

Jonas Kjærandsen

IG2NLP

Automated NLP Based Institutional Statement Annotation

Master's thesis in MACS
Supervisor: Christopher Frantz
June 2024

Jonas Kjærandsen

IG2NLP

Automated NLP Based Institutional Statement
Annotation

Master's thesis in MACS
Supervisor: Christopher Frantz
June 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science



Norwegian University of
Science and Technology

Abstract

The Institutional Grammar (IG) is a grammar for systematic study and analysis of institutions. The purpose of the IG is to formalize a way to define institutions and analyse institutions expressed in natural language in order to make those analytically accessible. The essential unit of analysis are so-called institutional statements that include strategies (e.g., conventions of behaviour), norms (socially enforced behaviour) and rules (legally enforced behaviour). These institutional statements are annotated or encoded with components of the IG to perform this analysis. The task of annotating institutional statements requires knowledge and expertise of the IG, as well as substantive time investment. The purpose of this thesis is to assist in the annotation of institutional statements. This assistance comes in the form of a prototype of an automated annotation software that bases its automated annotations on Natural Language Processing (NLP) techniques and a custom matching function to match components of the IG to parts of text based on the output of the NLP pipeline. The contributions were developed in stages, starting with an initial pilot prototype for the automated annotations based solely on dependency parsing and Part-of-Speech (POS)-tagging. This was developed alongside an updated user interface prototype for the Institutional Grammar Parser (IG Parser) annotation tool that was later integrated into the main IG Parser program. The pilot project was further extended through the development of IG2NLP with additional functionality, tooling and NLP techniques. Further, an Application Programming Interface (API) was developed together with a prototype integration of the API into the IG Parser, which extends the functionality of the IG Parser with automated annotation. The main contributions of this thesis are the development of a matching function from dependency parsing, coreference resolution, Named Entity Recognition (NER), POS-tags and uFeats to components of the IG and the integration of this in a prototype. The use of custom manually created rules based on NLP techniques gives reproducible results and shows the potential of using NLP to assist in the annotation task. The solution shows good performance on regulative statements consisting of single fully formed sentences. The solution also automates the annotation of constitutive statements and nested statements, however the accuracy is lower for these features. Further contributions are the testing methodology developed for testing the annotations and the discussion of limitations, strengths and future work. In the end, the solution developed in this thesis serves as a proof of concept that lays the foundation for further work in

automated annotation using NLP.

Sammendrag

"Institutional Grammar" (Institusjonell Grammatikk) er en grammatikk for systematisk studie og analyse av institusjoner. Målet til den Institusjonelle Grammatikken er å formalisere en måte å definere institusjoner og å analysere institusjoner uttrykt i naturlig språk for å gjøre institusjonene tilgjengelige for analyse. Den essensielle enheten av analyse er såkalte "institutional statements" (institusjonelle uttrykk) som inkluderer strategier (dvs. konvensjoner av atferd), normer (sosialt håndhevet atferd) og regler (lovlig håndhevet atferd). Disse institusjonelle uttrykkene er kommentert eller kodet med komponenter av den Institusjonelle Grammatikken for å utføre denne analysen. Oppgaven av å kode institusjonelle uttrykk krever kunnskap og ekspertise i den Institusjonelle Grammatikken og betydelig tidsinvestering. Hensikten til denne oppgaven er å hjelpe med kodingen av institusjonelle uttrykk. Denne hjelpen er i form av en prototype av et program som automatiserer kodingen av uttrykkene basert på Naturlig Språkbehandling (NLP) og en modell for å knytte komponenter av den Institusjonelle Grammatikken til deler av tekst basert på resultatene av Naturlig Språkbehandling på uttrykket. Bidragene til oppgaven ble utviklet i stadier. Først ble et innledende pilotprosjekt utviklet. Dette pilotprosjektet var en prototype som automatiserte kodeprosessen basert på avhengighetsanalyse og POS-tagging. Denne prototypen ble utviklet i parallell med en prototype på et oppdatert brukergrensesnitt til IG Parser kodeverktøyet (tekstredigeringsprogram for institusjonelle uttrykk), denne prototypen ble senere integrert inn i IG Parser kodeverktøyet. Pilotprosjektet ble videreutviklet gjennom utviklingen av IG2NLP programmet med ekstra funksjonalitet, verktøy og med koding basert på flere NLP teknikker. Videre ble et API utviklet sammen med en prototype for integrasjon av API inn i IG Parser, som utvider funksjonaliteten til IG Parser med automatisert koding. Bruken av en skreddersydd modell for å knytte komponenter til tekst basert på NLP gir reproducerbare resultater og viser potensialet av å bruke NLP til å hjelpe med kodeoppgaver. Løsningen viser god ytelse på "regulative statements" (regulerende uttrykk) som består av enkeltformede setninger. Løsningen automatiserer også koding av nestede uttrykk (uttrykk som inneholder andre uttrykk) og "constitutional statements" (konstitusjonelle uttrykk), men ytelsen er lavere på disse oppgavene. Videre bidrag inkluderer testmetoden utviklet for å teste kodingen og diskusjonen av begrensninger, styrker og videre arbeid. Alt i alt virker løsningen utviklet i denne oppgaven som et konseptbevis som legger grunnlaget for videre arbeid innen automatisert

koding ved help av NLP

Acknowledgement

I would like to take this chance to thank my supervisor, Christopher Frantz, for sharing his deep knowledge of the IG, his thorough feedback and deep insight throughout the duration of the thesis and preceding project work.

Contents

Abstract	iii
Sammendrag	v
Acknowledgement	vii
Contents	ix
Figures	xi
Tables	xiii
Code Listings	xv
Acronyms	xvii
1 Introduction	1
1.1 Topic	1
1.1.1 Research Questions	2
1.2 Outline	3
2 The Institutional Grammar	5
2.1 Regulative and Constitutive Statements	6
2.2 Components of the Institutional Grammar	7
2.3 Related Work	9
2.4 Challenges	11
3 Literature Review	13
3.1 Objective	13
3.2 Document Selection	14
3.2.1 Document Exclusion	15
3.2.2 Document Inclusion	16
3.2.3 End Notes	16
3.3 Data Analysis and Methodology	17
3.3.1 Quality Evaluation	20
3.4 Categories	22
3.5 Findings	23
3.5.1 Single Focus Techniques	25
3.5.2 Dependency and Constituency Parsing	32
3.5.3 Relation of Entities	35
3.6 Discussion	40
3.7 Literature Review Conclusion	41
4 Natural Language Processing	43
4.1 Utilized NLP Techniques	43

4.1.1	Stanza	43
4.1.2	Named Entity Recognition	44
4.1.3	Coreference Resolution	44
4.1.4	Stanza Universal Dependencies Models	45
4.1.5	Dependency Parsing	46
4.2	Large Language Models	47
5	Background work	49
5.1	Pre-processing	50
5.2	Initial Experiments	52
5.3	The IG Parser User Interface Pilot Project	55
6	IG2NLP	57
6.1	Development	57
6.1.1	Requirements	58
6.1.2	System Architecture	58
6.2	Matching Function Feature Development	60
6.2.1	Component Scoping	60
6.2.2	Component Coverage Extension	63
6.2.3	New NLP Technique Implementations	67
6.2.4	Constitutive Statement Handling	70
6.3	Software and Tooling	72
6.3.1	Installation and Requirements	72
6.3.2	IG2NLP	73
6.3.3	API	74
6.3.4	Frontend	76
6.3.5	Development Tools	78
6.3.6	Testbed	79
7	Testing and Evaluation	81
7.1	Testing Methodology	81
7.2	Testing Results	87
7.2.1	Testing Without Nesting	88
7.2.2	Testing With Nesting	90
7.2.3	Exclusions	92
7.2.4	Validation testing	92
7.2.5	Discussion and Limitations	100
8	Conclusion and Discussion	111
8.1	Conclusion	115
	Bibliography	117

Figures

3.1	Documents by year	17
3.2	Document type	17
3.3	Document type of work performed	18
3.4	Type of evaluation performed	19
5.1	Dependency parse trees, first with all relevant input data, second with words replaced by their component matches.	53
5.2	Partial dependency parse tree with advcl dependency and a comma signalling an Activation Condition.	54
5.3	Text editor window before and after visualization work.	56
6.1	Basic compound word combination example.	60
6.2	Basic logical operator parse tree example.	61
6.3	Nested component handling	65
6.4	IG2NLP program diagram	73
6.5	Text editor window with the advanced UI toggled on and off respectively.	76
6.6	IG2NLP Automation frontend modal.	77
6.7	Testing diagram	79
7.1	Component counts in training data	87
7.2	Article 1.1 Component conflict example	109

Tables

2.1	IG Components and their respective IG Script symbols	8
3.1	Quality assessment of filtered articles/papers	21
3.2	NLP technique usage in the included documents	23
3.3	ContextMEL Temporality LSTM results	26
3.4	ContextMEL Negation LSTM results	28
7.1	Regulative statement component metrics without nesting	88
7.2	Constitutive statement component metrics without nesting	89
7.3	Testing of statements without accounting for nesting	90
7.4	Testing of statements accounting for nesting	91
7.5	Verification regulative statistics category one	93
7.6	Verification regulative statistics category two	94
7.7	Verification regulative statistics category three	95
7.8	Verification regulative statistics all categories	96
7.9	Constitutive validation dataset statement component metrics without nesting	97
7.10	Testing of statements accounting for nesting on the validation dataset	98
7.11	Testing of statements accounting for nesting on the validation dataset	99

Code Listings

6.1	API JSON request format	75
6.2	API JSON response format	75
7.1	Dataset JSON format	84
7.2	Testing component JSON format	84
7.3	Testing partialMatches component matching JSON format	85

Acronyms

- advcl** Adverbial Clause Modifier. 52, 53, 62, 63, 103
- advmod** Adverbial Modifier. 66
- AKE** Automatic Knowledge Extraction. 37
- amod** Adjectival Modifier. 66
- API** Application Programming Interface. iii, v, 1, 10, 49, 58, 60, 62, 69, 71, 72, 74, 75, 77–79, 92, 112
- BERT** Bidirectional Encoder Representations from Transformers. 26, 28, 30, 31, 40, 112
- cc** Coordination. 61
- CLI** Command Line Interface. 72, 73
- CNN** Convolution Neural Network. 35
- conj** Conjunct. 61
- EHR** Electronic Health Record. 25
- ER** Entity Resolution. 23, 30, 52
- IAD** Institutional Analysis and Development. 5
- IE** Information Extraction. 24
- IG** Institutional Grammar. iii, vii, 1, 2, 5–11, 13, 27–29, 31, 33, 34, 40, 41, 49, 52, 57, 58, 60, 69, 70, 72, 78, 81, 87, 91, 92, 103, 107, 111–113, 115
- IG Parser** Institutional Grammar Parser. iii, v, 1, 8–10, 49, 51, 54–56, 58, 61, 66, 72, 74, 76, 77, 101, 107, 112
- JSON** JavaScript Object Notation. 58, 69, 73–75, 78, 79, 83–85

- LLM** Large Language Model. 3, 38, 41, 43, 46–48, 71, 101, 102, 104–106, 112, 114, 116
- LSTM** Long Short-Term Memory. 26, 28, 30, 35
- NER** Named Entity Recognition. iii, 9, 16, 23, 24, 26, 27, 30–32, 35, 36, 40, 41, 44, 52, 60, 78, 94, 101, 102, 106, 110–112, 115
- NLG** Natural Language Generation. 23
- NLP** Natural Language Processing. iii–v, 1–3, 10, 11, 13, 14, 17, 18, 23, 24, 26, 29–33, 35, 36, 40, 41, 43, 44, 47–50, 57, 59, 60, 63, 73, 74, 78, 82, 88, 91, 104, 111, 112, 115, 116
- NLU** Natural Language Understanding. 14, 23
- nsubj** Nominal Subject. 109
- obj** Object. 109
- obl** Oblique Nominal. 63, 66, 70
- POS** Part-of-Speech. iii, v, 1, 23, 24, 27, 34, 45, 58, 60, 61, 69, 73, 78, 111, 112, 115
- PSRL** Prioritised Semantic Role Labelling. 38
- RE** Relation Extraction. 30, 31, 35, 36, 40, 41, 112, 115
- REST** Representational State Transfer. 49, 58, 71, 72, 74
- RNN** Recurrent Neural Network. 26, 35
- SRL** Semantic Role Labelling. 38, 39

Chapter 1

Introduction

This chapter serves as an introduction to the thesis and starts with an introduction to the topic, followed by an introduction of the problem, the contributions of the thesis, and finally an outline of the rest of the thesis.

This thesis is intended for an audience with some basic prior knowledge in the field of Natural Language Processing and with a Computer Science background.

1.1 Topic

The IG is a grammar for systematic study and analysis of institutions. Annotation of institutional statements of the IG is a process that requires time, expertise, and effort. To assist in the annotation of such statements I am researching two avenues. First the use of Natural Language Processing together with custom matching software as a method of semi-automating the annotation process. By using NLP methods such as Dependency Parsing and POS-tagging structured information can be extracted from the institutional statements, which can be used to match components of the IG to elements of the statement text. Further, an updated interface to, and integration of the new software into the IG Parser annotation tool was developed to assist in the annotation process.

This thesis has two primary software artefacts, the IG2NLP automated annotation software and the IG Parser updated user interface and prototype API integration.¹²³

¹The IG2NLP software is available on GitHub: <https://github.com/Kjaerandsen/IG2NLP>

²The updated user interface is integrated into the main IG Parser GitHub Repository: <https://github.com/chrfrantz/IG-Parser>

³The API integration prototype is available on GitHub: <https://github.com/Kjaerandsen/IG-Parser-prot-vis/tree/api-integration>

1.1.1 Research Questions

This thesis seeks to answer the following research questions:

1. To what degree can NLP tools be used to automate the annotation process of institutional statements to the IG Script notation?
2. How accurate is automated parsing of text into IG Script notation using the solution?

The first research question can be divided into two subquestions:

1. Which NLP techniques could be used to help automate parts of the IG annotation process?
2. What are the current trends in the relevant papers on unstructured text processing with NLP?

These questions will be mainly addressed through the related work section (section 2.3), and the subsequent literature review in the third chapter (chapter 3) which gives a more general insight into the current landscape of NLP with a focus on techniques potentially applicable to automating IG annotation. This chapter is followed by a chapter introducing the specific NLP techniques and toolkit used in the prototype (chapter 4). Finally, the prototype and development chapters (chapters 5 and 6) go into specific detail on the way NLP techniques can be applied for this task.

The second research question is answered by using statistical metrics in the testing and evaluation chapter (chapter 7). Where the automated annotations produced by the developed IG2NLP prototype are evaluated by comparing the annotations to a manual annotation across different datasets.

1.2 Outline

The thesis is structured in chapters as described below:

- Chapter 2: introduces the Institutional Grammar, its background, and its use cases along with examples.
- Chapter 3: presents the background literature review performed to gain insight into the wider field of Natural Language Processing and to find NLP techniques that can potentially be used to automate the annotation of institutional statements.
- Chapter 4: goes into further detail on the NLP techniques used in the solution presented in this thesis, and presents an introduction and discussion of Large Language Models (LLMs) in relation to their potential strengths and weaknesses for the automated annotation task.
- Chapter 5: showcases the background and basis of the "IG2NLP" solution developed for this thesis, including information on the datasets used, pre-processing, and pilot projects.
- Chapter 6: showcases the IG2NLP software developed for this thesis, including various stages of development, principles behind the design, the implementation, surrounding tooling and integration, and a brief evaluation of the performance of the solution.
- Chapter 7: presents and discusses testing of the solution, this includes the results of this testing on various statements, its implications, the limitations of the solution, and potential ways to address these limitations.
- Chapter 8: summarizes the thesis, discusses potential future improvements and work, and concludes the thesis.

Chapter 2

The Institutional Grammar

The Institutional Grammar (IG) was first presented in 1995 in an article by Sue E. S. Crawford and Elinor Ostrom named "A grammar of institutions"[1]. Crawford and Ostrom presented a grammar of institutions with the "ADICO" syntax, containing the "*Attribute, Deontic, Aim, Condition and Or Else*" components, and presented a method of using the grammar for the systematic analysis of different types of institutions, namely shared strategies, norms and rules[1]. The purpose of the IG is to formalize a way to define institutions and analyse institutions expressed in natural language in order to make those analytically accessible. The Essential units of analysis are so-called institutional statements that include strategies (e.g., conventions of behaviour), norms (socially enforced behaviour) and rules (legally enforced behaviour). This initial paper was followed by several papers extending the IG, and using it in research, and analysis. A framework for this analysis was later formalized with the Institutional Analysis and Development (IAD) framework[2]. Following the formalization of the IAD framework Basurto et al. presented an attempt at coding institutional statements from U.S. legislation with the IG[3]. This article provided guidelines for identifying institutional statements, coding and analysing them, and further included a method of nested analysis looking at aggregating units of observation for analysis instead of only looking at individual units. The article showed how the IG can be used to describe policy and to get insight into roles, responsibilities and emphasis in the policy, and concluded with a set of limitations including inter-coder reliability and ambiguity[3]. To mitigate some of these limitations Siddiki et al. presented a revision of the guidelines discussed above for applying the IG and presented a new component to the grammar, the so-called "Object" in the article "Dissecting Policy Designs: An Application of the Institutional Grammar Tool"[4]. Siddiki et al. provide a definition of the Object as: "the inanimate or animate part of a statement that is the receiver of the action described in the aIm and executed by the agent in the Attribute"[5]. The purpose of the Object was to reduce ambiguity in coding by making a clear distinction between the Object component and the Attribute and further by helping to distinguish the other components in a statement. Another purpose of the paper was to increase inter-coder reliabil-

ity, in addition to helping in analysis. Another subsequent development to the IG was the introduction of the Nested ADICO (nADICO) by Frantz et al. in the paper "nADICO: A Nested Grammar of Institutions"[6]. This paper expanded on the nesting capabilities of the IG with refinements to allow capturing and expressing the intricacies of institutions. The paper further aimed to refine the differentiation of norms and rules[6]. Finally, Christopher Frantz and Saba Siddiki presented the IG 2.0[7]. The Institutional Grammar 2.0 included an updated IG syntax, supporting the annotation of both constitutive and regulative statements, alongside updated coding guidelines through the supplementary IG 2.0 Codebook[8]. In the later released "Institutional Grammar" Book Frantz and Siddiki present a formal syntax for annotating institutional statements in text with the IG Script notation[9].

This thesis focuses on the IG 2.0 and uses the IG Script notation of the IG 2.0 in the given examples and in the annotation work. The related work section on the other hand (section 2.3) will discuss related work that was performed with the prior "IG 1.0". The next sections will go over the two types of institutional statements; regulative and constitutive statements. This will be followed by an introduction to components of the IG and the IG Script notation, in addition to related work. Finally, a subsection on challenges in annotation using the IG is presented.

2.1 Regulative and Constitutive Statements

The IG 2.0 has components for both regulative and constitutive statements. In context of the IG 2.0 Regulative statements are statements that describe rules or norms that regulate behaviour, Frantz and Siddiki describe regulative statements as: "regulative statements describe actions for actors within particular contexts. They may further indicate prescription and consequences related to the referenced action." [10]. A basic example of this can be "Students must attend class in a given course, or else they will fail the course", in this case, the actor or student has a rule that must be followed. If the rule is not followed by the actor in the specified circumstance then there is a consequence of failing the course specified.

Constitutive statements on the other hand are statements that describe facts or describe the setting, this can be definitions of terms for example, or a description of roles. Frantz and Siddiki explain that "constitutive statements parameterize features of the institutional system within particular contexts"[10]. An example of a constitutive statement could be describing what a student refers to in the previous example. For example, "A student refers to a person currently attending an educational institution", where a basic description of what a student is in the context of the statement is provided. In addition to statements that are exclusively constitutive or regulative, there are hybrid statements with nesting where a part of the statement can be constitutive nested within a regulative statement for example.

This thesis provides an attempt at automating the annotation of both constitutive and regulative statements, while the problem of detecting whether a

statement is regulative, constitutive or hybrid is a challenge that is not solved in this thesis.

2.2 Components of the Institutional Grammar

The IG uses components to describe meaning in institutional statements. For regulative statements for example, this can be distilled into the main components of the sentence. For example, with the sentence "Students must attend class", the obligation is "must". This is the Deontic component of the statement. Then looking at the sentence we describe who or what the obligation belongs to, in this instance "students", which defines the Attribute component of the statement. Then, we know the actor, the obligation, and now the action is the Aim component, which is "attend". Finally, the Direct Object component describes what the direct object of the sentence is, or what the Attribute acts upon with the Aim. In this case, it describes what the student must attend, which is "class". By using these components of the IG we can extract meaningful information from the institutional statement. Further components of regulative statements also include Activation Conditions which describe the conditions for the statement to be in effect, such as an activation date, Execution Constraints describing constraints for the statement such as defining a timeline for completion of a task and Properties of various components giving additional context to the components. For constitutive statements, the Attribute component is replaced by the Constituted Entity, the Aim with the Constitutive Function, the Deontic by the Modal and the Objects by the Constituting Properties. The two statement types do however both use the Activation Condition, the Execution Constraint and the Or Else components.

The Institutional Grammar Script Notation

The IG Script notation is a way to annotate institutional statements in-text. By encapsulating words and sentences with a symbol such as "A" for the Attribute component, and a set of matching brackets. A basic example of an Attribute component is: "A(Entity)". The "A" refers to the specific component, and the parentheses signal that the component is not nested, and define the scope of the component. A component can contain multiple words: "A(Member State)", a suffix to link components and their respective properties: "A1(Entity) A1,p(Entity Property)", and semantic annotations: "A[entity=Member State](it)", which in this case describes who or what entity the "it" refers to. Further components can have internal scoping when dealing with logical operators For example, "A(Students [AND] Teachers)", and can have nested components contained within such as for an Activation Condition:

"Cac{if the A(Member State) A,p(concerned) I(decides not to address) the Bdir(recommendations [OR] substantial part thereof) }"

This example is the first part of statement 7.3.3 of the dataset used in the development and testing of this thesis. Subsequent examples will also be from the development dataset. For an example of a fully annotated statement with nesting, suffixes, logical operators and multiple components see the following:

*A(Member States) **A1,p**(relevant) **A1**(EU institutions), **A1**(bodies) and **A1**(agencies) **D**(should) **I**(ensure) that, **Cac**(in cases of large-scale cybersecurity incidents [**AND**] crises), **Bdir**{they **A**([Member states ...]) **I**(coordinate) **Bdir,p**(their) **Bdir**(efforts) **Cex**(through a Joint Cyber Unit which enables mutual assistance through expertise from Member State authorities [**AND**] relevant EU institutions [**AND**] bodies [**AND**] agencies)}.*

This example includes Attribute components linked to the Attribute Property "relevant" with a suffix, a nested Direct Object component, logical "and" operators inside an Execution Constraint, inference of the Attribute in the Direct Object and more. Through this example a wide range of the capabilities of the IG Script notation for in-text annotation can be seen, however there are additional features that are not present in this example such as semantic annotations, the Indirect Object component, and the components of constitutive statements.

A table showing the full list of components and their respective IG Script notation symbols as used in the IG2NLP software as presented in the IG Parser syntax guide can be seen below (table 2.1):¹

Symbol	Component
A	Attribute
A,p	Attribute Property
D	Deontic
I	Aim
Bdir	Direct Object
Bdir,p	Direct Object Property
Bind	Indirect Object
Bind,p	Indirect Object Property
Cac	Activation Condition
Cex	Execution Constraint
E	Constituted Entity
E,p	Constituted Entity Property
M	Modal
F	Constitutive Function
P	Constituting Property
Pp	Constituting Properties Property
O	Or Else

Table 2.1: IG Components and their respective IG Script symbols

¹IG Parser Syntax guide <https://ig-parser.newinstitutionalgrammar.org/help/>

These symbols will be used throughout this document to refer to their respective components. Please refer back to this table in such cases.

Through these various symbols the annotation of statements can be performed using a basic text editor, or pen and paper. Although, there are specific annotation tools more fit-for-purpose such as the INCEpTION platform² with the IG layers³, and the tailor-made IG Parser⁴, which is specifically designed and developed for this purpose.

2.3 Related Work

The task of automating the annotation process has been previously attempted. These previous attempts will be discussed in this section, along with their details, contributions, limitations and future work. This section is partially based on previous related work sections from the Research Project Planning (MACS4000) deliverable and the Advanced Project Work (APW) deliverable of the previous semester. However, this section goes into further detail and has been mostly rewritten. As discussed above these two papers do not use the IG 2.0, which has implications to the comparability of these results to the solution in this paper. There are also additional factors to consider when annotating using the IG 2.0 with the increased feature set leading to more alternatives and complexity when annotating.

Matia Vannoni proposed a method based on computational linguistics for extracting the "Attribute", "Object", "Deontic", "Neg", "Aim", Object Properties, Attribute Properties, and "Conditions" from institutional statements[11]. This method used dependency parsing to extract these features from text through a manual mapping-based approach. The "Or else" component was not covered in this solution. Vannoni et al. mentions integration of NER, and handling co-referencing as opportunities for future work, in addition to machine learning for future research. Another aspect for potential future work is the inclusion of topic modelling as a potential method for "categorizing conditions according to their type (time, space, and so on)"[12]. Potential limitations mentioned in the paper are in validation through comparison to a manual encoding of the dataset, as inter-coder reliability can cause inaccuracies in the manual encoding.

