilities
- Model 2 takes word order (i.e. the absolute word positions) into account
- Model 3 takes even a *fertility* parameter into account (fertility being the number of target words that can be generated from a given source word)
- Models 4 and 5 take the *context* (i.e. the relative position of the words) in which each word occurs into account in a broader sense

The GIZA++ toolkit [28], an open source implementation of the IBM models based on the initial GIZA package [5], is widely used, for instance in SMT systems such as Moses [23].

As mentioned in the introduction, however, alignment can be performed at different levels of abstraction: to bring this to the extreme, there are contexts in which we might want to align full sentences, or even whole documents. While the latter two levels of abstraction are not necessarily relevant to MT, only aligning individual

words is also, as discussed in the Introduction, not always the best option. For this reason, word alignment is often generalized to *phrase alignment*. In systems like GIZA++, whose output is a list of pairs of word positions indicating which words in the SL string are to be aligned with which words in the TL string (commonly referred to as the *pharaoh format*), extracting phrase alignments by combining the indices is realtively straightforward.

SL "*Word alignment with standard statistical tools*"

TL "*Allineamento parola per parola con strumenti statistici standar*"

`0-1 0-2 0-3 1-0 2-4 3-7 4-6 5-5`

- `0-1 0-2 0-3`: ⟨*"word", "parola per parola"*⟩
- `1-0`: ⟨*"alignment", "allineamento"*⟩
- `2-4`: ⟨*"with", "con"*⟩
- `3-7`: ⟨*"standard", "standard"*⟩
- `4-6`: ⟨*"statistical", "statistici"*⟩
- `5-5`: ⟨*"tools", "strumenti"*⟩
- `0-1 0-2 0-3 1-0`: ⟨*"word alignment", "allineamento parola per parola"*⟩
- ...

...

Figure 2.7: The pharaoh format output of optimal word alignment on a pair of Italian-English sentences and, below, some of the phrase alignments that can be derived from it.

However, it must be noted that, in the context of statistical MT, the term "phrase" is not to be intended in its grammatical sense, but can refer to any sequence of - typically contiguous - words. In any case, both for word and phrase alignment, the relation that is established with these statistical methods is merely between strings in different languages: there is no intermediate representation capturing the concepts themselves.

The parallel between MT and compiler pipelines proposed in Section 2.1.2.2 suggests instead that a more flexible approach, applicable at any level of abstraction, could be grammar-based. This is particularly useful in cases, not at all uncommon, where the minimal translation units are, in one or both the source and the target language, multiword - potentially discontinuous - expressions or even more complex constructions that phrase alignment is not able to handle.

Over time, various constituency grammar-based approaches have been proposed [27, 22]. These approaches, which make use of parallel treebanks, can be referred to as *tree-to-tree alignment* methods [42]. While theoretically appealing, however, their use in data-driven MT is limited, since they tend to suffer not only from the scarce availability of large-scale manually annotated corpora and the lack of robustness of the existing parsers, but also from the fact that, in general, the monolingual grammars used for parsing tend to be designed independently from each other, following different traditions, formalisms and linguistic theories [19], thus making it

often impossible to find a common representation describing a complete mapping from one tree to another [42].

2.2.3.2 Our approach

Given that GF solves the latter problem, that it provides a good basis for NLG and that the intermediate representations it makes use of preserve all the syntactic information present in a sentence, the idea of trying to perform CA by comparing GF ASTs seems tempting. However, since ASTs need to be obtained from raw - or, at most, barely sentence-segmented - text, CA would suffer from the same problem that, as mentioned in in Section 2.1.2.2, affects other tree-to-tree alignment methods and GF-based open-domain translation: the lack of a sufficiently robust analysis stage and, as a consequence, the inadequacy of the resulting parse trees.

As anticipated throughout this chapter, we attempt to solve this problem by taking advantage of dependency parsing. In particular, UD is our formalism of choice since its focus on "universality" (i.e. on abstracting away from cross-lingual differences) makes it as interesting as GF itself when it comes to the possibility of establishing mappings between trees in different languages. Furthermore, when it comes to MT itself, using UD allows us to take advantage of the tools that have been developed to leverage its similarities with GF (cf. Section 2.1.3.1), thus enabling GF target language generation.

Concretely, this means that the system we propose requires the following elements, whose reciprocal relations are shown in Figure 2.8:

- a UD parser
- an alignment module based on dependency tree comparison, which is the tangible output of this project and whose CE and CP components are described and evaluated in detail in Chapters 3 and 4 respectively
- a program, based on `gf-ud`, that converts the alignments into GF ASTs to then generate a domain-specific GF lexicon[8].

The CE and CP modules are implemented as part of the project, while the UD parser and `gf-ud` are independently developed open source software, even though the latter, being currently under development was adapted and extended as part of the process. In addition, we provide a simple GF-based translation script to evaluate the performance of the CA module to the test.

[8]CA is performed on UD trees, i.e. before conversion to GF, to reduce the possibility of errors. The reason is that, as mentioned in Section 2.1.3.1.4, in fact, such conversion is not obtained by means of a deterministic procedure and, as such, is not always guaranteed to be correct.

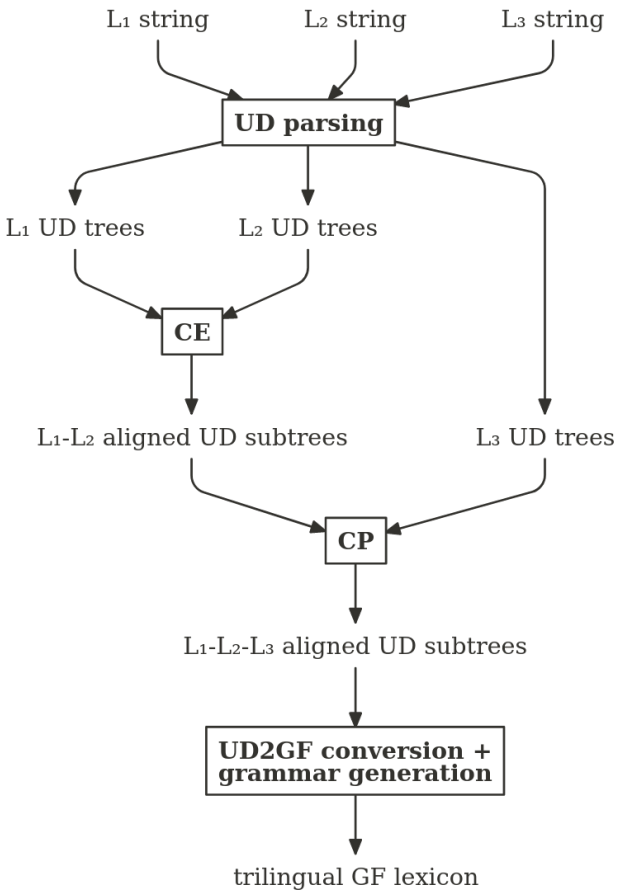


Figure 2.8: Relationship between the different components of the system proposed in this work. The diagram shows the minimal, trilingual use case in which both CE and CP can be made use of.

3

Concept Extraction

As mentioned in Section 2.2.2, extracting a set of concepts via linguistic comparison is the first and most important step of CA. This chapter describes our CE module, delineates our strategy to evaluate it independently from the other components of the system illustrated in Figure 2.8 and presents the results of such preliminary evaluation.

3.1 Method and implementation

For the reasons discussed in Section 2.2.3.2, our syntax-based approach to CE is based on comparing, instead on ASTs, dependency trees, and UD trees in particular. The task of dealing with UD trees is facilitated by the existence of a series of Haskell modules written in the context of the development of the aforementioned `gf-ud` and of some preliminary unpublished CA experiments[1].

In this context, as shown in Figure 3.2, dependency trees are represented as rose trees, i.e. tree data structures with an unbounded, variable number of branches per node.

[1] Aarne Ranta and Prasanth Kolachina, code available at `https://github.com/aarneranta/concept-alignment/tree/master/old_ca_versions/v1`

```
1 another another DET _ _ 3 det _ _
2 dependency dependency NOUN _ _ 3 compound _ _
3 tree tree NOUN _ _ 0 root _ _


RTree {
 root = UDWord {
  udID = 3, udFORM = "tree", udLEMMA = "tree", udUPOS = "NOUN", (...), udHEAD = 0,
  udDEPREL = "root", (...)},
 subtrees = [
  RTree {
  root = UDWord {
   udID = 1, udFORM = "another", udLEMMA = "another", udUPOS = "DET", (...),
   udHEAD = 3, udDEPREL = "det", (...)},
  subtrees = []},
  RTree {
  root = UDWord {
   udID = 2, udFORM = "dependency", udLEMMA = "dependency", udUPOS = "NOUN", (...),
   udHEAD = 3, udDEPREL = "compound", (...)},
  subtrees = []}]}


3 tree tree NOUN _ _ 0 root _ _
 1 another another DET _ _ 3 det _ _
 2 dependency dependency NOUN _ _ 3 compound _ _
```

Figure 3.1: The CoNLL-U and rose tree representation of a dependency tree. In the rose tree in the middle, fields left black in the CoNLL-U representation are omitted for compactness. The third representation, easier to read, resembles CoNNL-U notation while graphically showing the structure of the tree. This notation will be used for the other examples of this chapter.

As shown in Figure 3.2, the type of the nodes of such trees, `UDWord`, is a record type whose fields mirror those of a CoNLL-U file (cf. 2.1.3.1.1). Alignments are, as for Definition 3, simply pairs of UD trees.

```
data UDWord = UDWord {
  udID      :: Int,
  udFORM    :: String,
  udLEMMA   :: String,
  udUPOS    :: String,
  udXPOS    :: String,
  udFEATS   :: [String],
  udHEAD    :: Int,
  udDEPREL  :: String,
  udDEPS    :: String,
  udMISC    :: [String]
  }

data RTree n = RTree n [RTree n]

type UDTree = RTree UDWord

type Alignment = (UDTree,UDTree)
```

Figure 3.2: Fundamental data types used for representing dependency trees in the CA module.

3.1.1 Baseline

The original CE algorithm, dating back to the above mentioned CA experiments, has two stages, sketched respectively in Figure 3.3 and 3.4. The first consists in, given two trees corresponding to a sentence and its translation, aligning the entire trees by recursively sorting and padding their lists of subtrees. In particular:

- sorting is based on the UD label of their root
- what is meant by *padding* is the insertion of a dummy dependency subtree wherever a tree in the source (resp. target) language has a subtree with UD label l that is missing in its target (resp. source) language counterpart. This is useful, for instance, for dealing with cases where a determiner in a sentence in the SL is omitted in its translation.

The result of this first step is a pair of *perfectly aligned* trees, i.e. trees with identical shape that can later be used to extract the pairs of aligned subtrees.

```
align :: (UDTree,UDTree) → Alignment
align (RTree n ts,RTree m us) = (RTree n pts, RTree m pus)
   where
      (pts,pus) = unzip [align (t,u) |
      (t,u) <- padSubtrees (sortSubtrees ts) (sortSubtrees us)]
         where
            root (RTree n _) = n
            sortSubtrees = sortOn (udDEPREL . root)
            padSubtrees xs ys = case (xs,ys) of
               (x:xs', y:ys')
                  | k x == k y → (x,y):padSubtrees xs' ys'
                  | k x <  k y → (x,d):padSubtrees xs' ys
                  | k x >  k y → (d,y):padSubtrees xs  ys'
               (_,[]) → [(x,d) | x <- xs]
               ([],_) → [(d,y) | y <- ys]
               _ → []
               where
                  d = padUDNode -- dummy node used for padding
                  -- padding key to compare subtrees
                  k t = (udDEPREL (root t),
                        distanceDepNode (root t))
```

Figure 3.3: The alignment step of the basic CE algorithm. Given two full sentence trees, it aligns them by sorting and padding.

Once two sentence trees are aligned, extracting pairs of aligned subtrees is just a matter of recursively obtaining the lists of all subtrees of each sentence and zipping them together. Every time a pair of subtrees with depth greater than 1 is extracted, in addition, a new pair of subtrees whose only nodes are their roots is added. This makes it possible to find single-word alignments, which we refer to as *head alignments*, that would otherwise be ignored.

```
extract :: Alignment → [Alignment]
extract (t,u) = zip (subts' t) (subts' u)
    where
        subts' t = let ts = subts t in ts ++ map headt ts
            where
                subts t = t:concatMap subts (immediateSubts t)
                    where immediateSubts (RTree _ ts) = ts
                headt (RTree n _) = RTree n []
```

Figure 3.4: The extraction step of the basic CE algorithm. Given two perfectly aligned trees, it returns a list of pairs of aligned subtrees. To do that, it extracts all subtrees (at any depth, cf. `subst'`) of each member of the sentence alignment. For each extracted subtree, a new tree whose only node is its root is also created (cf. `headt`).

After extracting subtree alignments, it is possible and generally useful to remove the dummy subtrees added during alignment they contain.

Obviously, the "full-sentence" tree pair is itself an alignment, even though a trivial one, as it represents a sentence-level correspondence in a case where, due to the approach being syntactic comparison, the inputs are assumed to be sentences that correspond to each other[2].

