UNIVERSITÉ DE LORRAINE

Institut des sciences du Digital Management et Cognition

Going from UD towards AMR

Realisation Report

28 May 2019

for

UE 805 EC1 Supervised Project

(M1 Natural Language Processing)

by

Kelvin Han
Siyana Pavlova

Supervisors and Reviewers

Supervisors : Bruno Guillaume
Maxime Amblard

Reviewer : Chloé Braud

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Acknowledgements

We wish to thank our supervisors, Bruno and Maxime, for providing with us invaluable guidance and support in the project. We are grateful for their kindness in sharing their knowledge with us, for spending time to introduce us to the concepts of Abstract Meaning Representation and graph rewriting, guiding us, as well as pointing us towards exploring other topics and issues related to the direction of our project. We also wish to thank in advance our reviewer, Chloé, for taking the time to review our report.

Contents

Chapter 1

Introduction

This report presents our efforts to develop a system that takes sentences in natural language, parses them in the Universal Dependencies (UD) framework and applies a set of rewrite rules on the UD parses to produce Abstract Meaning Representations (AMRs) of the sentence. The rewriting system is supported with a lexical resource containing predicates from the PropBank dataset, a part of which is enriched with semantic role information.

In chapter 2, we discuss the motivation for our project and outline the background frameworks, namely UD, AMR and Grew. We also present the corpus used for evaluation.

In chapter 3 we describe the Core System underlying our project. Additionally, to aid the development process, we built tools and integrated existing packages in a Development Tools/Pipeline (see section 3.3 of the report). This development pipeline handles conversion between formats of representation for UD and AMR, evaluation of our rewrite system)

In chapter 4, we introduce the measures used for evaluating our system. We then present the results on the first 100 sentences from our corpus, with analysis on the best outputs, and outline some areas for improvement. Finally, we present the results on the remaining 1,462 sentences from the corpus, showing how our system performs on "unseen" data and with an larger, albeit incomplete, lexicon.

In chapter 5 we discuss the need for semantic role labelling for disambiguation. We present our approach to annotating a subset of PropBank's predicates and report our inter-annotator agreement. A proposal for an automated GRS identification using clustering is then outlined. Finally, we discuss some of the more challenging characteristics of AMR and draw similarities to related problems.

We summarise our work and outline some areas worth exploration in chapter 6.

Chapter 2

Background and Related Work

2.1 Motivation

The lack of appropriately annotated data has been a perennial problem within the field of natural language processing (NLP)[6]. While annotated resources at the surface realisation level - in the forms of treebanks for English and other languages, as well as under the Universal Dependencies (UD) framework - is relatively abundant; the amount of semantically, or semantically-oriented, annotated data is much more limited.

A key challenge of producing semantic, or semantically-oriented, annotations is the complexity of the task. To annotate semantically requires more than recognising parts of speech and linking dependencies; it requires an understanding of sentence meaning, identification of appropriate concepts and relations between them. This has been a barrier to the development of large sets of semantically-oriented annotated data.

However, the authors of [2] have demonstrated the viability of using graph rewriting methods to move from surface (syntactical) annotations to an intermediate surface-deep structure, then to a deep structure (akin to Chomsky's syntagmatic theory), and eventually to semantically-oriented annotations including Abstract Meaning Representation (AMR) and Restricted Minimal Recursion Semantics (RMRS).

Notably, they have designed a system, Grew, comprising a set of transformations and packages of transformations to move from surface Sequoia (a framework of French syntactical annotations) to deep Sequoia and eventually to AMR as well as to RMRS. The resulting deep structure-like annotations from the graph rewriting process has an F-measure score of 0.9598[1], lending support to the case that the Grew system may be a viable method for producing AMR annotations from surface annotations. However, no evaluation score is available for the deep Sequoia to AMR rewrites due to the lack of an AMR Bank in French.

Our interest lies in the potential of graph rewriting for enriching the available pool of data for AMR and other meaning representations. Currently, there

[1]Based on new edges, and excluding surface relations.

exists a multitude of semantic dependency formalisms, and a widely accepted framework has yet to emerge[2].

We see that, in the interim before agreed standards appears, effort will have to go towards developing different sets of semantic or semantically-oriented data. This current state-of-play has the effect of hampering entry into semantically-oriented data, as researchers await standards to emerge. Therefore we see a system that can satisfactorily generate semantically-oriented from surface annotations, as well as convert, or aid the conversion, between semantically-oriented annotations, as having the potential to contribute meaningfully towards the advancement of research in semantics-focused NLP, as well as development of downstream NLP applications.

2.2 Universal Dependencies (UD)

UD is a framework for syntactic annotation that emerged in 2014. The goal of the project is to build a cross-linguistically consistent treebank[15]. It aims to tackle problems arising from the fact that different languages had used different and incompatible annotation schemes, making multilingual research on syntax difficult.

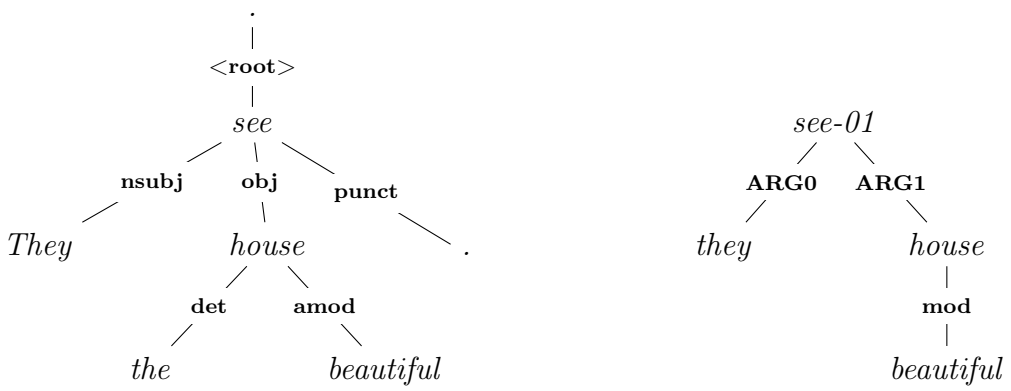
UD merges some of the previous efforts to create a universal annotation scheme, namely, the (universal) Stanford dependencies [5], the universal Google dependency scheme [14], the Google universal part-of-speech (POS) tags [17] and the Interset interlingua for morphosyntactic tag sets [20].

The principles driving the UD syntactic annotation are of *dependency* and *lexicalism*, meaning that syntactic words are the units of grammatical annotation, as opposed to phonological or orthographic words. It is important to note that UD offers a universal pool of POS tags, morphological features and dependency relations that languages can choose from. This means that the tag inventory is fixed and is meant to be used by all languages, but not all categories have to be used in every languages [15].

2.3 Abstract Meaning Representation (AMR)

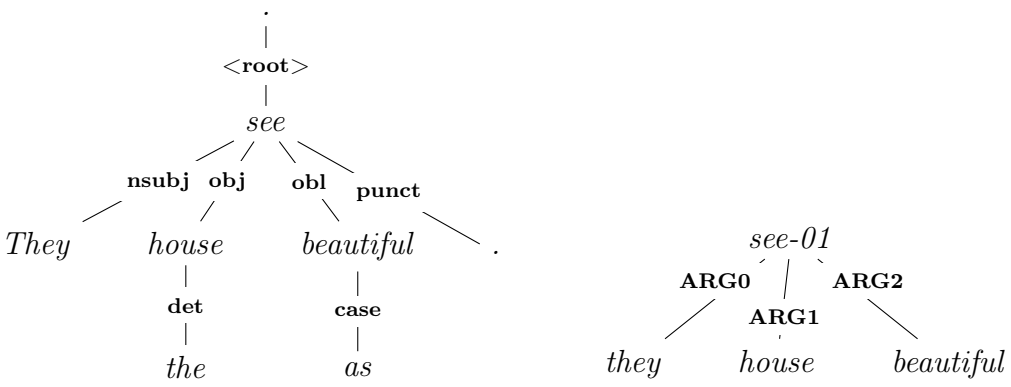
AMR is a formalism developed to capture relationships between semantic roles, i.e. predicate-argument structure of a sentence. The output of annotating sentences following the AMR framework, are directed acyclic graphs (DAGs). It is widely accepted in the AMR research community to refer to the resulting graphs as "AMRs". We follow the same notation throughout this report.

[2]Some of these formalisms, such as DELPH-IN Minimal Recursion Semantics, Predicate-Argument Structures and Prague Semantic Dependencies, are of the bi-lexical variety (whereby the semantic representations have a one-to-one correspondence with words in the surface realisation) [16]. These formalisms, part of the Semantic Dependency Parsing (SDP) family, arose as a response to the SemEval 2014 and 2015 Shared Tasks `http://sdp.delph-in.net`.



(a) Dependency tree (b) AMR

Figure 2.1 – Dependency tree and AMR of the sentence "They see the beautiful house."



(a) Dependency tree (b) AMR

Figure 2.2 – Dependency tree and AMR of the sentence "They see the house as beautiful."

AMR abstracts away syntactical information in the surface realisations (POS tags and syntactic dependencies). However, as seen in 2.1a and 2.1b, there can be sufficient information at the syntactic level to help differentiate between two sentences having similar words but with different meanings. Concepts in AMR are made up of English words or phrases (nominals such as boy, girl, etc. and verb senses), as well as different semantic roles associated with each of them, the latter being drawn from PropBank[3]. In addition, AMR specifies at least 60 of its own non-core roles[4].

A unique feature of AMR (compared to other semantically-oriented annotation frameworks such as RMRS) is the avoidance of multiple roots via the ability to express predicate-arguments in the form of inversed roles. As a result, an

[3]PropBank is in turn, developed by overlaying an additional semantic role labelling layer over verbs in the Penn Tree Bank [9])

[4]Such as :source, :destination, :path, :beneficiary, :accompanier, :topic, etc. For the detailed list, see [11]

AMR representation is simpler compared to other semantic formalisms such as Minimal Recursion Semantics (MRS)[5].