Another article by Rice et al. presents the development of a solution for automated annotation of policy texts[13]. This approach is also based on dependency parsing, but instead of using a manual mapping approach from the dependency parsing output to IG components, a machine learning approach is taken. Where a supervised model is trained to map the output of the statement after dependency parsing into IG elements, this comes with the benefit of potentially higher accuracy, but has a risk of overfitting to the datasets used in training. This can cause

²INCEpTION annotation platform website: <https://inception-project.github.io/>

³Institutional Grammar Layers for INCEpTION: <https://github.com/InstitutionalGrammar/IG-Inception-Layers>

⁴IG Parser website: <https://ig-parser.newinstitutionalgrammar.org/>

the solution to be less accurate on institutional statements in other fields, or with different sentence structures or formatting than what the model has previously seen. Another downside to training a model is the requirement of labeled training data. The specific components the solution covers are the "Attribute", the "Aim", the "Deontic", "Objects", "Conditions", and the "Or else".

Both of these articles use an earlier version of the institutional grammar without the nesting capabilities of the nADICO or the IG 2.0 and the semantic annotation of IG 2.0. Another aspect to consider is that due to the earlier version of the IG the list of potential components is smaller than that of the IG 2.0, for example the Condition component mentioned in these papers does not distinguish between Activation Conditions and Execution Constraints. Further, both papers rely on dependency parsing as the primary technique behind their mapping approach, which shows that there is potential in using NLP techniques for automated annotation of institutional statements, and that dependency parsing is an effective technique for this task. However, there are several additional available NLP techniques that the papers have not tested. Another aspect that both papers touch on is the use of machine learning models for the matching of IG components, with Vannoni et al. mentioning it as potential future work, and Rice et al. basing their solution on machine learning. However, basing the matching on a machine learning model can also introduce potential drawbacks such as the requirement of training data, reproducibility and the potential of overfitting to the datasets used in training.

To differentiate and expand upon this related work I will be looking into using additional NLP techniques for the automation, additionally, the solution described in this thesis is built for the IG Script notation of the IG 2.0, which uses a newer syntax with different characteristics and additional features. The addition of further features necessitates solutions for handling more statement data and detection of additional components. An example of the added complexity is the handling of nested component structures, with components potentially containing nested components within, and the handling of constitutive statements that have their own components, in addition to some shared components with regulative statements. For the initial prototype discussed in a later section (Section 5.2) the focus was on annotating the "Object", "Attribute", "Aim", "Deontic" and "Activation Condition" components using dependency parsing, to gauge the performance of a manual mapping approach based on dependency parsing for the IG Script notation. This prototype was then expanded upon with further component coverage, and the inclusion and utilization of additional NLP techniques (Section 6.1). Finally, another innovation of the solution in this thesis is the development of an API and the prototype integration into the IG Parser annotation tool, extending the utility of the solution by facilitating user input on request.

2.4 Challenges

Annotation of institutional statements is a multi-faceted issue with increasing levels of complexity depending on which features are utilized.

With the inclusion of further complexity through additional features the annotation becomes more resource intensive. As the more encompassing annotation requires increased knowledge of the IG to handle more advanced annotation tasks, and performing these additional annotation tasks also increases the time investment necessary. The primary challenge this thesis addresses is the annotation process, specifically the time and expertise required to annotate institutional statements. For more basic statements the solution presented in this thesis frees up time by allowing for rapid batch processing. While for more advanced statements manual intervention is still necessary in several cases, however the automation can assist in the groundwork by providing an initial annotation suggestion, thereby offering a more efficient semi-automated coding. An iterative approach may also be utilized by coders where they look at the statement, use the automated annotation, and then make iterative alterations to approach a more accurate annotation, or split up the statement in smaller parts that are separately ran through the solution.

The development of such an automation solution has its own challenges, in the expertise required, the necessity of varied datasets to base the automation on, selection of techniques or ground methods for annotation, reproducibility of results and the potential of overfitting the solution to a certain dataset or statement structure. The expertise required comes in the form of both knowledge of the IG and in NLP techniques to be able to detect and utilize patterns in statements to be able to create consistent rules or a method for automated annotation. The choices taken to approach a solution and mitigation of the various challenges outlined above will be presented and discussed throughout the thesis.

After initial research into the problem of annotation a literature review was performed to gain deeper insight into the field of NLP to identify possible candidate techniques for the inclusion in the automation process. The next chapter presents this literature review and its results.

Chapter 3

Literature Review

This literature review was first performed as a part of the Research Project Planning (MACS4000) course in the preceding autumn semester. It has subsequently been updated and newly released papers have been included. The following sections contain a literature review on Natural Language Processing in unstructured text with the purpose of finding out if there are natural language processing (NLP) techniques that can be used to automate the process of annotating institutional statements into IG Script notation.

3.1 Objective

The goal of this literature review is to find out what NLP techniques are relevant to automating the annotation of Institutional Statements. For this literature review, I have the following research questions:

1. Which NLP techniques could be used to help automate parts of the IG annotation process?
2. What are the current trends in the relevant papers on unstructured text processing with NLP?

Further, the goal of this literature review is to assess the field of NLP processing in unstructured text related to the goal of automating the annotation process, this includes learning the relevant trends, strengths, and weaknesses in the field. It was narrowed down to get a grasp on which NLP techniques can be used to add automation capabilities to the IG Parser. As such the literature review will be focused on techniques useful for in-text annotation and techniques relevant to the requirements of the Institutional Grammar.

3.2 Document Selection

The initial search resulted in **130,859** documents for NLP written out and in abbreviated form, while for both NLP and Natural Language Understanding (NLU) there were **10,719** documents. To narrow down the scope of the literature review to a manageable and relevant selection several steps have been taken. The focus will be on unstructured text. As such the initial search term has been narrowed down to:

(nlp OR (natural AND language AND processing)) AND ((natural AND language AND understanding) OR nlu) AND text AND unstructured

By completely spelling out NLP and NLU in addition to including the abbreviations the chance of accidentally missing documents is reduced. The search was performed in Scopus, which is a database of peer-reviewed literature such as journal articles and conference papers.

The initial search for this revised term resulted in **259** documents. This was narrowed down by limiting the language to English, then by specifying the document types to articles, book chapters, conference papers, reviews, and conference reviews, and finally the subject area to computer science. This resulted in a total of **163** documents. As the field is constantly evolving and to narrow down the amount of papers to a more manageable level the search was limited to papers from 2017 and onwards. 2017 was included as it showed a rather substantial increase in documents published compared to the previous years. Now the total was **127** documents, and after checking for availability through NTNU the final total of documents was **114** documents as of November 28th, 2023. The search was subsequently performed again a last time on the 14th of May 2024, in which two new relevant papers were found and introduced to this literature review.

3.2.1 Document Exclusion

After reading the abstracts and conclusions of the documents they were divided into categories and new exclusion criteria were formed to narrow down the scope of the literature review to a more relevant selection of documents. The exclusion of documents was done through categorizing them, and evaluating whether the focus of the document and the techniques used were relevant to automating in-text annotation of Institutional Statements into IG Script notation. In the following section, a list of categories used for exclusion will be presented.

- Document classification or clustering
Documents where the focus is on extracting information from documents and using it to classify or cluster documents. This includes applications such as creating knowledge graphs and word clouds. This was deemed irrelevant to the task as the focus is on annotating text in documents, not on information extraction or generating statistics of documents.
- Image or video captioning or information extraction
Extracting information from images or video is not directly relevant to in-text annotation. This includes efforts for text extraction from videos and images and captioning of images or videos.
- Question generation or answering and conversational agents
While question answering models and conversational agents could be used to assist a user in annotation. This would be a supervised or semi-automated process which is not the focus area of this project.
- Summarisation
As with the point above, summarisation is not directly relevant when dealing with a fully automated system for annotation.
- Dataset or corpus generation and documents focusing on improving training performance of models
These are subjects related to machine learning models and data-set creation which is outside of the scope of this project.

Further Exclusions

By further reading through documents with a focus on sentiment analysis this was deemed to be irrelevant. Sentiment analysis is the task of classifying text or parts of text with sentiment. Generally, this sentiment is in the form of positive or negative sentiment. This sentiment classification can be useful for evaluating user feedback which can be seen in the paper "How We Failed in Context: A Text-Mining Approach to Understanding Hotel Service Failures" by Shuyue Huang, Lena Jinggen Liang, and Hwansuk Chris Choi where they use sentiment analysis on hotel reviews to understand service failure[14], and the paper "Sentiment Analysis Application and Natural Language Processing for Mobile Network Operators' Support on Social Media" by Kingsley A. Ogudo and Dahj Muwawa Jean Nestor which covers using sentiment analysis to gauge customer satisfaction on mobile network

services[15]. Sentiment classification is not relevant to parsing institutional statements, however, as sentiment is not a factor in these kinds of statements, where the focus is on the logic and dependencies within statements, not on opinions or sentiment. Papers that focused on sentiment analysis were therefore excluded from this literature review.

3.2.2 Document Inclusion

After the initial reading of abstracts and conclusions, the inclusion criteria were updated to include documents that used or discussed the following NLP techniques:

1. Relation Extraction
2. Semantic Role Labelling
3. Temporal
4. NER / entity recognition
5. Co-reference resolution
6. Negation
7. Causality Detection
8. Dependency Parsing
9. Constituency Parsing

Papers with one or more of these techniques that were not excluded due to the exclusion criteria outlined above were included in the review.

3.2.3 End Notes

To not exclude relevant documents based solely on these criteria the abstracts and conclusions of each document were read, and where extra attention was needed the documents were skimmed through to see if they contained relevant information. Following the exclusion of papers based on the reasons outlined above there were 26 remaining documents. Following the updated search on the 14th of May, a further two documents were included, with an updated total of 28 remaining documents.

3.3 Data Analysis and Methodology

Through looking at the included papers they can be categorized by year, domain, NLP techniques, and document type. After classifying the documents they will be introduced in categories to introduce and discuss their findings. Before the techniques are related to automated annotation of institutional statements and trends in the documents are discussed as a whole.

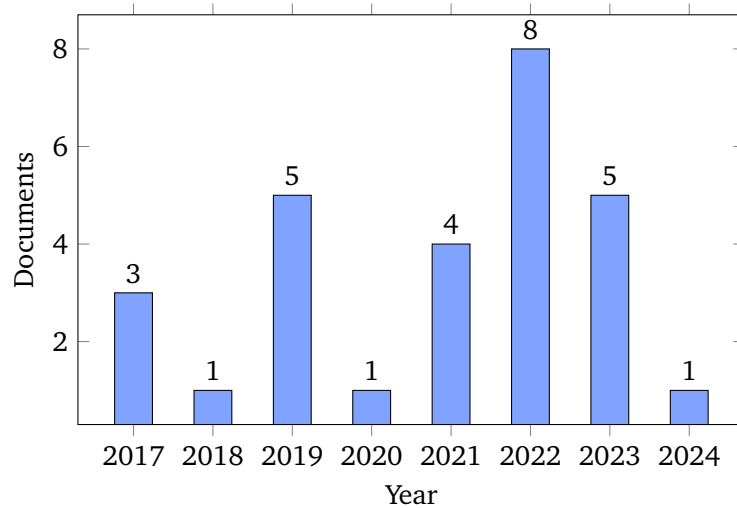


Figure 3.1: Documents by year

When looking at the release date of the papers in the table above (table 3.1) we see a relatively stable line, with a spike of relevant papers in the year 2022, and a lower number of relevant papers present in 2020 and 2018. The average amount of relevant papers per included year is just under four. These papers can be split into types of documents and the type of work done.

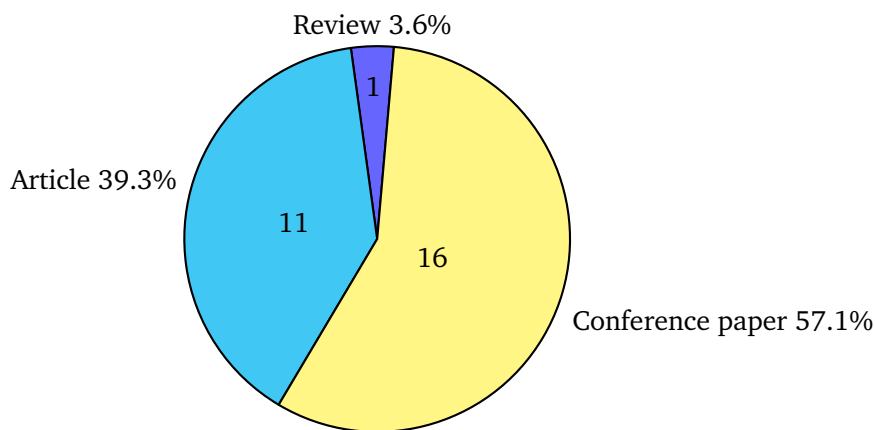


Figure 3.2: Document type

The chart above (figure 3.2) shows that the documents are primarily articles and conference reviews, with a single review document, and the majority of almost 54% of the documents are conference papers. The majority of conference papers can be an indication that the field is active as conferences have more rapid release cycles than other types of literature.

Further, we can look at the type of work performed in each paper. For this, they have been categorized roughly by their contribution to the space. With the following categories:

- Review or survey
Documents that give an insight into the field at large, or a subset of the field.
- Application of NLP
Documents which apply NLP techniques or tools to a problem.
- NLP technique development
Documents that develop something new within NLP, such as a new technique, or developing a model for a new domain.

This categorization can be seen in the pie chart below (figure 3.3:

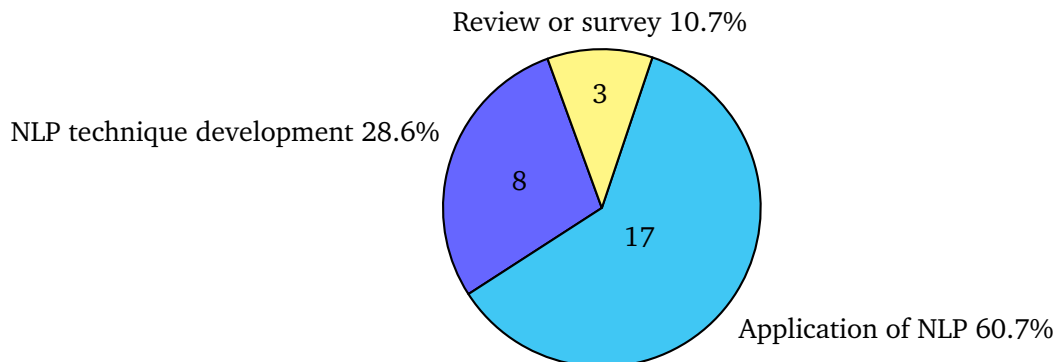


Figure 3.3: Document type of work performed

The majority of papers are in the category of application of NLP, with NLP technique development coming in second, and finally, there are two surveys and one review.

Further, the papers were divided into rough evaluation groups. Where the categories are based on the testing or evaluation performed in the papers. Including qualitative, quantitative, and both. This grouping excluded the surveys and the review as they have a different structure than the other papers. Qualitative evaluation is an evaluation where the authors describe the outcome and try to explain the underlying factors leading to the outcome and what can be done differently in the future. Quantitative evaluation is where the authors use methods such as statistics for precision and recall to evaluate performance. This evaluation categorization can be seen in the pie chart below (figure 3.4):

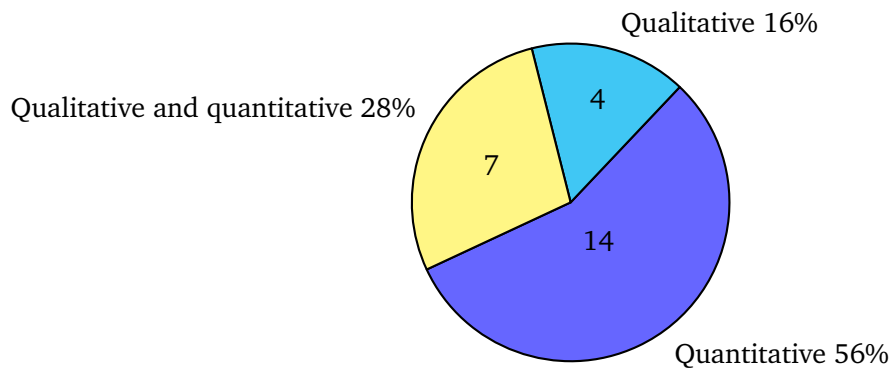


Figure 3.4: Type of evaluation performed

3.3.1 Quality Evaluation

To assess the quality of the included studies the following questions will be asked of each document and each question will be rated from one to five:

1. Is the aim of the paper clearly defined?
2. Are the results clearly presented and evaluated?
3. Are the techniques used general or applicable to other domains?
4. Is the focus of the paper relevant?

The ranking scale used is a five-point scale with one and two representing a disagreement with the question, three representing a weak agreement, and four and five representing stronger agreement with the question. The first question acts as a basic quality evaluation question of the paper that looks within the paper to see whether the aim is clearly defined. The second question looks at the results of the paper and whether they are made clear, and are properly presented. The third question looks at the techniques used in the papers and is an evaluation of whether they are more domain- or topic-specific or more general. Finally, the fourth point looks at the relevance of the paper's focus. This is a combination of evaluating the relevance of the techniques used for the goal of this literature review, which is finding techniques to help in the automation of institutional statement annotation or encoding. The other element evaluated in this point is the focus area of the paper, for example, if the paper is focused on extracting information on materials as in the paper "Materials information extraction via automatically generated corpus" then the focus area is not directly relevant, however the techniques used are relevant. This combination of lower topic relevance and higher technique relevance, or the inverse causes a ranking of three. If the score were to reach a number lower than three for this fourth question then the paper would be excluded from the literature review due to relevance. Further, a combined average score of less than four would be a reason to consider omitting the paper from the literature review. This average is made from all the questions weighted equally, this is due to the importance of all the questions. As they all cover important aspects for the literature review, the first two cover the quality of the papers, and the last two cover whether the techniques can be applied to other domains and the relevance of the paper. The combination of all these four points gives a final more encompassing evaluation.

The rankings of the included papers can be seen in the table below (table 3.1):

Document	1	2	3	4	Average
[16]	5	5	4	3	4.25
[17]	5	5	4	3	4.25
[18]	5	5	3	3	4
[19]	5	5	5	4	4.75
[20]	5	5	5	5	5
[21]	5	5	4	4	4.5
[22]	5	4	5	4	4.5
[23]	5	4	5	4	4.5
[24]	5	4	3	4	4
[25]	5	5	3	4	4.25
[26]	5	5	5	4	4.75
[27]	5	5	3	4	4.25
[28]	5	N/A	3	4	4
[29]	5	5	3	3	4
[30]	5	5	4	4	4.5
[31]	5	5	3	3	4
[32]	5	5	4	4	4.5
[33]	5	5	4	4	4.5
[34]	5	5	4	4	4.5
[35]	5	5	4	3	4.25
[36]	5	5	3	3	4
[37]	5	5	5	3	4.5
[38]	5	N/A	3	5	4.33
[39]	5	4	4	4	4.25
[40]	5	N/A	5	4	4.67
[41]	5	5	3	3	4
[42]	5	5	3	4	4.25
[43]	5	5	5	4	4.75

Table 3.1: Quality assessment of filtered articles/papers

The documents included all have averages between 4 and 5 which means that they are deemed as relevant papers with a sufficient quality to be included in the literature review. This assessment is based on the points discussed above, specifically, the last two points look at if the techniques discussed or used in the papers are relevant to this literature review and if the focus of the paper is relevant to this literature review. Papers that have focus areas that are not directly relevant to this literature review may be included if they use techniques or methods that are relevant. In such cases, the relevant parts of the documents will be discussed. The "N/A" is specific to surveys or overview papers where the main focus is on introducing other works in the field or the field at large.

3.4 Categories

To cluster the documents a set of categorizations has been made, as described in the methodology section. These categories include details on the documents such as their type of work, domain, techniques used, and more. In this subsection, I will present some statistics surrounding these categories and briefly discuss relevant points to the different categories.

For domains, there was a wide spread of domains covered in the selection, with a majority of the documents focusing on general-purpose techniques. A few papers also focused on languages other than English, either exclusively or inclusively. Although there were also papers spread out in specific domains such as a few papers in the clinical domain, three papers focusing on security or cybersecurity, and several other small groupings.

Further, all the papers were categorized by the techniques they applied or discussed. The following techniques were the main techniques which were deemed relevant:

1. Relation Extraction
2. Semantic Role Labelling
3. Temporal
4. Named Entity Recognition / Entity Recognition
5. Co-reference Resolution
6. Negation
7. Causality Detection
8. Dependency Parsing
9. Constituency Parsing

These techniques were grouped into three main groups with the first two surrounding defining a relationship between entities in a text. The second grouping was for items four to seven of the techniques. These are all techniques that look at a single aspect in the text respectively. Finally, dependency and constituency parsing. These techniques both look at the basic grammatical structure of the text at a sentence level. They also both output the results as a tree with structure dependencies related through the branches. The coverage of the various techniques discussed can be seen in the table below (table 3.2):

Technique	Documents	Percentage
Relation Extraction	14	50 %
Semantic Role Labelling	3	10.7%
Temporal	4	14.3%
NER / ER	21	75 %
Co-reference resolution	4	14.3%
Negation	1	3.6%
Causality detection	1	3.6%
Dependency Parsing	5	17.9%
Constituency Parsing	2	7.1%

Table 3.2: NLP technique usage in the included documents

When looking at the percentages we can see that NER / Entity Resolution (ER) are used by more than two out of three papers, showing that they are very prominent techniques. Further relation extraction is also used in a lot of documents, with almost half the documents performing or discussing some form of relation extraction. For the final grouping dependency parsing shows over twice the amount of usage as constituency parsing in these documents.

The following sections will go over each of these groupings, present the papers that are within each, and their findings.

3.5 Findings

NLP is a wide field with a multitude of different techniques and models developed to process text. Pajila et al. presents the landscape of NLP through a Survey[38]. Pajila et al. describe how NLP can be divided into components and these components are categorized into two main types, NLU and Natural Language Generation (NLG)[38]. For this literature review, the focus is on the field of NLU so the details on NLG are out of scope and will not be discussed. Further, the paper describes how NLP contains the components phonology, morphology, pragmatics, semantics, lexicology, and syntax, as core linguistic features of natural language. These features will be introduced below:

Phonology studies sounds, and how they are organized and developed.

Morphology is used to study words, how they are structured, how they can be created through the combination of morphemes, and how the form of a word can be used to indicate a variation. This includes adding "un" to "known" to create unknown, changing the meaning from suggesting something is known to suggesting the opposite and changing the inflexion of a word to signify tenses, for example, "I am driving" and "I drove".

Lexicology is the study of words and what they communicate in terms of meaning and is connected to POS-tagging. POS-tagging tags words based on the surrounding words and context, and uses the most likely tag for each word if there

are multiple alternatives. An example of a POS-tag is "AUX" which is the auxiliary that accompanies a verb and contributes additional meaning to the verb. I.e. "must" or "may" before a verb signifies if the verb is an obligation or a voluntary option.

Syntax looks at sentences at the phrase level instead of the word level. Looking for combinations of words that indicate more meaning than the words themselves. This method relies on keeping the structure of the sentence and therefore does not use the same pre-processing steps as other techniques may use. It is important to keep structure to keep the meaning consistent, for example, "John has a tall house" and "tall John has a house" imply that the house is tall and that John is tall respectively. Further for POS-tagging the structure can be important as the meaning of a word can change depending on the structure.

Semantics focuses on the meaning of phrases. A combination of words can signify meaning not directly written out. This is similar to and described by the phrase "reading between the lines", which does not mean actually reading between the lines, but instead finding meaning by considering the bigger picture. This can also include understanding which meaning a word has in context, as several words can have multiple meanings.

Pragmatics look at context and how the context can affect the meaning of language. For example, the phrase "I love dogs" in a normal conversation might be interpreted literally, while in a situation where a dog has made the floor dirty, the same phrase might convey dissatisfaction instead.

Further, the paper describes applications of NLP as chatbots and virtual assistants, sentiment analysis, machine translation, speech recognition, text summarization, NER, and information retrieval. In this literature review, the NER and information retrieval use cases are the most relevant. With NER described as "a major NLP application"[38], that is used to extract entities such as people and organizations from text. Pajila et al. describe how "NER involves tokenization, part-of-speech tagging, and dependency parsing."[38], showing how NLP techniques build upon each other to create more advanced functionality. In the other relevant use case of information retrieval, the paper describes Information Extraction (IE), "Information extraction (IE automatically extracts structured information from unstructured or semi-structured text input."[44]. This information includes entities and relations, in addition to events and facts. Finally, the document goes over the weaknesses of NLP. The weaknesses include ambiguity in language making it difficult to interpret text, and domain-specific language with domains having terminology which is unique to the domain. This can make it difficult to use NLP models when several domains include unique terms or meanings of words. Further training data can be a problem, with little or biased training data results can vary. Finally, explainability, where it can be difficult to understand why the tool has come to a conclusion, which might reduce confidence in the results.