[2]contrary to what it may seem, however, sentence-level alignment is in general not at a trivial task, as it is often the case, especially in certain language pairs, that one *orthographic* sentence, i.e. a sentence defined based on the presence of a full stop, maps to more than one orthographic sentences in the translated text, or vice versa.

```
4 aligned aligned ADJ _ _ 0 root _ _
 1 we we PRON _ _ 4 nsubj _ _
 2 are be AUX _ _ 4 cop _ _
 3 perfectly perfectly ADV _ _ 4 advmod _ _
```

```
3 allineati allineato ADJ _ _ 0 root _ _
 1 siamo essere AUX _ _ 3 cop _ _
 2 perfettamente perfettamente ADV _ _ 3 advmod _ _
```

⇓ sorting

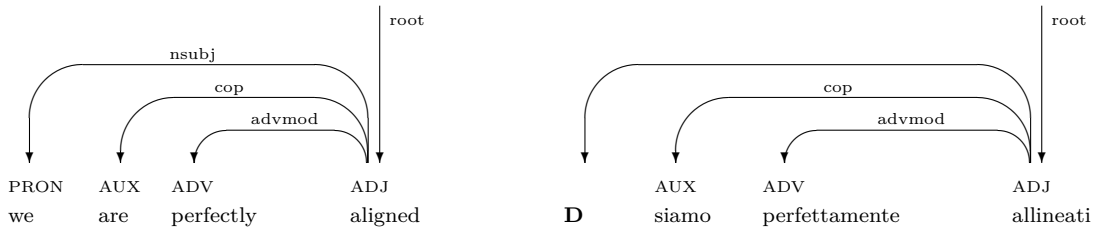
```
4 aligned aligned ADJ _ _ 0 root _ _
 3 perfectly perfectly ADV _ _ 4 advmod _ _
 2 are be AUX _ _ 4 cop _ _
 1 we we PRON _ _ 4 nsubj _ _
```

```
3 allineati allineato ADJ _ _ 0 root _ _
 2 perfettamente perfettamente ADV _ _ 3 advmod _ _
 1 siamo essere AUX _ _ 3 cop _ _
```

⇓ padding

```
4 aligned aligned ADJ _ _ 0 root _ _
 3 perfectly perfectly ADV _ _ 4 advmod _ _
 2 are be AUX _ _ 4 cop _ _
 1 we we PRON _ _ 4 nsubj _ _
```

```
3 allineati allineato ADJ _ _ 0 root _ _
 2 perfettamente perfettamente ADV _ _ 3 advmod _ _
 1 siamo essere AUX _ _ 3 cop _ _
 D
```



- ⟨*we are perfectly aligned, siamo perfettamente allineati*⟩
- ⟨*aligned,allineati*⟩ (head alignment)
- ⟨*are*⟩
- ⟨*siamo*⟩
- ⟨*perfectly,perfettamente*⟩
- ⟨*aligned,allineati*⟩

Figure 3.8: Steps of the baseline CE algorithm. First, the rose tree representations of two yet-to-align dependency trees. Just below, the two rose trees after sorting and padding. Finally, the graphical representations of the resulting perfectly aligned trees and the linearizations of the aligned subtrees extracted from them. Since, in Italian, the subject is implicit, a dummy node, marked as **D**, is added to the corresponding tree during the padding step.

As we will see in the following sections, such algorithm is useful both as a baseline and as a source of inspiration for our improved CE program.

3.1.2 Proposed improvements

While one of the most important ideas in the improved version of the CE module proposed in this thesis remains aligning (sub)trees with identical root UD labels, using this criterion alone makes our baseline both unable to detect many of the existing correspondences and prone to extract incorrect ones. This happens both when syntactic similarity is only apparent and when words are aligned based exclusively on the fact that they are the roots of two aligned subtrees[3]. As a consequence, the objective of this part of the project is to improve both the precision and the recall of the algorithm (or rather, for reasons that will be discussed in Section 3.2.2, two approximations of such metrics).

3.1.2.1 Multiple alignment criteria

One obvious way to increase the total number of alignments the algorithm detects is to, instead of considering as aligned only subtrees in *matching contexts*, i.e. whose heads are attached to the same node, and with matching UD labels, make use of a set of additional, possibly more relaxed, criteria. In the following, formal definitions of all the criteria used in the current implementation, including the original one, are given.

3.1.2.1.1 Label matching

As mentioned in section 3.1.1, the only alignment criterion the original version of the CE module makes use of is based on comparing the UD labels of each pair of trees candidate for alignment, i.e. of each pair of trees in matching contexts. Formally:

Criterion 1 (Matching UD labels) *Two dependency trees t, u will be aligned if their roots share the same UD label.*

When referring to UD labels we disregard, unless otherwise specified, subtypes.

3.1.2.1.2 POS-equivalence

As mentioned in Section 2.1.3.1, dependency trees provide information not only on the syntactic role of each word, but also on their grammatical category, represented as a universal Part Of Speech (POS) tag. Intuitively, if the words corresponding to the nodes of two trees in matching contexts have the same POS tags, the two trees are generally more likely to correspond to each other than if not. As a consequence, a useful relation to define between dependency trees is that of *POS-equivalence*:

Criterion 2 (POS-equivalence) *Two dependency trees t, u are POS-equivalent (and, as such, will be aligned) if $M_1 = M_2 \neq \emptyset$, where M_i is defined as the multiset of POS tags of all the meaning-carrying word nodes of t_i.*

The definition specifies that the two multisets should not contain the POS tags of all the words in the sentence but rather only those of the *meaning-carrying* ones,

[3] see Section 3.1.2.2.2 for a more detail discussion on head alignment.

referring to words belonging to a particular set of classes. Words that should generally be taken into account, in fact, correspond roughly to content words (cf. Section 2.1.3.1.2), but this term was deliberately avoided in Definition 2 due to the fact that it can be useful also to include some function words, for instance pronouns and some kinds of determiners. In particular, the current implementation considers as meaning-carrying all words that belong to an open class - defined as in the UD documentation [3] - and numerals, but this set of tags has been obtained empirically working with English-Italian and might not be ideal for all language pairs. On the other hand, words belonging to other classes, such as auxiliary verbs, adpositions and conjunctions can and should be in most cases ignored as they are often omitted or rendered with words with different POS tags when the sentence is translated to another language, especially if the two languages in the pair at hand differ significantly.

Applied alone, this criterion can be used to capture correspondences that would otherwise be missed, thus increasing recall, but a decrease in precision is also to be expected. Perhaps more interestingly, however, another way to apply this criterion is in conjunction with other ones, and in particular together with UD label matching (cf. Criterion 1), in context where high precision is more important than recall. We will get back to combining criteria in Section 3.2.4.

3.1.2.1.3 Handling divergences

While we do not want to make our CE module language pair-specific, there are many cases where parallel texts present significant and systematic cross-linguistic distinctions. Some of these distinctions, formalized in [15], have nothing to do with idiomatic usage or aspectual, discourse, domain or word knowledge and are not specific of particular language pairs, even though they do occur more often in some than they do in others. Drawing inspiration from [15] and [18], we refer to these distinctions as *divergences* and introduce a third alignment criterion based on them:

Criterion 3 (Known divergence) *Two dependency trees t, u will be aligned if they match a known divergence pattern.*

In [15], seven classes of divergences are identified: *thematic*, *promotional*, *demotional*, *structural*, *conflational*, *categorial* and *lexical*. However, because the author's proposed way to resolve divergences assumes the availability of more than merely syntactic information[4] and because, working with UD trees, we do not have access to anything but strictly syntactical and morphological annotations, the very task of identifying these distinctions at this level of granularity becomes, in our position, extremely challenging. Consequently, we refer to the simpler classification and less formal definitions proposed in [18], where promotional and demotional divergences are merged in a wider-coverage category of *head-swapping* divergences and where lexical divergences, the hardest to handle in the absence of any kind of semantic information, do not appear at all. In the following, we give definitions and examples

[4] in particular, of a lexicon of Root Lexical Conceptual Structures specifying, for instance, the logical subject, arguments and modifiers of each verb.

for each of these classes of divergences, specifying to what extent and how[5] each of them is handled in the proposed language-agnostic CE module. We do not attempt to cover all possible cases, but rather provide a few examples as a proof of concept.

Categorial divergence A categorial divergence happens when the translation of a word with POS tag P_1 is a word with a different POS tag P_2. The instances of this class of divergences our CE module handles explicitly are some of those that cannot be captured by Criterion 1 alone due to the fact that the difference in POS tags also causes the UD labels to be different, and in particular:

- cases where an adverb in the source language corresponds to an adjective in the target language (and vice versa: all rules the program is based on are symmetrical), e.g.

  Example 3 *"Roberta listens distractedly" VS "Roberta lyssnar distraherad"*

  where the English "*distractedly*" is an adverb and the Swedish "*distraherad*" is an adjective and, as such, would be labelled respectively as an `advmod` of "*lyssnar*" and as an `amod` of "*Roberta*"
- cases where a nominal modifier (`nmod`) in the source language is rendered as an adjectival modifier (`amod`) in the target language, like the following, where the English adjective "*doctoral*" becomes the Italian noun "*dottorato*", preceded by the preposition "*di*":

  Example 4 *"Herbert completed his doctoral thesis" VS "Herbert ha completato la sua tesi di dottorato"*

- cases where an adverb is rendered as an oblique, such as

  Example 5 *"Nicola studies consistently" VS "Nicola studia con costanza"*

There are indeed other divergences of this kind that can occur in a parallel text and cannot be captured by Criterion 1, such as

Example 6 *"Claudio is hungry" VS "Claudio ha fame"*

where, in the original sentence, the complement of the copula *am*, *hungry*, would be labelled as its `root` but, in Italian, the noun *fame* is the object of the root verb *ha*. It proved hard to detect cases like this via purely syntactic rules without causing the program to extract a lot of incorrect alignments as well, but if necessary it is easy to modify the program by removing or adding rules of this kind, potentially even language-specific.

Conflational divergence Conflational divergences involve the translation of two or more words in the source language using a single word that combines their meanings in the target language. A straightforward example is that of compounds, for instance

Example 7 *"Filippo is interested in game design" VS "Filippo är intresserad av spelutvekling"*

[5] unless otherwise specified, each divergence can be expressed as a set of simple rules in the form of boolean functions taking the two UD trees candidate for alignment as input.

but there are also other cases, like

Example 8 *"I'll come and say hello to Bruno and Andrea" VS "Verrò a salutare Bruno e Andrea"*

where the single word *salutare* is not a compound but expresses the same meaning as "say" and "hi" together. Divergences like the one described in the latter example are usually captured by Criterion 1, while compounds often need to be taken care of explicitly. For reasons that will become clear later, these cases are discussed in Section 3.1.2.2.2.

Structural divergence Structural divergences happen when the subject, object or indirect object of a sentence in the source language are rendered as obliques in the source language. For instance, in the sentences

Example 9 *"I called Francesco" VS "Ho telefonato a Francesco"*

what is the direct object in English (*Francesco*) becomes a prepositional phrase (*a Francesco*) in Italian. Even though in many cases these divergences can be captured by POS-equivalence, in order to ensure higher precision, the CE module handles all of them explicitly via rules that combine it with UD label checking and whose priority is higher than that of the basic Criterion 2.

Head swapping divergences Head swapping divergences always involve a head verb and a logical modifier and occur when the logical modifier is placed lower (resp. higher) in the source language sentence than in its target language counterpart. For instance, in the following example

Example 10 *"Anna usually goes for walks" VS "Anna brukar promenera"*

in Swedish, the logical modifier *usually* is implicitly expressed by the verb *brukar* itself, i.e. placed "higher up".

These divergences are, while not uncommon, hard to handle without access to a lexicon where entries for verbs are complete with a list of arguments and modifiers[6].

Thematic divergence Thematic divergences can happen when the logical subject of a sentence in the target language differs from its grammatical subject. An example of thematic divergence is the following:

Example 11 *"Yana likes books" VS "A Yana piacciono i libri"*

In this case, in the Italian translation, the (logical and grammatical) subject of the English sentence, *Yana*, is expressed, even though it is still the logical subject of the sentence, as an oblique; the object *books* becomes the grammatical subject (*i libri*) in Italian, while the head verb remains in both cases the root.

Similar to this are the cases where a passive sentence in the source language is rendered as active in the target language and, as a consequence, the subject and complements are swapped:

[6] on the other hand, developing the CA module further could be a way to construct such a "rich" lexicon, as will be discussed in Chapter 6.

Example 12 *"The game had only been tried by Andrea" VS "Solo Andrea aveva provato il gioco"*

One could object that divergences like the above (and some other) could - and maybe *should* - be avoided when translating a text. We will not dwell upon this, since what matters for the purpose of CA is not what is desirable in natural language translation but rather what divergences do occur in the human-translated parallel texts available for analysis.

Given that UD provides a subtype `pass` for `nsubj`s and auxiliary verbs, it is easier to explicitly handle cases such that of Example 12, while those that resemble Example 11 are left, for the moment, unhandled.

3.1.2.2 New extraction algorithm

The decision to use different alignment criteria simultaneously, together with the nature of the criteria described in the above themselves, implies that perfect alignment must be given up: the new algorithm is such that aligned subtrees are not extracted from a pair of sorted and padded sentence trees but rather obtained by direct comparison between the original trees. Aligning a pair of sentences means, then, checking if the corresponding UD trees t, u match any of the criteria at hand and, if they do, considering them as the members of an alignment, adding the alignment to a collection and recursively repeating the same procedures for all possible pairs t', u', where t' is a subtree of t and u' is a subtree of u.

```
type Criterion = UDTree → UDTree → Bool

alignExtract :: [Criterion] → (UDTree,UDTree) → [Alignment]
alignExtract cs (t@(RTree n ts), u@(RTree m us)) =
    if (not . null) applyingCriteria
        then (t,u):concatMap (alignExtract cs) [(t',u') |
            t' <- sortSubtrees ts, u's <- sortSubtrees us]
        else []
        where
            applyingCriteria = filter (\c → c t u) cs
            sortSubtrees = sortOn (udDEPREL . root)
```

Figure 3.9: Basic version of the improved CE algorithm. Instead of aligning full sentence trees and only then extract concepts comparing the structure of the trees and their UD labels only, it recursively extracts pairs of aligned subtrees directly. To do so, for each pair of subtrees occurring in the same context, it tries to apply multiple criteria, implemented as boolean functions telling whether two trees should become the members of an alignment.

3.1.2.2.1 Pruning alignments

In its most basic version, this algorithm does nothing to avoid subtrees in the source language to be aligned with multiple subtrees in the target language and vice versa. As a consequence, it systematically *overgenerates* alignments.

To avoid overgeneration, or rather to counter it, we need a way to select, in the very common case that the algorithm detects several alignments alternative to each other within the same sentence, the one that is more likely to be correct. A way to do this is to sort such alignments based on their "reliability" and only keeping the first alternative. The question becomes, then, how to sort the alignments.

To answer this, it must be first of all said that the criteria described in Section 3.1.2.1 are not to be considered equally reliable nor describe equally frequent situations. For this reason, a way to obtain a series of alignments that are, in part, implicitly sorted is to apply criteria (or particular combinations of criteria) in a specific order that can be determined empirically and is easily modifiable thanks to the fact that, in its current implementation, the algorithm outlined above takes a list of criteria assumed to be in order of priority as one of its parameters.

It can happen, however, that two trees match *more than one* of the criteria. For instance, two trees might have their roots sharing the same UD label (cf. Criterion 1) *and* be POS-equivalent. In such cases, it intuitively makes sense to consider them more likely to be exact, as there are literally more reasons to align them. This makes the implicit ordering that the solution just described produces "for free" insufficient and leads to the necessity to keep track of the set of reasons why an alignment has been extracted and only then sort the alternative alignments. From the implementative point of view, each alignment is simply associated with a `Set` of labels, whose type is in fact named `Reason` and has an ordering defined over. The final sorting is based on:

1. the number of reasons for alignment
2. the priority level of the highest-priority reason

These labels can even be part of the final output of the CE module, thus adding an ulterior level of explainability to the system it belongs to.