With this simpler structure in AMR comes increased intuitiveness, and the effort of manually annotating sentences in a semantically-oriented formalism has fallen, giving rise to a collaborative effort to build a bank (AMR Bank) of gold-standard AMR annotations [3]. While AMR Bank currently comprises only sentences in English, AMR is starting to become an attractive representation for researchers when working on NLP tasks that involve the handling of semantic information, including document summarisation and text generation [12].

Further, as pointed out by the author of [3], AMRs, based on their graph structures, can easily be represented by triples (or sets of triples)[6], which are machine readable and allows for easy comparability between AMRs.

2.3.1 AMR Bank: The Little Prince

The AMR Bank[7] is a limited distribution product of the Linguistics Data Consortium. Access to the entire AMR Bank is by licence. However, AMR was the subject of a SemEval Shared Task in 2016[8] and a component of AMR Bank (human annotated AMRs for sentences in the English translation of The Little Prince novel[9] was released. We utilised this dataset (AMR Bank: The Little Prince) for the development of our system. The dataset is in a single plain text file containing more than 1,500 sentences, separated with a regular pattern.

We chose to work on the SemEval 2016 Shared Task 8 data as it was the same dataset used in the development of the CAMR system [19], a transition-based AMR parser[10]. It was released in 2015 and was one of the first AMR parsers. The use of AMR Bank: The Little Prince, facilitates our use of CAMR as an objective benchmark for evaluating our system's performance.

2.4 Grew: Graph Rewriting for NLP

Grew is a graph rewriting tool for NLP. It has been written using the Ocaml[11] programming language and can be used as a Python library. Grew has been used for surface dependency syntax, deep dependency syntax and semantic representation but it can be used to represent any graph-based structure.

[5] Or its more widely developed variant, RMRS.

[6] The creators of AMR designed the initial framework with an intent to facilitate the manual AMR annotations of sentences, and selected the PENMAN notation for this aspect. Examples of how sentences are manually annotated in PENMAN notation can be found in the AMR specification. However, for the purpose of this report, in which we focus on machine reading and processing of AMR, we focus on graph and logical representations of AMRs.

[7] `https://catalog.ldc.upenn.edu/LDC2017T10`

[8] `http://alt.qcri.org/semeval2016/task8/`

[9] `https://amr.isi.edu/download/amr-bank-v1.4.txt`

[10] `https://github.com/c-amr/camr`

[11] `http://ocaml.org/`

The Grew library features a dedicated syntax for graph handling, where nodes can be specified including their feature structures. A sample graph including lemmas, part of speech tags and UD relations for the sentence *"I saw a magnificent picture"* can be seen below.

```
g = Grew.graph(‘‘‘graph{
W1 [lemma=‘‘I’’, upos=PRON];
W2 [lemma=‘‘see’’, upos=VERB];
W3 [lemma=‘‘a’’, upos=DET];
W4 [lemma=‘‘magnificent’’, upos=ADJ];
W5 [lemma=‘‘picture’’, upos=NOUN];
W2 - [nsubj] -> W1;
W2 - [obj] -> W5;
W5 - [det] -> W3;
W5 - [amod] -> W4;
}’’’)
```

The principle behind graph rewriting is in recognising certain patterns in a graph and transforming the recognised part using a set of commands (e.g. deleting or adding nodes and edges). In order to facilitate graph rewriting, Grew uses rules. A rule is made up of a pattern and a list of commands.

A pattern may appear several times in the same graph and thus the same rule needs to be applied to the graph multiple times. Furthermore, more than one rule may apply to a given graph. In some cases, it is important which rules are applied first. To allow for specifying the way rules are applied, Grew uses strategies. When applied to an input graph, a strategy produces a set of graphs. The simplest form of strategy is a single rule. Other types of strategies include:

- Sequence - `Seq(S1, S2)` - produces graphs obtained by applying S1 and then S2 to a graph
- Pick - `Pick(S1)` - picks only one of the solutions of S1
- Iteration - `Iter(S1)` - apply S1 to a graph for as long as it can be applied

Grew also offers a package system which allows for the grouping of sets of rules with a common objective. Packages are often used to group rules that should be applied together and it is therefore possible to use the name of a package as a strategy name.

Finally, while many rules can be written to model the general language rules that can be found in grammar, there are some which depend on specific lexical entries. To tackle this, Grew offers the possibility to create parameters for rules, using lexicons. The lexicon format used by Grew is simple: the first line of a lexicon specifies the fields used by the lexicon, and each of the following lines corresponds to a lexical entry. The field values are separated by tabs.

Chapter 3

Design and Implementation

3.1 System Architecture

Our main system consists of five modules as shown in Figure 3.1. A part of the pipeline ("Parser" and "UD to AMR") covers the main objective of this project and can be used independently. An additional set of modules ("Preprocessor", "AMR Graph to Text" and "Smatcher") were developed in order to allow us to evaluate the performance of our system against AMR Bank: The Little Prince, the human-annotated gold standard.

3.2 Core System

3.2.1 Parser Module

Our `parser.py` module takes a sentence in natural language and creates its UD parse tree. We rely on UDPipe[12], through its Python library[13], for parsing. We chose to use the `english-ewt-ud-2.3-181115.udpipe` model for UDPipe. Among the four UDPipe models available for English, it has the highest labeled attachement score (LAS) and it is based on the largest corpus: 16,622 trees, 254,854 tokens - and we expect its wider coverage to have better performance. A comparison between the four available UDPipe models for English can be seen in Table 3.1.

The core function in our module lets the user pass a sentence and a sentence ID. The latter is for writing the sentence ID into the CoNLL-U file created for that sentence. In addition to this, the module provides a function which allows the user to parse all sentences in a provided folder. The user is required to pass the `load_path` and `save_path`, where the `load_path` points to a folder containing text files, each consisting of one sentence in plain text, and `save_path` is the path to the desired directory where the CoNNL-U files produced by the parser should be saved.

[12] `http://ufal.mff.cuni.cz/udpipe`
[13] `https://github.com/ufal/udpipe`

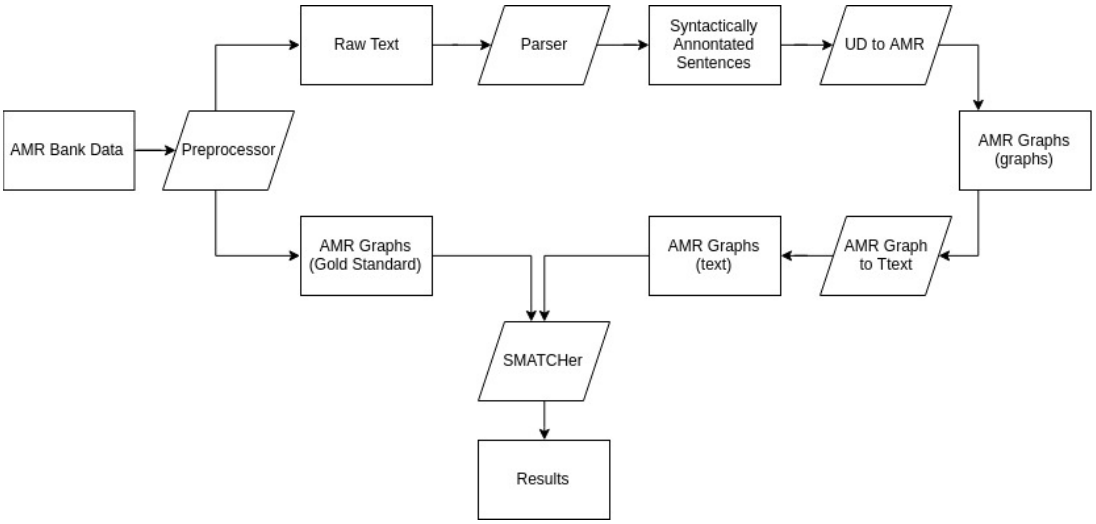


Figure 3.1 – System Architecture

Model	Trees	Tokens	LAS
english-ewt-ud-2.3-181115.udpipe	16,622	254,854	85.8%
english-gum-ud-2.3-181115.udpipe	4,399	80,176	83.8%
english-partut-ud-2.3-181115.udpipe	2,090	49,648	85.6%
english-lines-ud-2.3-181115.udpipe	4,564	82,816	78.3%

Table 3.1 – Trees and tokens per UDPipe model.

Throughout this chapter, we will present how our system behaves on a sample sentence. From The Little Prince, we chose the sentence "Once when I was six years old I saw a magnificent picture in a book, called True Stories from Nature, about the primeval forest." However, due to the size of the resulting output, we have decided to only work through a part of this sentence for the report, namely the fragment "I saw a magnificent picture."

Given the sentence "I saw a magnificent picture.", the parser will produce a CoNLL-U file with the following content:

```
# newdoc
# newpar
# sent_id = 1
# text = I saw a magnificent picture.
1 I I PRON PRP Case=Nom|Number=Sing|Person=1|PronType=Prs 2 nsubj _ _
2 saw see VERB VBD Mood=Ind|Tense=Past|VerbForm=Fin 0 root _ _
3 a a DET DT Definite=Ind|PronType=Art 5 det _ _
4 magnificent magnificent ADJ JJ Degree=Pos 5 amod _ _
5 picture picture NOUN NN Number=Sing 2 obj _ SpaceAfter=No
6 .  .  PUNCT . _ 2 punct _ SpaceAfter=No
```

A dependency tree for the same can be seen in Figure 3.2.