3.5.1 Single Focus Techniques

This subsection will cover techniques that focus on a single aspect of language. With techniques looking at temporal aspects, negation, resolution of co-references, and recognition of entities.

Temporal

Temporal aspects in this case surround techniques used to find or relate time or dates with text. For example, looking at timelines. Amy L. Olex and Bridget T. McInnes present a review of temporal reasoning in the clinical domain for timeline extraction[28]. In this case, the focus is on timeline extraction from Electronic Health Records (EHRs) and other clinical documents[28]. Temporal information expresses when an event occurs or did occur and potentially the duration of an event. For a basic example "The weather report says it will rain tomorrow", shows an event of rain, that will occur tomorrow or the day after today. Temporal information can be expressed in many ways, such as a reference to a holiday can indirectly express a date, or a reference to a previous event can express when the current one occurred. I.e. "the patient fell 2 months after knee surgery"[28]. To process temporal data in texts temporal annotation is used, which standardizes the formatting of text and relevant information such as events and temporal information. For temporal annotation, several models have been developed over the years such as TIDES for a general domain model and THYME-TimeML specifically for the clinical domain. There are several models for temporal information extraction. This paper takes it further by building on temporal information extraction to extract timelines, within the clinical domain. This adds complexity in finding clinical events and excluding events that are not medical but may appear in general models. Further, co-reference resolution is mentioned as a possible method to remove duplicates and detect instances where the same entity is referred to in several ways. Other tasks in timeline extraction in the clinical domain include ordering of events and visualization. The paper concluded that there is still a long way to go before timeline extraction can be put into practice. Olex and McInnes described that improvements are needed in temporal identification, event identification, co-reference resolution, the ordering of events, visualization, and handling the whole patient history, which can include duplicates and many documents.

Paula Chocrón, Álvaro Abella, and Gabriel de Maeztu present "ContextMEL" a method for classifying contextual modifiers in clinical text[25]. This paper also focuses on extracting information from EHRs. The system uses expert annotations to build a dataset that is used to train deep learning models[25]. These models are trained for three so-called modifiers in the paper, which are temporality, certainty, and negation. I will focus on the temporality aspect for this section, and revisit the negation modifier in the relevant subsection. ContextMEL is a system for training models, by first annotating data and then training models on the annotations. The data that is worked on is in Spanish and Catalan, however, the authors present the methods as domain-independent, so the same process could theoretically be

applied to other domains. The temporality task divides events into three categories, with the labels "Antecedent, Plan, Current Episode"[25]. Antecedent refers to clinical history, the plan is the future plan, in this case, it is commonly a treatment and the current episode is what is the current event. The classifiers were trained using two different methods. One is based on Long Short-Term Memory (LSTM) and the other is based on the Bidirectional Encoder Representations from Transformers (BERT) language model.

LSTM is a Recurrent Neural Network (RNN) introduced by Sepp Hochreiter and Jürgen Schmidhuber in 1997 in the paper "Long Short-Term Memory"[45]. LSTM was developed to solve the problem of error backflow in RNNs where error signals would either vanish or explode causing unstable learning. To solve this LSTM has an architecture allowing for a constant flow of error signals[45]. LSTMs are suitable for sequence learning tasks that involve remembering things that have occurred multiple times.

BERT is a language model presented by Jacob Devlin et al.[46]. The BERT language model works in two stages, first a pre-training phase on unlabeled data, followed by a fine-tuning phase with labelled data. This two-stage method allows BERT to be adapted to many tasks through fine-tuning, these tasks can include tasks such as NER, inference and question-answering[46]. For the temporality aspect, the performance of the BERT approach outperformed the LSTM approach by almost three percent in accuracy with an accuracy of 84.41%. The testing was performed on a set of 100 manually annotated examples. The results also showed precision and recall of the LSTM model for the Past, Present, and Future labels[25] as can be seen in the table below (table 3.3):

Tag	Precision	Recall
Past	77.93%	78.03%
Present	77.89%	77.93%
Future	89 %	88.80%

Table 3.3: ContextMEL Temporality LSTM results

Showing the best performance for the future tag, with similar performance between the past and present tags at around 78%.

Husari et al. present a method for analysis of cyber threat intelligence based on NLP[24]. In this paper, the domain is cyber security, as opposed to the earlier papers within the clinical domain[24]. The techniques used in this paper are dependency parsing, and temporal relation extraction using temporal anchors. Where certain keywords are used to relate dependencies to temporal data. For example "then" and "before" are used to show ordering or events. This ordering is used to reformat the text with a logical temporal order. The paper finds that "the types of malicious artifacts exhibit a high temporal dependency which indicates that some types of artifacts are preconditions to other types"[24]. However, the paper is limited by not presenting more details from the test results and in the methodology using only English temporal anchors or keywords. This is a limited approach that

can not pick up all temporal data in the text, so Husari et al. describe that they will expand the set of temporal anchors to improve the coverage.

Zaeem et al. present work in modelling and analysing identity threat behaviours through text mining[17]. Again the domain is security, this time with a focus on identity theft. The paper collects information from news stories and reports on identity theft[17]. The approach used is text mining, using a combination of pre-processing techniques, noun phrases, NER, and POS-tagging. The algorithm proposed collects news articles and theft reports and then processes these documents using a pipeline. The pipeline performs preprocessing and NER, and uses the collected data to create an identity theft record which is later analyzed. Within the theft record, two temporal aspects are relevant, "time selection" and "timeline". For the time selection, two methods are discussed, the first is looking at the publication date of the article, and the second is using NER. The timeline aspect correlates loss due to an identity threat with dates. The results of these aspects, in addition to other aspects that I did not discuss, are then analyzed, presented, and compared to a manual investigation showing comparable trends for the various categories.

For IG temporal information can be useful in discovering Activation Conditions or Execution Constraint components. Where a condition can be in the form of a certain period a rule is effective. For example, when discussing curfew it might only be in effect after ten and before six the next morning. This form of condition can be seen as a temporal anchor as discussed by Husari et al.[24]. It is worth discussing how applicable the techniques used in the papers are as the first two papers are within the clinical domain, which has a lot of domain-specific language and requirements, and the last two are within security. For the clinical papers, the first paper[47] discusses a lot of domain-specific models that are not directly applicable, however, the methodology behind creating them and solving the problem of timeline extraction can be replicated in other domains. ContextMEL[25] on the other hand is generalizable, however it does require training data. For the other two papers[24][17] they use general techniques which apply to several domains.

Causality Detection

A technique for detecting causality was presented by Khetan et al. in the paper "Causal-BERT: Language models for causality detection between events expressed in text"[35]. Khetan et al. present causality detection using a BERT-based method where BERT is fine-tuned to find relations between cause and effect or "other" between events in text[35]. The technique relates two events, with a relation of the type "cause-effect" or "other", these events the model finds relations between are expressed in the same sentence. A given example of a classified true positive is:

"<e1>Dehydration </e1>from <e2>fluid loss </e2>generally is the only major problem the virus can cause, most often in the elderly."[35]

In the example, there is a causal relation between fluid loss and dehydration. The model showed F1 scores of over 90% on three datasets, with F1 scores over 95% on the same datasets when pre-training and fine-tuning were performed.

While this method of causality detection shows promising results, whether it is useful for the parsing of institutional statements is unclear. There might be use cases in finding constraints for the statements for example, but this will need to be tested.

Negation

ContextMEL which was referred to above in the subsection about temporal aspects also has a modifier for negation detection[25]. This module finds cases whether some logic in a sentence is negated or not[25]. The accuracy of the solution based on the BERT approach was 95.16% and for the LSTM approach it was 95.70% as can be seen in the table below (table 3.4):

Tag	Precision	Recall
Positive	96.67%	95.47%
Negative	92.86%	94.71%

Table 3.4: ContextMEL Negation LSTM results

The negation module showed precision and recall of over 90% in both positive and negative. Additionally, the model outperformed the state-of-the-art NegEx model on the Spanish dataset with an accuracy of 94.22% versus 83.14%. Specifically, this refers to NegEx-MES which is applied to Spanish medical texts, however, there are also NegEx systems created for other domains.

This paper shows that negation detection can achieve a very high accuracy and be applied to specific domains such as the clinical domain in this case. When looking at IG negation detection might be helpful to verify that the logic of the statement is properly conveyed through the annotation.

Co-reference Resolution

When dealing with natural language text a common occurrence is the referencing of the same entity in different ways. A basic example of this is "John and Mary are studying in the library, he has an exam tomorrow.", where "John" and "he" are the same entity. To be able to process this text correctly it is important to understand that "he" is the same entity as "John". Co-reference resolution is a NLP technique aimed to solve this problem.

As discussed earlier the paper by Olex and McInnes describes co-referencing models for the clinical domain as a method for removing duplicate information and linking data between documents[28].

Swagata Acharya, Sourav Mandal, and Rohini Basak present a model for solving arithmetic problems using NLP and classification based on rules[36]. A part of this model is co-reference resolution which is used as a pre-processing step where pronouns are substituted with nouns using NeuralCoref[36].

Huang et al. describe "Building Cybersecurity Ontology for Understanding and Reasoning Adversary Tactics and Techniques"[30]. They describe using co-reference resolution through adopting "Ellipsis Subject Resolution (ESR), Pronoun Resolution (PR) and the Entity Resolution (ER) components from EXTRACTOR"[30]. These are all methods that are used in the paper to substitute co-references with the relevant subjects in the text. ESR detects sentences starting with a verb and looks for the most probable of possible subject candidates from earlier sentences to resolve the reference. PR resolves pronoun references and ER resolves noun or verb phrases referring to an entity in the sentence.

Kunal Khadilkar, Dr. Siddhivinayak Kulkarni, and Dr. Sitalakshmi Venkatraman propose an approach for automatic summarization of speeches and essays using knowledge graphs[23]. In this paper co-reference resolution is used as a step in the proposed method for generating knowledge graphs[23]. Co-reference resolution is in this case used to link entities with their preposition occurrences. With an example given of resolving "He" as "Adam" in the text. This way all branches which lead to Adam can be properly included in the knowledge graph.

As seen above co-reference resolution can be helpful in many contexts, Olex and McInnes describe using it for deduplication[23], while the other papers use the technique as a simple substitution technique as a processing step. For the IG it is important to specify all entities clearly, and the entities specified in the statements are vital to the meaning of the statement, therefore resolving co-references is important.

Named Entity Recognition / Entity Recognition

NER and ER are techniques for recognizing entities in text. These entities can be entities such as people, places, and organizations. The recognition of entities is a NLP technique which is often used in pipelines with other techniques, and some modern methods combine NER and other techniques. An example of this is presented in the paper "Multi-Entity Collaborative Relation Extraction"[26] where NER and Relation Extraction (RE) are performed collaboratively, this paper is discussed in further detail in the relation extraction subsection. Another paper "Structured Approach for Relation Extraction in Legal Documents" first uses NER followed by RE as two separate steps, this paper will also be discussed in more detail in the RE subsection. In this section, some of the papers using NER will be presented and discussed, as well as papers presenting developments to the technique or applying it to new domains.

Siddharth Mehta, Gautam Jain, and Shuchi Mala present an approach based on NLP for exploring social media posts and tags and clustering the data to locations[39]. Their methodology includes using NER to extract location tags, which are then used to cluster the information[39].

Pengyu Zhang uses NER to extract numerical facts from Chinese text in the paper "A Numerical Fact Extraction Method for Chinese Text"[31]. The paper proposes two NER methods, that supervised methods based on BERT and are called "NER combine" and "quantity MRC"[31]. These methods take two different approaches to finding entities and quantities related to the entities in Chinese text.

Santoso et al. present a method for extracting concept for ontology building based on NER[29]. They propose a model for NER using bidirectional-LSTM which takes an input of word embeddings in the form of vectors[29]. The model performs both POS-tagging and NER simultaneously on Indonesian text and shows good performance compared to other models.

Afnan Iftikhar, Syed Waqar Ul Qounain Jaffry, and Malik, Muhammad Kamran present their work developing a NER model for legal judgments in the paper "Information Mining From Criminal Judgments of Lahore High Court"[21]. They use a labelled dataset of criminal judgments with nine entity types to train three different classifiers[21]. The methods are "conditional random field", "Maximum Entropy classifier" and "Trigrams 'N' Tags". These methods showed average F1 scores of 0.96, 0.79, and 0.91 respectively. Showing that NER can be effectively adapted to and used in the legal domain.

Boudjellal et al. present a BERT-based model for NER on Arabic biomedical text[27]. The model is focused on a very specific domain of Arabic biomedical text, which comes with the problem of having fewer resources than English models or other more general models. The extracted entities are specific to the biomedical domain and include entities such as diseases and symptoms. The model was built by using AraBERT base weights and pre-training on the AraBERT base data in addition to biomedical Arabic literature and then fine-tuned. The model developed was named ABioNER and was compared to AraBERT and BERT multilingual cased

models, showing better performance than the two on two different entity types, showing an F1 score of 85.65. This score was higher than AraBERT with 83.69 and BERT multilingual cased which had an F1 of 82.12. Showing that pre-training a BERT model on domain-specific data can lead to increased performance on data within that domain.

Another paper using NER in the clinical domain is presented by Fang Dong et al. in the paper "Chinese Medical Named Entity Recognition Based on Pre-training Model"[42]. This paper proposes a NER model for Chinese text in the clinical domain based on RoBERTa (a pretraining approach for BERT presented by Yinhan Liu et al.[48]), adversarial training and hybrid encoding layers[42]. The model was tested on three medical case datasets and showed increased performance over the baseline.

Miha Štravs and Jernej Zupančič present a method for NER in the paper "Named Entity Recognition Using Gazetteer of Hierarchical Entities"[20]. The method uses word similarities based on lemmatization, stemming, word embeddings, and similarity at the character level to find predetermined entities in text[20]. By using lemmatization and stemming the entities can be found even when they are formatted in different ways such as with spelling errors or in different tenses. Other strengths of the solutions are that they can be adapted to different domains by changing the entity gazetteers and the performance can be increased by changing out the underlying techniques for stemming or lemmatization. The solution specifically was made for and tested on Slovenian and English text, showing better performance than Elastic Search in testing.

NER is a central NLP technique that is used in many domains and languages and for many use cases. Including domains such as the clinical domain[27][42], the English language, as well as Chinese[31][42], Arabic[27], Slovenian[20], Indonesian[29], and a multitude of other languages. For use cases, it is often used as a step in a NLP pipeline or framework, where the entities are further processed or used, for example in RE where relations between entities are extracted, or in co-reference resolution where co-references to entities are handled. NER has the potential to be useful for finding various IG components in institutional statements, such as "Objects", and "Attributes" and potential constraints for entities related to a time or a quantity.

3.5.2 Dependency and Constituency Parsing

"In computational linguistics, the term parsing refers to the task of creating a parse tree from a given sentence." [49]. Dependency and constituency parsing are two techniques that look at grammar at the sentence level and structure dependencies within the sentence in the form of a parse tree. The techniques approach this problem using two different types of grammar, but both end up with trees displaying the logical dependencies or structure of a sentence.

Dependency Parsing

With dependency parsing "the syntax of the sentence is expressed in terms of dependencies between words" [49]. These dependencies bind two words together, and have a direction, either left or right, and a dependency type. There are some additional requirements to the nodes, they all connect to the root, and every node has one branch connecting to the node, except for the root. Some of the dependencies can include objects and auxiliary. The exact dependencies vary based on the implementation. As discussed earlier Swagata Acharya, Sourav Mandal, and Rohini Basak presented a model for solving arithmetic problems using NLP [36]. In the methodology, they use dependency parsing to replace conjunctions used for joining two quantities. The example given is "Carolyn starts with 47 marbles and 6 oranges." [36], where the sentence is then split into two, both starting with "Carolyn starts with" and then presenting the item and quantity. The paper also used NER to find the entity that has a quantity, e.g. the marbles in the example above.

Lai et al. present an NLP approach to study floods and storms [34]. In the paper, they use dependency parsing as a method for connecting flood reason entities with locations [34]. The method uses a dependency parser on sentences in a document or article, and then trees are converted to graphs. The graphs are then connected at the head, and Dijkstra's algorithm is used to find the closest location to the flood reason entity to connect them. Spacy's dependency parser is used to create the parse trees. The method is compared to a method of nearest neighbour as well, which they found to perform worse. The dependency parsing method linked fewer total location links in the test, but had a higher contextual accuracy.

Muditha Tissera and Ruvan Weerasinghe present an approach for extracting knowledge from unstructured text based on grammatical structure [33]. In this work dependency parsing is used as the basis of extraction [33]. The dependency parser is used to get the grammatical structure, again the Spacy dependency parser is used. The authors then present a mapping approach that uses the dependencies to create triples (subject, predicate, object components). At the most basic level, this mapping is simply a process of looking at dependency tags and converting the data to a triple using basic rules, but the method is expanded upon. The expansion encompasses the selection of the most appropriate candidate based on the link to the root token, and further inclusion of prepositions, compound words, and more to enhance the meaning of the triple. Further details on this paper will be

discussed in the Relation of Entities subsection.

Huang et al. describe building a cybersecurity ontology as discussed in the subsection about co-reference resolution[30]. Another technique used in the paper is dependency parsing. They use Stanza for dependency parsing, which is a NLP package written in Python. Dependency parsing is used for extracting triplets from the input sentence. The method consists of collecting subject-object relationships and finding the closest verb to the object as the relation. Additionally, a method based on simple mapping rules is implemented. Further details on this paper will be discussed in the Relation of Entities subsection.

As discussed in the subsection about temporal information Husari et al. presented a method for analysis of cyber threat intelligence based on NLP[24]. As with the papers above the method used is a mapping approach based on dependency parsing[24]. Where a "Threat Action" is found by looking at the typed dependencies and their parent and child nodes.

Through the papers discussed a pattern emerges, of using dependency parsing as a tool for extracting structure from sentences, often using manual mapping approaches. The technique is also applied to different domains, with maths[36], flood and storm data[34], and cybersecurity[30] covered in these examples. This shows that the technique is general and can be applied to various domains. Similar approaches using manual mapping may be effective in parsing institutional statements using the IG as well.

Constituency Parsing

Constituency parsing is a technique where a sentence is divided into constituents represented in a parse tree. These constituents are phrases that belong to categories in the grammar, and the grammar is context-free[49]. For example, using constituency parsing on the sentence "The quick brown fox jumped over the lazy dog.", the sentence gets divided into a noun phrase for "the quick brown fox" and "jumped over the lazy dog" is a verb phrase. This is simplified, as there are other constituents below and above the verb phrase. Additionally, it is common to also show POS-tags for the words together with the constituents when presenting the parse tree.

Jose L. Martinez-Rodriguez, Ivan Lopez-Arevalo, and Ana B. Rios-Alvarado present an approach for constructing knowledge graphs using an OpenIE-based approach[19]. In the paper, they use constituency parsing as a step in the process of generating the knowledge graphs. Specifically constituency parsing is used to "group words into sub-phrases that function as a single unit"[19] in the form of noun-phrase units. These units are then later used to associate entities with information.

Özge Gürbüz and Onur Demirörs present an ontology bases methodology for creating business process models based on organizational guidelines[18]. They use constituency parsing together with a mapping function to categorize information in sentences into the categories: "what", "where", "why", "when", "who", and "how"[18]. This mapping function looks at a node and the parent node and maps the span according to a simple template. For example "when" maps to the constituency tags "PP" connected to "Root". These categories are then further matched into process entities.

These papers show that constituency parsing can be used to extract information from sentences, by grouping words in a structured way that implies meaning. The second paper is especially relevant in the categories it collects using constituency parsing. These categories can be related to IG with "when" and "where" for example possibly relating to conditions or constraints. Showing a mapping approach from constituents to IG as a possibility.

3.5.3 Relation of Entities

This subsection focuses on relating entities to each other through various NLP techniques. With the main goal of finding relations between entities.

Relation Extraction

Kartik Detroja, C.K. Bhensdadia, and Brijesh S. Bhatt present "A survey on Relation Extraction"[40]. In the paper relation extraction is introduced as a sub-task of information extraction, RE concepts are introduced before methodologies and deep learning methods are presented, followed by domain-specific implementations, additionally datasets for RE system evaluation are presented[40]. Information extraction is used to extract information from unstructured text, RE specifically extracts information in the form of semantic relations between entities. The example of "(Sardar Patel, birth_place, Nadiad)"[40] is given, showing that the person was born in the location. This is a tuple, with three elements, which can also be referred to as a triple. The process of RE can be performed separately from entity recognition, but there are also approaches that combine the two. In the paper, different methods for performing RE are presented and categorized into two main groups, traditional methods and deep learning methods. The traditional methods include rule-based methods, which rely on pattern-matching rules, these methods are domain-specific and rely on expertise for building the rules. Further, there are two main categorizations of supervised and unsupervised methods. With the distinction of unsupervised methods not requiring labelled data, while supervised methods require labeled training data. Semi-supervised methods need less labelled training data, and distant supervision methods use text together with knowledge bases to extract relations without manually labelled data. Deep learning methods include Convolution Neural Network (CNN) based methods which can both use supervised methods and distant supervision-based methods, methods based on RNN and LSTM, and finally encoder-decoder or transformer-based methods. Further in domain-specific relation extraction, the authors mention two main challenges. The first is that many RE models are trained on general data, which can have different word distribution than domain-specific text. The second is that models trained on general text do not perform well on domain-specific text. So, domain-specific relation extraction models might need to be trained on data from the specific domain to improve the performance.

Yan et al. present a semi-supervised information extraction framework based on using NER, a RE algorithm based on distance heuristics and an ordered neurons-LSTM network in the article "Materials information extraction via automatically generated corpus"[37]. This framework is semi-supervised and relies on automatically generating a corpus, which is used to train the semi-supervised information extraction technique[37]. Information extraction is the task of extracting structured information from unstructured text. The corpus in this paper is generated using NER to extract entities, and mapping rules with the tool Snorkel to generate the corpus. These mapping rules are explained with the superalloys as a way

to relate the superalloy with the solvus temperature of the alloy among other relations. Further, a relation extraction model was trained, with relations between the superalloys and their hardness, solvus temperature, and density. Showing F1 scores of 85.81, 89.27 and 94.02 respectively. This relation extraction model used an ordered neurons-LSTM network. Using this relation extraction model structured information could be extracted from unstructured text. A reason for using this semi-supervised approach is to reduce the amount of labelled data required for the information extraction framework, as many neural network techniques require large and accurate datasets for training.

Lai et al. also use NER and relation extraction in the paper "A Natural Language Processing Approach to Understanding Context in the Extraction and Geo-Coding of Historical Floods, Storms, and Adaptation Measures"[34]. As discussed earlier in the sections on dependency parsing and NER, this paper describes NLP techniques used for gathering information on floods and storms[34]. In this case, relation extraction is used to link city and street entities in the data, which is used as a step for geocoding and clustering the data in the paper.

Relation extraction is also used in the clinical domain as presented by Hasham Ul Haq, Veysel Kocaman, and David Talby in "Deeper Clinical Document Understanding Using Relation Extraction"[32]. This paper presents a framework for text mining of clinical documents using a combination of NER and RE[32]. The relation extraction performed in this paper is domain-specific, finding instances of medical conditions related to body parts, which can then again be related to subparts, or have other relevant relations such as measurements and their values, and units. This paper demonstrates how relation extraction can be a useful technique in specialized domains with domain-specific relations, just as it can be used with more general relations.

In the legal domain, Anjali K. Sasidharan and Rahulnath R. present an approach for RE in the paper "Structured Approach for Relation Extraction in Legal Documents"[43]. The authors describe an approach starting with NER, followed by triple (subject, predicate and object) extraction and RE. This process is used to generate knowledge graphs. These knowledge graphs help in showing connections between laws, concepts and cases, and can be helpful for analysis[43].

As presented above RE can be performed together with NER, which is what Liu et al. describe in the paper "Multi-Entity Collaborative Relation Extraction"[26]. The paper describes a method using NER in combination with RE for multi-entity RE[26]. The model uses four modules, the NER module, the RE module, a graph convolutional networks module, and a classification module. Where the graph convolutional networks are used to integrate the entities and relations to enhance the performance of the modules. Testing showed increased performance for different RE models in terms of F1 scores on two different data sets when used with the framework.

A way to represent relations in a structured form is to use triples, which can come in the form of subject-predicate-object pairs or other pairings between two entities and a relation. The papers below all use this form to represent relations.

As discussed in the section about dependency parsing Muditha Tissera and Ruvan Weerasinghe present an approach for Automatic Knowledge Extraction (AKE) from English news sources to triples[33]. In the paper, they extract information from news in the form of subject-predicate-object triples. The triple extraction uses a method based on dependency parsing where the triples are expanded upon with the inclusion of prepositions, compound words, and more to enhance the meaning of the extracted triple. An example of this expansion is given with the triple "countries" - "achieved" - "gains" expanded to "most countries" and "significant gains", this way the triple has more meaning[33]. The method was tested on two different news datasets and showed good performance. In the testing of the method on the BBC News dataset 92.6% of the extracted triples were meaningful. This technique of a rule-based triple extraction based on dependency parsing is interesting as the technique does not rely on a lot of training data and can potentially be adapted to other domains than news.

