Master thesis project proposal

Efficiently converting from UD to GF

Andreas Källberg anka.213@gmail.com

Suggested Supervisor at CSE: Aarne Ranta

Relevant completed courses student 1:

TIN092 Algorithms
TDA342 Advanced functional programming
DAT151 Programming language technology

March 15, 2023

1 Introduction

Grammatical Framework[**?**] (GF) is a formalism for describing natural language grammars as code, which allows converting between natural language and a language-independent abstract syntax. It is split into a generic resource grammar that covers morphology and the language as a whole and application grammars for a more narrow domain which allows a more semantic abstract syntax. When you try to write a grammar that covers a very wide domain, you will often get an over-generating grammar where each sentence can be parsed in very many ways into many different trees, where usually only one is the intended way.

Universal Dependencies[**?**] (UD) is another grammar formalism which uses machine-learning for parsing, which allows it to use context more in order to guess which interpretation was intended. The training data is based on a large set of sentences, which have been manually tagged with labels and a tree structure.

While the machine-learning based approach of UD allows it to guess the correct tree better for ambiguous sentences and allows it to handle grammatically incorrect sentences better, GF is much more capable when it comes to performing transformations on the sentences, while maintaining correct morphology and grammar. This makes it attractive to parse sentences using UD and then convert the parsed trees to GF trees in order to perform further transformations.

There exists a proof-of-concept implementation of a tool for converting between the trees for GF and UD, with the help of so called labels files which describe the mapping between UD labels and GF functions, called gf-ud, which contains both a component for converting from UD to GF, called ud2gf[**?**] and a component for converting from GF to UD, called gf2ud[**?**]. This work will focus on the ud2gf component.

The labels files can also contain macros, which allows constructing virtual GF functions during the translation from UD, which will be expanded to real GF functions at the end of the translation. These can, among other uses, be used to preserve information from the UD labels about subtrees, which can then be used at later points of the transformation to ensure that the desired GF tree is produced.

2 Problem

The current ud2gf implementation has some limitations. It quickly becomes extremely slow for sentences more than a couple of words and/or when using large GF-grammars, e.g. ones containing Wordnet[**?**]. Additionally, if the structure differs too much between the representation of a sentence in UD format and as a GF tree, it is not possible to describe the required transformation in the current "labels file" language.

```
1  cute  cute  ADJ  JJ  Degree=Pos  0  root  _  FUN=cute_A
2  ,  ,  PUNCT  ,  _  3  punct  _  _
3  fluffy  fluffy  ADJ  JJ  Degree=Pos  1  conj  _  FUN=fluffy_A
4  and  and  CCONJ  CC  _  5  cc  _  FUN=and_Conj
5  furry  furry  ADJ  JJ  Degree=Pos  1  conj  _  FUN=furry_A
```

Figure 1: The phrase "cute, fluffy and furry" as a textual UD tree

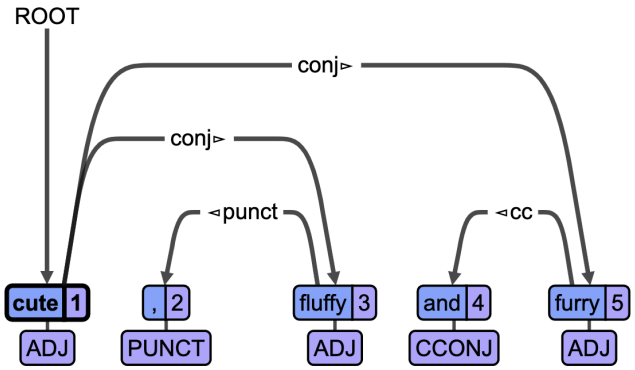

Figure 2: The phrase "cute, fluffy and furry" as a UD tree in graphical format

For example, the adjectival phrase "cute, fluffy and furry" would be described in UD format as in Figures 1 and 2.

while the GF version of the same tree, shown in Figure 3, would look like this:

```
ConjAP and_Conj (ConsAP (PositA cute_A)
                        (BaseAP (PositA fluffy_A) (PositA furry_A)))
```

Here we can see that in UD, the word "cute" is in the root, while the conjunction "and" is at the bottom of the tree, while in GF the conjunction is a direct child of the root. This transformation can not be preformed by the simple single-layer transformations that are available in the current macro-system for labels files.