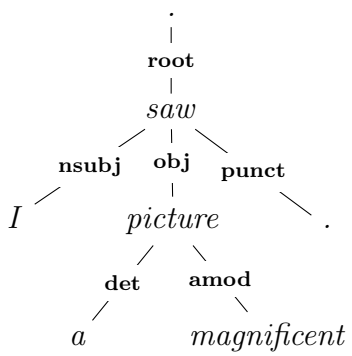


Figure 3.2 – Dependency tree of the sentence "I saw a magnificent picture"

3.2.2 UD to AMR Module

The `ud_to_amr` module takes a UD-annotated sentence stored in CoNLL-U format, similar to that produced by `parser` in subsection 3.2.1 above, applies a set of rewrite rules to convert the sentence into AMR form. The module returns the AMR in CoNLL-U format. We rely on the Grew system, through its Python bindings (Grew Python), for loading, rewriting and saving the rewritten AMR sentence. Our functions in this module are primarily convenience extensions on top of Grew Python functions. It is designed as an interface point for the different packages `UDPipe`, `Grew`, and `Smatch` (see subsection 3.3.1) and their outputs that are utilised in the rewriting of the AMR.

The key function in the module takes as input: (i) the UD-annotated sentence to be rewritten, (ii) the name of the file containing the GRS rewrite strategies, as well as (iii) the name of the GRS strategy to be applied on the UD-annotated sentence. It applies (ii) to the data loaded from (i). The `save_data` function calls the `amr_graph_to_conllu` module (see subsection 3.3.2) to convert the generated AMR in graph format into the CoNLL-U format and written to a plain text file.

The `ud_to_amr` module comprises two main parts. The first (see subsubsection 3.2.2.1) is a lexical resource based on PropBank predicates enriched with human-annotated semantic role labels. The second (see subsubsection 3.2.2.2) is a Grew Rewriting System (GRS) containing a graph rewriting rule base constructed by us.

3.2.2.1 Lexicons

In natural language, we often have words with more than one meaning. This can lead to ambiguity in the interpretation of sentences, and remains a challenge in natural language processing. For example, without contextual information, it is not possible even for listeners to completely distinguish the meanings of certain sentences such as *"John scanned the paper"*, let alone doing so computationally.

AMR is intended to capture semantically-oriented information, including (i) which sense of a predicate was used in the surface realisation of the sentence, as

well as (ii) the semantic roles associated with these predicates. However, a UD annotation does not provide word sense disambiguation. Also, in many cases there is no one-to-one mapping between UD tags (or sets of UD tags) to AMR tags (or sets of AMR tags). These hamper a direct transformation of a UD annotation to AMR. However, these two problems can be addressed by the use of lexicons.

Since AMR is based on PropBank[14], it is the most appropriate choice for the basis of our lexicon. A list of 8,733 frame arguments is available online[15]. Each line of this file contains a predicate and its respective argument structure in the format presented below.

```
...
indict-01 ARG0:  accuser ARG1:  accused ARG2:  crime
...
indispose-01 ARG1:  cause of unwillingness ARG2:  unfit one ARG3:  unwilling to
do this
indispose-02 ARG1:  cause of illness ARG2:  ill person
...
```

The notation structure is not consistent in the file and therefore cannot be used by our GRS rules directly. In the majority of cases, **ARG0** is the agent, actor, cause or stimulus of the predicate. However, there exist predicates for which this is not true. We discuss this aspect in further detail under section 5.1.

With this in mind, we devised a structure, based on the PropBank frame arguments file, which can be used by the GRS rules and would help us resolve the two main problems described above. This involved the enrichment of the lexicon with finer-grained semantic role labels for each predicate sense. The annotation procedure we used for the semantic role labelling enrichment is described in subsection 5.1.1.

3.2.2.2 Grew Rewriting System (GRS)

To build our rule base, we started by analysing UD relations in order to find mappings between them and the available AMR relations. Throughout this process we used the UD Guidelines[16], Grew-match[17], and our corpus to guide us in identifying the language phenomena to be captured by our system.

The structure for our rule base was inspired by the authors of [2]. The final rule base we built consists of 183 rules, grouped in 10 strategies. It is structured around the following:

- 3 files containing rules with unique structures:
  - `core_roles.grs` - 32 rules
  - `core_roles_acl.grs` - 4 rules

[14] `http://propbank.github.io/`
[15] `https://amr.isi.edu/doc/propbank-amr-frames-arg-descr.txt`
[16] `https://universaldependencies.org/u/dep/index.html`
[17] `http://match.grew.fr/`

  - `non_core_roles.grs` - 19 rules

- 5 files containing rules with repetitive structures:
  - `cc_to_op.grs` - 11 rules
  - `contrast.grs` - 71 rules
  - `mark_to_non-core.grs` - 5 rules
  - `poss_lemmas.grs` - 6 rules
  - `remove_punct_safe.grs` - 35 rules

- 1 file grouping rules into strategies:
  - `grs_amr_main.grs` - 10 strategies

The files in the first two groups contain rewrite rules for transforming UD structures into AMR structures. The files in the first group contain rewrite rules which are aimed at unique structures. Each of the files in the second group contains a set of rules, aimed at identical structures, but applying to different UD tags within it. These were necessary in order to ensure we cover all the different possibilities for a particular edge within the structure of interest.

The strategies contained in the `grs_amr_main.grs` file group the rules on the basis of their intended functionality. For example, the rules `Iter(remove_punct)` and `Iter(remove_det)` (aimed, respectively, at removing punctuation, and removing the determiners "a", "an" and "the") are a part of the `clean` strategy. This strategy is meant to be applied at the end of the rewriting process and remove the nodes that do not contribute to the semantic information represented by AMR.

3.2.2.3 Example of UD to AMR in action

Starting from the UD graph in Figure 3.2, we can follow how the sentence "I saw a magnificent picture." is transformed when passing through the UD to AMR module.

First, the rules for preparing predicates will fire. The only node that matches a predicate in our lexicon is "see". As shown in Figure 3.4, there are five entries for it in the lexicon. One of the resulting graphs after applying the predicate preparation rules, is the one shown in Figure 3.5a.

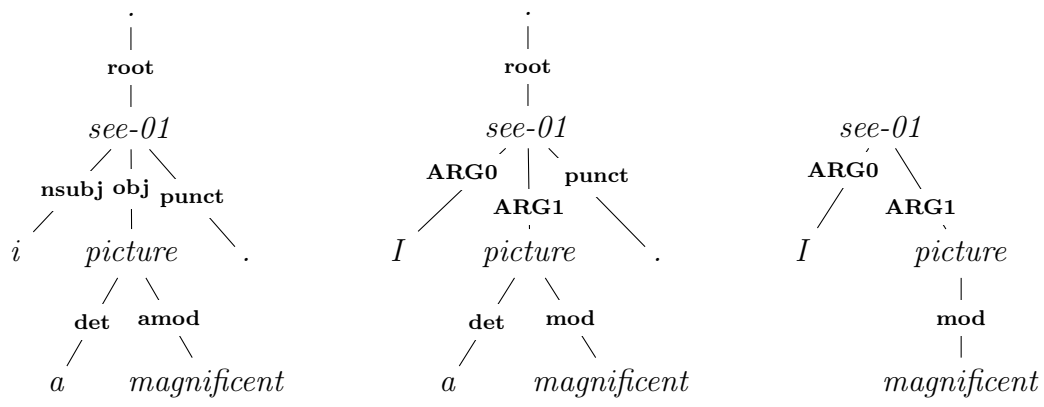
Following this, the rule `pred_nsubj_obj` (shown in Figure 3.3) will fire, transforming the `nsubj` edge to `ARG0` and the `obj` edge to `ARG1`. Then, a rule for changing `amod` to `mod` will produce the graph in 3.5b. Finally, a cleanup strategy will remove the punctuation, determiners and the root relation to give the final graph in 3.5c.

```
rule pred_nsubj_obj(lex from "lexicons/subcat/test_lexicon.lp"){
  pattern {
    V[cat=VERB, lemma=lex.lemma, arg0=agent|actor|stimulus|cause,
arg1=patient|theme|experiencer];
    nsubj_rel:  V -[nsubj]-> NSUBJ;
    obj_rel:  V -[obj]-> OBJ;
  }
  commands {
    del_edge nsubj_rel;
    del_edge obj_rel;
    add_edge V -[ARG0]-> NSUBJ;
    add_edge V -[ARG1]-> OBJ;
  }
} }
```

Figure 3.3 – GRS rule `pred_nsubj_obj`.

```
...
predicate arg0 arg1 arg2 ...
see-01 experiencer stimulus attribute ...
see-02 agent patient destination ...
see-03 agent - - ...
see-04 agent patient preposition ...
see-05 actor1 actor2 - ...
```

Figure 3.4 – Lexicon entries for the predicate “see”.



(a) Initial step (b) Intermediate step (c) Final output

Figure 3.5 – Transformations on the sentence “I saw a magnificent picture.”

3.3 Development Tools/Pipeline

3.3.1 Evaluation Metric

To evaluate the performance of our rewrite system, we have to require an objective evaluation metric to compare our rewritten AMRs with the gold-standard human annotated versions in AMR Bank. We used the semantic evaluation tool formulated by AMR's development team, `smatch.py` [18]. See chapter 4. As mentioned in subsection 2.3.1, the primary factor guiding our selection of Smatch as our evaluation tool was its use by the developers of CAMR. This is further supported by its continued acceptance as an evaluation metric by the community of researchers working with AMR[19].

Our `smatcher.py` module takes the AMR developers' `smatch.py` package (see chapter 4) and uses it via shell with Python's subprocess module. Two AMR sentences in text format (annotated in PENMAN notation) are compared in order to generate a similarity score between them. To ensure that the objectivity of the evaluation score is maintained, we leveraged `smatch.py` in the way it was provided by the developers of AMR. This meant that we had to convert AMRs in Grew format to the text format that is accepted by smatch.py, as well as utilise shell for control, as `smatch.py` utilises argparse.