The papers discussed earlier by Huang et al.[30], Kunal Khadilkar, Dr. Siddhivinayak Kulkarni, and Dr. Sitalakshmi Venkatraman[23], Jose L. Martinez-Rodriguez, Ivan Lopez-Arevalo, and Ana B. Rios-Alvarado[19], and by Özge Gürbüz and Onur Demirörs[18], also extract relations in the form of triples. Huang et al. use triples for building a cybersecurity ontology, the triplets are built with a rule-based approach based on dependency parsing[30]. Kunal Khadilkar, Dr. Siddhivinayak Kulkarni, and Dr. Sitalakshmi Venkatraman use triples as a way to create knowledge graphs from text, with all relations in the form of triples to the same object or entity being represented as nodes connected to that object or entity in a graph[23]. The paper by Jose L. Martinez-Rodriguez, Ivan Lopez-Arevalo, and Ana B. Rios-Alvarado also uses triples to create knowledge graphs[23]. Finally, Özge Gürbüz and Onur Demirörs present a method for going from organizational guidelines to business process models, where the processed guidelines are represented in the form of triples[18].

These papers show that relation extraction is useful in many different fields and that it can be used to extract meaningful information from unstructured text. Relation extraction might have use cases in finding the relations in the statements and relating them to components of the IG 2.0. How applicable this is to parsing institutional statements depends on how well the existing models translate to institutional statements, as supervised methods can require a lot of labelled training data and might not generalize well to other domains, which can be a limiting factor. However, the results from the paper by Muditha Tissera and Ruvan Weerasinghe[33] show that a rule-based mapping approach can be effective. Further, the extraction in the form of triples might have an impact on how useful the technique is for the parsing process which will require testing to gauge.

Semantic Role Labelling

Semantic Role Labelling (SRL) is a technique used to extract information from text at a deeper level than earlier techniques, these semantic roles describe the logic of the sentence with more context than the parse tree methods discussed earlier, by linking words to semantic roles and defining relations between them. Màrquez et al. describe the purpose of labelling these semantic roles as characterising events, by describing semantic relations of a predicate and associated words[50]. Semantic roles used in SRL can include the agent which is who is doing something, the predicate which is the main verb of the sentence describing the what and the recipient which is the recipient of the action in this case. Màrquez et al. provide the example "[The girl on the swing]Agent [whispered]Pred to [the boy beside her]Recipient"[50].

Kumar Manas and Adrian Paschke present "Semantic Role Assisted Natural Language Rule Formalization for Intelligent Vehicle"[41]. This paper uses SRL, soft rule-based selection restrictions, and large language models (LLMs) to extract predicates, arguments, and temporal aspects from natural language rules and instruction"[41]. This data is then processed using a LLMs to generate machine-readable rules for intelligent vehicles such as autonomous vehicles and drones. This way these rules can be interpreted by the intelligent vehicles.

James J. Nolan, Mark Stevens, and Peter David present an approach for automatically extracting technical information from open source unstructured data[22]. The approach used is a combination of NLP techniques including statistical topic modelling, entity extraction and disambiguation, and semantic role labelling[22]. Specifically, this paper describes the development of the tool Tech-Trakr which collects web data from specific sites, news, and documents and extracts information. This approach uses statistical topic modelling to classify and cluster documents based on discovered topics. Further SRL is used to find relations between entities extracted using entity extraction and disambiguation. Entity disambiguation is the process of resolving co-references to entities. With this extracted information the tool can create technology profiles showing attributes and relations and the tool can be used to navigate through the information sources related to the specific attributes.

Another paper that uses semantic role labeling is "Shoo the Spectre of Ignorance with QA2SPR" by Simone Scannapieco and Claudio Tomazzoli[16]. In this paper, the authors present the creation of an architecture for question answering where a novel technique of Prioritised Semantic Role Labelling (PSRL) is used to improve the identification of questions and ranking of answers[16]. The PSRL technique is built for Italian and is based on five phases consisting of tokenization, schank verb analysis, feature extraction, Italian logical analysis rules retrieval, and complement checking, matching, and priority. Schank verb analysis is a method used for representing conceptual dependencies in text. Where the two basic forms are dependencies between an actor an action and an object, and an object and a state. Through Schank verb analysis the PSRL method finds conceptual depend-

encies, in cases where there are several complementing dependencies to the same actor or object a ranking method is used. Where certain types of complements are ranked higher than others, mainly object or possession relations are ranked over location or time relations. By using this ranking approach the authors expand upon SRL to help in automatic question answering by ranking the answer candidates.

These papers show that SRL can be a useful technique for various domains to extract more meaningful information from text. These relations could potentially be useful for annotating institutional statements as well. With the basic pattern of Schank verb analysis in "Shoo the Spectre of Ignorance with QA2SPR" there are parallels between the form of actor, action, object, and the annotation of institutional statements with the "Attribute", "Aim" and "Direct Object". The same goes for the SRL pattern of actor-predicate-recipient, which has the same parallels.

3.6 Discussion

The field on NLP is vast and diverse. With a substantial amount of different techniques, frameworks, and implementations. For example NER can be used as a general technique to find entities in unstructured text or be adapted to the clinical domain or legal domain to find entities specific to those domains. The problem with domain-specific text is that it can contain domain-specific terms or phrases which can be difficult for general techniques to recognize. Therefore, these domain-specific models are created, however creating such models requires knowledge within the domain, or a lot of resources for creating mapping rules, or labelled datasets for training. To reduce the amount of resources needed for creating such models there are developments in semi-supervised and unsupervised methods which require less labelled data. The alternative to such approaches is mapping approaches, which instead requires expert knowledge and testing to create rules to structure information from text. For prominent techniques NER is used in a lot of the literature as a technique that is either the main focus of the paper or as a technique in a pipeline or framework. For example, RE centered on finding relations between entities and some of the newer methods for RE perform NER and RE together[26]. Several papers performing relation extraction present the relations in the form of triples[33][30][18][19][23], which is a structured representation of the relation between two entities.

For the first research question "Which NLP techniques could be used to help automate parts of the IG annotation process?" there are several techniques that show promise. From dependency parsing to relation extraction, all the included techniques show some potential for automating distinct aspects of the annotation process. The combination of these techniques through a mapping approach from the output of these techniques to components of the IG shows promise. How effective this is in practice will depend on the mapping approach used, and how well the models and techniques work for institutional statements.

The second research question "What are the current trends in the relevant papers on unstructured text processing with NLP?" can be answered by looking at the bigger picture. The most prominent techniques in the papers are NER and RE. Which are both used in a majority of the papers. Another trend in the papers is the use of BERT in various fields with BERT adapted to be used for causality detection[35], and NER on Arabic biomedical text[27] among other use cases. Finally, there are a lot of papers that use either machine learning-based approaches[27][25], or rule-based approaches[33][30]. The two approaches have tradeoffs, with machine learning approaches requiring training data and resources for training, while rule-based approaches require expert knowledge to create the rules or mappings.

A limitation of this review is that the focus is on using NLP for the parsing of institutional statements using the IG, as there is little research in combining NLP and IG. The related work shows that there are approaches using both machine learning and manual mapping together with NLP techniques for automating the

annotation process of institutional statements, and shows that dependency parsing can be useful for this task. Other than the related work, the included studies show many different techniques and use cases for NLP, but how effective these techniques will be for institutional statements will need to be tested. Further research will also need to be performed to find tools and implementations of these techniques which can be utilized.

Another limitation is that the literature review was performed by a single reviewer, which can lead to potential bias in the document selection and review process. To mitigate this a structured approach to the review was taken and the process with the reasoning behind the exclusions and inclusions was documented. With more reviewers, this risk could however be better mitigated.

Future work will be to research available tools for the various techniques and how they can be utilized for parsing institutional statements. This will include gradually testing techniques to see if they can be used to automate some aspects of the annotation process and to evaluate the performance of the techniques on institutional statements. With the advent of new conversational LLMs in the recent years new opportunities have opened up for performing advanced tasks through LLMs. Therefore, future work will include looking at LLMs and whether they can be useful for this application.

3.7 Literature Review Conclusion

Through looking at the 26 included documents and reading through their findings and developments I have discovered several NLP techniques which show promise for automating the annotation process of institutional statements with the IG. Techniques including NER to find entities, which has potential for finding the "Attribute" component in statements, and as a step in finding the related entities for the statement to find the "Aim" and "Object" using techniques such as dependency parsing or RE. How effective this solution will be depends on the aspects which are covered by the solution, the techniques and models used and the mapping approach taken. However, the related work (section 2.3) shows that there is promise in using dependency parsing for automation, and that both machine learning-based mapping[13] and manual mapping approaches[51] have potential. Another consideration is in the choice of specific models or approaches for the techniques which will be used. As domain-specific language can reduce the effectiveness of general models, how applicable general models are to institutional statements will need testing. Trends in the papers included the use of NLP in a majority of the papers, creating relations between entities using either RE or parsing methods and there was a good split between using machine learning methods and manual mapping approaches. Future work will include finding and testing models for the different techniques discussed in this review and expanding on the research by looking at new releases in the field. Another aspect in future work will be to look at LLMs to gauge whether they can be useful for the task at hand.

Chapter 4

Natural Language Processing

Natural language processing is a widely used field of technology where machines process natural language for tasks such as information extraction, clustering of information, analysis of user sentiment, and information generation. It is a wide field with various distinct use cases and is an actively developed field with broad support for different languages. In this chapter, relevant NLP techniques and tools will be presented and discussed concerning their potential for helping in the annotation process of institutional statements. First, a toolkit will be presented, followed by presenting various relevant techniques facilitated by the toolkit to further the understanding of techniques previously mentioned in the literature review, before a section on LLMs is presented.

4.1 Utilized NLP Techniques

As part of the literature review and an initial pilot project, the Stanza Python NLP library and certain NLP techniques have shown promise for the task at hand. These techniques will be discussed in further detail here, and in the following chapter this pilot project (section 5.2) will be presented with more in-depth information on the initial implementation.

4.1.1 Stanza

Stanza is a Python NLP Package with support for many different NLP techniques. The Stanza toolkit was presented by Peng Qi et al. in the paper "Stanza: A Python Natural Language Processing Toolkit for Many Human Languages"[52]. Stanza was chosen due to its features, the wide range of supported NLP techniques, the pre-trained models available, the documentation, and the interface for using Stanford CoreNLP.¹ By using this package a lot of the techniques that were deemed as potentially useful for automating the annotation of institutional statements could be utilized from the same package. Further, the interface to CoreNLP allows for

¹CoreNLP website: <https://stanfordnlp.github.io/CoreNLP/>

further feature expansion in the future. Another reason to use a single toolkit is that this reduces the complexity of the solution and allows all processing of the input data to be performed in a single pipeline which reduces the execution time. More details on the initial testing and why Stanza was chosen are provided in the section on the pilot project (section 5.2). The relevant techniques in Stanza include Named Entity Recognition, Coreference Resolution and POS-Tagging and Dependency Parsing, these techniques and their implementations in Stanza will be presented in the subsections below.

4.1.2 Named Entity Recognition

A commonly used NLP technique is Named Entity Recognition. NER is used to detect and classify entities in a sentence. This can help in structured information extraction and classification. For example, an entity detected as a Person or an Organization might correlate with an Attribute component, or in the case of constitutive statements a Constituted Entity component. However, in testing, this was not found to be a reliable metric, although with further developments and new models NER could be useful in the future. Concerning semantic annotations information from NER can be helpful to give additional context to the components such as information on time, or dates, or whether a phrase is a law. These NER classifications are used in the annotation process primarily to handle conditions. In this case, the time or date, and law categorizations are useful to add information through semantic annotations to condition components, specifically to Execution Constraint components. For more detail on the handling of semantic annotations see the development section (section 6.2.3). The NER model used is general purpose, and in cases where the institutional statements consist of a lot of domain-specific language this may cause problems with the domain-specific terms or words. In such cases a model trained on the domain in question may perform better, however, this would require training new models and may result in overfitting to the domain and reduce the applicability to a single domain.

4.1.3 Coreference Resolution

Another important factor in the annotation of institutional statements is the problem of co-referencing, or referencing entities, specifically Attribute and Constituted Entity components with pronouns such as "it", "them", "he", etc. For the statements to be explicit and to reduce the room for misunderstanding it is vital to resolve such occurrences and clearly state what the entity referred to is. To handle such occurrences the Coreference resolution technique shows promise. As the technique is specifically designed to handle such cases, the inclusion of the technique was therefore a requirement when selecting the correct tool for the application. In the initial experiments of the pilot project, Coreference resolution was not available in Stanza, however, it was at the time available to use through the CoreNLP interface. Further, in the 1.7.0 release of Stanza, a Coreference resol-

ution model was introduced.² The implementation is based on "Word-Level Coreference Resolution" by Vladimir Dobrovolskii[53] and "Conjunction-Aware Word-level Coreference Resolution" presented by D'Oosterlinck et al.[54]. Dobrovolskii presents a method of performing coreference resolution at the word-level, instead of the span level and subsequently reconstructing the spans to enable a lower complexity of $O(N^2)$ and to consider all potential coreferences[53]. D'Oosterlinck et al. expand on this word-level coreference resolution approach by dealing with one of the limitations of this approach, which is conjunctions such as the given example "Tom and Mary are playing. He is 7 years old. They are siblings"[55]. In this example the coreference must handle a conjunction of two persons, and recognize that "they" refers to both, and that "he" refers to Tom.

The Stanza implementation finds coreferences in statements and links them together using id's. This coreference resolution model is used to simply replace pronoun references to Attribute or Constituted Entity components with their respective parent component. An example is the recognition that "they" refers to an earlier annotated Attribute component with the text "Member States". For more details on the implementation into the IG2NLP software see the relevant section in the development chapter (section 6.2.3).

4.1.4 Stanza Universal Dependencies Models

The other relevant models in Stanza are based on the Universal Dependencies v2.12. This includes their POS-Tagger with uFeats, the Dependency Parser, Lemmatization and more. These models will be briefly discussed below along with their use cases for the automated annotation.

POS-tagging and UFeats

POS-tagging classifies each word in a sentence with a specific POS-Tag and this gives additional information to base the annotation of components on. The Stanza POS-tagger tags words with both a UPOS-tag (Universal part-of-speech tag) and an XPOS-tag (a treebank-specific POS). These POS-tags can often correlate with certain components and are used as a part of the process to match words with components in the IG2NLP software. For example, Attributes and Constituted Entities often correlate with the PROP (proper noun) POS-tag of the UPOS, the Constitutive Function component is often correlated with a VERB and Modals of constitutive functions correlate with the MD (modal) POS-tag of the XPOS. Further the POS-tagging model of stanza includes morphological features with the so-called "uFeats". These universal morphological features include information such as the tense of a word, whether a word is in singular or plural form and more. The singular or plural classification is used in semantic annotation for entities (Attributes and Constituted Entity components), while the classification of word tense may be helpful in the future to handle cases where the Aim is in the past tense.

²Stanza 1.7.0 Release on GitHub: <https://github.com/stanfordnlp/stanza/releases/tag/v1.7.0>

This is specifically relevant to cases in the dataset where an Aim in the base statement is in past tense and the manual annotation has changed the tense to present tense, however, there are further considerations to make before automating this process.

4.1.5 Dependency Parsing

Dependency parsing is a technique for generating parse trees with structural information of sentences or natural language texts. This technique works by finding dependencies within a sentence and creating branches in the form of head connections between these dependants to extract structured information from text. This process follows the concept of a dependency grammar, with phrases or words connected through head relations. The end result of dependency parsing is a parse tree with each word connected through branches to the root node. The dependency parsing technique is analogous to the annotation process, where parts of statements are classified and connected through annotation. A substantial part of annotating institutional statements lies in the detection of the dependencies of a sentence, for detecting and determining the various components and their scope in a statement. Most of the matching of components to the statement text in the IG2NLP solution is based on the dependency parse tree. For example, early testing in the pilot project which will be presented in further detail in the subsequent chapter (section 5.2) showed a high correlation between the root node of a dependency parse tree and the Aim (I) component of an institutional statement. Where the root is often a verb that is contained in an Aim component in the statement.

Lemmatization

The lemmatization model is used to normalize the words in an input statement to their lemma form. For example in several cases in the dataset, there are Aim (I) components where the Aim component is in past tense such as the phrase "carried out" which is annotated as "I([carry out])" in the present tense. In this case, the lemma of the word "carried" is carry, and could be used to automate this process of changing the tense of the aim. However, changing the tense of words in a statement can change the internal context of the statement. Because of this the method of using lemmatization or uFeats to change Aim tense is not yet implemented.

This thesis bases its solution solely on the techniques mentioned in this chapter, however with the current rapid innovations and developments in the field of LLMs they show promise as a logical next step for the solution. The benefit of using these techniques over LLMs is the reproducibility of the results based on their deterministic operation, with consistent output given the same input and models used. This ensures reproducible results and consistent annotation performance. However, with these techniques, there are limitations that may potentially be solved through the introduction of further techniques or by using LLMs. The following

section will go over some of the strengths and weaknesses of LLMs, and presents potential use cases in automated annotation.

4.2 Large Language Models

In recent years a lot of progress has been made in the LLM front, with the release of OpenAI's ChatGPT, Google Bard, Meta Llama and more LLMs. These LLMs offer conversational AI models that can respond to human language and perform a variety of tasks. In the paper "A Comprehensive Overview of Large Language Models" Humza Naveed et al. present an overview of current LLMs[56]. Use cases of LLMs presented include usage in Law to code datasets, summarize content and explain legal terms, and usage of general purpose LLMs for a range of tasks such as drafting documents, answering questions and summarizing content[57]. However, Naveed et al. note that the quality of the answers is correlated with the quality of the training data, so applying a general LLM to IG 2.0 Annotation may have limitations due to the recency of the IG 2.0 specification release and the amount of available materials. Relevant challenges of LLMs mentioned include the potential for bias, overfitting and hallucinations which are answers that are incorrect due to input misinterpretation, conflicting answers or simply factually incorrect. Other challenges include cost and privacy concerns[58]. These challenges can be detrimental when dealing with LLMs, the possibility of incorrect annotations due to hallucinations, bias or overfitting can be problematic. Further, cost can be a major limitation in adopting LLMs, and privacy concerns may make LLMs unsuitable for handling confidential data. Finally, a potential concern is the reproducibility of the results, when prompting a LLM with the same question multiple times, multiple different answers can be obtained. This behaviour may be beneficial for human users in cases where a different explanation of a term is wanted, or where a hallucination is detected by a human user. Where the user can then ask the question again to obtain a new answer. However, for certain workloads and domains (e.g., information systems in business contexts, scientific computing) a consistent and reproducible output may be preferable. Because of these limitations using LLMs as the sole method for automated institutional statement annotation may be limited, however, LLMs may be helpful as a step in the annotation process. An LLM can for example be prompted with specific tasks such as to verify the scope of a component, or to help distinguish between condition types. With a basis of NLP techniques such as the ones described above a consistent annotation can be achieved with reproducible results, and an LLM can potentially serve as a tool for improving aspects based on this initial base encoding. In this case LLMs can potentially be used to indicate errors in the encoding, to distinguish between component types in cases of ambiguity or as a method of checking the type of statement. This way the annotations can be more consistent and reproducible, while still benefiting from the capabilities of LLMs.

The next chapter (chapter 5) will introduce the pilot projects leading up to the work in this thesis, which uses some of the techniques described in this chapter.

This is followed by a chapter (chapter 6) on the development of the IG2NLP solution which builds upon these prototypes and implements the rest of the NLP techniques described in this chapter, with the exception of LLMs.

Chapter 5

Background work

In this chapter, the background for the development of the IG2NLP software will be presented

IG2NLP is a solution for automated annotation of institutional statements using the IG Script notation of the IG 2.0. The solution covers basic aspects of encoding such as component detection, and component scoping, and more advanced features such as the detection of nested conditions and adding semantic annotations to components. These features are available through a program meant for offline batch processing of statements, in addition to a Representational State Transfer (REST) API for annotations on request. Further, the work includes feature updates to the IG Parser annotation tool for institutional statements and a prototype of the IG Parser with the automated annotations integrated through the REST API. This chapter describes the background work underlying the main development of the project, this includes the steps taken in pre-processing the datasets used for testing and development and initial pilot projects used as a basis for the work performed in this thesis.

As previously discussed in the section on Natural Language Processing there is great potential in using NLP for annotation purposes. Methods such as Dependency Parsing and Named Entity Recognition can give information on the structure and meaning of a sentence. Which together with a "matching function" can be used to detect components in institutional statements. This is the foundation of IG2NLP, namely the use of NLP techniques together with a custom matching function as a method for enabling semi-automated annotation of institutional statements. In the following subchapters pre-processing of data will be discussed, before the solution is presented in steps related to its development, first initial testing will be presented and discussed, followed by iterations on the software and their effects.

5.1 Pre-processing

To properly develop and evaluate an automation solution it is vital to have enough high-quality data to be able to both develop the solution and evaluate it over time. As the solution is based on dependency parsing, the structure of input statements has a direct impact on the annotated results. Therefore a breadth of input statements with varying structures is vital to enable the solution to adapt to different structures and to handle more edge cases. A breadth of data can also help mitigate potential over-fitting. Through developing and testing the tool on a wider array of statements from several backgrounds.

In addition to the pre-processing steps that will be described below the statements were also divided into two groups of statements, relating to the two types of institutional statements; constitutive and regulative statements. In the future, it may be possible to develop a proper method for automatically detecting whether a statement is regulative or constitutive, but for the initial testing and development, the statements were manually separated instead. Following these steps, the initial dataset used for research and development consisted of statements from European Union climate regulation and an ENISA Cybersecurity Act. With a total of **63** regulative statements and **22** Constitutive statements. The dataset of statements was used for the conceptual development of the automated annotation solution. As the solution does not rely on any machine learning or large language models the underlying datasets used in development are not trained on, in the same sense as they would be in a machine learning environment. The matching of components to text is done through a custom matching function based on output from various NLP techniques and should not run the same risk of overfitting as automated training solutions. This is due to the nature of the matching function, as it uses manually created rules based on structural data from the NLP pipeline used and is not trained on the statements themselves. However, the rules are based on patterns found in the dataset, so there is still some potential for overfitting, or for edge cases not contained in the dataset.

Because the solution is based on an input statement and needs to be compared to a manually annotated control, two items per statement need to be processed; the input statement, and the manual annotation of that statement. The pre-processing of these two items per statement can be divided into the following distinct steps:

1. For the input statements extra line breaks were removed, and special characters were either replaced or removed.
2. Additionally for several statements in the data sets the manually annotated statement was only based on parts of the input statement text. To allow for a one-to-one comparison between the version annotated through automation and the control these input statement texts were truncated to only include the same statement text as in the manually annotated statement text where possible. So any text preceding or following the manually annotated statement text was omitted.

3. The manually annotated statements were further modified to ensure a consistent annotation style between statements and to allow for direct semi-automated comparisons between the manual and the automated annotations. This included steps such as removing grammatical articles such as "the, a, all and any" from components such as Attributes, Constituted Entities, and Direct Objects. For example, if the manually annotated statement was "A(the Member State)" it would be modified to "the A(Member State)". Thereby ensuring consistent formatting and allowing automated comparisons.
4. Further processing included the removal of line breaks and special characters in the input statements and in some cases different handling of logical operators for a consistent style. For example, a statement with two consecutive Attribute components, "One and Two", can be annotated both as:

"A(One) and A(Two)" and "A(One [AND] Two)"

In such cases, the IG Parser handles both versions with the same implied logical "And" linkage between the Attributes, and therefore statements were modified in cases where the automated and manual annotations differed to facilitate semi-automated testing.

5. Suffixes (id's used to distinguish between components of the same type, or to link properties to specific component instances) were also removed from components to simplify the comparison process. However, in future iterations, if suffix handling is implemented then this step will be removed from the pre-processing.
6. Finally, in the dataset of the ENISA Cybersecurity Act punctuation was moved outside of components, ergo if a component included the ending full stop of a sentence, the full stop was moved outside the scope of the component.

In the following section initial experiments in automated annotation of these statements will be presented and discussed, followed by iterations on these initial experiments, new functionality introduced over time, and evaluation of the final solution.

5.2 Initial Experiments

This subsection outlines initial experiments with automated annotation of institutional statements, these experiments were performed as part of the Advanced Project Work (APW) course in the preceding autumn semester and the following text is partially based on the report text and work for that course.

To test whether dependency parsing combined with a manual matching function was a viable method for automating the annotation of institutional statements a pilot project was created. This pilot project consisted of two phases. Firstly tool selection and evaluation. Where a set of criteria was created to select tooling for the project. These criteria included:

- Free to use
- Recently updated or in active development
- Includes models for NER or ER
- Includes models for Dependency parsing
- Includes models for Coreference resolution
- Has pre-trained models available

The initial search was based on a broad search on open search engines, in addition to the inclusion of tools from research papers from the initial literature search and review. The alternatives were then narrowed down by excluding tools based on the requirements and finally, initial test programs were developed for Stanza and Spacy to decide between the two. At the time of the experiments Spacy had a Coreference resolution model available in the *spacy-experimental* branch, however, this model relied on an earlier version of Spacy and was not updated at the time of testing to the newer versions, or included in the main Spacy tool. While Stanza also had a Coreference resolution model available through Stanford CoreNLP. Which could be used with Stanza through an interface in Stanza that calls CoreNLP. The initial testing to decide between the two solutions used basic dependencies with a direct matching between dependency parsing dependencies and IG components. These components were limited to Attributes, Aims, Deontics, and Direct Objects. In the end, the choice was Stanza due to its integration with Stanford CoreNLP, which opens the possibility of using CoreNLP functionality in the future if beneficial. Further, it had the benefit of supporting Coreference resolution in the newest version of the program through the CoreNLP interface. Although the main deciding factor was the performance of Stanza in testing compared to Spacy, where the Stanza solution based on basic matching rules had a lower percentage of False Positive component detection. Following these initial tests the Stanza matching function was iterated upon with initial experimentation of condition detection based on the Adverbial Clause Modifier (advcl) dependency. This initial experimentation showed potential in dependency parsing as a base method for annotation, however, the performance was still lackluster with a very high false positive rate for Direct Object components and a true positive rate of around 30% for Attribute and Aim components, Deontic components had

a high true positive rate however of **94.44%** in the testing. This testing was performed on a smaller dataset than the testing that will be presented later, limited to only regulative statements and a total of **18** statements.