3.1.2.2.2 Aligning heads

As mentioned in Section 3.1.1, the original version of the algorithm creates, every time an alignment is extracted, an additional one for the for the heads of its two members, which we refer to as a *head alignment*. Doing this is a crucial part of the algorithm, as the following example, concerning a straightforward to align pair of sentences, shows:

Example 13 *Consider the English sentence "Enrico eats a banana" and its Italian equivalent "Enrico mangia una banana". As the figure below shows, their trees are perfectly aligned without any need for padding or sorting:*

Without head alignments, the output of the algorithm described so far would be:

- ⟨*Enrico eats a banana, Enrico mangia una banana*⟩

- ⟨*Enrico, Enrico*⟩
- ⟨*a banana, una banana*⟩
- ⟨*a, una*⟩

while introducing head alignment leads to detecting two additional, equally relevant one-word correspondences[7]*:*

- ⟨*eats, mangia*⟩
- ⟨*banana, banana*⟩

While it is of extreme importance not to miss alignments like the last two in the example above, aligning heads is not always appropriate, especially when, as in the present case, common translation divergence patterns are one of the criteria. For instance, when two trees are aligned because of a categorial divergence such as that of Example 4, where *doctoral* corresponds to *di dottorato*, it is at least questionable to also align *doctoral* with *dottorato*. Consequently, in the current implementation of the algorithm, each alignment criterion is associated with a flag telling whether heads should also be aligned whenever a new alignment is extracted because of that criterion.

In a way, even though it is handled differently from the others discussed so far, that of aligning heads could also be seen as another alignment criterion:

Criterion 4 (Heads of matching trees) *The roots n_1, n_2 of two trees t, u will be aligned if t and u are aligned.*

There are also cases where some sort of head alignment is desirable but it is not as simple as creating an additional alignment from the roots of each pair of aligned subtrees. As a consequence, an important part of the improved CE module developed within this project is a function `alignHeads` that performs head alignment with the necessary caveats. Following are the two special cases its current implementation is able to handle correctly despite being less straightforward. As for translation divergences, there may well be other cases that can be taken care of similarly: the objective of what has been implemented so far is mostly to demonstrate how this can be done.

Auxiliaries When the translation counterpart of a verb in the SL is composed of a lexical verb and one or more auxiliaries (or vice versa), it is desirable to align the SL verb to the entire TL subtree composed of the head verb and its auxiliaries, like in the following example:

Example 14 *"Many important decisions were taken by Tommaso" VS "Många viktiga beslut togs av Tommaso"*

Achieving this is relatively straightforward: before aligning a pair of head verbs, their dependents labelled `aux` must be looked for. In case the head verb in the TL has one or more such dependents, but its SL counterpart does not, the TL member

[7] note how, even with head alignments, not all potentially interesting alignments are found. In this example, for instance, it could be useful to align the verb phrase "*eats a banana*" with its Italian counterpart "*mangia una banana*", whose members are also not, strictly speaking, subtrees of the original sentences. The approach we use for head alignment, as we will discuss in Chapter 6, could be generalized to cases like this.

of the new alignment will not be composed by the head exclusively, but will include the `aux` subtrees.

Compounds We mentioned compounds as a type of conflational divergence (cf. Section 3.1.2.1.3). Given the variety of ways in which the translation equivalent of a compound can be expressed and the subsequent variety of related UD labels (a compound can be rendered as another compound, as a `compound` (space-separated) expression, as a noun modified by an adjective and/or another noun, as a `flat` multiword expression, as a combination of these things...), compounds are slightly harder to align than main verb + auxiliaries constructions in practice, but the solution is conceptually the same: before aligning two heads, we check whether the list of their immediate dependents contains anything labelled `compound`, `flat`, `amod` or `nmod`. If that's the case in, for instance, only the sentence in the source language, then it is likely that the head of the tree in the target language is in fact a compound. If so, it is aligned not just with the head of the tree in the source language, but with the tree composed of the head and the list of subtrees labelled as `compound`, `flat`, `amod` or `nmod`. In this way, our program is generally able to find the counterparts of a compound even when they are very complex, such as in the following case:

Example 15 *"I took a course on Machine Learning techniques" VS "Jag deltog i en kurs om maskininlärningstekniker"*

3.1.2.2.3 Counting and reusing alignments

We mentioned how knowing why a certain alignment was identified is important to know to what extent we can trust the program to having taken the right decision. When working on a full parallel text instead of on a single sentence, another important information is the number of occurrences of that alignment (disregarding all sentence-specific information such as the linear positions of its nodes) throughout the entire corpus: unless it is the result of a systematic error (and even in this case, having it presented as a first-class alignment will help finding the mistake), an alignment occurring multiple times can be considered "more reliable" as it is in a way "corroborated". As a consequence each alignment produced by our CE module is associated with a pair whose first element is the set of reasons for that specific alignment and whose second element is its total number of occurrences.

Furthermore, the fact that an alignment has already occurred can be used as an additional, backup criterion for when none of those described in section 3.1.2.1 apply:

Criterion 5 (Known alignment) *Two dependency trees t, u will be aligned if such alignment belongs to a set K of known ones.*

In the case at hand, K is initialized as empty and iteratively augmented with the results of aligning each pair of sentences, but it is not hard to imagine cases in which starting with a nonempty set would be useful.

3.2 Evaluation

We conclude this chapter by describing the data and the approaches used to evaluate the CE module described in Section 3.1 independently from the subsequent stages of the proposed pipeline and by presenting the results of such early evaluation.

3.2.1 Data

3.2.1.1 PUD treebanks

Because we want part of our evaluation to be independent from the quality of the dependency trees obtained by automatically parsing sentence-aligned text, a portion of the data used for this purpose consist in manually annotated data.

In particular, we use a subset of the Parallel UD (PUD) corpus, a set of handcrafted[8] multilingual treebanks in CoNLL-U format created for the CoNLL 2017 shared task on Multilingual Parsing from Raw Text to Universal Dependencies [1]. This prevents the CE module from failing because of parse errors, even though a small number of annotation inconsistencies and imprecisions is still present.

PUD treebanks are available in 20+ languages, of which we selected Italian, English and Swedish, and are composed of 1000 sentences taken from the news domain and from Wikipedia. Due to the lack of a gold standard to refer to in terms of CE and to the consequent need to manually assess the correctness of each alignment obtained, for most of the evaluation we only use the first 100 of these sentences.

3.2.1.2 Course plans

When it comes to testing the program on raw text to be parsed automatically, we use two bilingual sentence-aligned corpora consisting of course plans from the Department of Mathematics and Computer Science of the University of Perugia (for English-Italian) and from the Department of Computer Science and Engineering (CSE) shared between the University of Gothenburg and the Chalmers University of Technology (for English-Swedish). For brevity, we will refer to these two datasets as to the DMI and CSE corpora throughout the text.

Part of this data were collected and sentence-aligned specifically for this work, but a core of CSE plans had already been gathered as part of another thesis project [17].

Our parser of choice is UDPipe [39]. For parsing the two corpora, we use language models trained on two distinct parallel treebanks: for English-Italian, we selected ParTUT, an originally Italian-only treebank including texts of various genres, while for English-Swedish our choice was LinES, a larger bilingual treebank [2].

[8] for some languages, dependency labels were actually automatically converted to UD format from other standards. Furthermore, manual annotation often only concerns some of fields of the CoNLL-U files, but some manual annotation is generally involved in assigning POS tags and dependency relations.

3.2.2 Evaluation metrics

While precision and recall are two well-known performance metrics that would be very well suited to the evaluation of our CE module, the lack of a CE gold standard forces us to approximate them with respectively:

- the number of correct alignments the program is able to extract
- the ratio between such number and the total number of alignments extracted

Determining whether an alignment is correct is not, however, a completely trivial task, since there are correspondences which are correct but not easily reusable in contexts other than that in which they were found. For instance, in the following case:

Example 16 *"He missed the boat" VS "Ha perso il treno"*

it is hard to deny that *boat* has been "translated" as *treno*, "train", so our CE module would - and should - be able to detect it, but in most situations it is by no means desirable that such correspondence at the word level is made use of, for instance by storing it in an bilingual lexicon! As a consequence, each alignment can be marked as:

- correct and useful for translation (`+`)[9]
- correct but not useful for translation (`=`). This means that the CE module is working as expected and that we are are faced with a divergence in the broader sense of the term, i.e. potentially due to an idiomatic usage of language
- incorrect (`-`).

For instance, if we feed the CE module the pair of sentences of Example 16, as a result[10] we get:

```
+he missed the boat|ha perso il treno[UD,POS]1
+missed|ha perso[UD,POS,HEAD]1
=the boat|il treno[UD,POS]1
=boat|treno[UD,POS,HEAD]1
+the|il[UD]1
```

Furthermore, the absence a reference solution makes it so that comparing the updated versions of the module with the baseline described in Section 3.1.1, thus focussing on measuring the improvements with respect to it rather than the quality of the results under absolute terms, is in practice the best way to assess the results. While not ideal, repeatedly performing this kind of evaluation, starting already during the early stages of its development, proved to be useful to understand the impact of each of the changes made, thus guiding further modifications and helping debugging the program when necessary.

[9] at least to some extent: we observe that perfect translation equivalents that can be replaced with each other in all contexts are extremely rare, just like perfect synonyms.

[10] this is what the actual output of the program in evaluation (or "linearized") mode after annotation looks like. The abbreviations in square brackets refer to alignment Criteria 1, 2 and 4 described in the above.

3.2.3 Implementation details

In order to be able to perform the type of evaluation described in the above Section and because manual inspection of the results was indispensable, the program was made able to write alignments in the reader-friendly output format shown above. The notation is the following:

```
linearized SL tree|linearized TL tree[R1,R2,...,Rn]N
```

where `[R1,R2,...,Rn]` is the list of criteria the members of the alignment match and `N` is the number of occurrences of the alignment in the corpus.

With alignments in this format available, a simple way to evaluate a specific version of the CE module is to manually assess the correctness of each of the correspondences the program identifies and compute some statistics about them. A separate Haskell module, `EvAlign`, was written to parse files in the above format and compute a series of useful statistics:

- the total number of alignments found
- the number of *distinct* alignments, i.e. the number of alignments with distinct linearizations
- the percentage of correct alignments
- the percentage of correct alignments that are also useful for translation
- the percentage of alignments found due to each combination of criteria, and how many of them are correct.

While labelling all alignments by hand was necessary to evaluate the baseline, as long as the corpus does not change, it is possible to evaluate later versions of the program semi-automatically, as the correctness of many alignments is most likely already known thanks to the first round of manual annotation or to any of the successive ones. `EvAlign` can in fact also be run interactively with a labelled file and an unlabelled file as input. When this is the case, the program tries to label as many alignments as possible based on the content of the labelled file (which, of course, can even be the result of merging multiple files obtained with different versions of the program), asking the user to assess the correctness of newly found ones only. If run in this modality, the program also displays some more statistics that make comparison it with the older, labelled file easier, such as the number of new correct alignment found and that of incorrect alignments lost.

3.2.4 Evaluating CE against a baseline

Table 3.1 compares the results obtained with the baseline with those obtained with the proposed improved version on the 100-sentence manually annotated corpus described in Section 3.2.1, both for English-Italian and for English-Swedish.

	baseline		**improved version**	
	en-it	**en-sv**	**en-it**	**en-sv**
distinct alignments	1097	1257	1198	1314
correct (+ and =)	830 (58.12%)	995 (79.15%)	964 (80.46%)	1105 (84.03%)
correct and useful (+)	776 (54.34%)	976 (77.64%)	896 (74.79%)	1082 (82.28%)

Table 3.1: Comparison between the baseline and the improved version of the CE module using manually annotated sentences from the PUD treebank in two different language pairs, English-Italian and English-Swedish.

As the table shows, for both language pairs, the raw number of distinct alignments extracted increases, which suggests an increase in recall. Most importantly, the percentage of correct alignments, our approximation for precision, increases significantly, and in particular by more than 20% for English-Italian, the language pair most used during development. There are also significant, but easily explainable differences between the two language pairs considered: in general, results are better for English-Swedish due to the syntactical similarity between the two.

3.2.5 Evaluating specific criteria

When it comes to the improved version, it is also interesting to look at criterion-specific statistics, summarized in Table 3.2.

One of the main things the results suggests is that, even though most alignments are still found by applying the original criterion (cf. Criterion 1 in Section 3.1.2), a combination of criteria 1 and 2 is still able to detect many correspondences (more than 30% of the total for both language pairs, if we combine regular and head alignments extracted in this way) while having higher (>90%) precision.

This data also give an empirical confirmation of the observation made in Section 3.1.2.2.2 about the problem that the baseline has when it comes to head alignment, indicating that POS-equivalence is especially important when extracting head alignments. The small size of the corpus cause Criterion 5 to remain extremely marginal. Cross-language pair differences are, in this case, minimal, confirming that the criteria used and their priority order are not specifically tailored for the language pair used for under-development experiments.

criteria	**alignments extracted**		**correct (+ and =)**	
	en-it	**en-sv**	**en-it**	**en-sv**
matching UD labels	39.39%	37.71%	78.60%	80.24%
POS-equivalence	3%	1.82%	83.33%	83.33%
known divergence	2.08%	1.36%	52%	55.55%
known alignment	0%	0%	-	-
matching UD labels + POS-equivalence	20.95%	24.71%	93.03%	94.76%
matching UD labels + head alignment	21.78%	19.61%	67.81%	76.74%
POS-equivalence + head alignment	1.33%	1.29%	75%	64.70%
matching UD labels + known alignment	0.25%	0.45%	66.66%	83.33%
matching UD labels + POS-equivalence + head alignment	10.85%	12.69%	95.38%	90.41%

Table 3.2: Criterion-specific statistics. The leftmost column specifies a criterion or combination of criteria. The central column indicates the percentage of alignments extracted because of each set of criteria, and the rightmost one specifies how many of them were marked as `+` or `-`.

3.2.6 Working with raw text

Of course, it is also interesting to compare the performance of the CE module on manually annotated data with the results obtained on the automatically parsed course plans corpora, summarized in Table 3.3.