3.3.2 AMR Graph to Text and CoNLL-U

When comparing AMR graphs, Smatch takes as input two text files, each containing an AMR graph in its textual format. We have the gold standard AMR graph in this format. However, our `UD_to_AMR` module produces AMR graphs in Grew format. Therefore, we needed a rewriting module to convert a graph in Grew format into a graph in text format. Our `amr_graph_to_text` module provides a function which takes a Grew graph as input and returns its textual representation. Since the function needs only a lemma, a concept (as produced by the `UD_to_AMR` module) and a list of relations for each node, a simplified AMR graph in Grew format containing only these features for the sentence "I saw a magnificent picture" is presented below. We have also presented the produced text representation.

[18] `https://github.com/snowblink14/smatch/blob/master/smatch.py`

[19] For example, Lyu, Chunchuan and Titov, Ivan, *"AMR Parsing as Graph Prediction with Latent Alignment"*, 2018

```
['1':  ['concept':  'I',
        'lemma':  'I',
        []],
 '2':  [ 'concept':  'see-01',
        'lemma':  'see',
        [['ARG0', '1'], ['ARG1', '5']]],
 '4':  ['concept':  'magnificent',
        'lemma':  'magnificent',
        []],
 '5':  ['concept':  'picture',
        'lemma':  'picture',
        [['mod', '4']]]]
```

```
(s / see-01
  :ARG0 (i / i)
  :ARG1 (p / picture
    :mod (m / magnificent)))
```

Chapter 4

Evaluation Measures and Results

4.1 Semantic Match, Smatch

We utilise the `Smatch` evaluation measure for assessing the performance of our GRS. This measure and tool was developed by a group within AMR's developers [4]. Smatch works by finding the maximum F1 score possible between two AMRs. The Smatch metric views each AMR as a collection of triples, each of which comprise a pair of concepts and the AMR relation associating them. By measuring the degree of overlap of the triples set from each AMR, the Smatch algorithm develops a view of the structural similarity of the two AMRs.

Smatch works with three distinct types of triples - instance, relation and attribute triples. Instance triples tie nodes to IDs, one is present for each distinct node. Relation triples correspond to AMR relations between nodes. Attribute triples capture information about the properties of specific nodes - "TOP" (pointing to the root node), mode and polarity, to name a few. Triples are equal in weight regardless of their type.

As our objective is to develop a GRS such that it is able to produce the correct AMR of a sentence from its UD annotation, we chose to evaluate our GRS by returning all possible outputs (see section 2.4) from the application of our rules. For each sentence, we evaluate each of its outputs from our GRS against the corresponding gold standard AMR from the AMR Bank. The output AMR (from our GRS) with the highest Smatch score (F1) is recognised as our system's output. Subsequently, our reported system performance is computed based on the set of best scores for each of the evaluated sentences.

This approach allows us to better evaluate the AMR production potential of our system, and leaves the question of selecting the right output from the system to other disambiguation methods (for example, finer GRS rewrite rules or with the support of human annotators). The results of various tests on our system is contained in the next section.

	Our Final GRS		
	Min	**Max**	**Arithmetic Mean**
Precision	0.00	1.00	0.47
Recall	0.00	1.00	0.46
F1-score	0.00	1.00	**0.46**

Table 4.1 – Min, Max and Average Precision Recall and F1-score for our final GRS.

	Base			Base+Lex			Interim GRS		
	Min	**Max**	**Arithmetic Mean**	**Min**	**Max**	**Arithmetic Mean**	**Min**	**Max**	**Arithmetic Mean**
Precision	0.00	1.00	0.27	0.00	1.00	0.41	0.00	1.00	0.44
Recall	0.00	1.00	0.28	0.00	1.00	0.44	0.00	1.00	0.45
F1-score	0.00	1.00	**0.27**	0.00	1.00	**0.42**	0.00	1.00	**0.43**

Table 4.2 – Min, Max and Average Precision Recall and F1-score for Base, Base+Lex and Iterim GRS.

4.2 System Evaluation

We evaluated the quality of the rewritten sentences using the Smatch metric, assuming the highest scoring graph for each sentence as the system output (see section 4.1 above). We evaluated our system with two different sets of inputs. The first set is the first 100 sentences of The Little Prince corpus and the second set is the remaining 1,462 sentences of the corpus.

4.2.1 First 100 Sentences

We applied our GRS on the first 100 sentences of The Little Prince corpus. These sentences range in length from three to 41 tokens, with the average sentence length being 13.35 tokens. The highest average F-score for the first 100 sentences of The Little Prince corpus, after applying our **Final** GRS is **0.46**. Table 4.1 provides the minimum, maximum and average[20] precision, recall and F1-scores.

In addition, we tracked the changes in the F-score of our system as we progressively added a test lexicon (see subsubsection 3.2.2.1 and section 5.1) and more GRS rules. Table 4.2 provides an overview of these experiments.

The bars in red relate to results (**Base**) after application of our preparatory GRS rules, those in blue relate to results (**Base+Lex**) after the addition of a controlled experimental lexicon with complete coverage for the sentences , green relates to results (**Interim**) after the application of our interim set of basic GRS rules. As can be seen from Figure 4.1, the application of the controlled lexicon begins to shift the F-score distribution rightwards (i.e. a better evaluation result across the board for the 100 sentences). The addition of the interim rules, further

[20]We provide the arithmetic average precision and recall scores for a more granular view of these scores over the 100 sentences, since the minimum and maximum for these are 0.00 and 1.00 respectively. We note however, that the sentences are of significantly varying lengths, and as such, an arithmetic average of these precision and recall scores may not be a truly informative measure.

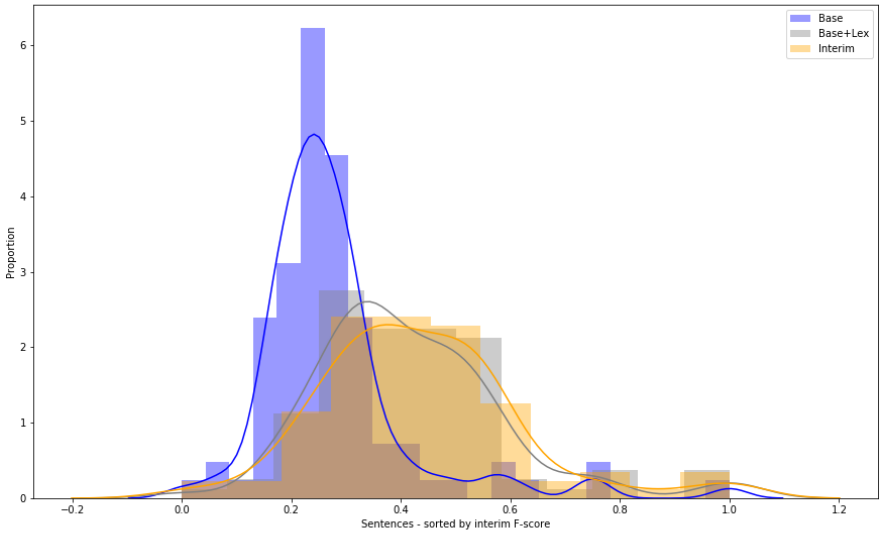


Figure 4.1 – Distribution of F-scores for rewrites of first 100 sentences of The Little Prince, for *Base* to *Base+Lex* to *Interim*.

improved the evaluation results, as can be seen by a further rightward shift of the distribution.

The following sections, analysing the rewritten sentences with the highest and lowest Smatch F-scores, are based on the application of our **Final** GRS rules.

4.2.1.1 Best Outputs

While we get a maximum F1-score for some of the sentences, further investigation shows that these are sentences containing only one word each (*"Yes,"* and *"What!"*). The only rewrite rule that applies to each of these is the one for deleting punctuation. This explains why there is a perfect match between our result and the gold standard.

The highest meaningful F1-score that we get is 0.88. It comes from the sentence *"Something was broken in my engine ."*. The gold standard AMR for the sentence and the AMR produced by our system are presented in Figure 4.4.

The AMR graph on the right is obtained by applying the following rules and strategies:

- strategy `prepare_predicates`, which finds the sense "break-01" for "break";
- rule `remove_aux_pass`, which removes the `aux:pass`, when the predicate has an `nsubj:pass` relation too
- rule `pred_nsubjpass`, which is applied when there is a passive subject, and transforms the `nsubj:pass` relation from "break" to "something" into an `ARG1` relation;

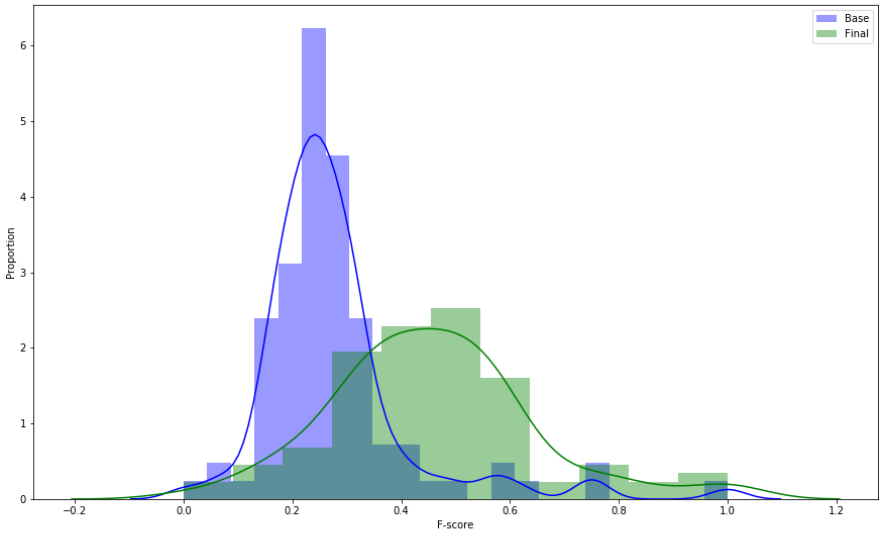


Figure 4.2 – Distribution of F-scores for rewrites of first 100 sentences of The Little Prince, for *Base* to *Final*.