Limitations of this initial experiment were in the simplicity of the implementation, and the underlying tools and models. The initial matching function was only based on direct dependency on component matching, which has the limitation of improper component scoping. See the image below for an example of this direct matching (figure 5.1):

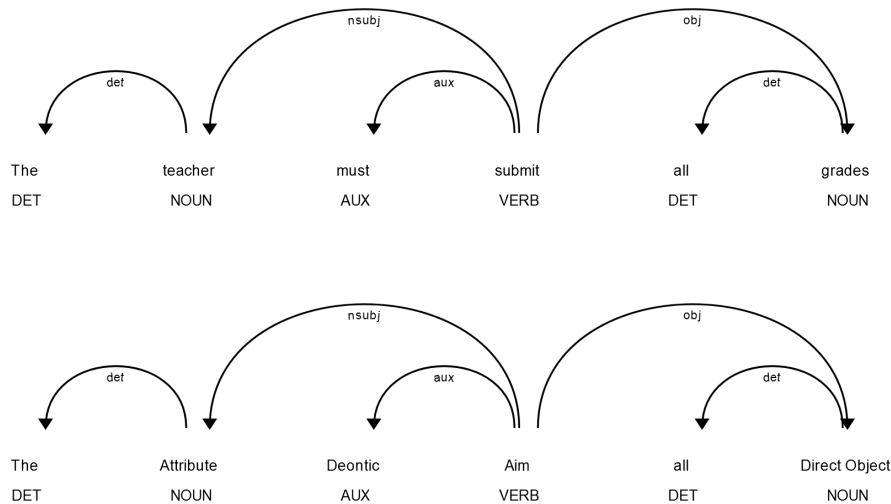


Figure 5.1: Dependency parse trees, first with all relevant input data, second with words replaced by their component matches.

An example of this scoping limitation is that Direct Objects can include several words such as "such recommendations", or nested statements consisting of several different components. There were initial experiments with broadening the scope of components and their coverage, however this was basic and inaccurate. These scope-broadening techniques were divided into two. The first was logical operator handling where if the component end was followed by a logical operator then both the operator and the subsequent word were included in the component. So, for example with the text "doctors and nurses", if "doctors" was detected as an Attribute component, then "and nurses" would also be included in the component. However, in some cases, this led to incorrect encapsulation in some cases where the logical operand was not connected directly to the preceding word and instead referred to the larger sentence structure. To mitigate this the matching function was updated as part of the thesis project to traverse the parse tree properly to check the head of each logical operator. The other scoping extension was for Activation Conditions. The Activation Condition rule was a very specific implementation, relying on an *advcl* dependency, the component being located at

5.3 The IG Parser User Interface Pilot Project

As a part of the Advanced Topics in Software and Systems Engineering course, a prototype for an updated user interface for the IG Parser institutional statement annotation and visualisation tool was developed.¹² This subsection will present the background for this work, the development goals and requirements, and finally the resulting artefact from this work. This section is based on the report for the course and has been modified to be more concise and to use it as an introduction to further work performed as part of the thesis artefacts.

The basis of the work on user interface improvements to the IG Parser was a heuristics evaluation based on the heuristics proposed by Mohamed Benaida in the paper "Developing and extending usability heuristics evaluation for user interface design via AHP"[59]. In this heuristics evaluation, the IG Parser along with two other annotation tools was evaluated, and a less thorough overview of an additional three annotation tools was performed. Through this analysis, a set of limitations to the IG Parser was discovered, and this led to requirements and ideas for the new user interface. The primary points of improvement were annotation speed, ease of use, and error prevention. Firstly for annotation speed and ease of use, an updated interface was developed with buttons for each different component type and alternate annotation options. The old method of annotation relied entirely on user text input, while the new interface has two primary input methods added, which are selection-based input with buttons for the various components, and a second method that allows for keyboard input using keybindings for each component type. In addition to this, the whole page was updated to facilitate keyboard navigation, modals were developed to give annotation information in the editor itself, instead of relying on external resources or resources on a different page of the website, and confirmation prompts were added to the example selection to prevent users from accidentally overwriting their work. Finally, the addition of colour coding to visualize the component annotations was implemented. This was intended to make distinguishing annotations from the text more clear and fast, and to help in distinguishing between different component type annotations. A comparison between the two user text editing windows can be seen below (figure 5.3):

¹GitHub repository for the prototype: <https://github.com/Kjaerandsen/IG-Parser-prot-vis>

²As of the time of writing an updated version of the prototype is integrated into the main IG Parser page located at: <https://ig-parser.newinstitutionalgrammar.org/>

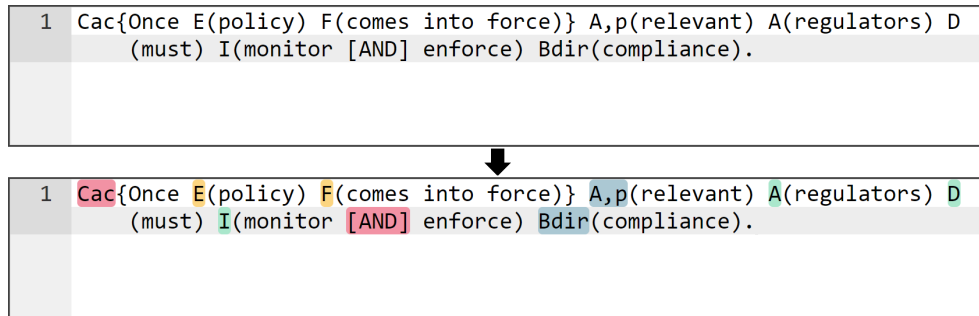


Figure 5.3: Text editor window before and after visualization work.

Here you can see the colour coding divided into categories with separate colours per category. Categories are divided into components shared across statement types (both regulative and constitutive) are pink, constitutive statement exclusive components are yellow, regulative statement exclusive components are green, and finally blue for Properties and Objects.

Following selective user testing, the developed prototype of the editor UI served as a basis for the refinement of the IG Parser, building the basis for the integration of the IG2NLP software for automated statement annotation, a feature that will be presented and discussed in a later section (Section 6.3.4).

Chapter 6

IG2NLP

In this chapter, the development of the IG2NLP software will be introduced followed by sections with detailed discussions on the various developments.¹ These developments are based on the starting point with the pilot projects outlined above, followed by the requirements and system architecture outlined in this section, further the subsequent section goes over the matching function, and iterations to cover the different limitations of the pilot project, leading into details on the software artefacts and tooling.

6.1 Development

The primary development process consists of a primary goal connected to a development branch on the git instance. This goal is worked towards in its specific branch, and all developments are tested on the datasets to evaluate the artefacts. Using the manual annotations as a reference and the previous iteration as a baseline makes it quick to recognize whether the new developments lead to improvements or regressions. This testing was assisted by a multi-level implementation of logging which facilitates error handling, and information gathering at several levels to help in testing and debugging. For smaller self-contained functions a test-driven approach was utilized with unit tests to validate all base cases of the functions. After the goal is reached and the artefact has been tested on the test data set the changes can be merged into the main branch again, leading to a new goal and development cycle. The development of the matching function consisted of a three-step process. The first step was pattern recognition, where the dataset was looked at, together with information on the IG and annotation, and output from the NLP pipeline was looked at. This included running dependency parsing among other NLP techniques on the testing data, visualizing the output in the form of dependency parse trees and text files, and finally looking at the output to discover patterns in structure, dependencies, pos-tags or other factors linking the output of the various NLP techniques to IG components. This led to the second

¹The IG2NLP software is available on GitHub: <https://github.com/Kjaerandsen/IG2NLP>

step of creating rules based on the findings. These rules consisted of rules at various levels, at the most basic level a simple matching between either a POS-tag or a dependency and a component. More advanced examples include dependencies reliant on the presence of other components connected to the component, for example, a Property of a component, and finally components containing more than one word. These components can include for example an entire phrase, several logical operators, or even several internal nested components. The rules for these components are therefore more complex and can include positive and negative lookahead, recursive matching function execution and logical operator handling. Finally, after new rules are formulated, they are tested on the dataset. Using the manual annotations as a reference and the previous iteration as a baseline makes it quick to recognize whether the new rules lead to improvements or regressions. If the new rules cause performance regressions, then they may be reworked in the second step, the pattern may be looked at again to consider alternative approaches, or the rule may be scrapped entirely.

6.1.1 Requirements

The goal of the software is to assist in the annotation of institutional statements with the IG Script notation. To enable this the following requirements were defined:

- Handling of components for regulative statements
- Handling of components for constitutive statements
- Consistent output
- Support for batch processing
- REST API
- Logging or another method for signalling errors, warnings, and suggestions
- Documentation with instructions for how to use the software and input structures used

6.1.2 System Architecture

The IG2NLP software consists of two primary applications in the form of Python programs:

1. IG2NLP:
The primary software artefact takes an input of statements in a structured JavaScript Object Notation (JSON) format and can batch annotate them as either constitutive statements or regulative statements.
2. API:
A REST API was made for the integration of the automated annotation tool to other software such as the IG Parser.

In addition to these two primary software artefacts, a broad set of tooling was developed with additional functionality primarily focused on the development of the software, and evaluation of its performance.

This suite of tools includes:

- `Cache.py` - a program for caching the output of the NLP pipeline, which allows for rapid iteration on matching rules, or testing of different parameters without necessitating the extra work of processing all statements again.
- `ParseTrees.py` - a program for generating dependency parse trees in the form of HTML documents for a dataset.
- `dependencyParsing.py` - a program for generating a single dependency parse tree in the form of a served webpage.
- A test suite that will be described in the subsequent testing chapter.

The development and functionality of this suite of tools will be presented and discussed in the subsequent sections, starting with the matching function.

6.2 Matching Function Feature Development

This section will go over and present features developed as part of the thesis work. The focus in this section lies on the matching function itself, highlighting individual aspects by reiterating the associated challenge, and how it is addressed. The matching function as described in the section on the pilot project in the previous chapter (Section 5.2) is the main function that handles the matching of components in the IG to output from the NLP pipeline. This matching function uses a combination of POS-tags, dependency parse trees, NER data and more to match components of the IG to parts of statement text. This section starts from the baseline described in the prototype section, followed by describing developments to the scoping of components, continuing with additional component coverage to handle more component types, and finally introduces the implementation of additional NLP techniques and their impact, usefulness, and limitations. Other parts of the software such as extra visualization programs, the test suite, and the API will be covered in a later section (Section 6.3).

6.2.1 Component Scoping

Compound words

In the English language compound words are a common occurrence. In several natural language processing techniques words are compounded to assign logic to the correct scope of words. In such cases multiple words are handled as a single token, this is called "Multi-Word Tokens". For the IG2NLP application cases where two or more words should be treated as one were common, for example, the Attribute "A(Member States)", it does not make sense to split the words "Member" and "States". To reduce the complexity of the matching function and the complexity of the scoping of components in such cases a pre-processor was created. The pre-processor handles compound words and combines them into single words for future matching as can be seen in the figure below (figure 6.1):

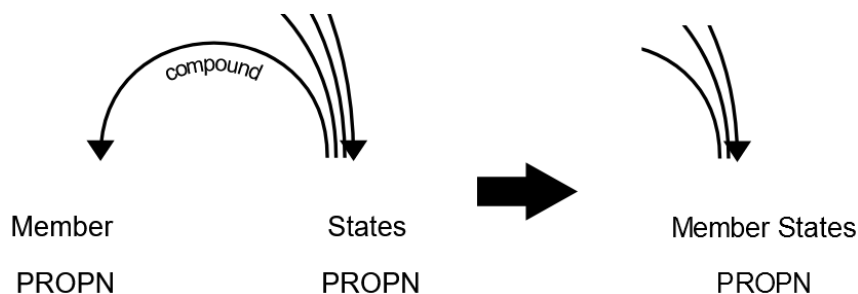


Figure 6.1: Basic compound word combination example.

The same basic combination is performed for some more complex cases as

well, such as in compounds longer than two words, or with a punct (punctuation) dependency between the two words (words bound by a dash), for example, "long-term" is treated as a single word.

Logical operator handling

In institutional statement annotation, logical operators consist of three possible logical operators: the logical and ([AND]), logical "and/or" ([OR]; inclusive or) and the "either or" ([XOR]; exclusive or). At the most basic level, a simple word-matching function can be used to detect such operators and annotate them as logical operators. This function would work by going through each word one by one, and checking whether the word converted to lowercase is contained in the list of supported logical operators. However, as previously discussed in the section about the pilot project, this does not work in all sentence structures and may cause invalid inclusion of words into components. Another alternative with the same basic structure is to use dependency parsing and look for the Coordination (cc) dependency, or the "CCONJ" POS-tag from POS-tagging. This can further be extended to find the proper scope of the logical inclusion using the Conjunct (conj) dependency. An example of a simple dependency parse tree with a logical operator can be seen in the figure below (figure 6.2):

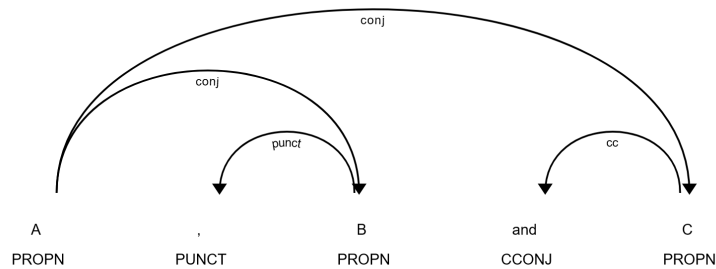


Figure 6.2: Basic logical operator parse tree example.

Further components with a larger scope may contain internal logical operators as part of their contents. In these cases, a detection function handles detecting internal logical operators and potentially handles internal conflicts. For example in the third IG Parser example statement there is an Aim (I) component with the base text "review, and reward or sanction". In this case there is both a logical "AND" and a logical "OR". This would then result in the second logical operator being scoped or encapsulated with parentheses to allow for parsing. This results in the component:

I(review [AND] (reward [OR] sanction))

However, the manually annotated statement annotates the "OR" as an "either or" ([XOR]; exclusive or) instead of an "and/or" ([OR]; inclusive or) as in the

automated solution. This is a limitation of the automated solution, where there is currently no mechanism for detecting whether an "OR" should be an "and/or" or an "either or". Another limitation is the internal scoping, the current implementation handles cases where there are both "OR" and "AND" logical operators in a component, and encapsulates the operator with its surrounding words to make the statement parseable, however, this scoping may need manual refinement to accurately convey the underlying meaning. To indicate this to the user a comment is added to the API output suggesting manual review, and a warning is logged and printed to the terminal of IG2NLP with the message "WARNING: Found both "and" and "or" logical operators in component, please review manually to solve potential encapsulation issues."

Activation condition detection and scoping

In the previous section on the pilot project the basic scoping and detection mechanism for Activation Conditions was presented. Where the detection was based on positioning in the sentence at the start of the sentence, and the presence of a dividing comma to signal the end of the component scope. To further develop this detection the mechanism was updated to also support Activation Conditions in other parts of the input statement. The main detection mechanism is still based on the advcl dependency. However, instead of relying on the start of the sentence and a comma to end the component scope, the scoping is now handled through two directional inclusion functions. A positive lookahead, and a positive lookbehind for words connected to the advcl dependency directly or indirectly. This way the dependency parse tree is properly utilized to scope the condition and the scoping is independent of the position of the advcl dependency.

In the next subsection on component coverage extension, the mechanism for differentiating between the Activation Condition and Execution Constraint components, and the detection and handling of nesting within components such as those two will be presented. Further component coverage implementation will also be presented, including the implementation of the components used in constitutive statements and component properties.

6.2.2 Component Coverage Extension

The initial pilot project only covered the following components:

- Attribute
- Deontic
- Aim
- Direct Object
- Activation Condition

To expand the functionality of the matching function, and to increase the usefulness of the software this list of supported components needed expansion to cover all possible components, both for regulative and constitutive statements. This meant adding handlers for the following components to the matching function:

- Indirect Object
- Execution Constraint
- Constituted Entity
- Modal
- Constituting Properties
- Constitutive Function
- Or Else

and properties for all the components that support properties, such as Direct Object Properties. In this section, the handling of these components will be introduced, along with different limitations and potential solutions starting with the detection of Execution Constraints and distinguishing them and Activation Condition components.

Execution Constraints and differentiation of conditions

By reviewing the datasets and comparing the structure of the sentences, the dependencies, and other output data from running the NLP pipeline on the data and comparing this with the manually annotated statements several patterns emerged. Two of these patterns were specific to Execution Constraints, the first being an overlap between the mechanism for detection of Activation Condition components and Execution Constraint components. As in some cases the advcl dependency corresponded with an Execution Constraint instead of an Activation Condition. The other pattern was a link between the Execution Constraint component and the Oblique Nominal (obl) dependency. This pattern on the other hand had less overlap with Activation Conditions. So the matching function based on the advcl dependency needed a method of distinguishing between the two condition types. For this a simple method of looking at the internal dependencies within the included scope of the component was used, if the words of the component contained two or more internal obl dependencies then the component is annotated as an Execution Constraint instead of an Activation condition. Using this method

the two condition types can share a single handler, and be differentiated using a simple function. For the obl dependency, the primary detection is for Execution Constraints, however, there was some overlap with other components such as objects, which is mitigated using position information and adjacency to other components. Although, this does not eliminate the issue entirely and further work is necessary to improve the annotation accuracy.

Nested component handling

The next challenge for conditions was in handling cases where the conditions contained nested structures with other components within themselves. Take the start of statement 7.3.3 from the dataset as an example:

"if the Member State concerned decides not to address the recommendations or a substantial part thereof [...]".

In this case the matching function detects the phrase at the start of the statement as an Activation Condition component:

"Cac(if the Member State concerned decides not to address the recommendations or substantial part thereof)".

The matching function then looks for nested components within the Activation Condition, if the requirements are met then the internal components are also annotated as can be seen below:

"Cac{if the A(Member State) A,p(concerned) I(decides not to address) the Bdir(recommendations [OR] substantial part thereof)}".

To detect and handle such cases of nested components a method of recursion was utilized. Where for components that have support for nesting within the matching function (currently the matcher has implemented this for Activation Conditions, Execution Constraints, and Or Else components), the matching function is run again on the contents of the component. The results of the next matching function are then checked, with a basic rule for regulative statements requiring at least an Aim and an Attribute component to be present, and for constitutive statements a Constituted Function and a Constituted Entity are required. If these components are present then the brackets type is changed to indicate a nested component, and the results of the other matching function replace the text of the component. On the other hand, if the matching function does not detect the necessary components, then the text is simply kept as is, and the component is annotated as not nested. See the figure below for a basic workflow visualization of the parser (figure 6.3). Note that the entire figure exists within the matching function as an individual match, and the end block represents the matching function going to the next word, where this block can again be called if the matching function detects the subsequent words as a potential nested component.

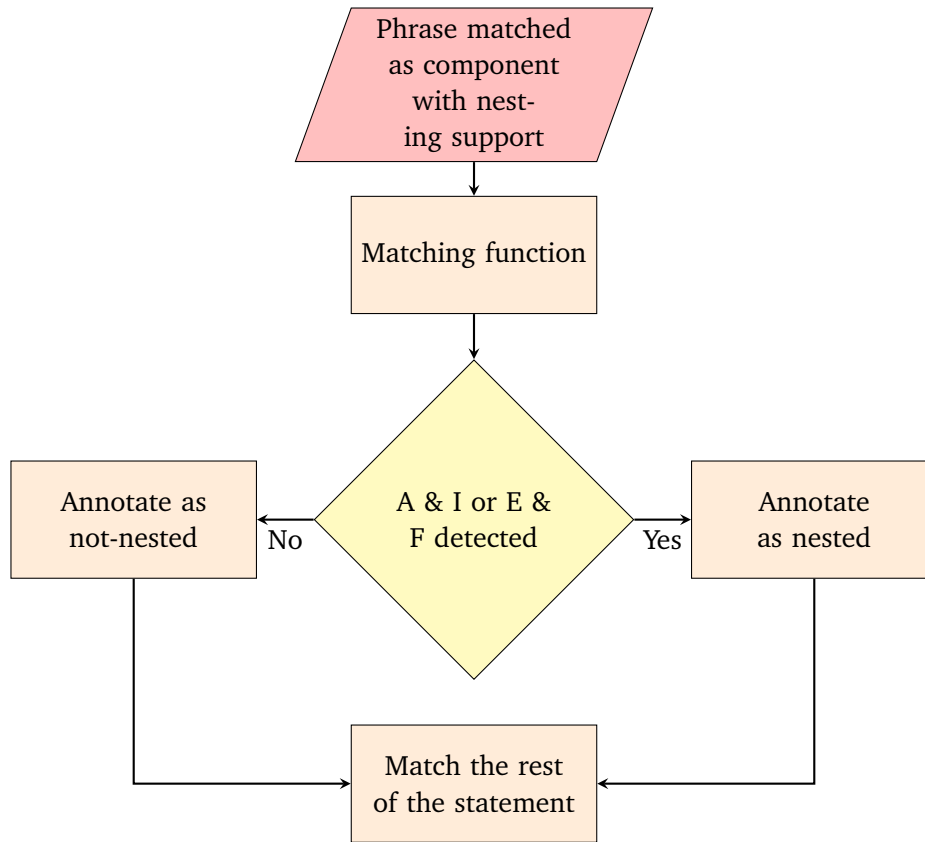


Figure 6.3: Nested component handling

In the model above the functionality shown exists within the matching function, with the starting point being a case of the matching function where it calls itself on the contents of a component. The end case is the nested matching function exiting and the original matching function continuing to match the rest of the statement if there are additional words.

The limitation of this approach is shared with one of the primary limitations of using dependency parsing as the main information source for annotation; the solution is unable to handle inference. As in many statements, it is not explicitly stated that the Attribute is the actor of the Activation Condition for example, and it is instead inferred from the context, e.g., in the statement 7.3.1 from the development dataset:

"Where recommendations are issued in accordance with paragraph 2 [...]"

The manually annotated statement infers the Attribute in the Activation Condition as "the Commission":

"Cac{Where A([Commission]) I([issues]) Bdir(recommendations) are issued Cex(in accordance with paragraph 2)} [...]"

Detection and handling of such cases is an unsolved challenge for now. However, there are potential avenues for solving such issues that will be described at a later stage in this thesis.

Or Else component handling

The third component with nesting support in the solution is the Or Else. For the detection of the Or Else component, a simple dictionary approach was used. Where if an "or" word is followed by "else" or contains the word "otherwise", then the rest of the sentence is annotated as the contents of an Or Else component. Unique to the Or Else is that the component is always treated as a nested component, and it shares the basic method of reusing the matching function on its contents with the conditions.

Properties

Another factor to cover in the annotation of institutional statements is the properties of components. The IG Parser supports the following property components:²

- Attributes Property
- Direct Object Property
- Indirect Object Property
- Constituted Entity Property
- Constituting Properties Properties

To detect such components a combination of positioning, dependencies, and head connections of words is used. For example, a subset of dependencies connected to components such as the Attribute can be used to detect a property of the component, or included in the component depending on context. For example an Adverbial Modifier (advmod) or an Adjectival Modifier (amod) dependency. A common factor in attribute detection is this dependency head connection to another component with support for properties. Other dependencies are also used in certain contexts to detect attributes, for example, in some cases, an obl dependency is used to indicate a property of a Direct Object, Indirect Object, or Constituting Property component instead of an Execution Constraint.

Limitations to the method used in the IG2NLP software include difficulty in differentiating between instances where a property should be included in the main component, or kept as a separate property component, and handling nesting in properties. All properties in the IG 2.0 support nesting, however, this was not present in the test data used for development and was therefore not covered in the software.

The Constituting Property component mentioned above as supporting nesting is a component used in constitutive statements. In the following section, the

²List shows a subset of the component list from the IG Parser Syntax Guide page <https://ig-parser.newinstitutionalgrammar.org/help/>

introduction of constitutive statement handling will be presented, along with implementation details and alterations required to adapt the matching function to also be able to handle constitutive statements. Before that, the introduction of new NLP techniques to the matching function will be presented, including their use cases and limitations.

6.2.3 New NLP Technique Implementations

Coreference resolution

As discussed previously the 1.7.0 release of Stanza introduced Coreference resolution, in this section the implementation of this Coreference resolution model into the IG2NLP software will be presented and discussed.