	DMI (en-it, 798 sentences)	**CSE (en-sv, 498 sentences)**
distinct alignments	352	529
correct (+ and =)	243 (69.03%)	368 (69.56%)
correct and useful (+)	229 (65.05%)	351 (66.35%)

Table 3.3: Performance of CE on barely sentence-aligned data, parsed automatically. Two distinct corpora of different size have been used for the two different language pairs, so the two columns are not to be compared with each other, but rather with Table 3.1.

As it is to be expected, results are not as good for automatically parsed sentences as they are for manually annotated data. In particular, the percentage of correct alignments drops - but interestingly it is virtually the same for both language pairs. The difference between the two pairs seems instead to be in the total number of alignments found: despite English-Italian corpus being significantly larger, more concepts are extracted for the Swedish-Italian one.

3.2.7 Comparison with methods from Statistical MT

Another kind of evaluation that can be performed before the aligned dependency subtrees are converted to GF ASTs to be fed to the last stage of the pipeline is to compare them with those obtained by means of application of existing statistical algorithms. Among the well known solutions mentioned in Section 2.2.3.1, we have chosen to focus on `fast_align` [16] because of its ease of use and installation.

A Haskell script converts the CoNNL-U files that are fed to `AlignTrees` into the input format required by `fast_align`, so that the exact same tokenization is used.

A second script converting its output into that used by `EvAlign` allows to reuse all of the evaluation infrastructure described in Section 3.2.2. Finally, Table 3.4 summarizes the results of a comparison between the improved CE module and `fast_align` run with 10 iterations in EM training and symmetrizing the alignments found by running the program in both directions.

Because `fast_align` is a purely statistical approach, we trained it not only on the same 100-sentence subset of PUD used for evaluating `AlignTrees` against the baseline, but also on the full 1000-sentences dataset. In the latter experiment, the alignments obtained for the last 900 sentences were discarded before evaluating the results. Due to the time required to manually annotate the numerous alignments extracted by `fast_align`, this evaluation has been conducted only for the most challenging language pair, English-Italian. For a fairer comparison, since `fast_align` works at the word level, only the one-to-one, one-to-many and many-to-one word alignments produced by the CE module were kept.

	`AlignTrees`	**`fast_align` 100**	**`fast_align` 1000**
distinct alignments	716	1440	1435
correct	536 (74.86%)	410 (28.47%)	656 (45.71%)
correct and useful	491 (68.57%)	371 (25.76%)	590 (41.11%)

Table 3.4: Comparison between the improved CE module and `fast_align` trained on the first 100 only (column 2) and on the whole PUD corpus (column 3).

As the table shows, since `fast_align` tries to find a correspondent in the TL for each word in the SL, the number of alignments found by our CE program is roughly half of that of those found statistically. The percentage of correct alignments is, however, much higher for our system, even when the full PUD corpus is used in the training step of `fast_align`. This shows that our approach is particularly well suited to smaller datasets and can even be used on individual sentence pairs, provided that they are analyzed correctly.

4

Concept Propagation

After a set of concepts is known, it can be useful to be able to look for the corresponding concrete expressions in a new language. In this chapter, we describe our approach to this task and present an evaluation of the corresponding module.

4.1 Method and implementation

As mentioned in Section 2.2.2, Concept Propagation (CP) is the task of finding the concrete expressions corresponding to a set of known concepts, represented by a set of alignments and/or their abstract representations. Since our method for propagating concepts makes use of dependency tree alignments only, CP can take place before and, in general, independently from a potential conversion of the UD trees into GF trees.

There are two scenarios in which CP can prove useful:

1. when working on a parallel text A in more than two languages. In this case, CP can be used, after CE has been performed on a pair of languages $\langle L_1, L_2 \rangle$, to propagate the extracted alignments in a third version of the text in a new language L_3. In this case, the expectation is for the program to be able to propagate the vast majority[1] of the concepts in the set
2. when working on two bilingual parallel texts A and B with one language in common, say A in two languages $\langle L_1, L_2 \rangle$ and B in two languages $\langle L_2, L_3 \rangle$. After performing CE on the first pair, CP can be used to look for the extracted concepts in the L_2 version of the second and, if they do appear in such text, find their counterparts in L_3. Of course, the number of concepts that can be propagated in this scenario can be significantly lower, especially if the two parallel texts do not belong to the same domain.

Our algorithm is meant to be as general as possible, and as such it is not optimized for the first scenario, but rather covers both without making any assumptions in terms of sentence alignment[2]. It works as follows: for each alignment obtained by performing CE on text A, we consider the dependency tree in language L_2. We then

[1] even though handling common translation divergences help, free translation involving radical differences in sentence structure can make it so that some concepts lack a clear counterpart in the text in L_3.

[2] a way to make the algorithm more efficient for the first scenario could in fact be to rely on the fact that the n-lingual parallel text is sentence-aligned, keep track of the sentence from which a particular concept has been extracted and only try to find a equivalent in the corresponding L_3 sentence.

Figure 4.1: The two CP scenarios compared. On the left, Scenario 1, when CP is applied on three translations of the same text. On the right, Scenario 2, dealing with two distinct bilingual texts.

look for such tree among all subtrees the L_2 version of text B is composed of. If a sentence in the L_2 version of text B does contain a_i^2, we align such sentence to its L_3 counterpart using the same procedure as described in Section 16 for CE. Finally, if multiple possible correspondences are found, the L_3 dependency subtree which is closest in depth to a_i^2 is selected. This is meant to avoid, for instance, mistakenly aligning a full sentence tree to a dependency tree extracted when aligning heads, which would otherwise be likely, as their heads are obviously both labelled `root`. In the next two sections, we discuss some important details of the algorithm, and in Section 4.1.2 in particular will get to other issues related to head alignment.

```
propagate :: [Criterion] → ([UDTree],[UDTree]) → UDTree →
    Maybe Alignment
propagate cs ([],_) _ = Nothing
propagate cs (t:ts,u:us) c =
    let as = sortAlignments (alignExtract cs (t,u))
    in case find (\(sl,_) → sl == c) as of
        Nothing → propagate cs (ts,us) c
        a → a
    where
        sortAlignments = sortOn (depthDiff . fst)
            where
                depthDiff t = abs (depth t - depth c)
```

Figure 4.2: The CP algorithm. The concept to propagate is looked for among the SL subtree of each alignment obtained by comparing the sentences of the new corpus. If it is found, the alignment it belongs to is returned. If there are multiple possible alignments, the alignment whose TL side is closest in depth to the concept itself is returned.

4.1.1 Ignoring details of UD trees

One point that becomes of extreme importance when dealing with Scenario 2, i.e. when the text used for CP is not a translation of the one used for CE, is the way we look for subtrees in the L_2 version of text B. Since we work with two different L_2 texts, in fact, we cannot expect text B's subtrees to match L_2 concepts extracted from T_1 exactly: as we saw in Section 3.1, nodes of UD trees contain information,

particularly about the position of each word and its head, that should be disregarded in this context.

To make the CP algorithm more robust to minor parse errors, our decision was to consider two (sub)trees equivalent based on a comparison of just their shape and their `FORM`, `LEMMA`, `UPOS` and `DEPREL` fields. In this way, not only all optional fields (which might well differ or be blank for only one of the texts), but word position information is also ignored.

4.1.2 Propagating head alignments

The propagation of head alignments is by far the most challenging aspect of CP. This is due to the fact that the head of a dependency tree t, as defined for the purposes of CE (cf. Section 3.1.2.2.2), is not, strictly speaking, a subtree of t (even though in a broader sense it is, of course, a part of t). When head alignments are extracted, in fact, their members are either:

- trees composed exclusively of the root of a dependency subtree
- trees composed of the root of a dependency subtree and some of its immediate subtrees. As discussed in Section 3.1.2.2.2, this happens in cases such as when the root in the SL is a compound (written as a one-word) and its TL counterpart is a multiword.

Not only, then, is it necessary to look for each concept to propagate among all the dependency subtrees of each SL sentence of the corpus CP is performed on, but for each subtree it is also necessary to check whether the concept could be a head of such subtree. This is implemented by means of a function `isHeadOf` specular to `alignHeads`.

If this is the case, i.e. if the concept the program is trying to propagate is the head of a certain SL subtree, then the resulting alignment should be with the head of the TL counterpart of such subtree.

4.2 Evaluation

In the following three sections, we describe both the CP experiments performed and their results. Clearly, for all the experiments, we make use of the evaluation script used for CE, which has been extended to also compute CP-specific statistics, such as the percentage of (successfully) propagated concepts.

4.2.1 Preliminary testing

One way to check the CP module for obvious flaws before trying to assess its performance in the two scenarios described above is to try to re-obtain the alignments obtained with the CE module via propagation. This means running both programs on the exact same input treebanks as used for CE. It represents a special case of Scenario 1 in which the program is expected to be able to propagate all alignments, with no exceptions.

The fact that many of the alignments obtained via propagation are the same as the ones extracted initially is generally a good indication that the program is working according to expectations. However, this has to be taken with a grain of salt: as discussed in the above, the program looks for one possible counterpart per concept only and is provided no indication about what sentence in particular it should look into, so that in presence of multiple valid alternative alignments the user should not expect it to identify the same correspondences as the extraction procedure, but rather just correct, potentially different correspondences.

4.2.1.1 Experimental results

This kind of pre-evaluation has been run on the first 100 sentences of the PUD treebanks in English and Italian, one of the corpora used for evaluating CE. Most importantly, the experiment confirms that the CP module is in fact able to find all the alignments extracted by CE. The results also show that the vast majority of the propagated alignments is identical to its extracted counterpart, and that most often, even when it is not, that does not imply that the propagated alignment is incorrect. Out of a total of 224 (18.69%) incorrect propagated alignments, only 28 (12.5%) introduce an alignment error which was not already present in their extracted counterparts, and there are even instances where propagated alignments are, conincidentally, correct even though their extracted counterparts are not.

4.2.2 Scenario 1

For evaluating how well CP works in the first scenario, the corpus we use is the same as for testing, but in three instead of two languages: the concept extracted by means of comparing the English to the Italian version are propagated to Swedish, using both English and Italian UD trees as the starting point.

Considering that we are working with three versions of the same text, the expectation is for the program to be able to propagate the vast majority of the concepts with a precision similar to that obtained during the extraction stage, even though free translation, unhandled translation divergences and annotation inconsistencies can make it possible for some of them not to be found in the Swedish translation or not to be aligned correctly.

As for when it comes for CE, comparing the results of CP for two different language pairs, English-Swedish and Italian-Swedish, can be interesting to see how much easier it is to propagate concepts in two more syntactically similar languages.

4.2.2.1 Experimental results

As Table 4.1 shows, the experimental results of this type of evaluation match the above expectations. First, the number of alignments the program is able to propagate is, for both language pairs, smaller than that obtained in the testing phase but still relatively large, while the amount of propagation errors increases marginally. Furthermore, the English-Swedish pair gives slightly better results than the Italian-Swedish pair, both in terms of the number of alignments the program is able to propagate and in terms of the amount of alignment errors.

It is interesting to notice, however, that the amount of errors is significantly (> 5%) inferior when concepts are propagated to Swedish starting from English, as opposed to Italian, dependency trees. This is most likely due to the syntactical features shared by the former two languages. This means that not all Italian-English extraction errors are propagated and it seems to suggest that, if we were to only use the trilingual alignments obtained by English-Italian extraction followed by English-Swedish propagation, some of the initial incorrectly extracted alignments would be automatically discarded.

	en-sv	**it-sv**
propagated	1019 (85.05%)	979 (84.64%)
tot. errors	133 (13.05%)	187 (19.1%)
CP-introduced	75 (56.39%)	84 (44.91%)

Table 4.1: Results of propagating the Italian-English alignments obtained via CE on the first 100 sentences of the PUD corpus to Swedish, using either the English or Italian subtrees as a starting point.

4.2.3 Scenario 2

The PUD corpus is composed of extracts of texts on a variety of subjects. As a consequence, performing an evaluation of CP in Scenario 2 using, for extraction and propagation, two different subsets of the corpus, would not help assess whether the module is well suited for the most relevant subcase of such scenario, i.e. when working with two texts belonging to the same domain.

On top of evaluating the program on PUD data dividing the set of sentences used so far into two equally sized subsets, then, the two course plans corpora described in Section 3.2.1.2 have been also used to put the program to the test.

These two small bilingual corpora allow us to perform two additional experiments:

1. to extract English-Swedish alignments from the CSE corpus and propagate them to Italian using the DMI corpus
2. vice versa, to extract English-Italian alignments from the DMI corpus and propagate them to Swedish using the CSE corpus

4.2.3.1 Experimental results

4.2.3.1.1 Propagation with texts in different domains

When it comes to the first set of experiments, where we use PUD data, being the subsets of selected sentences extremely small and having the correctness of most alignments been already assessed in previous experiments, it is easy to work with all possible pairs of languages. The results, summarized in Table 4.2, show that, in general, only a small fraction of the extracted alignments are actually propagated in this case. As mentioned above, this is to be expected since different sentences belong to different domain. A closer look to the set of alignments returned by propagation, the vast majority of which has single words as members, shows that most of them are

common function words, which, alone, are not always relevant for a GF-based MT pipeline. Common determiners such as articles, for instance, can be better handled by RGL grammars alone. Prepositions, on the other hand, are only useful with some information about the context in which they occur in, since it is customary for them to be used differently across different, even closely related languages. However, some common content words, like ⟨*always, sempre, alltid*⟩, ⟨*new, nuovi, nya*⟩ and ⟨*same, stesso, samma*⟩ are also aligned correctly.

	en-it-sv	**it-en-sv**	**en-sv-it**	**sv-en-it**	**it-sv-en**	**sv-it-en**
extracted	638	638	687	687	608	608
propagated	92 (14.42%)	92 (14.42%)	98 (14.26%)	84 (12.22%)	101 (16.61%)	87 (14.37%)
tot. errors	46 (50%)	21 (22.82%)	42 (42.85%)	24 (28.57%)	21 (20.79%)	28 (32.18%)
CP-introduced	33 (71.73%)	11 (52.38%)	21 (50%)	12 (50%)	12 (57.14%)	21 (75%)

Table 4.2: Results of CP for various language pairs using PUD (non-homogeneous) data. For each column, the first two languages are the ones used for extraction and the second two the ones used for propagation.

The table also shows, like the results obtained from Scenario 1 suggest too, that the order in which languages are used is not completely irrelevant: while the percentage of propagated alignments does not vary widely, the amount of errors is significantly lower when the propagation step involves the two syntactically more similar languages, English and Swedish. The fact that CP works better with English-Italian than with Italian-Swedish, on the other hand, may simply be a result of the fact that the latter language pair was never used for testing the program during its development, which might have caused some of the criteria to be not completely language pair-independent.