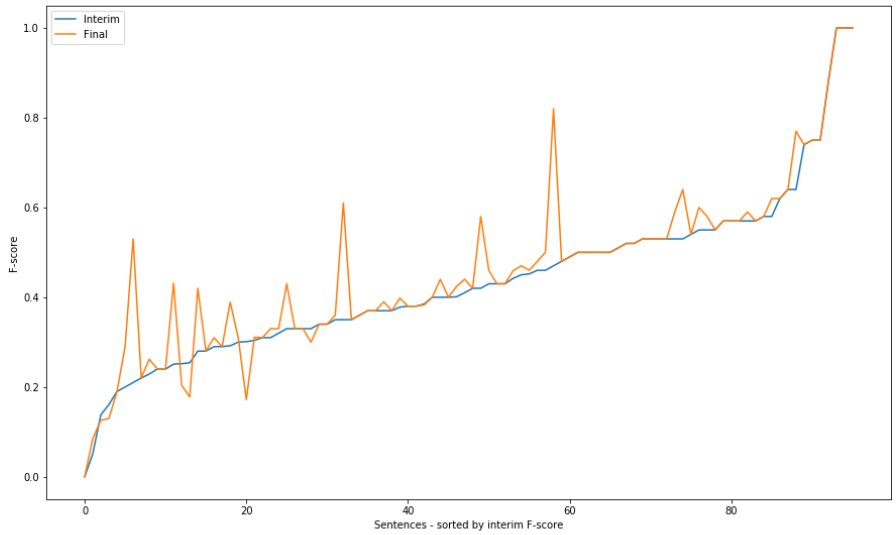


Figure 4.3 – F-score for each sentence, change from *Base* to *Final*.

```
(b / break-01
  :ARG1 (s / something
    :location (e / engine
      :poss (i / i))))
```

```
(b / break-01
  :ARG1 (s / something)
  :obl (e / engine)
    :poss (i / i))
```

Figure 4.4 – Gold AMR (left) and produced AMR (right) for *"Something was broken in my engine ."*

- rule `det_as_pers_pron`, which transforms the `det` relation from "engine" to "my" into a `poss` relation;
- strategy `rename_poss_concepts`, which transforms "my" into "i";
- strategy `clean`, which removes punctuation, the `case` marker "in" and the root of the sentence.

The gold AMR graph contains four nodes, which result into four instance triples, three relations (between *break-01* and *something*, *something* and *engine*, and engine and *i*), which result in three relation triples, and one attribute triple (*break-01* being the root of the graph), making a total of eight triples. Similarly, we have four instance triples, three relation triples and one attribute triple for the AMR produces by our system, resulting in a total of eight triples. We have an overlap of six triples (four instance relations - one for each of the nodes mapped correctly, two relation triples - `poss` transformed correctly and `ARG1` indentified correctly, and one attribute triple - the root). Thus we obtain a precision of 0.88 and a recall of 0.88, giving us an F1-score of 0.88.

The reason why the `obl` relation is not transformed is that our system does not treat the majority of oblique relations, as we do not have a reliable way to differentiate between different types of nominal modifiers. Although there exists a database of prepositions classified by their modification indicators[21], there remains significant overlap between many prepositions (e.g. the preposition "in" can refer to a locational or a temporal modifier), and we did not expect the use of it to substantially help in disambiguation.

4.2.1.2 Areas for Improvement

Using the **Final** GRS , the majority of the rewritten sentences have a Smatch F-score between 0.30 and 0.50. Although this is an improvement over the baseline, it is below that of the external benchmark we have chosen - CAMR which reported a Smatch score of 0.63.

Our analysis identified three main reasons for the quality of the rewrites for these sentences. Firstly, the coverage of our GRS rule base is incomplete and our system is not able to rewrite for all the core and non-core roles in AMR. Secondly, there are certain concepts which are predicates derived from nouns, (see the example for the noun "drawing" in Figure 4.5) and we have yet to develop our GRS to handle such cases.

[21] `http://www.clres.com/db/classes/ClassTemporal.php`

```
...  (t / thing
    :ARG1-of (d / draw-01
      ...
```

```
   ...  (d / drawing
  :...  (...
```

Figure 4.5 – The noun "drawing", reified and represented in AMR Bank (left) and without reification (right)

```
(s / smile-01
  :ARG0 (p / person
    :ARG0-of (h / have-rel-role-91
      :ARG1 (i / i)
      :ARG2 (f / friend)))
  :manner (g / gentle)
  :manner (i2 / indulgent))
```

```
(s / smil
  :nsubj (f / friend
    :poss (i / i))   :advmod (g /
gently)
  :conj (i2 / indulgently))
```

Figure 4.6 – Gold AMR (left) and produced AMR (right) for *"My friend smiled gently and indulgently."*

Finally, we developed our GRS rules on the basis of their most prototypical cases. Given that we are working with an automated UD parser, it is possible that there may be variations in the parser output (either as a valid parse with a different structure or an error), thereby preventing a GRS from firing, or wrongly causing one to fire. Figure 4.6 presents one such case where the lemma "smile" was parsed to be "smil" and thus did not match the predicate "smile" in our lexicon.

One out of the 100 rewritten sentences have a Smatch score of zero. The rewritten version of this sentence, together with its gold AMR counterpart is presented below:

```
(c / cause-01
  :ARG0 (a / amr-unknown)
```

```
(w / why )
```

Figure 4.7 – Gold AMR (left) and produced AMR (right) for *"Why ?'*

The sentence was not rewritten at all because it consists of a single wh-word - "why", and our GRS does not currently contain rules to treat wh-sentences, which are represented in AMR with the "amr-unknown" concept[22].

4.2.2 Remaining 1,462 sentences

Subsequently, we conducted two experiments to establish that our system can be run across the entire corpus, as well as to evaluate the efficacy of our GRS on the remaining sentences in The Little Prince corpus. As we developed our GRS rule base (from **Base** to **Final**) using examples from the first 100 sentences of the corpus, running the system on sentences other than the initial 100 somewhat mimics the performance of the system on unseen data. The Smatch F-score for

[22]Except when the wh-sentence involves a relative clause, in which case inverse roles, i.e. ARG-of, are used.

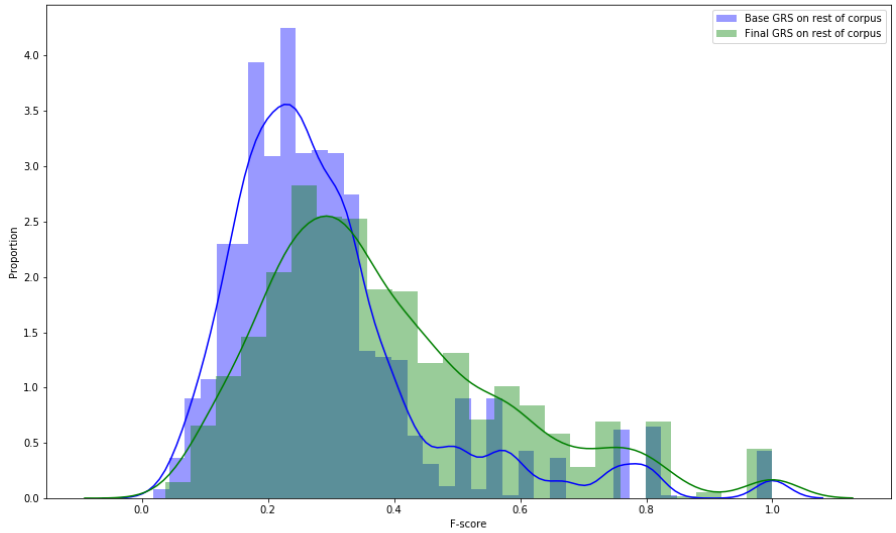


Figure 4.8 – Distribution of F-scores for rewrites of remaining 1,462 sentences, change from *Base* to *Final*.

the 1,462 sentences after applying our Final GRS is **0.38**. The distribution of the F-score for the 1,462 sentences, as well as the shift in the distribution from the application of our **Base** to **Final** GRS can be found in Figure 4.8.

Chapter 5

Discussion

5.1 Semantic Role Labelling for Disambiguation

A lemma may have different senses that each have different argument structures (in terms of having a different set of semantic roles specified). AMR inherits PropBank's semantic roles framework. The semantic role labels (SRL) in PropBank are shallow[23] and its ARG0 and ARG1 roles relate to "Proto-Agent" and "Proto-Patient" roles respectively[8][24]. Using the argument structure of a predicate, it is possible to:

- rely on UD syntactic and part-of-speech information to help with disambiguation[25].
- specify the correct ARG0 or ARG1 when rewriting a UD relation to an AMR relation[26].

We identified a set of PropBank annotation guidelines providing a mapping to VerbNet semantic roles[27]. Although this mapping is not present in the latest version of the PropBank annotation guide, and contains only 27 SRLs (instead of the 31 in VerbNet), we leveraged a subset of it to specify a set of SRLs that will

[23]PropBank has up to 10 main argument roles (ARG0 to ARG9) which are coarser-grained compared to VerbNet's (where 31 SRLs are specified).

[24]As an illustration, PropBank has two senses for the lemma `indispose`. The first of which has no **ARG0** role specified, and its **ARG1**, **ARG2** and **ARG3** roles are specified as agent, patient and theme respectively. The second sense of `indispose` also has no **ARG0** role, but it only has **ARG1**, **ARG2** specified, and these are specified as cause and experiencer respectively.

[25]Using the lemma `indispose`, which has two senses in PropBank, a surface realisation with `indispose` and which has an indirect object specified (prototypically linked to its parent with an iobj UD relation) cannot refer to the second sense of `indispose`. Unlike with `indispose-01`, the specificiation of `indispose-02`'s argument structure does not accept a third participant.