In testing the Stanza Coreference resolution model had some issues in detecting spans across sentences, as in an entity that does not end in the sentence it starts in, further the scoping of the entity Coreferences was inaccurate compared to the manual annotation of the same entity. The first problem resulted in the pipeline stopping, and is an error in the implementation, to mitigate this the IG2NLP software has configurable batch sizing, which allows limiting the pipeline to running a single sentence or statement at a time, which prevents spans crossing sentences. Another problem is that the solution treats the entire batch of documents as a shared document, and detects entities across statements. This cross-statement coreferencing may not be beneficial in all cases, for example when batch-processing datasets from different domains. To prevent this the batch size can be adjusted to one, or the datasets can be annotated separately. Finally, the scoping of entities was not always accurate to the scoping of the same entities in the manually annotated statements, the basic implementation works by finding the longest reference of an entity. To increase the accuracy of the entity scope the matching function updates the scope itself by using its own longest Coreference detection. This works by looking at words with the same Coreference "chain" or id and taking the longest component contents from that chain as the full entity name to use when replacing for example pronouns with their respective entities. Say an entity is "John Doe", and it is referred to in a statement as "John", "he", "John D.", and "John Doe", then the longest reference is "John Doe", which would be used in eventual replacements and semantic annotations if enabled.

The use-case of coreference resolution in the matching function lies in better detection of Attributes and Constituted Entities, and in replacing pronouns such as "him/her", "it" and "they" with the specific entity they refer to.

In the same spirit, this can be used to add a shared "Entity" semantic annotation to each instance of an entity in a statement. E.g. if an Attribute component is "A(Member States)" and the Attribute is later referred to as "they", then "they" can be supplemented with the Attribute component e.g.:

"... *they* ..."

is supplemented with the Attribute component contents resulting in:

"... *A([Member States]) they* ..."

Additionally, if semantic annotations are enabled both instances can be annotated with the semantic annotation "[Entity=Member States]". This makes it clear that both Attribute components refer to the same entity.

In the following subsection, more information on semantic annotations and their implementation in the software will be presented.

Semantic Annotation

Semantic annotations of the IG Extended and IG Logico are used to encapsulate extra information into components. This extra information can contain information such as the specific Entity name as described above, information about the context of a constraint, and if an entity is singular or plural.

The IG2NLP currently has three different semantic annotations supported. The method described above of annotating the Entity based on Coreference resolution. In addition to this, NER is used to look for date or law entities. These two entities are then matched to the semantic annotations "[**ctx=tmp**]" (temporal context) and "[**act=law**]". This can help add additional context. The temporal context is however a very general ctx annotation, and it may be possible and beneficial to look into developing this further to be more specific such as differentiating between a point in time (**tim**), a time frame (**tfr**) and frequency (**fr**) as described in the context taxonomy of the IG 2.0 Codebook[8]. The final semantic annotation currently supported is the number annotation, signalling whether an entity is singular, or plural with the semantic annotation "[**number=sing**]" or "**number=plur**" based on universal morphological features (UFeats) from the Stanza POS-tagger.

These semantic annotations are all parameterized using environment variables, a .env file for the IG2NLP program or as variables in the JSON request body of the API. This makes it possible to enable or disable various semantic annotations according to user needs, For example in some use cases, the "number" annotation may not be valuable, while for others such as simulation or modeling it may be useful.

In the future, it may be worth researching methods to extend the semantic annotation support of the matching function to cover more different semantic annotations. For example annotating whether an entity is animate, or inanimate, could also help distinguish between regulative and constitutive statements. Such changes would, however, require testing to gauge their efficacy and usefulness.

Everything leading up to this point has been focused on describing the handling of regulative statements, there are, however, also constitutive statements that need separate handling. In the following subsection, the handling of constitutive statements will be introduced, starting with how the matching function was adapted to handle constitutive components. Followed by a discussion of similarities, differences, limitations, and future improvement possibilities.

6.2.4 Constitutive Statement Handling

Constitutive statements are a vital part of the IG to define elements of institutional statements or to give background information for statements. For example, defining what specifically is meant when referring to a student. For example defining that students in the context means students admitted to a certain educational institution at the current time, or students in a certain field, within an age range, etc.

There are two primary challenges to automating the annotation of constitutive statements. First, distinguishing constitutive and regulative statements requires context information, or a set of criteria and data to distinguish between the two types. Second, constitutive statements have their own set of components that are separate from regulative statements, in addition to some overlapping components.

The overlap between regulative and constitutive functions is both in the shared components (Activation Conditions, Execution Constraints, and the Or Else) and components with similar structure or purpose. The Attribute and Constituted Entity for example serve the same purpose in their respective forms of statements, the same for Modals and Deontics, however, there are subtle differences. A Modal does not signify that there is a consequence for not following the rule or norm, while a consequence is often defined or implied in regulative statements connected to the Deontic.

As described above there is a lot of overlap between the two statement types, so the first task when adding constitutive statement handling was to look into ways of adapting the old matching function to handle constitutive components. For the initial implementation, a simple one-to-one conversion was used. Where the functions for the shared functions were utilized, Deontics were instead matched as Modals, Aim components as Constitutive Functions, and Objects were primarily converted to Constituting Properties. However, this initial implementation was not accurate in all cases, and alterations needed to be made to better encapsulate constitutive components. The *obl* dependency for example was used to detect Constituting Properties instead of Execution Constraints in several cases. Several other minor alterations were also made to increase the accuracy of the constitutive statement handling.

After adaption of the matching function, the next challenge lies in detecting whether a statement or a part of a statement is constitutive. This remains an unsolved problem, however, there are several possible ways of solving the problem.

In the meantime the API implementation returns both a constitutive and a regulative match for each input statement, and the IG2NLP software has a parameter for deciding whether to annotate a dataset as constitutive or as regulative. These are stop-gap solutions, however, and a detection mechanism would help improve the usefulness and usability of the software solution. Some potential methods for detecting whether a statement is constitutive include the following:

- Looking at the detected entity (Attribute or Constituted Entity) and checking if the entity is an animate (regulative) or an inanimate object (constitutive).
- A basic dictionary-based approach for Modals and Deontics, as Modals are generally permissive in nature, which can be reflected in the wording used.
- Identifying the nature of the Constitutive Function or Aim may also give insight into whether the statement or section is constitutive.
- With the advent of new highly capable LLMs it may be possible to train a LLM for the task of statement type detection. Through giving contextual information and basic steps for the task. This may result in problems with reproducibility and accuracy, however.
- With a large training dataset a model can be developed to distinguish between regulative and constitutive statements, however overfitting to the dataset, and generalizing to statements from different domains may be a problem.

As presented above there are many potential methods of solving this problem. In the meantime manually separating the statements is required. In the subsequent section, more details on the complete software will be presented, including the main application, the REST API, the test suite, and other tooling.

6.3 Software and Tooling

Throughout the Thesis work, several programs have been developed. This includes the automated annotation software in the form of a Command Line Interface (CLI) application, a REST API, an updated interface to the IG Parser and integration of IG2NLP into the IG Parser, tooling to help in the development of the matching function, and a test suite for evaluating the annotations. In this section, these different programs will be presented, along with their use cases, implementation details, and other relevant details.

6.3.1 Installation and Requirements

The main IG2NLP software stack consists of Python 3 programs. They have all been tested on Python 3.11 and have requirements specified in a "requirements.txt" file provided with the programs. To install the requirements the pip package installer can be used with the command "pip install -r requirements.txt". If the system has a GPU with Cuda support then Pytorch with Cuda needs to be installed.³ Additionally, "useGPU" needs to be enabled through configuration of the software. Further details on the software will be provided in subsequent sections on the various parts of the software stacks, and instructions are available in the project repository.

The IG2NLP main program and API are primarily restricted in terms of parallel execution and memory. As the Stanza pipeline scales with core count the pipeline runs faster with more cores, or with GPU processing as GPUs are especially suited for parallel execution. Running the accurate models for the Stanza pipeline which are enabled by default requires a lot of memory. When testing on a fresh Ubuntu server installation the system used around 13 gigabytes of RAM when running the API on the CPU. Based on this it is recommended to have at least 16 gigabytes of RAM for the software. This requirement decreases if a GPU is used for the pipeline, and increases if the batch size is increased. Further, it is possible to use faster models for the Stanza pipeline using configuration options of the software, which would lessen both the execution time and the memory requirements at the cost of accuracy.

The IG Parser prototype can be downloaded from the GitHub repository.⁴ Running the IG Parser prototype requires Golang and has been tested on Golang 1.21. Building the program can be done using the command "go build -o ig-parser.exe ./web", alternatively the program can be run using the command "go run ./web/-main.go" from the root folder.

In addition to the methods outlined above, there are docker container setups for both the IG2NLP REST API and the frontend that can be run using docker. Instructions for using the docker environments are provided in the respective repositories with the dockerfiles.

³Pytorch local installation instructions: <https://pytorch.org/get-started/locally/>

⁴IG Parser prototype <https://github.com/Kjaerandsen/IG-Parser-prot-vis>

6.3.2 IG2NLP

The main IG2NLP application is a Python CLI application that takes the input of a structured JSON document of base statements. This set of base statements is annotated through automation to either regulative or constitutive statements based on an input parameter. The basic inner workings of the system can be seen in the diagram below (figure 6.4):

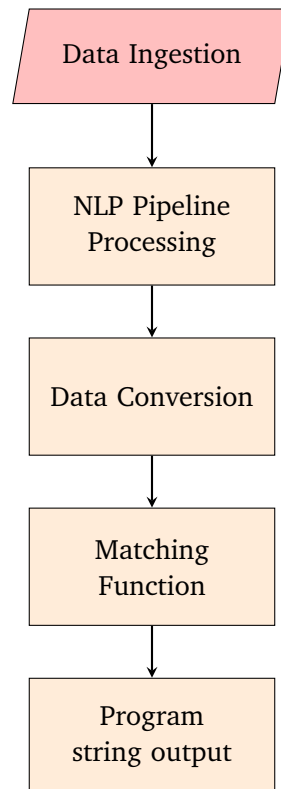


Figure 6.4: IG2NLP program diagram

Firstly the input JSON data is read from the specified file, then the Stanza NLP pipeline is run on each statement. After the NLP pipeline, the resulting data is used to populate a list of custom Word class instances, with the containing text, dependencies from dependency parsing, Coreference resolution data, POS-tags, uFeats, etc. Finally, the matching function is run on this data and the output is written to a specified JSON file.

To facilitate a wider set of use cases several parameters can be used to parameterize the annotation. This includes:

1. a Constitutive or Regulative statement toggle.
2. Enabling or disabling Coreference resolution.
3. Enabling semantic annotations and toggling specific semantic annotations.
4. Changing the models used for the NLP pipeline, from accurate to default, to fast presets.
5. Other parameters not related to annotations directly such as input, and output files, the batch size used in the NLP pipeline, whether to use CPU or GPU processing, and using cached pipeline results.

6.3.3 API

Following the development of the IG2NLP software, a REST API was developed to allow for integration into other software such as the IG Parser. To facilitate proper integration into other software applications the API was developed with versioning and parameters to enable a consistent implementation that can support several use cases and to ensure backward compatibility. The API also supports configuration on the hosting side to choose which models to use, and whether to enable GPU processing. This way the software can be tailored to different needs and backgrounds. A hosted instance using faster models may have less accuracy, however the processing time is drastically reduced.

This is a trade-off that may be worth considering if the API were to be made available publicly. For end users of the API, the JSON data format supports per-statement matching parameters. This allows end-users to enable various forms of semantic annotation and if enabled at the host side toggling of Coreference resolution.

To use the API a new JSON data structure was created for the task. The main principles behind the format are a focus on maintainability and future-proofing. As this is a developing field with frequent new releases of different NLP technique implementations. Therefore, several choices in the data structure were made to allow for iterations to the underlying technology, whilst maintaining backwards-compatibility through parameterization and versioning of the API. Requests are performed as HTTP POST requests to the `/ig2nlp` endpoint. An example of a request can be seen below (code listing 6.1):

```
1  [{
2      "stmtId": "1",
3      "origStmt": "word",
4      "apiVersion": 0.1,
5      "matchingParams": {
6          "coref": true,
7          "semantic": true,
8          "semanticNumber": true
9      }
10 }]
```

Code Listing 6.1: API JSON request format

The JSON data above also allows for several additional statements to be added to the same request. Each statement can have a different "apiVersion" parameter, and different matching parameters that change the resulting annotations accordingly. As the current implementation of the annotator does not have a method for distinguishing between a regulative and a constitutive statement the API returns both a version of the input text annotated as regulative and as constitutive. The response JSON data format can be seen below (code listing 6.2):

```
1  [{
2    "apiVersion": 0.1,
3    "commentConst": "",
4    "commentReg": "",
5    "encodedStmtConst": "F(word).",
6    "encodedStmtReg": "I(word).",
7    "matchingParams": {
8      "coref": true,
9      "semantic": true,
10     "semanticNumber": true
11   },
12   "origStmt": "Attribute.",
13   "stmtId": "1"
14 }
```

Code Listing 6.2: API JSON response format

In addition to the two annotations of the input text, the output also includes comments from the matching function and the parameters and API version used for each statement. The comments can include information to suggest further manual inspection of the statement annotations, such as explaining that coreference resolution was used to replace a word with an entity, or an indication of multiple conflicting logical operators, where manually verifying the scoping is suggested.

6.3.4 Frontend

Following the user interface prototype work in the preceding pilot project (Section 5.3) several alterations and additions have been made to the repository. To keep the prototype up to date, and ready for integration it has been updated several times to include changes in the main IG Parser repository. Further, several bug fixes, and new features have been implemented. These features include the option to toggle the new editor interface. Toggling the interface is implemented as a simple button that toggles the new visualizations for symbols, and the new input options as can be seen below (figure 6.5):

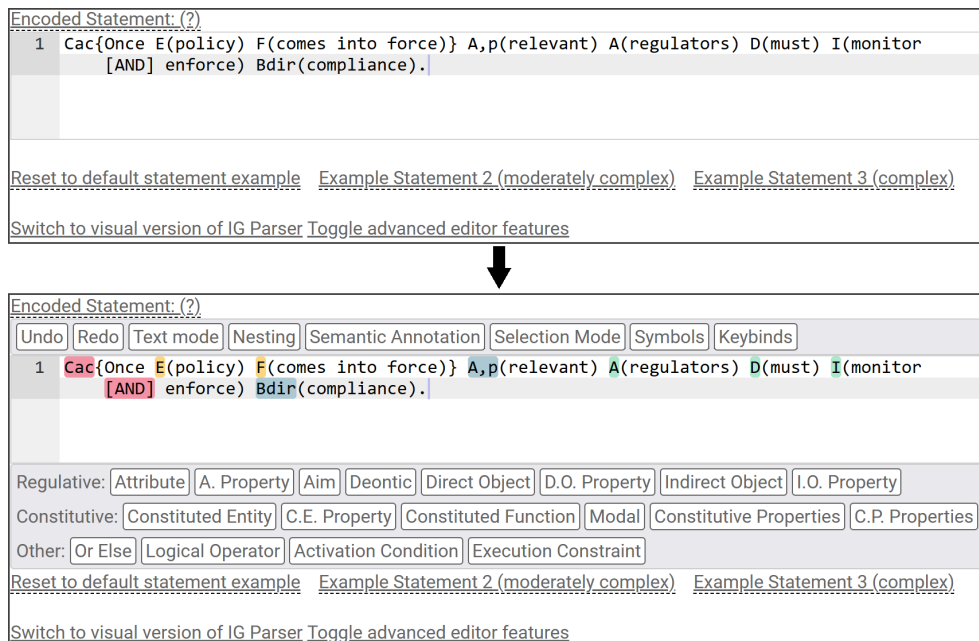
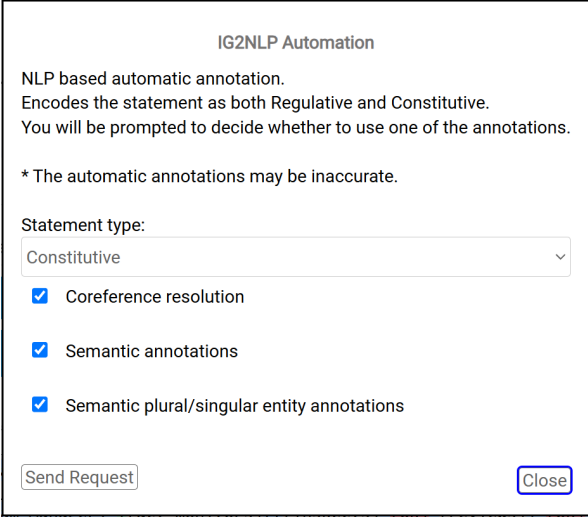


Figure 6.5: Text editor window with the advanced UI toggled on and off respectively.

Further, the IG2NLP automated annotation software was integrated into the IG Parser using the API described above. Through clicking the "IG2NLP Automation Interface" a modal is opened which allows the user to select options for the IG2NLP automation, and send a request for automated annotation. This modal can be seen in the figure below (figure 6.6):



IG2NLP Automation

NLP based automatic annotation.
Encodes the statement as both Regulative and Constitutive.
You will be prompted to decide whether to use one of the annotations.

* The automatic annotations may be inaccurate.

Statement type:
Constitutive

- Coreference resolution
- Semantic annotations
- Semantic plural/singular entity annotations

Send Request Close

Figure 6.6: IG2NLP Automation frontend modal.

The resulting annotation is subsequently used to fill the editor window, and any comments from the api are displayed above the editor. These comments give additional information surrounding the annotation, such as whether an entity has been injected to replace a pronoun, or if there are different types of logical operators in a component. The comments serve as additional information for human coders and give a starting point for fine-tuning the annotations. For example, with a comment on the logical operators, the implication is that the human coder should look at the scoping of the logical operators to verify or correct the scope.

6.3.5 Development Tools

In addition to the main software, several tools have been developed to aid in the development process. This includes visualization tools, a basic cache implementation for the NLP pipeline data, and other tools to view the results of the various NLP models.

When developing the matching function the process of updating, running the NLP pipeline, the matching function, and checking the results a bottleneck became apparent. At each start of the program, the NLP pipeline needs to be initialized and then accessed to process each statement, in batch or sequentially. To speed up this process a basic caching program was developed. The cache works by running each statement through the pipeline, converting the format to the custom Word format used by the software, and then saving this data to a JSON document. This JSON document can then be retrieved by the IG2NLP program to skip the pipeline initialization and execution steps. Thereby saving most of the execution time, and allowing for more rapid iterations and testing. In the future, this caching solution can potentially be implemented for the API as well, so that a user can request the same statement with different parameters, without requiring the server to re-run the pipeline on the statement.

Another step in the matching function development is pattern recognition, finding and recognizing where various dependencies, POS-tags or named entities correspond with elements of the IG. To help in researching, and testing several additional smaller programs were made. Firstly in visualization, the Spacy Display visualizer was used to visualize the parse trees from the dependency parsing method. This was implemented in two different programs; **parseTrees.py** for batch processing of a data set, and **dependencyParsing.py** for free input single statement visualization. Further programs to compare the different named entity recognition models were made, and **pipelineData.py** creates a text file from a dataset with relevant data from the dependency parsing, POS-tags, uFeats and NER. This file can be used to look for patterns in the statements to discover new opportunities for the automated annotation software to utilize.

6.3.6 Testbed

The final part of the software stack is the test suite, a set of tools for semi-automated testing of the annotations. The test suite goes through a JSON dataset consisting of manually annotated and statements annotated through automation, and compares the annotations at different levels to evaluate the accuracy of the annotations compared to the manually annotated control data. The basic workflow includes the steps as outlined in the diagram below (figure 6.7):

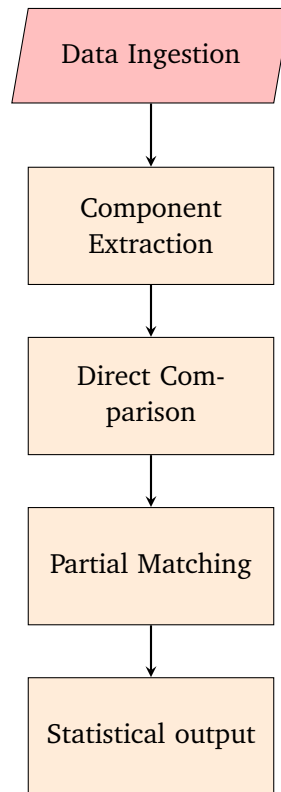


Figure 6.7: Testing diagram

Data ingestion takes a set of statements as outlined above with their annotations. Component extraction works with a combination of regular expressions and a custom extraction method for picking up the entire component text. As part of this workflow pre-processing is used to remove ids from components and semantic annotations are currently not evaluated using this solution. After the components are extracted a separate program can be used to compare the component lists from the manual and the automated annotations. This is done through a process with several steps of granularity that will be described in the next chapter.

To perform some basic load testing on the API a simple program was also made, the program works by selecting a small batch of statements at random from a dataset, and sending a request to the API with those statements. Through

running this program over time some basic performance and requirement testing can be performed.

More details on the test-suite and the specifics of the testing methodology will be presented together with results in the next chapter.

Chapter 7

Testing and Evaluation

In this chapter the testing of the IG2NLP artifact will be described in detail, starting with the testing methodology, and the reasoning behind each decision in the development of the so-called "test bed". Followed by testing of the artifact at different levels of complexity and a discussion of the performance and its implications to the wider field of institutional grammar research.

7.1 Testing Methodology

To perform tests on the performance of the automated annotation solution a so-called "test bed" has been developed. This consists of comparisons between statements annotated through automation, and their manually annotated counterparts. At the most basic level, there is the option of directly comparing the two texts character by character, however, there are coding styles and formatting that may cause this most basic method to suffer in terms of accuracy and quality of the evaluation.

To mitigate this the first step taken was the pre-processing step as outlined in the previous chapter. The next step lies in scoping the comparison. Instead of comparing character by character a method of comparing components at several levels is adopted. This can be divided into two primary parts, firstly counting true positives, false positives and false negatives. However, there are intricacies in the annotation of institutional statements that make this method lackluster in some areas. For example, in the test dataset, there are several partial matches of various degrees and "types". This partial match should not be counted as a direct mistake in many cases. Examples of partial matches covered are incorrect scoping of components and incorrect component typing. This partial match is the second level of testing where another category is added of matches that are partially correct, which includes clear examples of potential areas of improvement in a consistent format. Examples of partial positives and their criteria will be provided below followed by an introduction and explanation of the testing methodology.

An example of partial matches includes components with properties, i.e. an Attribute in the IG can have an Attribute Property connected, such as in the manually

annotated version:

"the A,p(acting) A(Manager)"

In such cases, a partial positive could be the same section annotated as:

"the A(acting Manager)"

The opposite of this can also happen, where a component is split into a component and a property, instead of annotated as a single component due to context, in such cases that would also be counted as a partial positive. I.e.:

Manual: "A(acting Manager)" and Automated: "A,p(acting) A(Manager)"

Another common occurrence in testing is difficulty in differentiating between different component types in certain situations. For example, differentiating between an Activation Condition and an Execution Constraint often relies on the context within the sentence and is difficult using the NLP techniques that the IG2NLP solution bases its annotations on. Such occurrences are also covered in the partial positive category of the test bed. This partial matching based on incorrect component annotation was limited to component pairs of Activation Conditions (Cac), Execution Constraints (Cex) and Objects (Bdir and Bind) or Constituting Properties (P). This is because these components are closely related and can be hard to distinguish between.

Further for the initial testing logical operators were ignored, however in the future it might be worth looking into methods of more accurately covering logical operators and testing such solutions. A limitation of the software now is the inability to distinguish between the logical "and/or" ([OR]; inclusive or) and the "either or" ([XOR]; exclusive or), and in the scoping or encapsulation of logical operators within components.

Additional edge cases that were manually handled, such as with missing determiners (a, or the for example), cases with a third level of properties (Pp,p), where this was adjusted to a second-degree property instead (Pp), and division of components linked by a logical "and" where the manual and automated annotations differed for example if the manually annotated version is:

A(a) and A,p(property) A(b)."

In this case if the automated annotation is:

A(a [and] property b)"

Then the automated annotation would be manually altered to the following:

A(a) and A(property b)"

This form maintains the same exact meaning in the language, but allows matching both attributes from the manual annotation to the automated annotation, with the first being a true positive, and the second being a partial positive because of the combination of the Attribute and the property in the same component.

Finally, when testing components without considering nested components, the inferred components within a nested component are ignored. See the section on nesting and inference for more detail on component inference (section 7.2.5).

To help in the evaluation of the solution a set of tools were created that helped automate the evaluation process. The tooling worked in a simple two-step process, first, the dataset was parsed to extract all relevant components and extra data such as semantic annotations. Following this, a three-step comparison is performed component by component. This process works as follows:

1. The components are directly compared, where both the content and the component type need to be direct matches, this is used to detect true positives. Where the components are equal except for nesting, i.e. the manually annotated version detects a component as nested, while the statement annotated through automation does not, the match is counted as a partial positive.
2. After direct comparisons the component contents are compared across component types, i.e. if an Execution Constraint is annotated as an Activation Condition component then a partial positive match is counted.
3. Finally, scoping is handled, where the solution looks for partial content matches for the partial positive counter and tries to group all components that partially match another component. This can for example be as discussed above where an Attribute and an Attribute Property component are annotated as a single Attribute component. These matches detected through automation are then appended to an output list of partial positive matches where they can be manually reviewed. After manually reviewing the pool of automated partial matches the rest of the components can be counted as either false positives, or false negatives, depending on whether they originate from the manually -, or the statement annotated through automation.