Finally, the percentage of errors introduced by CP is significantly higher than that recorded for Scenario 1. One factor that contributes to this is the fact that function words, which are by far more numerous in the propagated alignments than content words, are often used differently in different languages and contexts. When it comes to the languages we are considering, this applies in particular to prepositions: because it is hard to know what the correct translation equivalent is without looking at the specific sentence, when CP is performed on a text different from the one used for CE, many preposition-to-preposition alignments have to be marked as incorrect even though they might be correct in some specific cases.

4.2.3.1.2 Propagation with texts within the same domains

Quantitatively, the results of the last pair of experiments, performed on two different automatically parsed bilingual corpora composed of course plans of Computer Science programmes, are to some extent similar to those obtained using different texts in different domains (cf. Tables 4.3 and 4.2). Only a tiny fraction of the extracted alignments is propagated and errors are numerous, as in the previous setting.

	sv-en-it	**it-en-sv**
extracted concepts	1950	1823
propagated	205 (10.51%)	200 (10.97%)
tot. errors	66 (32.19%)	61 (30.5%)
errors introduced by CP	33 (50%)	33 (54.09%)

Table 4.3: Results of CP for using two different bilingual corpora belonging to the same domain of course plans. For each column, the first two languages are the ones used for extraction and the second two the ones used for propagation.

More than the quantitative results, however, it is interesting to take a look at some of the propagated concepts. Because the text are similar both stylistically and, most importantly, in terms of content, the program is in this case also able to find trilingual correspondences between content words, sometimed specific of the domain. Examples of such alignments are ⟨*skills, capacità, färdigheter*⟩, ⟨*functional, funzionale, funktionell*⟩, ⟨*exam, prova, tentamen*⟩, ⟨*course, corso, kurs*⟩, ⟨*prerequisites, prerequisiti, behörigheter*⟩, ⟨*lectures, lezioni, föreläsningar*⟩ and ⟨*knowledge, conoscenza, kunskap*⟩.

Sometimes, however, the meaning of a word is highly context-dependent and as a consequence, even if both the pairs of extracted and propagated alignments are correct, the corresponding trilingual alignment we could infer isn't (e.g. ⟨*learning, conoscere*⟩ and ⟨*learning, inlärning*⟩).

Finally, in some rare cases, even correspondences regarding longer expressions, such as ⟨*the aim of the course, l'obiettivo del corso, syftet med kursen*⟩, can be found. In this case, the English-Italian correspondence is not especially useful, since the phrase could be translated almost word by word between the two languages, but the translation to Swedish is not literal, thus making the alignment useful for the purposes of MT.

5

Concept Alignment in Machine Translation

We conclude this work by documenting a first attempt to put our CA component to use in a MT system. The chapter starts with a discussion of how GF translation can benefit from using concepts extracted with our method as translation units. After that, we give a step-by-step description of how this is achieved, also indicating the changes made to the CA module to adjust it for this specific use case. Finally, Section 5.2 focuses on the actual evaluation of the resulting system and presents its results.

5.1 Method and implementation

As discussed in Section 2.1.2.2, GF translation consists in parsing SL strings to obtain the corresponding ASTs to then linearize them according to the concrete syntax of the TL. As well as describing the morphological and syntactical features of the languages involved, a multilingual grammar suited for this task defines a *translation lexicon*, i.e. a set of word senses and the corresponding expressions in the various languages.

Automating CA can be seen, then, as a first step for automatically generating such lexicon. An important point is that our approach, as opposed to word alignment techniques, identifies concepts at different levels of abstraction. This allows us to construct lexical entries not only for the grammatical categories that typically correspond to single words (such as nouns, adjectives and adverbs), but also for phrases (with the term "phrase" now intended grammatically, as opposed to its meaning in SMT). Doing so enables GF translation, designed to preserve grammatically, to also render idiomatic expressions found in the data used for generating the lexicon correctly.

Obtaining a multilingual GF grammar from a set of aligned UD trees, however, is not immediate. In the following three sections, we illustrate the different steps of this process. We then introduce the simple GF-based MT module used for the experiments discussed in Section 5.2 and describe the adjustments made to the CA component in order to optimize it for the task at hand.

5.1.1 UD-to-GF conversion

The output of both CE and CP is, unless when linearized for evaluation purposes, a set of aligned UD trees in CoNNL-U format. In order to be used for GF-based translation, such trees need to be first converted into GF ASTs, which are then used to generate a set of grammar rules constiuting the above mentioned multilingual GF lexicon.

Conversion is based on annotations defining the relation between UD trees and GF ASTs, which we refer to as *dependency configurations*. While going into all the notational details is beyond the scope of this project[1], an example of one such configuration is

```
#fun PredVP nsubj head
```

which specifies that a `PredVP` (predicative Verb Phrase), defined in the RGL as a function of type

```
PredVP : NP -> VP -> Cl
```

i.e. a function taking two arguments, an `NP` (Noun Phrase) and a `VP` (Verb Phrase) and returning a `Cl` (clause), corresponds to a UD tree whose head, the VP, has a `nsubj` attached to it. Appendix B lists the dependency configurations used for UD-to-GF conversion in the context of this project.

As discussed in Section 2.1.3.1.4, the UD-to-GF conversion, while being more complex than its reverse, can be performed by `gf-ud`, a program offering several functionalities useful to work with GF and UD simultaneously. Such program, however, cannot be fed the alignments obtained via CE and CP directly. This is due to the fact that the members of most of the relevant alignments are subtrees (or heads of subtrees) and not complete UD trees. While this makes no difference in terms of the Haskell data type we represent them with (cf. Figure 3.2), trying to print them to files in the corresponding CoNNL-U notation with no postprocessing leads to malformed UD sentences.

The problem, which is due to the lack of a root labelled as such and illegal `ID` and `HEAD` values, was solved by implementing a simple normalization procedure, now part of the CA library. Called every time a UD subtree has to be converted to CoNNL-U, it first replaces the UD label (`udDEPREL`) of its root node with `root` and assigns the special value 0 to its `udHEAD` field. After that, it proceeds to "rescale" the `udID` and `udHEAD` fields of all other nodes so that they end up into the range $[1, n]$, where n is the number of nodes of the extracted subtrees[2].

[1] a comprehensive description of `gf-ud`'s annotation scheme is available at `github.com/GrammaticalFramework/gf-ud/blob/master/doc/annotations.md`.

[2] see Section 2.1.3.1.1 and Figure 3.2 for definitions of all the fields of UD trees and their CoNNL-U representations.

```
1  sadly  sadly  ADV  _  _  5  advmod  _  _
2  this  this  DET  _  _  4  det  _  _
3  malformed  malformed  ADJ  _  _  4  amod  _  _
4  subtree  subtree  NOUN  _  5  nsubj  _  _
5  needs  need  VERB  _  _  0  root  _  _
6  postprocessing  postprocess  VERB  _  _  5  xcomp
```

```
1 this this DET _ _ 3 det _ _
2 malformed malformed ADJ _ _ 3 amod _ _
3 subtree subtree NOUN _ _ 0 root _ _
```

Figure 5.1: A vanilla noun phrase subtree (highlighted in black) in the context it was extracted from (gray) and the same subtree after normalization.

Another practical problem that arises when trying to transform a CoNNL-U tree into a GF AST is that `gf-ud` easily becomes remarkably resource-intensive when dealing with ambiguous, large trees. This is due to the fact that conversion in this direction is a nondeterministic search problem [37]. For this reason, it can be useful to limit the size[3] of the alignments that are actually kept after the extraction (or propagation) procedure. Since our CA component works at all levels of abstraction, thus also aligning full clauses, this turned out to be essential and became one of the command-line parameters of the final executables, as mentioned in Appendix C.

5.1.2 Grammar generation

The result of this UD-to-GF conversion is still a series of tree alignments, but their resulting format makes a substantial difference, since GF ASTs can be used to obtain the rules of a generative grammar. `gf-ud` also provides a way to do this, given an *extraction grammar* and a *morphological dictionary* of the languages of the alignments.

What we refer to as an *extraction grammar* is of course a GF grammar, whose objective is to define the set of basic categories and syntactic rules the entries of the automatically generated GF lexicon is going to be based on.

A *morphological dictionary*, on the other hand, contains correspondences between a large number of lemmas and various word forms. In this case, the morphological dictionaries themselves are implemented in GF.

The program responsible for generating the GF lexicon given these components has been written in parallel with - but not as part of - this thesis project and is still under development. As a consequence, some programming work was required to automate the few parts of the process that still had to be performed manually and some (minimal) postprocessing is sometimes required before the automatically generated grammars can be compiled. This happens for instance when Italian words are incorrectly POS-tagged as verbs and, not having a word ending typical of such

[3] in this context, the size of an alignment is defined as the number of nodes of its largest member.

category, produce pattern matching errors when the corresponding strings are passed to the constructor generating all the verb forms.

5.1.2.1 Extending the extracted grammar

The extraction grammar is designed for the grammar generation module to produce grammar rules corresponding to small concepts: basic categories, noun phrases, verb phrases etc. As a consequence, if used directly, grammars resulting from the above described generation step can only deal with very simple predicative sentences, with verbs in the present tense of the indicative mood, and their constituents, such as

Example 17 *"this sentence is simple"*

Of course, some limited variation is still possible. For instance, the determiner can change or disappear, number can vary, the complement of the copula can become a noun and an adjectival modifiers and prepositional phrases can be added to each noun phrase. For instance:

Example 18

- *"the sentence is simple"*
- *"a sentence is simple"*
- *"sentences are simple"*
- *"these sentences are simple"*
- *"this sentence is an example"*
- *"this short sentence is simple"*
- *"this sentence of the text is simple"*

For languages that are covered by the RGL, however, it is also extremely easy to extend such grammars with preexisting syntax rules so to make it possible to generate more complex clauses and sentences and add useful function words. The grammars used for the experiments that will be described in Section 5.2, for example, have been extended also to cover negative and interrogative forms, the past and future tense of verbs in the indicative mood and comparative adjectives. This allows for plenty of variations, such as:

Example 19

- *"this sentence isn't simple"*
- *"is this sentence simple?"*
- *"this sentence was simple"*
- *"this sentence will be simple"*
- *"this sentence is simpler than that sentence"*

Of course, variations can be combined, so that the grammar can produce sentences like the following:

Example 20 *"Won't these short sentences be simpler than that long sentence?"*

This is accomplished by making the automatically generated grammars extend not only the extraction grammar itself, but also a module that simply imports the necessary standard RGL categories and functions.

5.1.3 Translation

Maybe counterintuitively, the MT module itself is by far the simplest component of the entire system. Built using PGF, the API meant to be used for embedding GF grammars in Haskell programs, it simply makes use of GF parsing and linearization. The only difference between translating inside the GF shell and using this ad-hoc program, apart from the increased ease of use of the latter, are that:

- while GF can parse the SL string in a variety of ways and, as a consequence, return multiple TL linearizations, our program only outputs the first candidate translation, preliminarily discarding trees containing terminal nodes that have no linearization in the TL[4]
- to facilitate the experiments, our standalone program can translate several newline-separated sentences at once.

5.1.4 Adjustments to the CA component

While the domain of application of CA we focus on in this work is automatic translation, the system described so far is not in any way optimized for it and can also be made use of in other contexts. For instance, it is easy to imagine using it to improve the user interface of online translation memories or to facilitate reading parallel texts in the context of language learning.

In our particular setting, however, some practical problems arise and some adjustments can be done in order to make the CA software more well suited to the generation of a translation lexicon.

In the following section, we discuss two ways to mitigate the impact of the annotation errors that inevitably derive from applying automatic dependency parsing on raw, possibly noisy data. Both techniques aim to increase the number of extracted alignments without significantly affecting precision. Section 5.1.4.2, going in a sense in the opposite direction, describes the strategies we apply to filter out irrelevant alignments.

5.1.4.1 Dealing with parse errors

As discussed in Sections 3.2 and 4.2, both our CE and CP modules give promising results on manually annotated data. However, when run on UDPipe-generated dependency trees, performance, and recall in particular, is significantly affected by errors happening at the parsing stage. Because parsing itself is a process consisting of several stages, errors differ based on when they happen. In particular, UDPipe goes through the following steps, each potentially causing one ore more particular kinds of errors:

1. ***tokenization***, which can, although it is unlikely for the languages considered in this work, lead to errors in identifying word boundaries
2. ***morphological analysis***, which can cause lemmatization and, consequently, POS-tagging errors

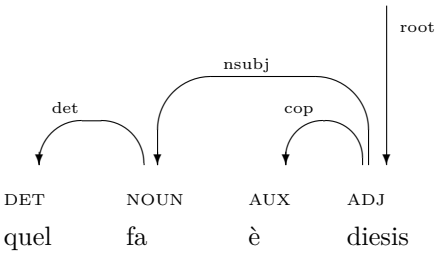
[4] a linearization can be missing, for instance, if a concrete syntax is postprocessed by simply getting rid of an "illegal" verb causing compilation problems (cf. Section 5.1.2)

3. actual ***dependency parsing***, the potential cause of attachment and labelling errors

In this context, we focus on the syntax-level errors: attachment and labelling. Of course, lemmatization and POS-tagging errors can be the first cause of one such error. In particular, the fact that a word is lemmatized incorrectly can make it so that it is also attributed the wrong POS tag and, as a consequence, assigned the wrong dependency label and/or attached to the wrong node.

Example 21 *Consider the Italian sentence "quel fa è diesis" ("that F is sharp"). Notoriously, in Italian, the word "fa" can stand for the musical note F. Since "fa" is also the third person singular of the present indicative of the verb "fare" ("to do", "to make"), it is likely to be lemmatized so (instead of as "fa"). If so, it will also probably POS-tagged as a* `VERB` *and assigned a dependency label typical of verbs (in this case, with no other lexical verbs in the sentence, most likely* `root`*, thus producing an attachment error as well), while the correct annotation is*

```
1 quel quello DET _ _ 2 det _ _
2 fa fa NOUN _ _ 4 nsubj _ _
3 è essere AUX _ _ 4 cop _ _
4 diesis diesis ADJ _ _ 0 root _ _
```



Nontheless, because we do not want to intervene on the parser directly (nor to modify its results), we ignore the causes and focus on the syntax-level errors directly.