[26]For example, UD nsubj/obj dependency relations, together with a predicate's argument structure can allow us to select the correct rewrite of a relation to ARG0/ARG1. See for example, the pred_nsubj_obj rule in our GRS within `https://github.com/siyanapavlova/AMR_annotations/blob/master/grs/core_roles.grs`

[27]`http://clear.colorado.edu/compsem/documents/propbank_guidelines.pdf`

provide us with a finer level of semantic roles. We select 25 of the more common SRLs present in PropBank argument frames[28]. The objective is to obtain a test version of the SRL-enriched PropBank lexicon[29] that will allow us to test the feasibility and efficacy of using a semantic roles-enriched PropBank lexicon. Our set of evaluation results (see section 4.2) indicate that the use of such an SRL-enriched lexicon enhances the performance of our GRS.

5.1.1 Annotation Approach

A total of 497 PropBank predicate senses were annotated jointly by both authors of this report[30]. Initially, 500 senses were randomly sampled from PropBank's 8,733 senses. The size of each sample is five (i.e. 100 samples were drawn from PropBank). This was done to ensure that the common portion of the annotation task will have contiguous blocks of senses from a single predicate lemma[31]. Out of the 500 sampled senses, three of the predicate senses contained ARGM[32] roles which we did not plan to analyse and annotate at this stage and were subsequently left out from the annotation task, therefore leaving 497 senses for joint annotation.

Additionally, in order to simulate the performance of a GRS with a complete SRL-enriched PropBank lexicon, the authors identified the predicates present in the first 100 sentences of The Little Prince, and similarly added the SRL annotations to this. This involved an additional 337 senses from PropBank that were split between both authors for annotation.

We note that certain predicates in PropBank have multiple semantic roles possible for a single argument. Therefore, there is a need for further disambiguation for these. One possibility is to rely on syntactic clues (such as type of prepositions used). For instance, the predicate sense fill-06, meaning *"to make or become full, containing, up to capacity"*, has an ARG1 role that can be filled by one of these semantic roles: (i) container, (ii) patient, or (iii) theme. Depending on whether it is concrete, whether it is structurally altered by the action of filling, and/or whether the ARG1 participant has been used intentionally etc, the semantic role for ARG1 would be different.

[28]These are namely: actor1, actor2, agent, asset, attribute, beneficiary, cause, destination, experiencer, extent, instrument, location, material, patient, patient1, patient2, product, recipients, source, stimulus, theme, theme1, theme2, time, topic.

[29]A partial coverage of PropBank, and having an intermediate depth of SRLs that is between PropBank and VerbNet.

[30]Both authors are first-year postgraduate (Masters) students in natural language processing who are advanced English speakers (a native and a C2-level).

[31]For instance, we have board-02, board-03 and board-05 within the common annotation task

[32]In PropBank, ARGM roles capture information about modifiers such as temporal or locational information. A PropBank lexicon annotated together with finer-grained ARGM roles, together with UD's modifier-class dependency relations, could help with disambiguation in the rewriting process.

5.1.2 Inter-annotator Agreement (IAA)

The IAA result for our annotation of the common 497 senses was **0.65**. This is computed with Cohen's kappa as the agreement across the 1,273 argument roles annotated[33]. This IAA result is broadly in line with IAA reported by other recent studies [7] and [18], which also note the difficulty of achieving excellent[34] annotator agreement in semantic role labelling. Additionally, we also computed the Cohen kappa scores in the following to obtain more granular view of our annotator agreement:

- The average of the agreement for each sense: **0.64**. A deeper look in this average gives us an idea of which predicates and their senses have the most (and least) semantic role agreement between the annotators.
- The agreement within each PropBank argument role: **0.67**. This provides an understanding of the assignment of semantic roles within each PropBank argument role. Heatmaps of the confusion matrices for PropBank's ARG0-ARG5[35] can be found in Annex A

5.2 GRS Identification with Clustering

We investigated a method to identify, in a computational manner, syntactic patterns that are candidates as the bases for GRS rules. The objective of such an approach is to obtain, from the labeled AMR Bank: The Little Prince data, the minimal set of distinct UD syntactic patterns that identify a particular AMR relation. This section summarises our learning and preliminary conclusions from the investigation.

We identified the `Grew`, `NetworkX`, and `scikit-learn` packages as useful for developing this phase of work. We believe that graph edit distance (GED) using `NetworkX` together with `scikit-learn`'s spectral clustering could help discriminate (and associate) between different (and similar) syntactic structures that rewrite into a particular AMR relation.

5.2.1 Proposed Approach

We propose to segment the UD-annotated sentences of The Little Prince based on AMR relations. Each segment is the set of sentences that contain a certain AMR relation, according to the AMR Bank. We do this for the set of relations within the AMR guidelines. For each AMR relation's segment of sentences, we propose to carry out the procedure in Table 5.1:

[33]Cohen's kappa was selected as it is designed for measuring agreement between two annotators accounts, which suits our case. It also accounts for chance agreement.

[34]Typically determined as being above 0.80 [10]

[35]Assignments to ARG6-ARG9 roles are rare within PropBank and none of the 497 senses in the joint annotation task had roles specified in the ARG6-ARG9 range

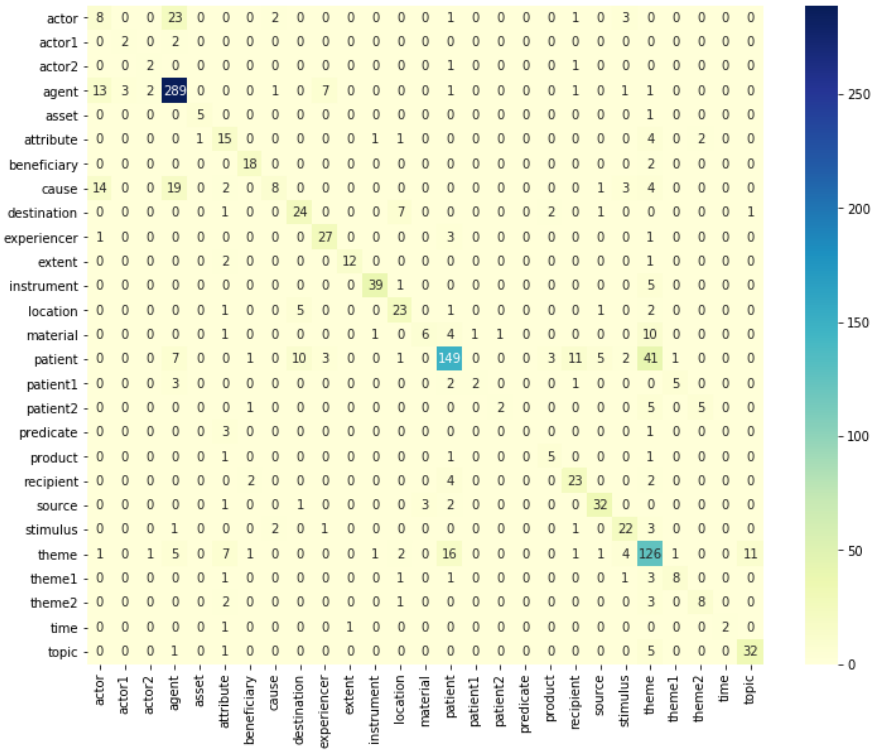


Figure 5.1 – Heatmap and confusion matrix of the inter-annotator agreement across all arguments.

Procedure for one AMR relation
Input: GREW UD graphs [1]
1. Load UD graphs and corresponding AMR graphs in Grew[2]
2. Find one-to-one correspondence between AMR and UD
3. Generate NetworkX (nx) DiGraph objects
4. Preprocess by removing nodes and edges for functional POS and relations[3]
5. Get largest subgraph
6. For each decreasing step-size:
a. Prune subgraph by deleting leaf nodes [4]
b. Compute pairwise similarity between all subgraphs with nx [5]
b. Iterate through n_cluster range. For each n_cluster value:
i. Fit SpectalClustering algorithm and predict[6]
ii Evaluate cluster labels[7]
iii. Store Evaluation results
7. Return optimal step-size and n_cluster setting
8. Recover syntactic pattern. For the optimal step-size and n_cluster setting:
a. Retrieve the subgraphs for every cluster
b. Find the maximal common subgraph in each cluster
c. Reconstruct and return syntactic patterns

Table 5.1 – Procedure for Identifying GRS Candidates

[1] Graphs with part-of-speech tags and dependency relations as node and edge attributes respectively are a suitable representation for our case. These better capture the syntactic patterns in UD to AMR GRS rules, as well as speed up the GED and isomorphism computations with node and edge matching in NetworkX.

[2] These GREW UD graphs are for sentences whose gold AMR parses contain at least one instance of the AMR relation being investigated.

[3] We expect certain POS-tags and dependency relations (such as punctuation, reparandum, goeswith and dep) will not contribute to discriminating between clusters of syntactic patterns that relate to an AMR label. It may help to conduct a covariance analysis between every dependency relation and POS-tag to identify insignificant dependency relations/POS-tags (i.e. with the least total covariance across the board).

[4] Only if the maximum depth (i.e. longest path from the root node) is more than the current step-size.

[5] We propose the use of graph edit distance (GED), with the primitive operations of adding and deleting for nodes and edges, as well as appropriate cost settings (to bring one graph into isomorphism with another). Although computing GED is an NP-hard problem [1], this is not an issue for our procedure as the sizes of our UD graphs (and their subgraphs) are small.

[6] We believe SpectralClustering to be suitable for this task, because it effectively brings a high-dimensional problem into a low dimensional one. It takes an affinity matrix and identifies a segmentation of the samples that best *"divide the data points* [of a dataset] *"into several groups such that points in the same group are similar and points in different groups are dissimilar to each other"*[13].