The steps outlined above match component contents based on the text content of the component, but ignores square brackets ([]), parentheses (()), logical operators (and, or, xor), commas (,) and two articles (a and the). This is done to simplify the automated comparison, and reduce the amount of manual intervention required. Because of these exclusions the testing does not test logical operator handling. The basic input structure for the testing program consists of a JSON document with a list of statements, including their base text, names (an identifier for the statement), the manually annotated statement, and the IG2NLP annotated statement as seen below (code listing 7.1):

```

1  [
2  {
3    "name": "Name",
4    "baseTx": "relevant union legislation",
5    "manuTx": "Bdir,p(relevant) Bdir(Union legislation) ...",
6    "autoTx": "Bdir(relevant Union legislation) ..."
7  }
8  ]

```

Code Listing 7.1: Dataset JSON format

This dataset consisting of such statements in JSON format is then processed to extract components in the basic format (code listing 7.2):

```

1  {
2    "Content": "relevant Union legislation",
3    "Nested": false,
4    "componentType": "Bdir"
5  }

```

Code Listing 7.2: Testing component JSON format

These components are organized by component type(**A**, **A,p**, ...), and their source(manually annotated or annotated through automation). In the future, this component object can be extended to include information such as suffixes and semantic annotations for more thorough comparisons.

Finally, after all components for each statement in the dataset have been extracted, this data can be read and the components from the manually and IG2NLP annotated statements can be compared. As explained above the comparison is done at two levels, resulting in a count of True Positives, Partial Positives, False Positives, False Negatives, and a total per statement. Direct matches are added to the True Positives counter and removed from the component list, subsequently partial positive matches are added to a list as can be seen in the JSON excerpt below (code listing 7.3):


```

1  "partialMatches": [{
2    "manuTxComponents": [{
3      "Content": "Union legislation",
4      "Nested": false,
5      "componentType": "Bdir"
6    }, {
7      "Content": "relevant",
8      "Nested": false,
9      "componentType": "Bdir,p"
10   }],
11   "autoTxComponents": [{
12     "Content": "relevant Union legislation",
13     "Nested": false,
14     "componentType": "Bdir"
15   }]
16 }]}

```

Code Listing 7.3: Testing partialMatches component matching JSON format

Finally, all components not matched directly or partially, are added to the "extraComponents" for the two annotations. Where these can be manually checked, and the extra components annotated by IG2NLP are counted as False Positives and the other extra components are counted as False Negatives.

With all the examples shown here of JSON data, there can be additional items, i.e. more than one component, or more than one partial component match per statement.

This testing process was performed at two levels, one where nesting was ignored completely, and one where the detection of nested components was also evaluated. The introduction of nested statements adds several layers of complexity to the solution. It does not only need to be able to detect and scope components, but then also go through each component with support for nesting and detect whether or not the component should be nested, and accurately annotate the components inside such nested components as well. To evaluate this properly all cases of nested components in the datasets were compared, to first see whether they were detected as nested, subsequently if they were properly scoped, and finally the accuracy of the internal component detection and annotation was evaluated. The way this comparison works is that if a component is annotated as nested in the manually annotated version and not in the automated annotation then the components inside the nested component of the manually annotated component are all counted as false negatives. If, on the other hand, the manually annotated statement has a false positive nested components, then all components within that nested component will be counted as false positives, and if both have annotated the component as nested, then the internal components can also be compared the

same way as all other components are compared. The total amount of regulative statements in the initial testing was **53**, and for constitutive statements, there were **19** in total spread across two datasets from two different domains (climate regulation and cybersecurity). The number of statements used is lower than the total amount of statements in the dataset due to the exclusion of statements with both regulative and constitutive sections and certain complex structures. The justification to these excluded statements as well as more detail will be presented in a later section (Section 7.2.3)

The following section will go over several rounds of testing performed, the results, and the implications of the results. Including a discussion of results at the various levels of testing, and an initial discussion on some of the strengths and limitations of the automated annotation.

7.2 Testing Results

This section goes over the testing results, starting with testing without nesting, followed by testing with nesting, and a discussion of excluded statements, the reasoning behind their exclusion, and how they relate to the limitations of the software. Finally, a discussion of the strengths and limitations of the software will take place. The sections below refer to components by their respective IG Script symbols, refer to the table in section 2.1 on page 8 for the component names. A figure showing how many of each component occurred in the training data set used for testing can be seen below (figure 7.1):

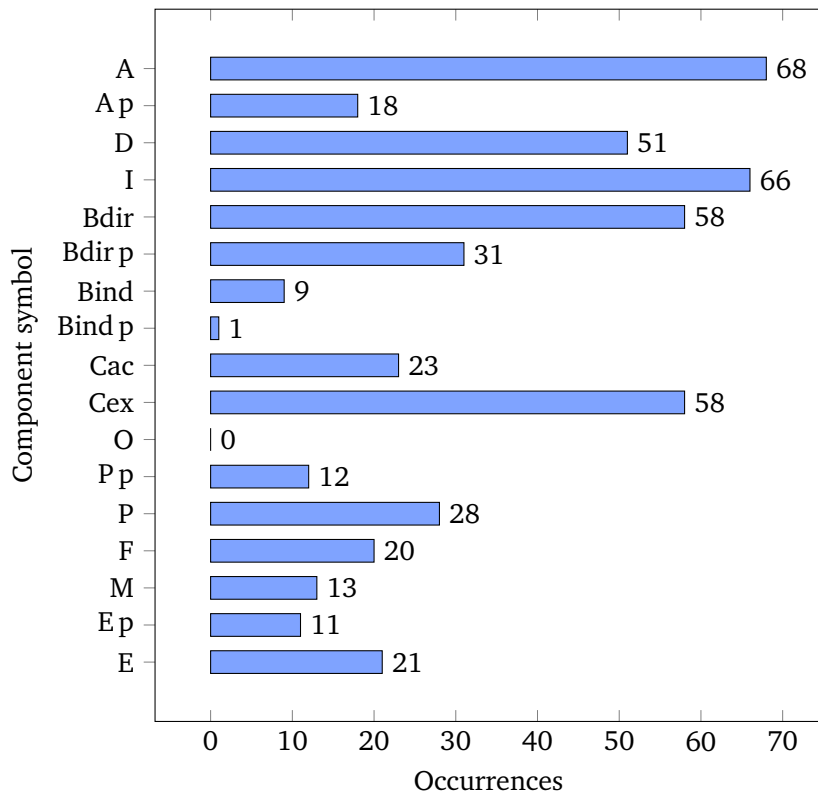


Figure 7.1: Component counts in training data

7.2.1 Testing Without Nesting

In the testing data the presence of nested component structures was limited to eight regulative statements and one constitutive statement, and one additional regulative statement had a false positive nested component. This section will expand on the previous section by testing these statements and discussing the performance of nested component handling.

Regulative statements

Symbol	TP	PP	FP	FN	Total count
A	86.21%	8.62%	3.45%	1.72%	58
A,p	68.75%	12.50%	6.25%	12.50%	16
I	65.00%	18.33%	10.00%	6.67%	60
D	92.45%	0.00%	3.77%	3.77%	53
Bdir	50.88%	36.84%	7.02%	5.26%	57
Bdir,p	18.18%	77.27%	0.00%	4.55%	22
Bind	22.22%	66.67%	0.00%	11.11%	9
Bind,p	0.00%	100.00%	0.00%	0.00%	1
Cac	80.00%	15.00%	0.00%	5.00%	19
Cex	46.67%	42.22%	4.44%	6.67%	45
O	N/A	N/A	N/A	N/A	N/A

Table 7.1: Regulative statement component metrics without nesting

As you can see in the table above (table 7.1) there are certain components with fairly high accuracy in detection and scoping. The Aim, Deontic and Activation Condition components all have True Positive rates of 80% or more. This means that the automated annotation of these components are reliable and fairly accurate on the datasets used in this testing. This in part supports the general usability of general-purpose NLP techniques for the detection of general components that commonly mirrors actors, deontic signals and conditions in natural language. For other components such as the Attribute Property and the Aim the true positive rates are lower. However, when including the partial positive metric the total accuracy is above 80%. This also applies to Objects and the Execution Constraint components. However, the true positive rate for these components are considerably lower. This indicates a combination of difficulty in scoping or component typing, and provides less confidence in the annotation. As such the object and Activation Condition annotations should be seen more as suggestions than as strict annotations as differentiating between these components offers challenges, and the scoping is also challenging in many areas. These difficulties will be described in further detail in the discussion section of this chapter (section 7.2.5). The next table below (table 7.2) shows the statistics for constitutive statements without nesting:

Constitutive statements

Symbol	TP	PP	FP	FN	Total count
E	50.00%	22.73%	9.09%	18.18%	22
E,p	27.27%	45.45%	0.00%	27.27%	11
F	54.17%	16.67%	20.83%	8.33%	24
M	100%	0.00%	0.00%	0.00%	13
P	31.25%	37.50%	15.63%	15.63%	32
Pp	0.00%	71.43%	21.43%	7.14%	14
Cac	33.33%	66.67%	0.00%	0.00%	3
Cex	33.33%	33.33%	16.67%	16.67%	12
O	N/A	N/A	N/A	N/A	N/A

Table 7.2: Constitutive statement component metrics without nesting

In this table the results for the Modal component have 100% accuracy.¹ However, for the other components the accuracy is lower than for their regulative counterparts. The Constituted Entity and Constitutive Function components have around 50% true positive rates, for the Constitutive Function component this is mainly due to scoping of the component. With more work on the scoping function, this accuracy could be improved further. In the case of Constituted Entities, the problem is in the detection mechanism based on dependency parsing. This method seems to fail in the context of constitutive statements due to overlap with the Constituting Properties, this failure will be discussed more in the discussion section (section 7.2.5). For the rest of the constitutive components, there is more overlap in the detection than there is for regulative statements, and the annotations are less accurate. This can be a result of different requirements for constitutive statements where the method for automated annotation fails. Notably, constitutive statements have a more general applicability than regulative statements, lowering the syntactic specificity of the individual components. Another practical constraint in the context of this thesis is the limited amount of constitutive statements available in the development process and for evaluation compared to regulative statements.

In the next subsection the testing will be expanded to include nested component structures and the effects of this will be discussed.

¹This is of limited surprise, given the focal scoping of modal signals and their similarity to deontics on the regulative side.

7.2.2 Testing With Nesting

The total amount of statements of the test set that exhibited nesting features was nine statements. One of these statements is a constitutive statement and the other eight are regulative statements.² In addition to these eight regulative statements there is a regulative statement where an Activation Condition was falsely detected as nested by the automated solution.

Out of these statements, there are a total of twelve components that are nested. Out of these twelve nested components the automated solution properly detected and encapsulated seven of these components, five of them were false negatives, and an additional false positive was detected. If we count the false positive in the total this gives a true positive rate of 53.8%, a false negative rate of 38.4% and a false positive rate of 7.69%. However, this only takes the main component into account, the handling of nested components also deals with internal component annotations which brings further components to annotate, and further complexities.

Reflecting the multi-stage parsing of nested statements (an initial parsing that only identifies the first-order components, such as Attributes, Aims, etc., followed by a deep parsing within all those components, the results presented here take a differentiated approach to clearly dissect classification performance on general identification vs. nesting. The first table (table 7.3) below hence highlights the results for the same set of statements without taking nesting into account. The second table (table 7.4) includes the consideration of nesting, increasing the total number of components across statements, inadvertently affecting the results:

Symbol	TP	PP	FP	FN	Total count
A	84.62%	7.69%	7.69%	0.00%	13
A,p	60.00%	20.00%	20.00%	0.00%	5
I	50.00%	0.00%	25.00%	25.00%	12
D	100%	0.00%	0.00%	0.00%	9
Bdir	55.56%	44.44%	0.00%	0.00%	9
Bdir,p	33.33%	66.67%	0.00%	0.00%	4
Bind	66.67%	33.33%	0.00%	0.00%	3
Cac	66.67%	33.33%	0.00%	0.00%	6
Cex	20.00%	60.00%	0.00%	20.00%	5
E	100.00%	0.00%	0.00%	0.00%	1
F	100%	0.00%	0.00%	0.00%	1
M	100%	0.00%	0.00%	0.00%	1
P	0.00%	100%	0.00%	0.00%	1
Pp	0.00%	50.00%	0.00%	50.00%	2

Table 7.3: Testing of statements without accounting for nesting

²Overall, the more limited number of statements reflects the special nature of nesting that does not occur in all institutional statements.

The table above has omitted components that were not present in the statements tested. These numbers serve as a baseline to compare the results in the next table (table 7.4). The following table shows the same statements, but with nesting of components enabled to gauge how this affects the accuracy of the annotation.

Symbol	TP	PP	FP	FN	Total count
A	66.67%	11.11%	11.11%	11.11%	27
A,p	50.00%	25.00%	12.50%	12.50%	8
I	45.83%	8.33%	12.50%	33.33%	24
D	72.72%	0.00%	27.27%	9.09%	11
Bdir	39.13%	26.09%	8.70%	26.09%	23
Bdir,p	33.33%	66.67%	0.00%	0.00%	11
Bind	40.00%	40.00%	0.00%	20.00%	5
Cac	55.56%	22.22%	11.11%	11.11%	9
Cex	36.36%	45.45%	0.00%	18.18%	11
E	100.00%	0.00%	0.00%	0.00%	2
F	50.00%	50.00%	0.00%	0.00%	2
M	100%	0.00%	0.00%	0.00%	1
P	0.00%	100%	0.00%	0.00%	2
Pp	0.00%	66.67%	0.00%	33.33%	3

Table 7.4: Testing of statements accounting for nesting

Looking at the results in this second table the first thing to note is that the addition of nesting adds several layers of complexity. This includes the total matches counted, which increased from 72 to 139. The addition of nesting resulted in nearly double the amount of matches. Further, there are many challenges to deal with in nested components, first in detecting that a component should be nested, then in internally annotating components, and inference of components. Inference of components is the process of including an Attribute and or Aim component in a nested statement from outside the direct statement text, for example when annotating a section of a statement as a nested statement the Attribute may be inferred from the base statement, see the subsequent relevant section for more detail (section 7.2.5). In the end the addition of nesting reduced the accuracy for most of the components, some more than others. For example, the Aim and Attribute components are often inferred from context in nested components, which results in new false negatives because the automated solution is currently unable to infer components in this manner. These difficulties and potential solutions will be discussed more in the discussion section (section 7.2.5). Before this more in-depth discussion, the next sections go over exclusions from the testing dataset, as well as the results achieved on another dataset not used in the development process.³

³Note that this work uses off-the-shelf models without training requirement. The use of a second dataset, however, allows us to test for any possible biases introduced during the conceptual mapping from NLP techniques to IG concept.

7.2.3 Exclusions

Certain statements were excluded from the testing above due to being complex cases where the direct comparison of the manual and automated statements was not possible. This was primarily caused by complex cases where the manual annotation of the statements was done by rewriting the statement text entirely.⁴ Additionally, statements that consisted of both regulative and constitutive sections (hybrid statements) were filtered out. An example of this is the first example in the IG Parser:

Cac{*Once E*(policy) *F*(comes into force)} *A,p*(relevant) *A*(regulators) *D*(must)
I(monitor [AND] enforce) *Bdir*(compliance).

In this case, manual intervention is necessary to detect that the internal components of the Activation Condition are constitutive. Manually fixing this could be done through generating both a constitutive and a regulative annotation and manually handling this (using the API both a regulative and constitutive annotation is generated), or in the future, with a statement type detection mechanism, this problem could be solved.

7.2.4 Validation testing

To validate the results of the testing an additional dataset was acquired and used to evaluate the results. This dataset was unseen during the development of the solution and serves as a test for the validity of the solution and how the methods apply to other datasets, while controlling for unintentional biases during development. The validation dataset consisted of 81 regulative and 27 constitutive statements. The same pre-processing procedures as outlined earlier (section 5.1) were followed for the dataset. Further, duplicate statements were removed, and parentheses in the base texts were removed. Finally, primarily for the second category as defined in the next subsection the Attribute component and in one case the Deontic component was inferred from outside the statement text. In such cases, the base text did not contain the component text. This also affected one of the statements in the third category. For these statements, the component text which was inferred from outside the statement was added in manually before testing. Illustrating this using an example, consider the base text below:

Any funds used to support in-state renewable electricity generation facilities pursuant to this section shall be expended in accordance with the provisions of this chapter.

This statement text corresponds to the following manually annotated variant:

⁴This is permissible practice in IG coding, e.g., to compensate or correct for stylistic features of the language in which institutions are expressed.

A([Commission]) Bdir(Any funds) Bdir,p(used to support in-state renewable electricity [...])

For this statement, and similar statements the Attribute text was added to the start of the sentence. The adjusted statement hence reads:

Commission Any funds used to support in-state renewable electricity [...]

Regulative statements

Looking at the testing results, the statements can be roughly divided into three categories, firstly basic statements consisting of a single full sentence, secondly sentences consisting of a phrase with an external Attribute inserted or inferred and finally statements consisting of multiple sentences, or colons and semicolons. The first category contains 56 out of the 81 statements (69%), the second category contains 17 (21%) and the third contains 8 (10%). For these three categories, the performance of the solution varies. The first category is handled well by the solution and the results for this can be seen in the table below (table 7.5):

Symbol	TP	PP	FP	FN	Total count
A	80.70%	15.79%	3.51%	0.00%	57
A,p	0.00%	16.67%	50.00%	33.33%	22
I	79.37%	4.76%	12.70%	3.17%	63
D	98.04%	0.00%	1.96%	0.00%	51
Bdir	27.59%	67.24%	5.17%	0.00%	58
Bdir,p	10.53%	81.58%	5.26%	2.63%	38
Bind	0.00%	100%	0.00%	0.00%	10
Bind,p	0.00%	100.00%	0.00%	0.00%	1
Cac	50.00%	50.00%	0.00%	0.00%	16
Cex	50.00%	50.00%	0.00%	0.00%	28

Table 7.5: Verification regulative statistics category one

In these statements, the Attribute, Aim and Deontic components all have true positive rates around or above 80%. Further, the false positive and false negative rates are rather low, with the exception of the Attribute Property component. The performance in these statements is rather high due to their similarity in structure to the dataset used for development.

Moving on to the second category the solution struggles as the statements are phrases and not full sentences. An example of this "phrase structure" is the statement below:

Commission Any funds used to support in-state renewable electricity [...]

Here the "Commission" is added from outside the phrase and not naturally integrated into the sentence. This causes a couple of problems, first, there are issues where the root of the statement does not correspond with the Aim component, second, there are component overlaps of the Aim component, the Attribute and the Direct Object. Handling such phrases may be possible, but the current implementation is built for fully formatted sentences. See the table below for component matching statistics for these statements (table 7.6):

Symbol	TP	PP	FP	FN	Total count
A	10.71%	17.86%	39.29%	32.14%	28
A,p	0.00%	0.00%	100%	0.00%	6
I	3.85%	46.15%	34.62%	15.38%	26
D	70.59%	5.88%	0.00%	23.53%	17
Bdir	11.76%	23.53%	0.00%	64.71%	17
Bdir,p	10.00%	70.00%	0.00%	20.00%	10
Bind	0.00%	80.00%	0.00%	20.00%	5
Bind,p	0.00%	100.00%	0.00%	0.00%	1
Cac	50.00%	0.00%	50.00%	0.00%	2
Cex	60.00%	20.00%	0.00%	20.00%	15

Table 7.6: Verification regulative statistics category two

In these specific cases, the dependency parser often detects the inserted "Commission" word as a separate sentence, which gives a separate parse tree and gives an Aim component annotation. Where this behaviour is not present, the root handling still does not correspond with the proper aim in all cases. This limitation of root handling can potentially be mitigated in the future by checking whether the root is a verb, and if not, then looking for a verb connected to the original root. This way a new root can be generated, and the discovery of the Attribute component could be simplified, see the figure showing the basic dependency-based matching for reference (figure 5.1). In this figure, the root of the tree is the Aim and a verb, which is what the matching function expects. The other problem with these statements is the structure. Since the statements are not fully formed sentences, the current solution struggles to classify those properly. In the future, it may be possible to write a version of the matching function for such phrases. However, at this initial stage, this may be overfitting to this dataset, which uses a specific phrase structure which is not a natural sentence construction. Another avenue to improve the annotation of these statements would be to improve the Attribute detection mechanism, given that central a problem in these statements is that Direct Object components are detected as Attributes due to the sentence dependencies and structure, while with an updated NER model or another approach, this could be mitigated. This mitigation would work by giving additional information to classify the component as either an Attribute component or a Direct Object. A final note is the Aim component, where in this set of statements the true positive rate is low, this is due to additional context added in the manually annotated ver-

sions of these components. Where the manually annotated version may be "I(be expended [expend])" while the automated solution annotates the same phrase as "I(be expended)". These matches are then counted as partial matches. For more discussion on this and other limitations see the subsequent section (section 7.2.5).

Finally, for longer statements which either consist of several sentences, or that are bound by semi-colons the IG2NLP program struggles to properly encapsulate everything in the same way the manual annotation does. This is primarily an issue of properly scoping very long Activation Condition and Execution Constraint components. Where these statements either have such components starting in the first sentence of the statement and spanning across multiple following sentences, or where such components contain semicolons. In the cases of semicolons, the manual annotation replaces the semicolons with the logical "and", and extends the component to cover the rest of the text. See the table below for component matching statistics for these statements (table 7.7):

Symbol	TP	PP	FP	FN	Total count
A	30.43%	0.00%	65.22%	4.35%	23
A,p	0.00%	0.00%	100%	0.00%	10
I	23.33%	3.33%	73.33%	0.00%	30
D	41.67%	0.00%	58.33%	0.00%	12
Bdir	33.33%	55.56%	11.11%	0.00%	9
Bdir,p	.00%	25.00%	50.00%	25.00%	4
Bind	0.00%	100%	0.00%	0.00%	3
Bind,p	0.00%	100.00%	0.00%	0.00%	2
Cac	25.00%	50.00%	25.00%	0.00%	8
Cex	0.00%	40.00%	60.00%	0.00%	5

Table 7.7: Verification regulative statistics category three

As can be seen in the table above these statements display relatively frequent false positives, an aspect that is due to the scoping issue. Phrases that should be contained within an Activation Condition or Execution Constraint are instead annotated as either a separate statement in the case of multi-sentence annotations or simply merged with other components in cases where semicolons are present. In the future, it may be possible to add handling for lists of phrases bound by semicolons, but there is a potential risk for overfitting in corresponding rule formulation. Hence, a larger dataset of annotated statements would be necessary to base and validate the rules on. For cases where a component spans across multiple sentences, this may be possible to automate, but the current solution treats each sentence in the input as a separate statement, which causes a lot of false positive components in this case.

Finally, when combining all three categories of regulative statements the combined accuracy can be seen in the table below (table 7.8):

Symbol	TP	PP	FP	FN	Total count
A	51.85%	12.96%	25.93%	9.26%	108
A,p	0.00%	4.54%	86.36%	9.09%	22
I	48.74%	13.44%	32.77%	5.04%	119
D	83.75%	1.25%	10%	5.00%	80
Bdir	25.00%	57.14%	4.76%	13.10%	84
Bdir,p	12.82%	75.00%	7.69%	7.69%	52
Bind	0.00%	94.44%	0.00%	5.56%	18
Bind,p	0.00%	100.00%	0.00%	0.00%	4
Cac	42.31%	46.15%	11.54%	0.00%	26
Cex	47.92%	39.58%	6.25%	6.25%	48

Table 7.8: Verification regulative statistics all categories

As can be seen in the table above the Attribute, Attribute Property, and Aim components accuracy suffers due to the false positives in the third category of statements. The Direct Object, Aim and Attribute components also suffer due to the second category of statements, where there are problems in the detection of the Aim and distinguishing between the Attribute and Direct Object components.

The following subsection goes over the results for the constitutive statements of this dataset and will be followed by a subsection testing the annotations of nested components in the dataset.