5.1.4.1.1 Working at the clause level

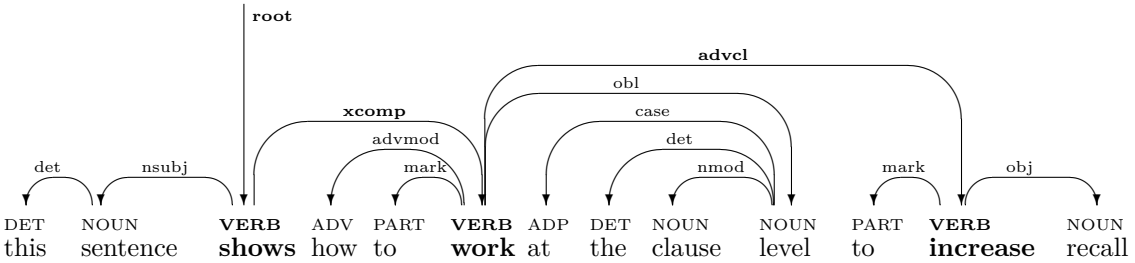
At the highest level of abstraction, the parser often fails to correctly identify the clauses a sentence is composed of. This happens in a variety of ways: at times, clauses are simply not identified as such or, on the contrary, non-clauses are labelled as clauses. Most often, however, clauses are simply not attached to the right node and/or given a label that, while still identifying them as clauses, does not give correct information about their type.

To address this problem, the CE module has been modified so that it can try to align individual clauses instead of sentences. The idea is to obtain, from each pair of sentence trees to align, a pair of lists of subtrees whose heads are labelled `root`, `csubj`, `ccomp`, `xcomp`, `advcl` or `acl`[5].

These lists of "clauses"[6] are then sorted by dependency label and distance from the root, so that in the absence of annotation errors, when there is in fact an evident correspondence, they appear in the same position. Finally, the program tries to align every possible pair of clauses and the pruning procedure described in 3.1.2.2.1 is applied to only keep the ones that are more likely to be correct.

[5] all these clausal labels are defined in Appendix A.2

[6] "clauses" is in quotes since what we are referring to as a clause includes its subclauses too. This means that the sentence is not segmented into clauses in the literal sense of the term, but rather that one tree per verb is extracted from them.



- "*this sentence* ***shows*** *how to work at the clause level to increase recall*"
- "*how to* ***work*** *at the clause level to increase recall*"
- "*to* ***increase*** *recall*"

Figure 5.3: A sentence and its decomposition into clauses. Note how the sentence is not actually segmented, but rather used to obtain a set of subsentences, each having their main verb, marked in bold, as root.

5.1.4.1.2 Dealing with unaligned subtrees

Clearly, however, parse errors do not happen exclusively at the clause level: smaller units, e.g. verb arguments, can also be misclassified or attached to the wrong node. To partially work around the problem, the CE module supports what one could refer to as "alignment by exclusion": after the extraction procedure described in Section 3.1.2 - or its modified version outlined in the above paragraph - is carried out, the program can try to find correspondences between the remaining unaligned subtrees. The idea is the same as above: subtrees are sorted, aligned and lastly alignments are pruned. This is done ignoring, if necessary, the context in which the subtrees occur and potentially making only use of a subset of the alignments criteria, for instance the strictest and safest ones.

This is useful in cases such as that illustrated in Figure 5.4, as well as in the event of actual attachment ambiguity or when the two versions of the text present syntactical differences that cannot be handled with the simple divergence patterns introduced in Section 3.1.2.1.3.

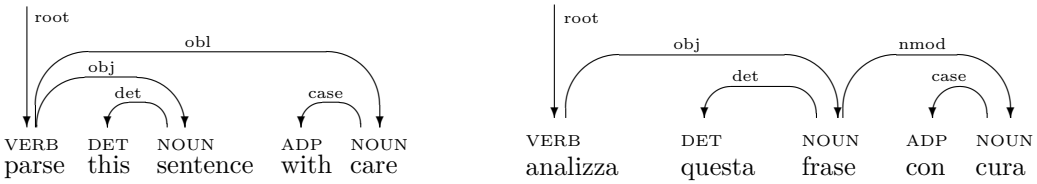


Figure 5.4: A case where alignment "by exclusion" is beneficial. In this case, while the two sentences should appear to be structured identically to a human reader, the Italian parse tree contains a (reasonable) attachment error. As a consequence, vanilla CE would not be able to align the prepositional phase "*with care*", attached to the main verb "*parse*", to its Italian exact counterpart "*con cura*", incorrectly linked to the noun "*frase*" as an `nmod`. A second CE pass trying to put unaligned subtrees in relation with each other, however, is able to detect the POS-equivalence between the two phrases, thus producing three new correct alignments: ⟨*with care, con cura*⟩, ⟨*care, cura*⟩ and ⟨*with, con*⟩.

The results of this kind of optimization, however, as well as those of working at the clause level, vary widely depending on the quality of the parse trees and can be detrimental, in terms of precision, if applied to high-quality datasets.

5.1.4.2 Selecting relevant alignments

The pruning procedure described in Section 3.1.2.2.1 is aimed at filtering out alignments that, because of their number of occurrences and of the set of criteria thanks to which they have been extracted, are less likely to be correct. However, not all the alignments extracted in this way are relevant for the purposes of MT.

On the one hand, most alignments that do not contain any content word are not useful for the generation of a domain-specific grammar, as many common function words are already covered by the GF RGL[7].

On the other hand, the concepts represented in the GF grammar should intuitively, in most cases, be "as small as possible". If we reconsider the example in Figure 1.1 (sentences *"finding useful correspondences is not exactly a piece of cake"* and *"trovare corrispondenze utili non è proprio scontato"*), for instance, while the multi-word correspondence ⟨*a piece of cake, scontato*⟩ is indeed useful there is usually no point in keeping correspondences such as ⟨*useful correspondences, corrispondenze utili*⟩, even if they are correct, when it it also possible to extract the alignments ⟨*useful, utili*⟩ and ⟨*correspondences, corrispondenze*⟩: one-word and, in general, shorter alignments are more easily reusable in different contexts, especially since there are language-specific RGL rules capable of handling word order properly. As a consequence, for the purposes of grammar generation, the CA modules should in this case only return the following set of alignments:

1. ⟨*finding, trovare*⟩
2. ⟨*useful, utili*⟩
3. ⟨*correspondences, corrispondenze*⟩
4. ⟨*is, è*⟩
5. ⟨*not, non*⟩
6. ⟨*exactly, proprio*⟩
7. ⟨*a piece of cake, scontato*⟩.

If we consider that the RGL can already handle most function words correctly, we can even discard all alignments that do not contain any content word (in this case, alignments 4 and 5, which leaves us with only 5 alignment useful to GF).

In some situations, however, larger alignments (such as ⟨*useful correspondences, corrispondenze utili*⟩) are considered to be relevant since they help understand which words can be safely used in conjunction with each other. For instance, while the Italian adjective "*brutto*" (most often translated as "*ugly*") is often used in conjunction with the word "*notizia*" ("*news*") in expressions such as "*brutte notizie*", in English the adjective "*bad*" (as in "*bad news*") is definitely more appropriate. Discarding a potential ⟨*bad news*, *brutte notizie*⟩ alignment is then not always the right choice,

[7]small alignments containing both function and content words, such as prepositional phrases, are on the other hand extremely useful, as they show which specific function word should be used in a particular context.

since in a context in which there are many alternative translation equivalents for the word "*brutto*" it provides useful information about what adjective to use to refer to the noun "*news*".

Because of this, and since CE could have applications other than MT, selection in this sense is performed - optionally - after the extraction phase itself and implemented as an independent function. It works as follows: first, alignments that do not present any content word are filtered out. Then, among the remaining ones, *superficially* perfect alignments *containing* other extracted alignments are also discarded.

To clarify such algorithm, it is worth defining perfect alignment - which was already mentioned in Section 3.1.1 - more rigorously.

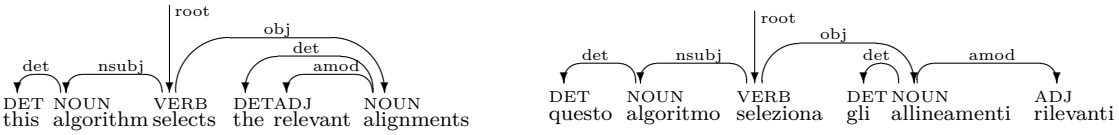
Definition 4 *An alignment $A = \langle e_1, ..., e_n \rangle$ is perfect if its member dependency trees $e_1, ..., e_n$ are identical both in their topology and in the dependency labels assigned to their nodes.*

The "shallow" version of perfect alignment the algorithm described above refers to, which we call *superficially perfect alignment*, occurs when the definition applies at least to the roots of the two member trees and the roots of their immediate subtrees.

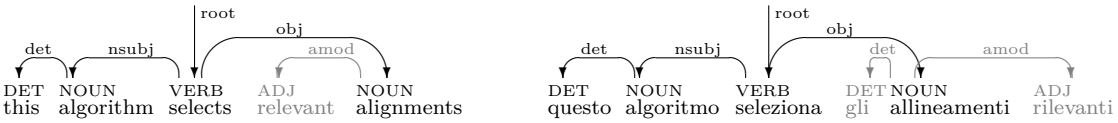
Example 22 *Consider the following pairs of English-Italian sentences:*

1. *⟨this algorithm selects the relevant alignments, questo algoritmo seleziona gli allineamenti rilevanti⟩*
2. *⟨this algorithm selects relevant alignments, questo algoritmo seleziona gli allineamenti rilevanti⟩*

While the two English sentences are translated to Italian in the same way, alignment is perfect only in the first case, where the determiner "the" is present. The two trees are in fact identical if not for the words themselves and their order.



In the second case, however, alignment is still "perfect" if we only compare the root node and its immediate dependents, thus ignoring the determiner "gli", which is absents in English, and the adjectival modifier "relevant/rilevanti":



Another concept introduced by this algorithm is that of alignments *containing* each other.

Definition 5 *An alignment $A = \langle e_1, ..., e_n \rangle$ contains an alignment $B = \langle f_1, ..., f_n \rangle$ if, for each of B's members, f_i is a subtree of e_i.*

Thus, the algorithm is safe - in the sense that it does not get rid of any potentially meaningful alignment - in that it gets rid of superficially perfect alignments whose subtrees are also members of smaller alignments. This means that when sentences vary widely syntax-wise, as it happens in cases where translation is not literal or in the presence of parse errors, the resulting extracted concepts can still contain a large number of nodes.

5.2 Evaluation

To evaluate the effectiveness of the translation method described in the previous section, a first, small-scale experiment was designed. The main idea is to automatically translate a set of English sentences to Italian and Swedish, make native speakers of the TLs correct the output sentences, so to obtain a set of reference, both grammatically and semantically correct translations, and compare them to the original machine-generated ones.

The remaining part of this chapter describes the experiment in detail. Section 5.2.1 deals with the automatically generated GF lexica in play, while Sections 5.2.2 and 5.2.3 describe the way the corpus of English sentences and their reference translations were obtained. After that, Section 5.2.4 introduces the evaluation metrics used in the experiments. Finally, the results of such experiments are presented in Section 5.2.5.

5.2.1 Lexica

For this first experiment, two distinct GF lexica, built starting from the alignments extracted from the DMI and CSE course plans corpora respectively, were generated as described in Section 5.1.2. Both were then enriched with the syntactic constructs listed in Section 5.1.2.1.

While building a trilingual grammar would have been optimal, the idea was abandoned because of the scarce amount of domain-specific terminology obtained by CE + CP, even making use of the adjustments described in Section 5.1.4, for all three languages, as we commented on in Section 4.2.1.1.

This means that only alignments obtained via CE are put to the text in this first experiment, while CP was not made use of at all. Using PUD treebanks instead of the two course plans corpora could have been a partial solution to the problem, but it would have resulted in a lexicon of terms and expressions belonging to the most disparate contexts. Most importantly, using manually annotated data would also have meant not testing all the key stages of the pipeline described in Section 2.2.3.2, where CP, unlike UD parsing, is seen as something optional.

5.2.2 The English corpus

Even avoiding CP, the small size of these corpora, the limitations of the language models used for parsing and the issues encountered in the UD-to-GF conversion make both the lexica resulting from this process relatively small. This, together

with the issues GF parsing suffers from, discussed in Chapter 2, makes working on arbitrary sentences taken from documents similar to those the course plans corpora are composed of extremely hard.

As a consequence, our choice for this first MT experiment was to generate the sentences to translate directly in the GF shell, making use of its random AST generation functionality. As discussed in Section 5.1.2.1, in fact, the (extended) CSE and DMI grammars allow for arbitrary variations - both grammatical and in terms of content words - over a small set of basic sentences.

Grammatical variation (e.g. turning indefinite forms of nouns into definite forms, altering the tense of verbs etc.) is meant to highlight how the output of GF translation typically consists of well-formed sentences. Changes of content words, on the other hand, allow to test a larger variety of extracted concepts.

While it is indeed possible to generate both an initial subset of sentences and the corresponding variations entirely randomly, care has been put in selecting only sentences that are semantically plausible. This is meant to facilitate the task of the human translators and to avoid having them insisting in trying to find some meaning in the sentences and correcting them using excessively rare word senses.

The results of this process are two small testing corpora, each consisting of 50 sentences in English, generated using the abstract syntax and the English concrete syntax of the DMI and the CSE grammar respectively. The TL concrete syntaxes of the two grammars (Italian in the former case and Swedish in the latter) were only used at a later stage, to make sure that all sentences did have a linearization in both the source and the target language. A linearization can in fact be missing, even though rarely, because the grammar generation stage omits linearization rules derived from pairs of trees that cannot share the same root category.

5.2.3 Reference translations

Reference translations were obtained by asking two native speakers of Italian and Swedish also proficient in English to compare the original English sentences to their automatically translated counterparts and to correct the latter. The two participants were instructed to only make the minimal changes necessary to obtain, starting from the output of the translation module described in 5.1.3 a set of grammatically and semantically correct translations.

This makes the reference sentences as similar to the automatically obtained ones to evaluate as possible, allowing a more meaningful automatic evaluation, as we will discuss in the following Section.

5.2.4 Evaluation metrics

The evaluation metric used for this experiment is the BLEU score [29], a widely adopted, simple, language-independent and inexpensive measure that claims good correlation with human judgments, as implemented in [26].

Following conventions, we report the cumulative n-gram scores for values of n from 1 to 4 (BLEU-1 to BLEU-4). However, being a significant portion of the sentences

of length 4 or less, we also report BLEU-1 to BLEU-3 scores, BLEU-1 to BLEU-2 scores and scores obtained considering unigrams only.