The translation described by a labels file is not one-to-one and there are often many possible GF trees that a UD tree could be translated to. The possible trees are currently ranked by completeness, as in how many of the words are included in the generated tree. However this ranking is incomplete and in case two possible trees, with the same GF category, cover the same words, an arbitrary tree will be chosen. A better choice could be to also check the linearization of these trees and rank those whose linearization is more similar to the original string higher. It would also be possible to completely exclude trees with differing linearization, but that would run counter to the goal of robustness.

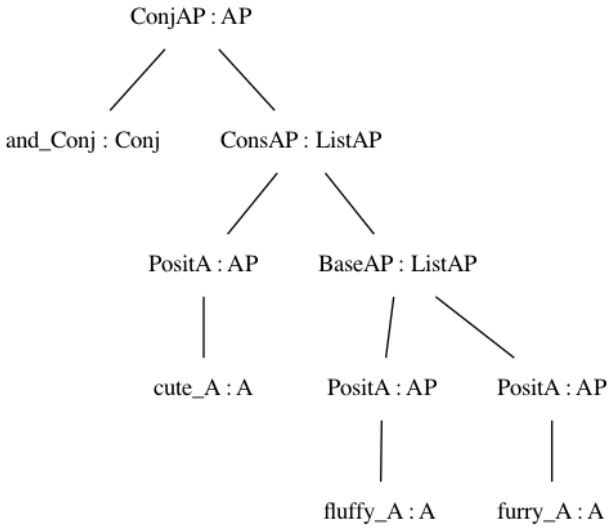

Figure 3: The phrase "cute, fluffy and furry" as a GF tree in graphical format.

Another problem is that it can sometimes be difficult to debug why a rule in a labels file is not firing, so it would be useful to have a debugging tool to help diagnosing such issues.

3 Context

This work is mostly a continuation of the work in [**?**].

A practical problem for which this tool can be useful can be found in [**?**]. In the Future Work section, under "Robust fall-back options", gf2ud is mentioned as a possible solution to making the parser more robust.

4 Goals and Challenges

1. Analyze algorithms and improve performance. The main challenge here is to find an algorithm for finding matching trees without exponential complexity on the number of children of a node in the UD tree.

2. Improve flexibility of macro language, allowing changing the structure of the trees while translating from GF to UD. One challenge here is in figuring out either how to change the algorithms to support these more advanced transformations or to find a way to allow them without needing to change the algorithms.

3. Write a debugging tool, which analyzes exactly what it is that prevents a rule in a labels file from firing or what prevents that tree from being selected. One challenge here is how to explain to the user the issue for all the possible things that can go wrong. There is also an engineering challenge in making an algorithm that figures out what went wrong and why.

4. If there is time, update the algorithm to look at the linearization in order to try to select trees that match the original string as closely as possible and evaluate what difference this makes. This is only possible with improved performance from goal 1, but keeping it fast will still be a challenge. There are also some challenges here in how it should interact with the advanced macros in goal 2. We also need to handle when a GF tree has multiple linearizations, e.g. if it results in a conjugation table.

5 Approach

5.1 Performance

1. Finding the main source of slowness, which was done with profiling.

2. Analysing the current algorithms, which are based on brute-force, trying all combinations, with some simple filtering.

3. Finding a better algorithm, which avoids exploring paths that could never be the correct answer and which avoids duplicate work.

4. Analysing algorithmic complexity of both algorithms and testing the practical performance to confirm the results.

5.2 Flexibility

In order to allow changing the shape of trees when translating from UD to GF, the macro language needs to be expanded. A first prototype of this can be done with minimal code changes by making macro expansion recursive and then representing the code for the transformation in Church-encoding, inspired by lambda-calculus.

This approach can be evaluated by seeing how well it covers different tree shape changes for different trees one would encounter.

It could also be worthwhile to make a more user-friendly version of the advanced macros that can be understood without knowing about Church-encoding

5.3 Debugging tool

Go through each component of the algorithms in order to find where applying a rule can go wrong and add detection for them. Additionally try out the

debugging tool on a real grammar, e.g. in the context of [**?**], in order to find edge-cases which were not handled by the debugging tool.

5.4 Linearization-aware translation

Here we need a way to determine which linearization is actually closer and when to keep multiple options for a later stage of the translation. Some care also needs to be taken in determining which trees can be tossed away and which ones need to be kept around for a later stage.

Evaluating if the results are better with this version can be done in a large part by comparing the input string with the resulting string after translating to GF and linearizing.

6 References