[7] As the clustering is done without gold labels, we anticipate using one or more of the following measures (Silhouette, Calinski-Harabaz, and Davies-Bouldin scores) to evaluate the quality of the cluster labels.

5.2.1.1 Other Considerations

Although we have a working pipeline for generating the subgraph cuts, fitting and evaluating the clustering algorithm, there are however, a few outstanding aspects that have to be addressed. These include:

- Identifying the appropriate graph edit costs that help in meeting the clustering objective - discriminate (and associate) unlike (and like) syntactic patterns that produce an AMR relation from UD annotation.
- We expect that a single syntactic pattern can map to multiple AMR relations. Therefore, it is necessary to expand the procedure, by increasing the subgraph stepsize, when the same syntactic pattern rewrite to more than one AMR relation.
- Examine the application of the procedure on AMR relations that inherently do not have a one-to-one matching with any token at surface realisation level (e.g. reified roles such as :accompanier, :age, be-located-at-91 etc).

5.3 Characteristics of AMR

5.3.1 Annotation Flexibility

The AMR guidelines [11] allow annotators some freedom in lexical choice. This is a complication for the rewriting process as we cannot simply rewrite by finding a one-to-one mapping between the lemma of the surface realisation with the senses in the PropBank lexicon. However, it is possible that other lexical resources (such as ontologies like WordNet[36] lexical networks such as SpiderLex[37], or dictionaries) can help to identify and rank potential candidates to select in the rewriting of the surface lemma.

Besides the freedom in lexical choice, the current AMR guidelines also allow flexibility in other areas. For instance, although AMR "prefers" representing a sentence as a directed acyclic graph[38], a small amount of cyclic AMRs are considered legal and it is represented as so in the guidelines. Relatedly, although AMR allows the conversion of non-core roles to core roles (reification) to avoid acyclic graphs, it does not clearly prescribe when this should be done.

We are of the view that although this relaxes the annotation complexity for AMR, it may be problematic for the development of systems to expand AMR. In particular, the annotation flexibility means that multiple "correct" AMRs could be made for a sentence. However, the AMR Bank only contains one single gold representation for a sentence. This gives rise to a challenge similar to that faced

[36] `https://wordnet.princeton.edu/`

[37] `https://spiderlex.atilf.fr/` a lexical resource developed by researchers at Computer Processing and Analysis of the French Language (ATLIF) institute in Nancy, France capturing *Lexical Systems* for both French and English

[38] See the "Cycles" section of the latest version of the AMR Specifications (v1.25) on `https://github.com/amrisi/amr-guidelines/blob/master/amr.md` and chapter 2

in machine translation, regarding how to assess a representation for a given meaning if there are more than one possible representation.

Chapter 6

Conclusion and Outlook

We built the pipeline for a system that takes the surface representation of a sentence, parses it in UD and moves it towards AMR. Our system leverages available packages including Grew and UDPipe, for natural language processing with graphs and for parsing in UD respectively. We also developed a Python program to facilitate the production of a semantic role labels (SRL)-enriched lexicon. A test lexicon comprising 834 predicate senses (about 10 percent of the senses in PropBank) was annotated with SRLs by the two authors of this report, with an inter-annotator agreement of 0.65.

Our pipeline was run on the sentences of The Little Prince novel, and experiments indicate that the inclusion of the test lexicon as well as a partial set of Grew Rewriting System (GRS) rules base increases the F-score of the rewritten sentences when compared to their gold representations. Our system was developed with the first 100 sentence of the Little Prince ans the application of our final GRS, comprising 55 key (non repeating) rules, on the first 100 sentences returned an F-score of 0.46. The application of the GRS on the entire Little Prince corpus returned an F-score of 0.38. Given that our pipeline remains in a development stage - with a partial set of GRS and lexicons, these results appear promising. A fuller set of GRS rules, SRL-enrichment of all the senses in PropBank, and other lexical resources are expected to improve the performance of our system.

Additionally, we also studied the feasibility of identifying, in a computational manner, syntactic patterns that are candidates for GRS rules. We laid out a proposed procedure for doing so with spectral clustering and using graph edit distances. We also built a pipeline that would iterate through subgraph cuts and cluster numbers in order to identify the optimal clusters and subgraph sizes for identifying syntactic structures that correspond to a certain AMR relation. Before application of the procedure, further studies on the appropriate cost settings for primitive graph operations that distinguishes between clusters of subgraphs is needed.

Bibliography

[1] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. volume 1, 01 2015.

[2] Guillaume Bonfante, Bruno Guillaume, and Guy Perrier. *Application of Graph Rewriting to Natural Language Processing*, volume 1 of *Logic, Linguistics and Computer Science Set*. ISTE Wiley, April 2018.

[3] Johan Bos. Expressive power of abstract meaning representations. *Computational Linguistics*, 42:1–8, 06 2016.

[4] Shu Cai and Kevin Knight. Smatch: an evaluation metric for semantic feature structures. In *ACL*, 2013.

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

[6] David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. A closer look at skip-gram modelling. *Proc. of the Fifth International Conference on Language Resources and Evaluation*, 01 2006.

[7] Wojciech Jaworski and Adam Przepiórkowski. Semantic roles in grammar engineering. pages 81–86, 01 2014.

[8] Daniel Jurafsky and James Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, volume 2. 02 2018; Draft.

[9] Paul Kingsbury, Benjamin Snyder, Nianwen Xue, and Martha Palmer. Propbank as a bootstrap for richer annotation schemes. In *The 6th Workshop on Interlinguas: Annotations and Translations, 2003.*

[10] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.

[11] Shu Cai Madalina Georgescu Kira Griffitt Ulf Hermjakob Kevin Knight Philipp Koehn Martha Palmer Nathan Schneider Laura Banarescu, Claire Bonial. Abstract meaning representation (amr) 1.2.5 specification. 2018.

[12] Kexin Liao, Logan Lebanoff, and Fei Liu. Abstract meaning representation for multi-document summarization. In *COLING*, 2018.

[13] Ulrike Von Luxburg. A tutorial on spectral clustering, 2007.

[14] Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 92–97, 2013.

[15] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan T. McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal dependencies v1: A multilingual treebank collection. In *LREC*, 2016.

[16] Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Zdenka Uresová. Towards comparability of linguistic graph banks for semantic parsing. In *LREC*, 2016.

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

[18] Chenguang Wang, Alan Akbik, Laura Chiticariu, Yunyao Li, Fei Xia, and Anbang Xu. Crowd-in-the-loop: A hybrid approach for annotating semantic roles. In *EMNLP*, 2017.

[19] Chuan Wang, Nianwen Xue, and Sameer Pradhan. A transition-based algorithm for amr parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–375. Association for Computational Linguistics, 2015.

[20] Daniel Zeman and Philip Resnik. Cross-language parser adaptation between related languages. In *Proceedings of the IJCNLP-08 Workshop on NLP for Less Privileged Languages*, 2008.

Appendix A

IAA Confusion Matrix by Argument

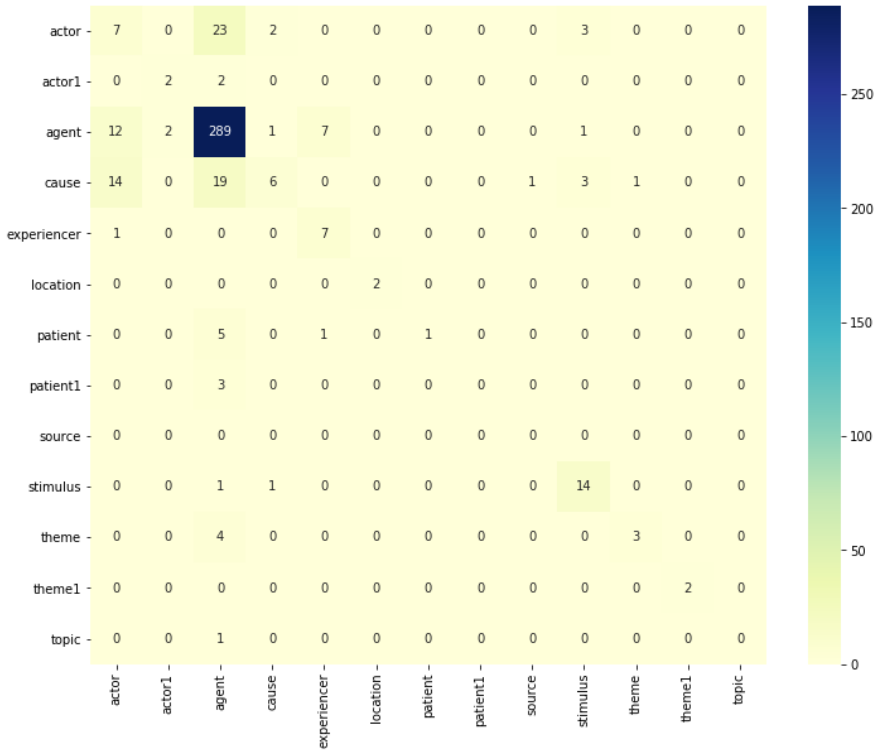


Figure A.1 – Heatmap and confusion matrix of the inter-annotator agreement for ARG0.