The way in which the reference translations were obtained is strongly connected to the choice of this specific evaluation metric. Without going into all the details, it is in fact important to know that the BLEU score is computed by counting matching n-grams in the candidate translation to n-grams in the reference text. This means that, if the reference translations are obtained independently from the automatic ones, BLEU scores can easily become misleading.

In a preliminary experiment conducted in this way, for instance, the Italian translation "*il docente discute la Didattica*" of the English sentence "*the teacher discusses the didactics*", was given the minimum score despite being completely correct and even idiomatic[8]. Looking at the two reference translations used at the time, it appears clear that such a low score is due to the capitalization of the word "Didattica" and the arbitrary (but equally correct) lexical choices made by the human translators, who rendered the sentence as "il professore discute i metodi didattici" and "l' insegnante discute la didattica"[9]. Coincidentally (but not unlikely for a language like Italian), this is a case in which these lexical differences affect the choice of the determiners used in the various translations (cf. "l'insegnante" vs. "il professore" and "la didattica" vs. "i metodi didattici"), producing a BLEU-1 to 4 score of 0.

5.2.5 Experimental results

Corpus-level BLEU scores for the automatic translations of the 50+50 sentences of the testing corpora generated as described in Section 5.2.2 are summarized in Table 5.1.

	DMI (en-it)	**CSE (en-sv)**
BLEU-1 to 4	55.4	61.27
BLEU-1 to 3	62.75	67.77
BLEU-1 to 2	70.6	74.3
BLEU-1	79.33	80.99

Table 5.1: BLEU scores obtained by comparing one candidate automatic translation per sentence to a reference translation obtained by making the necessary corrections to the automatically generated ones. Both the initial English sentences and their automatic translations were generated using the two course plans grammars.

These synthetic figures are useful to give an idea of the general quality of the translations: overall, although with relatively low scores, English-to-Swedish translation

[8]typically, the adjective "didattica" is not use in its plural form "didattiche" when nominalized. The capital D is often used, for this particular term, in official university documents.

[9]The terms "docente", "professore" and "insegnante" are almost perfect synonyms, even though "docente" is mostly used in formal documents to refer to university lecturers, "professore" is more common in secondary school and colloquial speech, and "insegnante" is the most generic. As such, it can also refer to elementary school teachers and in some cases even sport coaches. The expression "metodi didattici" is a perfectly sensible replacement for "didattica".

works significantly better than English-to-Italian. Looking back at the results obtained for CE (cf. Section 3.2), this is not excessively surprising, since both precision and recall for English-Swedish are consistently higher than they are for English-Italian. When it comes to the results obtained for CE on the course plans corpora in particular (cf. Table 3.3), however, the difference in precision between the two language pairs is negligible. This makes the reason for the substantial difference observed in the BLEU scores obtained for the two different corpora unclear, at least at a first glance.

Looking at sentence-level scores is, however, sometimes more interesting. Regardless the corpus, scores assigned to individual segments range from the minimum possible value of 0 to the perfect score of 100, which indicates a perfect correspondence between the automatic and the reference translation.

Examples of sentences that were assigned a perfect BLEU-1 to 4 score are "*the library provides useful textbooks*" (translated to Italian as "*la biblioteca fornisce libri utili*") in the DMI corpus and "*this lab is more difficult than the exam*" (whose Swedish translation is "*den här laborationen är svårare än tentamen*") in the CSE corpus.

On the other hand, it is easy for shorter sentences to be assigned the minimum BLEU-1 to 4 score even when they only contain a single grammatical or semantic error. This is the case, for instance, of the sentence "*the test is oral*", whose last word, "*oral*" is translated as "*dura*" ("*hard*") instead of "*orale*" due to an alignment error. Nonetheless, it is worth noticing that this is one of the many cases in which, even though the fact that only the first translation candidate is taken into account hides it, the correct alignment ⟨"*oral*","*orale*"⟩ is also found by the CE module. Moreover, the such valid correspondence has a higher number of occurrences and matches stronger alignment criteria than the wrong one ⟨"*oral*","*duro*"⟩. This suggest that, in contexts where higher precision is necessary, there are obvious improvements that can be done either in the choice of the alignment criteria or at a later stage, when relevant alignments are selected.

5.2.5.1 Error analysis

A problem with using the BLEU score as the only evaluation metric is the fact that it makes no distinction between content and function words, thus not allowing an evaluation focused specifically on the extracted concepts. The small size of the corpus, however, allows for some error analysis. While postprocessing the automatic translations, the participants were asked to indicate what kind of errors they encountered in each sentence (grammatical, semantical or both). Their observations are summarized in Table 5.2.

Interestingly, while most errors are due to wrong alignments, the main difference between two corpora lies in the number of translations that only contain grammatical errors. This explains the significant difference observed in the BLEU scores.
In Italian, grammar errors often involve contractions such as "*del*" ("*di*" + "*il*", in English "*of the*"), some of which are systematically rendered as two separate words. Another common case is that of wrong (or at least very confusing) adjective collocation, such as in the translation "*il libro presenta una tecnica con miglioramenti*

type of errors	DMI (en-it)	CSE (en-sv)
semantical	23 (46%)	23 (46%)
grammatical	10 (20%)	3 (6%)
semantical and grammatical	3 (6%)	4 (8%)

Table 5.2: Number of automatically translated sentences containing only semantical, only grammatical and both semantical and grammatical errors in the two synthetic course plans corpora.

vari utile" ("*the textbook presents a useful technique with various improvements*"), where the adjective "*utile*" ("*useful*"), referred to "*technique*" ("*tecnica*") is placed far from such noun, making the sentence hard to interpret[10]. Grammatical errors in Swedish are less common and, apart from long adjectives never being turned to the comparative degree periphrastically (e.g. "*relevantare*" instead of "*mer relevant*"), less systematic. Only in one case, for instance, ("*programbiblioteken*"), gender is incorrect.

Some alignment errors are also interesting to analyze. In English-Swedish, while several compounds, such as ⟨*computer science, datavetenskap*⟩, are aligned correctly, there are cases in which a single-root noun is translated as a compound (e.g. ⟨ *theory, automatateori* ⟩). This is not necessarily an actual alignment error, as it might rather be that the English version of the text was being less specific than its Swedish counterpart, thus producing an alignment that, during evaluation, would have been marked as correct but not generally reusable (=).

"Reasonable" alignment errors appear in the DMI corpus sentences too. The English for "*attendance of lessons*", for instance, becomes simply "*frequenza*" in the Italian translation. Such word is in fact often used alone to replace longer expressions such as "*frequenza delle lezioni*", just like in English '*of lessons*" can be omitted when what must be attended is evident from the context. Another interesting example is the alignment ⟨ *class, classe* ⟩, which causes the sentence *"I will attend the class"* to be (incorrectly) translated as *"io seguirò la classe"* instead of *"io seguirò la lezione"* even though the correspondence is in fact valid in most of the numerous contexts in which *"class"* is not to be intended as a synonym of *"lesson"*.

[10] the manually postprocessed translation is "*il libro presenta una utile tecnica con miglioramenti vari*".

6

Conclusions

The main objective of this thesis project was to develop a syntax-based Concept Alignment system and put it to the test in the context of domain-specific MT. The approach we propose basically consists in analyzing texts with a dependency parser, UDPipe, and comparing the resulting trees. For the MT evaluation, such dependency trees are converted to Abstract Syntax Trees, from which the rules of a multilingual generative phrase-structure grammar are derived. The framework we make use of when it comes to this grammar, GF, makes having a grammar sufficient to perform simple MT experiments.

The tangible fruit of this work is a Haskell library, as well as a number of executables offering an easy to use and configure interface to perform both variants of CA, extraction and propagation, and a variety of kinds of evaluations.

To assess the quality of the alignments obtained with such system, the extraction and propagation components were first tested without taking the target application context into account.

When it comes to CE, the core part of this work, comparison against a pre-existing basic implementation shows significant improvements in terms of both the quality and the quantity of the resulting alignments. With respects to traditional statistical approaches, one of the most useful consequences of our syntax-based approach is that it makes it possible to identify correspondences between multiword, even discontinuous expressions. Another important advantage over standard word alignment techniques is that, even though it can still benefit from large amounts of data, our system works consistently well even when run on extremely small corpora. This is confirmed by one of our experiments, where we restrict it so to only extract one-to-many and many-to-one alignments and compare it with `fast_align`. Furthermore, the alignment criteria our software makes use of are easy to modify, add and remove, paving the way to a variety of experiments.

CP, a less studied variant of CA, has been explored in two different scenarios. On the one hand, it can be useful to generate large multilingual lexica if applied to more-than-bilingual parallel corpora. On the other, it can also be used to identify shared terminology between two bilingual corpora having one language in common. The first experiments show promising results, but suffer from the scarce availability of sentence-segmented large bilingual corpora.

Finally, the CA module was adapted to the task of MT. The results of the first, small-scale experiments conducted in this sense highlight the strengths of GF in terms of NLG but also indicate the need for a more aggressive alignment selection strategy, as the numerous correct alignments are sometimes overshadowed by

alternative incorrect ones.

Future work

These results, while encouraging, suggest that there is still much room for improvement and that work in this area can be carried on in many different directions even constaining ourselves to CA *per se*, without focussing on its MT applications.

Further integration with statistical alignment techniques

With regards to the CA component itself, it is important to remember that both the quantity and the quality of the alignments are a direct consequence of those of the dependency trees they are obtained from. Apart from working on dependency parsers directly, a possibility is to experiment more with hybrid alignment approaches, integrating the results of statistical methods (which only depend on the quantity and quality of raw data rather than on that of their analyses) with those of our rule-based system. One way to do that has already been implemented, but has not been made use of in the final MT experiments due to the small size of the corpora involved.

Aligning verb phrases

Early in this work we noticed how only aligning subtrees is not enough to identify all the concepts that are present in a pair of sentences. As a consequence, significant effort has been made in also aligning subtree heads correctly. While this increases the number of concepts the program is able to find, it does not cover all cases. For instance, aligning complete verb phrases (i.e. verbs together with their arguments) could prove extremely useful. Generalizing the approach used for head alignment could be a way of addressing the issue and would make it possible to use CA to construct "rich" lexical entries for verbs, whose lack, as discussed in Chapter 3, currently makes it hard to handle some types of common translations divergences.

Iterative CA

The current implementations of CE and CP try to apply a set of priority-sorted criteria one after another to each pair of sentences in the corpus, allowing for later ranking of the alignments obtained based on the criteria their members match. Another possibility is to go through the entire corpus multiple times, for instance using only the strictest criteria first and only falling back to the less reliable ones at later iterations, until no more alignments are found. The same hypothetical iterative algorithm can have the size of the alignments, and not (or not only) the criteria they have to satisfy, as a parameter.

Optimizing propagation for multilingual corpora

CP was not the main focus of this work. As noted in Chapter 4, however, optimizing it for the simpler scenario in which it is applied to a more-than-2-lingual parallel text is straightforward and would improve both the quality of the results and, arguably, running time. Once this is done, generating multi-, as opposed to bi-, lingual domain-specific grammars will be more realistic, especially if `gf-ud`'s grammar generation functionality will also be further developed in parallel.

Generalizing CE to n languages

As an alternative to this optimization, CE could be generalized to an arbitrary number of languages. The expectation is for the resulting concepts to be of slightly larger size than those obtained with the current system, since comparing more trees simultaneously increases the possibility of encountering small divergences, this term being intended in its broad sense. This approach would probably help identifying longer idiomatic expression and could provide a better way to deal with multilingual corpora, even though, while intuitive, it does present some implementation challenges.

Stricter and language pair-specific criteria

When it comes to using the concepts as actual translation units, getting rid of incorrect alignments is crucial. While doing so manually is still more feasible than extracting them manually, it would be desirable to devise better alignment selection policies or stricter alignment criteria. One idea would is to tune the alignment criteria for the language pair at hand, both by removing the noisier ones for semantically related languages and by adding language-pair specific criteria. The criteria-specific statistics reported in Section 3.2 can serve as a starting point for work in this direction.

Bibliography

[1] Multilingual parsing from raw text to universal dependencies.

[2] Universal dependencies documentation.

[3] Universal pos tags. *UD documentation*. `https://universaldependencies.org/u/pos`. Accessed: 2020-11-04.

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

[5] Yaser Al-Onaizan, Jan Cuřín, Michael Jahr, Kevin Knight, John D. Lafferty, I. Dan Melamed, Franz-Josef Och, David Purdy, Noah A. Smith, and David Yarowsky. Statistical machine translation. Technical report, John Hopkins University Summer Workshop `http://www.clsp.jhu.edu/ws99/projects/mt/`, 1999.

[6] Ted Briscoe, John A Carroll, and Rebecca Watson. The second release of the rasp system. In *Proceedings of the COLING/ACL 2006 interactive presentation sessions*, pages 77–80, 2006.

[7] Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[8] Sabine Buchholz and Erwin Marsi. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the tenth conference on computational natural language learning (CoNLL-X)*, pages 149–164, 2006.

[9] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

[10] 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, 2005.

[11] Noam Chomsky. *Syntactic structures*. Walter de Gruyter, 1957.

[12] Haskell B Curry. Some logical aspects of grammatical structure. *Structure of language and its mathematical aspects*, 12:56–68, 1961.

[13] 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 *LREC*, volume 14, pages 4585–4592, 2014.

[14] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating typed dependency parses from phrase structure parses. In *Lrec*, volume 6, pages 449–454, 2006.

[15] Bonnie Dorr. Machine translation divergences: A formal description and proposed solution. *Computational linguistics*, 20(4):597–633, 1994.

[16] Chris Dyer, Victor Chahuneau, and Noah A Smith. A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, 2013.

[17] Eriksson, Gabrielsson, Hedgreen, Klingberg, Vestlund, and Ödin. Grammar-based translation of computer science and engineering terminology. 2020.

[18] Nizar Habash and Bonnie Dorr. Handling translation divergences: Combining statistical and symbolic techniques in generation-heavy machine translation. In *Conference of the Association for Machine Translation in the Americas*, pages 84–93. Springer, 2002.

[19] Nitin Indurkhya and Fred J Damerau. *Handbook of natural language processing*, volume 2. CRC Press, 2010.

[20] Michael Johnson. Compositionality. *The Internet Encyclopedia of Philosophy.*

[21] Aravind K Joshi and Yves Schabes. Tree-adjoining grammars. In *Handbook of formal languages*, pages 69–123. Springer, 1997.

[22] Hiroyuki Kaji, Kida Yuuko, and Yasutsugu Morimoto. Learning translation templates from bilingual text. In *COLING 1992 Volume 2: The 15th International Conference on Computational Linguistics*, 1992.

[23] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[24] Prasanth Kolachina and Aarnte Ranta. From abstract syntax to universal dependencies. In *Linguistic Issues in Language Technology, Volume 13, 2016*, 2016.

[25] Jan Landsbergen. Machine translation based on logically isomorphic montague grammars. In *Coling 1982: Proceedings of the Ninth International Conference on Computational Linguistics*, 1982.

[26] Lena Morgenroth. Comparison and implementation of mt evaluation methods. 2011.

[27] Makoto Nagao. A framework of a mechanical translation between japanese and english by analogy principle. *Artificial and human intelligence*, pages 351–354, 1984.

[28] Franz Josef Och and Hermann Ney. Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting of the Association of Computational Linguistics (ACL)*, 2000.

[29] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[30] Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*, 2011.

[31] Chris Quirk, Arul Menezes, and Colin Cherry. Dependency treelet translation: Syntactically informed phrasal smt. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 271–279, 2005.

[32] Owen Rambow and Giorgio Satta. Synchronous models of language. *arXiv preprint cmp-lg/9605032*, 1996.

[33] Aarne Ranta. Grammatical framework. *Journal of Functional Programming*, 14(2):145–189, 2004.

[34] Aarne Ranta. *Grammatical framework: Programming with multilingual grammars*, volume 173. CSLI Publications, Center for the Study of Language and Information Stanford, 2011.

[35] Aarne Ranta. Explainable machine translation with interlingual trees as certificates. In *Proceedings of the Conference on Logic and Machine Learning in Natural Language (LaML 2017)*, pages 63–78, 2017.

[36] Aarne Ranta. *Computational Grammar: An Interlingual Perspective*. Unpublished, 2020.

[37] 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, 2017.

[38] Stuart Shieber and Yves Schabes. Synchronous tree-adjoining grammars. In *Proceedings of the 13th international conference on computational linguistics*. Association for Computational Linguistics, 1990.

[39] Milan Straka, Jan Hajic, and Jana Straková. Udpipe: trainable pipeline for processing conll-u files performing tokenization, morphological analysis, pos tagging and parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 4290–4297, 2016.

[40] Zoltán Gendler Szabó. Compositionality. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2020 edition, 2020.

[41] Lucien Tesnière. Éléments de syntaxe structurale. 1959.

[42] Jörg Tiedemann. Bitext alignment. *Synthesis Lectures on Human Language Technologies*, 4(2):1–165, 2011.

[43] Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational linguistics*, 23(3):377–403, 1997.

[44] Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 523–530, 2001.

[45] Daniel Zeman. Reusable tagset conversion using tagset drivers. In *LREC*, volume 2008, pages 28–30, 2008.

A

Universal POS tags and dependency labels

While some of them are also discussed briefly in Section 2.1.3.1, this appendix provides systematic definitions for all the main UD Universal Part Of Speech tags and dependency relations used in the examples of this work. Complete lists and a good number of examples in several languages can be found in [2] and [36].

A.1 `UPOS` tags

The official UD documentation [2] distinguishes between *open class* or *content* words and *closed class* or *function* words.

A.1.1 Open class words

When it comes to open class words, the categories that are relevant to this work are the following:

- `NOUN`, the class of words typically denoting a person, place, thing, animal or idea. Unlike nouns in traditional grammars, `NOUN`s are intended for *common* (as opposed to *proper*) nouns only
- `PROPN`, the class of nominals used as names (or part of names) of a specific individual, place, or object, i.e. *proper nouns*
- `VERB`, designating events and actions. This category refers to content verbs only, while auxiliaries should be assigned the tag `AUX` (cf. A.1.2)

A.1.2 Closed class words

Among closed class words, the categories used in the examples of this thesis are:

- `PRON`, defined as in traditional grammars: the class of substitutes for nouns or noun phrases, whose meaning is recoverable from the context
- `AUX`, the class of function words that accompany the lexical verb of a verb phrase and express grammatical distinctions not carried by such lexical verb. The most common example of verbs that fall into this category is perhaps that of *tense auxiliaries* (e.g. "*has*" in the verb phrase "*has completed*"), but *passive auxiliaries*, like "*was*" in "*was completed*, modal auxiliaries like "*must*" and "*should*" and verbal copulas should also be tagged as `AUX`

- `DET`, a wide-coverage class for the different types of determiners, i.e. words that modify nouns or noun phrases expressing the reference of the noun phrase in context. It includes articles, pronominal numerals and words like *all* and *every*.

Finally, UD groups together categories that apply to tokens that are not even necessarily considered words in traditional grammars, such as symbols. The only category of this kind mentioned in this work is `PUNCT`, which refers of course to punctuation.

A.2 DEPRELs

Dependency relation are used to label links connecting the word nodes of a dependency tree with their heads, indicating the syntactic relation occurring between them.

The only "exceptional" label is `root`, which connects a fake node to the `root`, i.e. usually the main `VERB`, of a sentence. To avoid discrepancies between languages, the `root` of a sentence is always supposed to be a content word, meaning for instance that, if the main verb is a copula, it is its complement that ought to be labelled `root`.
The other dependency labels we refer to in this text can, like POS tags, be grouped according to the nature of the nature of the dependent they link to a head.

A.2.1 Open class dependents

Open class words can be assigned various kinds of labels:

- labels linking the core arguments of a verb to the verb they refer to (i.e., in case of a single-clause sentence, to its `root`):
  - `nsubj`, indicating the nominal subject of a clause (a noun, proper noun, pronoun or numeral)
  - `obj` and `iobj`, indicating its direct (resp. indirect) object. With *direct object* we refer to the noun phrase that denotes the entity acted upon or which undergoes a change of state or motion, while the *indirect object* of a verb is a nominal phrase that is a core argument of the verb but is not its subject or (direct) object
- `obl`, linking complements (i.e. non-core arguments, introduced by prepositions) to the verb they refer to
- labels for modifier words:
  - `amod`, linking adjectives to the nominals they modify
  - `advmod`, linking adverbs to the words (which can belong to several categories) they modify
  - `nmod`, used for marking the relation between a nominal and another `NOUN`, `PNOUN` or noun phrase
  - `nummod`, linking number phrases to the `NOUN` they modify with a quantity
- `flat`, used to connect the words following (in terms of position in the sentence) the head (i.e. the first word) of a flat multiword expression, and `compound`, playing a similar role in compound written as two or more separate words

- labels linking non-`root` verbs to other verbs, specifying what kind of clause they are heads of:
  - `csubj`, for clausal subjects
  - `ccomp`, for clausal complements, i.e. clauses that function as objects of a verb
  - `xcomp`, for open clausal complements, i.e. clausal complements whose subject is determined by the higher clause
  - `advcl`, for adverbial clauses modifying a predicate
  - `acl`, for clauses modifying a nominal

A.2.2 Closed class dependents

When it comes to closed class words, commonly used dependency labels are:

- `cop`, linking copulas to their complements
- `aux`, linking an `AUX` to the lexical verb it refers to
- `det`, marking the link between a determiner and the nominal it refers to
- `case`, linking words (such as pre- and postpositions) marking the case of a `nmod` to the `nmod` itself in languages that do not express case morphologically

Finally, corresponding to the POS tag `PUNCT` is `punct`, which links punctuation marks to the head of the clause they belong to.

B

Dependency configurations

This appendix provides the complete list of the abstract dependency configurations used for the UD-to-GF conversion of the alignments extracted by the CA module.

B.1 Category annotation

Category annotations describe a mapping between GF categories and UD `UPOS` tags (cf. Appendix A.1).

```
#cat A        ADJ primary
#cat Adv      ADV primary
#cat Det      DET primary
#cat N        NOUN primary
#cat PN       PROPN primary
#cat Prep     ADP primary
#cat Pron     PRON primary
#cat V        VERB primary
#cat Conj     CCONJ primary
```

B.2 Function annotations

Function annotations, on the other hand, put GF functions in relation with UD `DEPREL`s (cf. Appendix A.2). The following list also serves the purpose of enumerating the types of concepts GF lexica automatically generated as described in Chapter 5 consist of.

```
#fun AdjCN       amod head
#fun AdvVP       head advmod
#fun ComplV      head obj
#fun DetCN       det head
#fun AdvCN       head advmod
#fun PredVP      nsubj head
#fun PrepNP      case head
#fun PrepCN      head nmod
#fun FlatPN      head flat -- PN -> PN -> PN
#fun CompoundN   compound head
#fun PrepPP      head obl
```

C

Installing, using and configuring the CA module

This appendix provides installation and usage instructions for the CA software developed as part of this thesis project, whose source code, as well as this report, is available at `https://github.com/aarneranta/concept-alignment`.

C.1 Installation

To compile the CA module, we recommend the Haskell Stack[1]. To build the project, clone the above mentioned GitHub repository, move into the corresponding directory and run `stack build`. This will generate a haskell library, named `concept-alignment`, and the following five executables:

- `extract-concepts`, the CE module described in Section 3
- `propagate-concepts`, the CP module described in Section 4
- `evalign`, a script for evaluating CP and CE
- `generate-grammar`, for automatically generating a grammar as described in Section 5.1.2
- `translate`, the simple MT module presented in Section 5.1.3

C.2 Usage

C.2.1 `extract-concepts`

Using the CE module with the default parameters is straightforward:

```
stack exec -- extract-concepts SL.conllu TL.conllu
```

The program, however, supports a number of command-line options. Most importantly:

- `-file=FILE` can be used to specify where to write the resulting alignments. Unless otherwise specified (see below), the output consists in two new aligned `.conllu` files, stored at the chosen location, whose names are prefixed with `SL` and `TL` respectively
- `-linearize` specifies that concepts should be in the linearized format described in Section 3.2.3, useful for evaluation purposes

[1] `https://docs.haskellstack.org/en/stable/README/`

- `-maxsize=INT` sets the maximum size of the extracted alignments
- `-all` specifies that the selection step described in Section 5.1.4.2 should be skipped
- `-clauses` enables clause segmentation (cf. Section 5.1.4.1.1)
- `-rest` enables a second pass of alignment "by exclusion" (cf. Section 5.1.4.1.2)
- `-pharaoh=FILE` is used to specify a file in pharaoh format (cf. Figure 2.7) to use as backup

C.2.2 `propagate-concepts`

CP requires two, instead of three, `.conllu` files, one with the concepts to propagate and the following two containing the corpus of annotated sentences where their should be looked for. As a consequence, running it with the default parameters should be done as follows:

```
stack exec -- propagate-concepts SL_concepts.conllu SL.conllu
TL.conllu
```

All `extract-concepts` options that are also relevant for CP (i.e. all of them excepts `-maxsize` and `pharaoh`) are also valid for this second executable.

C.2.3 `evalign`

The evaluation scrip `evalign` can be run in three different "modes":

1. single-file mode (`stack exec - evalign annotated.txt`): given a pre-annotated file in the linearized format described in Section 3.2.3, it prints out basic statistics about precision, recall and amount of reusable alignments.
2. extraction interactive mode (`stack exec - evalign extraction annotated.txt new.txt`): given an annotated and a new, possibly yet-to-annotate file containing linearized alignments, it allows interactive minimal annotation of the latter (if needed) and, on top of printing out the basic statitics, it compares the new alignments to the old ones, telling how many correct and incorrect alignments were lost and/or found
3. propagation interactive mode (`stack exec - evalign propagation annotated.ca new.ca`): similar to extraction interactive mode, excepts that the statistics are CP-specific (percentage of successfully propagated alignments, number of errors introduced by CP etc.)

The command line option `-reasons` can be used when criterion-wise statistics are needed.

C.2.4 `generate-grammar`

The grammar generation module can be run as follows:

```
stack exec -- GenerateGrammar path_to_extract_grammar
paths_to_morphodicts paths_to_aligned_conllu_files
```

Here, `path_to_extract_grammar` should point to the extraction grammar in `.pgf`[2]

[2]Portable Grammar Format, generated by the GF compiler.

format. `paths_to_morphodicts` stands for the path of the morphological dictionaries of the languages involved, also in `.pgf` format. Similarly, `paths_to_aligned_conllu_files` is to be replaced with the files containing the concepts in CoNNL-u format. These files should be, in terms of the languages they are written in, in the same order as the morphological dictionaries.

C.2.5 `translate`

The simplicity of the translation module makes it also extremely easy to use. The string or list of newline-separate strings to translate comes from the standard input and the only argument is the path to the automatically generated GF grammar to be used, again in `.pgf` format. Here is an example of using the program to translate a single sentence:

```
echo "this sentence will be translated" | stack exec -- translate
Extracted.pgf
```

C.3 Configuration: modifying the alignment criteria

As mentioned in Section 3.1.2.1, modifying the alignment criteria the program makes use of requires little effort. All criteria are in fact defined in a separate Haskell module `Criteria`[3] that exports only the list of criteria to be used. In the current implementation, it looks like this:

```
criteria :: [Criterion]
criteria = [udpos, ud, divs, pass, pos]
```

Removing and/or changing the priority of the criteria is then just a matter of altering such list.

Adding new criteria is also simple, but it requires an understanding of the data type `Criterion`, defined in module `ConceptAlignment`:

```
data Criterion = C {
    func :: UDTree → UDTree → Bool,
    reas :: S.Set Reason,
    headAlign :: Bool,
    strict :: Bool
}
```

As the above code block shows, `Criterion` is a record type whose fields are:

- a boolean function `func` specifying a rule to decide whether two trees should be aligned. For instance Criterion 1 (UD label matching) is implemented as

```
udMatch :: UDTree → UDTree → Bool
(RTree n ts) `udMatch` (RTree m us) =
    udDEPREL n == udDEPREL m
```

[3] `https://github.com/aarneranta/concept-alignment/blob/master/Criteria.hs`

- a set of `Reason reas` to be used for ranking the alignments, shown in the linearized files and used to compute criterion-specific statistics. Reasons, and not alignment rules (`func`s), are what mirrors criteria as defined in Section 3.1.2.1 exactly. A `func` can define in fact a more specific alignment rule, such as "type of categorial divergence in which an adjective is replaced by adverb", while a `Reason` is in practice a more coarse-grained label associated to potentially many rules
- two boolean flags, `headAlign` and `strict`, the former specifying whether head alignment should be performed for UD tree pairs matching that criterion, the latter marking the criterion as either strict (i.e. to be also used for "alignment by exclusion", cf. Section 5.1.4.1.2) or not.