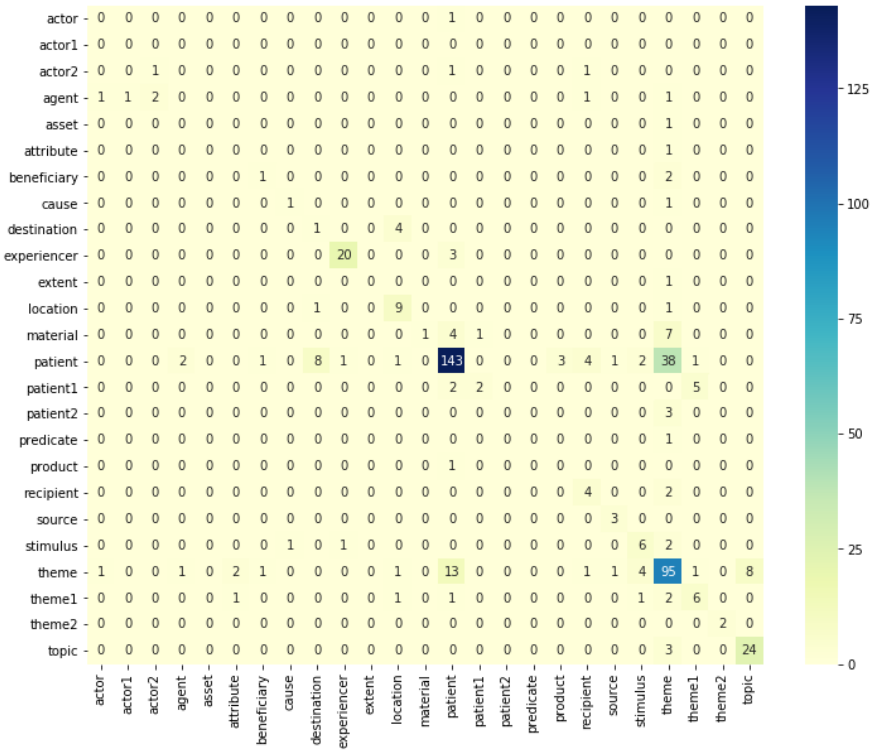


Figure A.2 – Heatmap and confusion matrix of the inter-annotator agreement for ARG1.

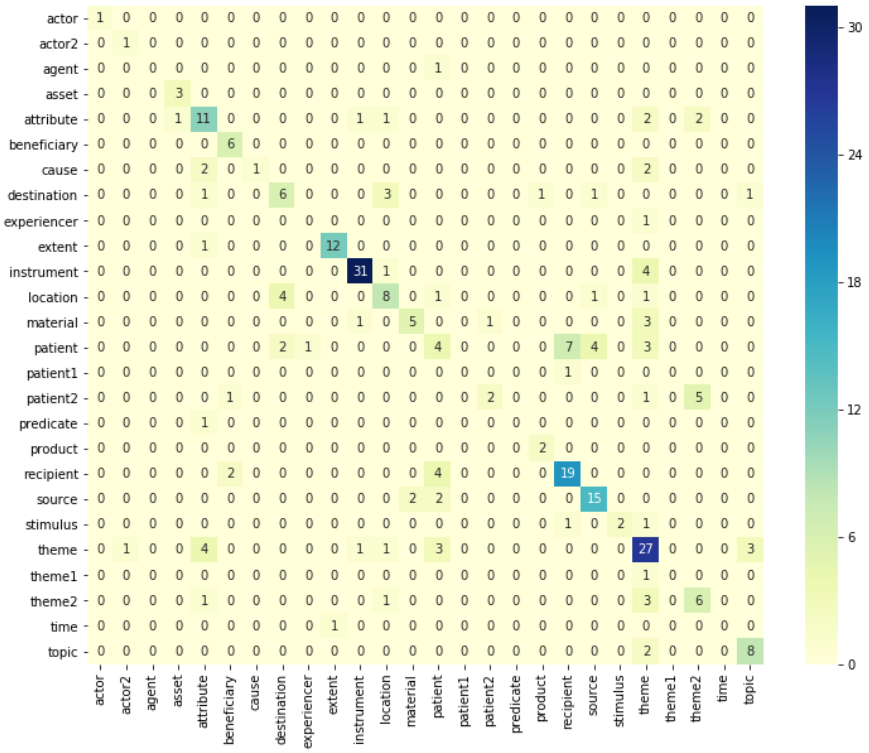


Figure A.3 – Heatmap and confusion matrix of the inter-annotator agreement for ARG2.

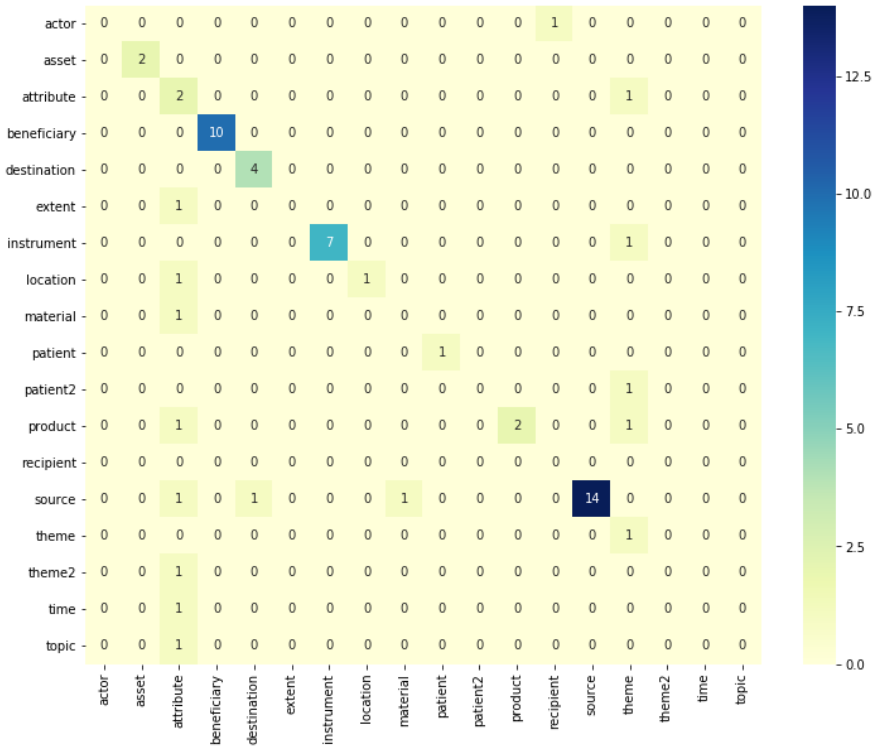


Figure A.4 – Heatmap and confusion matrix of the inter-annotator agreement for ARG3.

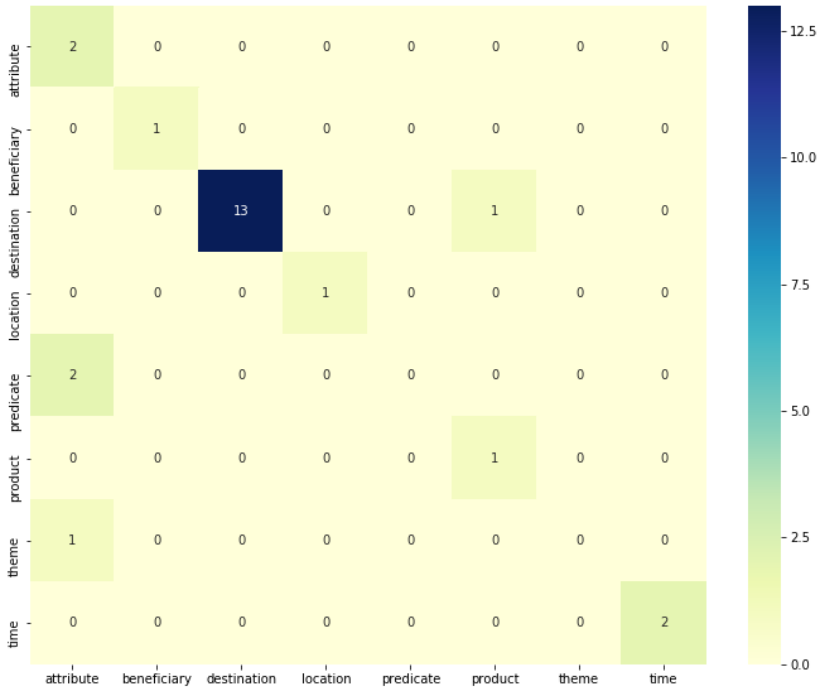


Figure A.5 – Heatmap and confusion matrix of the inter-annotator agreement for ARG4.

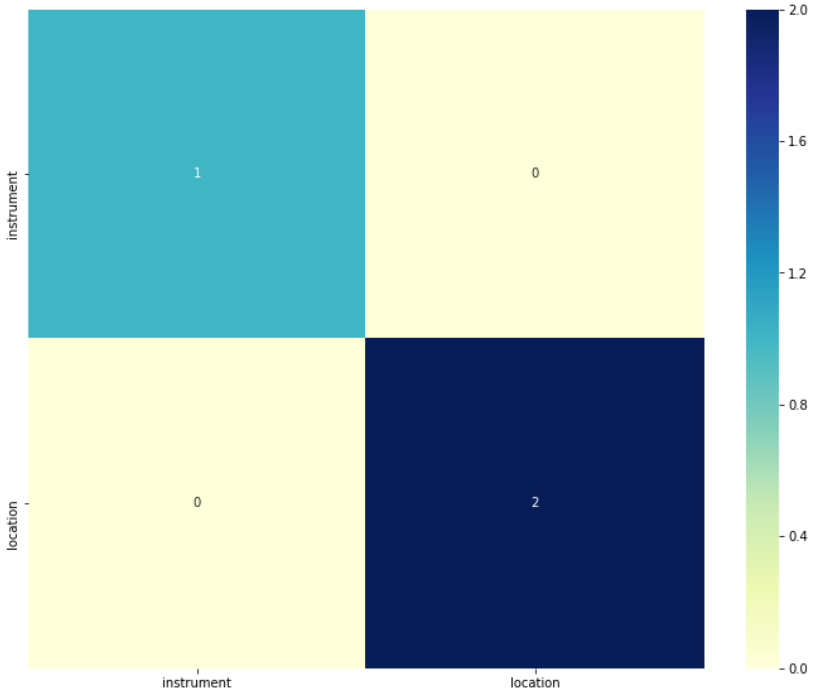


Figure A.6 – Heatmap and confusion matrix of the inter-annotator agreement for ARG5.