Constitutive statements

Symbol	TP	PP	FP	FN	Total count
E	25.00%	34.38%	15.63%	25.00%	32
E,p	16.67%	75.00%	0.00%	8.33%	12
F	21.43%	26.19%	38.10%	14.29%	42
M	76.92%	0.00%	15.38%	7.69%	13
P	3.57%	53.57%	7.14%	35.71%	28
Pp	0.00%	85.71%	0.00%	14.29%	7
Cac	0.00%	66.67%	33.33%	0.00%	3
Cex	22.22%	66.67%	11.11%	0.00%	9

Table 7.9: Constitutive validation dataset statement component metrics without nesting

As can be seen in the table above (table 7.9) the component accuracy for constitutive statements in the validation dataset is lower than in the dataset used in development. With the exception of the Modal component, all components have a sub 30% true positive rate, which indicates problems in scoping and detection of components. The lower accuracy can be explained in three ways, firstly the development relied on a small dataset of constitutive statements. The size of the development dataset gives fewer points of data to base the rule creation on, and less variance which can lead to conceptual overfitting in rule development. The second aspect lies in limitations of the method used to automate the annotation which includes problems in differentiating between component types in some cases and limitations of the models used. Finally, the dataset contained sentence structures that the employed dependency parser struggled with due to the stylistic specificities of the input statement. This caused the root of the dependency parse tree not to centre on the Constitutive Function component in the affected statements, which caused false positive Constitutive Function component detection, and false negative Constituted Entity detection, among other side-effects. This may be due to the specific formatting of the base text which in many cases consists of phrases that have been substantively modified in the encoding process. The structure of some of the input sentences is in several cases more reminiscent of phrases than full sentences which the IG2NLP program was developed for. An illustrative example is the statement text below:

"[environmental and public benefits] including improved air quality"

In this statement, the root of the dependency parse tree is "benefits", which is a noun. This is contrary to the statements in the training data where the root generally was a verb and corresponded with the Aim or Constitutive Function of the statement. This behaviour is the same as described above when talking about the second category of "phrase" statements. In the case of the statement above "including" is the Constitutive function, and is detected as a verb, while the root of the parse tree is "benefits" which is therefore annotated erroneously

as the Constitutive Function. Another problem is in scoping of longer components where it can be challenging to fully encapsulate everything accurately, while trying to distinguish between the main component and its properties. More discussion on the limitations and possible additions to IG2NLP are subject to the next section (section 7.2.5).

Nested components

Only a subset of the validation dataset was fully annotated with nested components. This included five statements. This subsection will compare the results of the automated annotations with these statements to gauge how well the detection and accuracy of the nested component detection works. Out of these five statements, there were a total of seven nested components. Out of these six components, two were detected by the automated solution with accurate scoping, and four were not detected. This gives an accuracy of 2/7% on detection, and a false negative rate of 5/7% which is lower than on the development dataset. Further, one of these two matches had the wrong component type, where an Execution Constraint was annotated as an Activation Condition. This lower accuracy is partly due to a limitation of the detection mechanism which relies on finding an Attribute component and an Aim component, and in four of the seven nested statements the Attribute was inferred through context which the solution is currently unable to do.

In the first table (table 7.10) below you can see the results for these statements when not taking nesting into account, in the second table (table 7.11) nesting is also tested, which increases the total amount of components in the statements and changes the results:

Symbol	TP	PP	FP	FN	Total count
A	80.00%	20.00%	0.00%	0.00%	5
I	100%	0.00%	0.00%	0.00%	5
D	100%	0.00%	0.00%	0.00%	4
Bdir	60.00%	40.00%	0.00%	0.00%	5
Bdir,p	25.00%	75.00%	0.00%	0.00%	4
Cex	0.00%	100%	0.00%	0.00%	3

Table 7.10: Testing of statements accounting for nesting on the validation dataset

Because of the low amount of statements the amount of total manually annotated components is low, and the amount of different component types covered in these statements is also low. In the next table nested component structures are also counted, which due to only two out of seven nested components being detected will cause a large amount of extra false negatives. This is because of the components which are annotated within the nested components. The updated results can be seen in the table below (table 7.11):

Symbol	TP	PP	FP	FN	Total count
A	41.67%	16.67%	0.00%	41.67%	12
A,p	0.00%	0.00%	0.00%	100%	1
I	50.00%	8.33%	0.00%	41.67%	12
D	100%	0.00%	0.00%	0.00%	4
Bdir	42.29%	42.29%	0.00%	14.28%	14
Bdir,p	11.11%	44.44%	0.00%	44.44%	9
Cex	22.22%	33.33%	0.00%	44.44%	9

Table 7.11: Testing of statements accounting for nesting on the validation dataset

Reviewing the numbers in the table above, a reduction in the true positive rate of the Attribute and Aim is apparent. This is due to the necessary presence of an Attribute and an Aim component in a nested regulative statement. Further, the total component count shows an increase, and the amount of false negatives has a drastic increase due to these nested components and their internal components. To increase the detection rate of nested components a method to handle Attribute inference could be an option, this problem of nesting an inference is discussed in more detail in a later subsection (section 7.2.5). Before that, the following section presents a discussion of the results of this testing and some of the limitations to the IG2NLP software, before going into detail on limitations to the automated annotations and potential ways to mitigate them in the future.

7.2.5 Discussion and Limitations

In this subsection, the results of the testing will be discussed. Starting with a general overview of the results, followed by a discussion of the strengths and weaknesses of the solution. Finally, an in-depth discussion of some of the specific limitations of the solution, and how they are addressed or can be addressed in the future.

From the numbers shown in the preceding sub-chapters, it is apparent that the performance of IG2NLP is better for regulative statements than it is for constitutive statements. This is based on a combination of issues. Firstly the sample size of constitutive statements was around a third the size of the sample size of regulative statements. With more data to base the automated annotations on a wider range of possible sentence structures and word combinations was present. This meant that the solution could be tested more thoroughly, and could be made to adapt to a wider set of statement structures. Additionally, for constitutive statements, a recurring issue was in dependency overlap between Constituted Entities and Constitutive Properties. Where the structural overlap led to incorrect annotations in both directions, causing lower annotation accuracy for both of these components. This limitation, and several others will be discussed in further detail in a subsequent paragraph. The results of the automated constitutive annotations should currently be viewed as more of a suggestion than an accurate annotation. However, with more training data and by introducing further tools for the purpose of cross-validation or refined multi-stage detection it might be possible to achieve higher performance. At the same time, these results and metrics provide a sound baseline performance on which future solutions can build and relate to.

For regulative statements, the accuracy of Attributes, Deontics, Execution Constraints, and Activation Conditions are all rather high with around 80% true positive rates on the training data set. However, there are still improvements to be made in extending the support for nesting to cover all relevant components. There is also currently a limitation to the annotation of Attribute components, where in some cases Direct Objects are incorrectly annotated as Attributes, either because of similar dependency structures or as a side effect of the solution not handling component inference, which will be presented and discussed in further detail in a subsequent paragraph. This issue was more apparent in the validation data set with the second category of "phrase" statements. Further, execution Constraint components also have some overlap with other component types which will need to be addressed to improve the accuracy of the annotations. Additionally, Indirect Object occurrences were limited in the dataset used to develop the solution, so to improve the Indirect Object annotation accuracy more training data is necessary.

With the validation data set some of the strengths and limitations of the solution became apparent. Firstly, statements consisting of a single sentence without the use of semicolons, or multiple sentences the solution handled well, with performance comparable to the performance on the data set used in development. Secondly, with smaller statements consisting of phrases with the Attribute com-

ponent included from outside the statement, the solution struggled with the Attribute, Aim and Direct Object components. This was in some cases due to the root node of the dependency parse tree not corresponding with the Aim component, and in other cases due to difficulties in differentiating Attributes and Direct Objects and the sentence structure. This caused an increase in false positives and false negatives for these components. These difficulties could potentially be remedied in the future with a traversal mechanism for the root, and an NER model for detecting whether a component should be an Attribute or a Direct Object. For this challenge, it may also be possible to use an LLM. Finally, the third issue was with longer statements consisting of multiple sentences, or an Execution Constraint or Activation Condition containing one or more semi-colons. In this case, the solution struggled with the additional sentences or parts after the colons, where the solution handles these as separate statements. This behaviour causes a lot of extra false positives. The two primary limitations discussed indicate that the solution may be overfit to full-sentence statements. This overfitting is partly due to the method of dependency parsing used, where the main matching is based on dependency parsing and with the introduction of other mechanisms to detect certain components the performance can potentially be increased for a wider range of statements. Another reason is the datasets used which consisted mostly of statements consisting of statements with a single fully formed sentence, and the size of the dataset used in the development. Because of the relatively low number of statements used in the development process, there are potential edge cases, such as these statement types mentioned where the solution may struggle. A final limitation to note is scoping of components where there is currently a bug in scoping of components where for one of the statements of the validation dataset there was an extra opening bracket which needed to be manually removed in order to make the statement valid. However, this could be automatically detected by running the automated annotations through the IG Parser where an error message would indicate this, and signal the requirement of manual review.

To improve the performance of the model more data to be used in the development process would help to find patterns, and edge cases and to test the rules created. Further, with more data, it may be possible to test more specific matching function rules, such as testing the use of a dictionary-based approach for certain component types, especially for components with recurring terms and limited stylistic variation. A dictionary-based approach could for example be helpful for detecting Deontic components throughout a statement, by looking for keywords such as "shall" and "must". This may also be a way to help distinguish Activation Condition and Execution Constraint components with keywords such as "before" and "after" which can be used to indicate that the component is an Activation Condition.

Further, when looking at nested components the solution is fairly accurate concerning the nested components that are in fact detected, both in scoping and internal annotation. However, there are limitations in the detection rate, and in cases where the Attribute and Aim or Constituted Entity and Constitutive Function

components are inferred outside the direct component text which the automated solution does not currently handle. This limitation will be discussed more in a following subsection (section 7.2.5). Another problem is in statements consisting of nested parts that are regulative or constitutive in a constitutive or regulative statement respectively. This case necessitates a method for detecting whether the component contents are regulative or constitutive which is currently an unsolved issue. However, it may be possible to develop a way to detect whether a statement is regulative or constitutive through the use of an LLM or through an algorithm using a combination of keywords, and NER or a classification technique.

Reviewing the test process itself, the current process is semi-automated with automated component extraction from annotated statements and automated component comparisons. However, there is a lot of manual work necessary to validate and correct the automated component pairing for the partial positive metric and the automation struggles when dealing with nested components. For nested component structures manual review was necessary in most cases.

In the following subsections, various limitations of the automated annotations will be presented along with potential solutions or mitigation strategies.

Execution Constraints and overlapping components

Execution Constraints throughout the data set have proven themselves difficult to accurately annotate due to an overlap in dependencies from dependency parsing with other component types. This includes Activation Conditions, Objects, both direct and indirect, and for constitutive statements the Constituting Properties. The primary mechanism used for the detection of Execution Constraints used is the oblique nominal (obl) dependency. The Universal Dependencies website defines the obl as a relation used for "a nominal (noun, pronoun, noun phrase) functioning as a non-core (oblique) argument or adjunct." [60]. While this dependency generally corresponds with the Execution Constraint in the form of detecting the condition, there are also several cases where the software annotates a Constituting Property (P) or an Object as an Execution Constraint. For example in a part of a statement from the EU climate regulation dataset:

*Manual: **Bdir**(recommendations) to that **Bind**(Member State)
Automated: **Bdir**(recommendations) **Cex**(to that Member State)*

In the example given above the automated annotation differs from the manual in that the Indirect Object(Bind) is annotated as part of an Execution Constraint (Cex). Another example of the same behaviour can be seen below:

*with the **Bdir**(State of the Energy Union report) **Bdir,p**(prepared in the respective calendar year in accordance with Article 35 of Regulation (EU) 2018/1999), to the **Bind**(European Parliament) and to the **Bind**(Council).*

***Cex**(with the State of the Energy Union report prepared in the respective calendar year in accordance with Article 35 of Regulation (EU) 2018/1999), **Cex**(to the European Parliament [**AND**] to the Council).*

Here the first example shows the manual annotation, which annotates the statement text as a combination of Objects, while the automated annotation annotates the text as Execution Constraints.

Further, another challenge is that the components Activation Condition and Execution Constraint have a lot of overlap in detection using dependency parsing. Currently, there is one primary mechanism for the detection of Activation Conditions, which also overlaps with Execution Constraints, which is the adverbial clause modifier (advcl) dependency. The adverbial clause dependency is described on the Universal Dependencies website as "a clause which modifies a verb or other predicate (adjective, etc.), as a modifier not as a core complement. This includes things such as a temporal clause, consequence, conditional clause, purpose clause, etc." [61]. This aligns perfectly with conditions of the IG, however, there is some overlap in this dependency, where the condition in some cases suggests an Activation Condition, and in other cases suggests an Execution Constraint. To detect when an advcl dependency is connected to an Execution Constraint component the obl dependency is looked for within the scope of the detected component, if

two obl dependencies or more are detected, then it is annotated as an Execution Constraint instead of as an Activation Condition. Using this mechanism has proven quite reliable, however there are cases where this is inadequate. For example in the CyberSecurityAct17 statement below:

*A(ENISA) D(should) I(support) Bdir(Member States [AND] Union institutions)
Cex(in identifying emerging cybersecurity risks and preventing incidents)*

In this statement, the manual annotation can be seen above. The only difference between the manual and automated annotation is the last component, where the automated solution annotates it as an Activation Condition (Cac) instead of the correct Execution Constraint (Cex). To mitigate such cases in the future the use of LLMs may be fruitful. By prompting a LLM with information on the difference between the two component types and asking which is more likely the performance may be improved. Although, this will require testing to verify, and evaluate the solution's efficacy. As the models used for the various NLP methods used in the software are in active development the detection and distinction of the various components may also improve in the future through more accurate underlying data from these models, or through the development of new models. These methods for improvement of the solution also apply to the other overlaps, such as with Object component overlaps like previously discussed.

Aim handling

The Aim (I) component shows the action of the statement. Detecting the Aim of a statement at the most basic level can be performed through using the root of the dependency parse tree, where the root is generally a verb, and corresponds with the Aim of the statement. However, in some cases, the Aim consists of several words or is nested within another component which adds complexity to the requirements of an annotation solution. In several cases in the datasets, there are multi-word Aim components that are annotated as an Aim and a Direct Object by the automation solution, while the manually annotated version encapsulates both into the Aim component. This can be seen in the two examples below with the manually annotated version shown first, then the version annotated through automation:

I(receive payments) | I(receive) Bdir(payments [...])

I(make use) | I(make) Bdir(use)

Another issue is in the tense of the Aim. Where in several of the manual annotations in the dataset the tense of the verb in the Aim component is changed. For example, in one statement the base text is:

"Those activities are to be carried out"

In this case, the manually annotated Aim component is "I([carry out])". To handle such cases it may be possible to use the universal features (uFeats) from the Stanza POS-tagger to change the verb tense. In the case of "carried out" the word "carried" has the lemma "carry" and the uFeats includes "Tense=Past" showing that the word is in the past tense.

However, changing the tense of the verb can alter the relationship between Attributes and Objects in some cases so testing this is crucial. Potentially integration with an LLM can be used to verify the tense change in such cases. For the other scoping issue mentioned earlier, this may be preventable through a more clever matching function, however in that case more data is necessary to base the new rules on to prevent overfitting and false positives.

Property handling

Another challenge for the solution lies in handling component properties. Say the two words "competent" and "authority". Depending on the context the first word may be deemed an Attribute and the the words would then be annotated:

$$A,p(\textit{Competent}) A(\textit{authorities})$$

This is the case for the NisDirective 11 statement. However, if this is the only reference of a (second word) in the statement, then it may be more accurate to annotate the whole phrase as a single Attribute component:

$$A(\textit{competent authority})$$

This is the case for NisDirective5 where a single competent authority is annotated as an Attribute.

This distinction between the main component and its properties is mostly based on contextual information and is therefore challenging to automate. For now, the distinction between a property of a component and the primary component is mainly handled by looking at relevant dependencies and checking the length of the potential property. If the potential property consists of a single word, then it is included in the component it is connected to, if it is longer than a single word, then it is annotated separately as a Property.

Another relevant example of logical operators and the challenge in handling them can be found at the start of the JointCyberUnit7 statement:

"Member States and relevant EU institutions, bodies and agencies"

In this statement, there is a list of Attribute components logically linked with the logical "AND". In this case, the automated annotation gives the result:

$$A(\textit{Member States [AND] relevant EU institutions [AND] bodies [AND] agencies})$$

However, the "relevant" is annotated in the manually annotated version as a Property. This necessitates splitting "Member States" and the other Attributes, giving the annotation:

*A(Member States) A1,p(relevant) A1(EU institutions), A1(bodies) and
A1(agencies)*

It is worth noting that all the Attributes are logical "AND" linked in the second example as well, and all the "A1" attributes could be contained in a single attribute "block" using the "[AND]". The following annotation is logically equivalent to the previous:

*A(Member States) A1,p(relevant) A1(EU institutions [AND] bodies [AND]
agencies)*

In this statement, the automated annotation is inaccurate as the Attribute Property is not handled properly. This is a limitation in both logical operator handling and Property handling. To fix this inaccuracy improvements in Property handling will need to be made to consistently distinguish Properties and their connected Components, and the logical operator handling will need to take such "internal" Property occurrences into account. To improve the detection of properties an accurate NER model may be helpful to detect whether the property is actually a property or part of the entity itself, otherwise looking at the wider statement may give insight to solve the issue or an LLM could be prompted for verification.

Nesting and inference

Several components of the IG 2.0 support nesting. A basic requirement for a component with nesting is the presence of an Attribute (A) and an Aim (I) component internally for regulative statements, or a Constituted Entity (E) and a Constitutive Function (F) for constitutive statements. The IG2NLP software uses a simple mechanism to detect nesting within components and annotate such components accurately.

The basic flow of nested component handling can be seen below:

1. Component detection.
2. Component scoping.
3. Matching function runs on the component contents.
4. Check for component pair of Attribute (A) and Aim (I), or Constituted Entity (E) and Constitutive Function (F).

For a visualization of the nested component handling please refer to the graphic in section 6.3 on page 65. After this workflow, if the component pair is found the component is annotated with curly brackets and the result of the matching function on the internal text is kept. If the component pair is not found the component is annotated with parentheses and the internal text is kept.

This method works for several basic cases of nested structures in statements, however, the solution is limited in a few ways. Firstly, the components where nesting is supported in IG2NLP are limited to a subset of the full set of components with nesting support. This subset includes Activation Conditions (Cac), Execution Constraints (Cex), Direct Objects (Bdir), and the Or Else (O). Further, there are statements where the nested structures of a statement, such as an Activation Condition are constitutive, while the main statement is regulative, or the opposite.

Take the first example from the IG Parser as an example:

***Cac**{Once **E**(policy) **F**(comes into force)} **A,p**(relevant) **A**(regulators) **D**(must)
I(monitor [**AND**] enforce) **Bdir**(compliance).*

Here the top-level statement is regulative, while the first Activation Condition contains a nested constitutive statement. Such cases would require a method for detecting whether a statement is regulative or constitutive to handle. Such a method is currently not developed, and these cases would then incorrectly be annotated as regulative or constitutive throughout.

Finally, there is the challenge of component inference for nested components. In several cases throughout the datasets, there are nested structures where the Attribute and Aim components in a nested component structure are inferred from context, i.e.:

Where the Commission finds, after due consideration of the collective progress assessed in accordance with Article 6(1) [...]

***Cac**{Where the **A**(Commission) **I**(finds), **Cac**{**A**(Commission) **I**([considers]) after due consideration of the **Bdir**(collective progress) **Bdir,p**(assessed in accordance with Article 6(1))} [...]*

This example shows that the Attribute is repeated within the second Activation Condition (Cac) component through inference, and the Aim (I) is retrieved from within the component text.

Handling such cases is no trivial matter, as the inclusion of external components to make a structure a valid nested structure can cause false positives. Where the absence of an Attribute (A) and an Aim (I) or a Constituted Entity (E) and a Constitutive Function (F) is incorrectly circumvented through incorrect inference. The challenge would then lie in finding a consistent factor that indicates that a component should be nested with component inference.

Another aspect to consider is the training data used in the development of IG2NLP, as the data had a limited amount of certain structures such as no occurrences of the Or Else (O) component, Attribute Properties (A,p) with nesting, Constituted Entity Properties (E,p) with nesting, etc. Developing rules for annotating such components as nested would introduce the risk of false positives and inaccurate annotations. Therefore, the components with support for nested annotations in the solution were limited to a subset, while the Or Else (O) component was included in this subset. This is because the Or Else (O) is always nested, and there is therefore a smaller risk of incorrect annotation.

All in all, there are several challenges in nested statement structure handling. Distinguishing between regulative and constitutive statements would help in cases where nested structures are not of the same type as the base statement. For inference, a method of consistently detecting where inference is both possible, and accurate would need to be developed, and finally, the list of supported components for nesting can be broadened in the future if enough relevant data can be collected to base the annotations on.

Distinguishing Constituted Entities (E) and Constituting Properties (P) and Attributes (A) and Direct Objects (Bdir)

One problem in constitutive statements is handling Constituted Entities and Constituting Properties. There are several cases where there are overlapping dependencies and structures between the two in the datasets. For example, in Article 1.1 the start of the statement is:

"This Regulation establishes a framework for the [...]"

Looking at the dependency parse tree for this statement (figure 7.2):

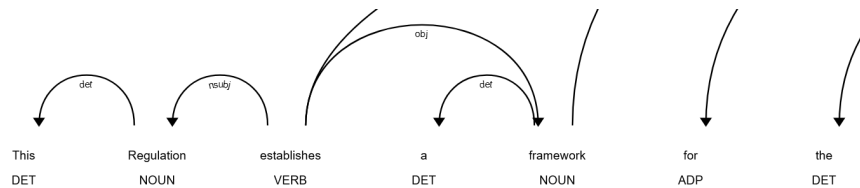


Figure 7.2: Article 1.1 Component conflict example

In the figure above the VERB "establishes" is the root of the dependency parse tree. Using the basic matching rules this should be annotated as the Constituted Function (F) component, a Nominal Subject (nsubj) dependency to the root of a sentence is handled as a Constituted Entity (E) component, and the Object (obj) dependency is treated as a Constituting Properties component (P). If annotated using the regular rules of the matching function this statement would then be annotated as:

"This E(Regulation) F(establishes) a P(framework) Bp(for the[...])"

However, the manually annotated statement has the Constituted Entity and the Constituting Property swapped around:

"This P(Regulation) F(establishes) a E(framework) E,p(for the[...])"

In this example the matching function is unable to detect this edge case, and as such the annotation is wrong. Further this can be seen in the subsequent statements *Article 1.2* and *Article 1.3* respectively:

"This Regulation sets out a [...]"

"This Regulation applies to [...]"

In the case above both statements start with the same format of a nsubj dependency connected to the root of the dependency parse tree. They also both have the word Regulation, however in the manually annotated version of these statements the first "Regulation" is annotated as a Constituting Properties (P), and the

second as a Constituted Entity (E) component. For manual review the difference in annotation can be detected by looking at the wider context within the statement, however, this is challenging to automate. This example also displays a weakness of simple matching-based rules, where both reliance on dependency parse tree data, and dictionary-based approaches would result in similar errors.

In the future, such cases may be handled better using NER models for example, or other methods to try to extract more information to base the annotations. For regulative statements, the same problem occurs with the Attribute (A) and Direct Object (Bdir) components. This was more of an issue in the validation dataset statements that consisted of phrases. In this case, a method for recognition of the animacy of entities (Attributes are generally animate objects) may help distinguish such edge cases, or potentially an NER model could solve this issue.

Chapter 8

Conclusion and Discussion

This thesis presents work using NLP techniques to automate the annotation of institutional statements of the IG using the IG Script notation. The purpose of the thesis is to answer the following research questions (with the first question being augmented with additional subquestions):

1. To what degree can NLP tools be used to automate the annotation process of institutional statements to the IG Script notation?
 - 1.1. Which NLP techniques could be used to help automate parts of the IG annotation process?
 - 1.2. What are the current trends in the relevant papers on unstructured text processing with NLP?
2. How accurate is automated parsing of text into IG Script notation using the solution?

The first chapter introduced the thesis, the underlying motivation and research questions, as well as showcasing an outline of the rest of the thesis. This was followed by a second chapter introducing the IG, the types of institutional statements and the IG Script notation.

The first research question was answered in parts throughout the thesis. The sub-questions to this research question were primarily answered in the related work (section 2.3) and literature review (chapter 3), and are reflected in the following. A promising avenue highlighted in earlier work was the automated annotation of the earlier IG 1.0 that explored the potential of dependency parsing and mentions of NER for the extraction of institutional statement information. The literature review went into detail on NLP and introduced research and developments of NLP in several sectors. This included relevant techniques such as NER, dependency-parsing, POS-tagging, coreference resolution, relation extraction and various other techniques. These techniques were then discussed in relation to their potential for automation of the annotation process. Through this literature review, a more general overview of potential NLP techniques was presented.

Following the literature review is the NLP chapter which narrowed down the relevant NLP techniques (chapter 4), presented the Stanza toolkit which was used

and presented a discussion on LLMs and their potential for the annotation task.

After the toolkit and relevant techniques were presented the fifth chapter introduced the pilot projects performed leading up to this thesis (chapter 5). This chapter introduced two pilot projects, one using dependency parsing and post-tagging to automate the annotation process and the other introducing an updated user interface for the IG Parser annotation tool. Through the development of these prototypes, the use of dependency parsing and POS-tagging for automated annotation was tested. The results of this testing achieved good results for the Attribute, Deontic and Aim components: However, there were several limitations in the scoping of components, the amount of components covered by the solution and the handling of nesting.

The prototypes were iteratively developed as part of the thesis, incrementally exploring additional NLP techniques to be embedded in the IG2NLP software (chapter 6). This development included the introduction of further NLP techniques to base the annotations on, the development of an API and a subsequent prototype integrating the API into the IG Parser, and extra tooling for visualization, caching and testing of the solution.

Through the review of the related work and exploration using an initial prototype, the potential of dependency parsing and POS-tagging for automated annotation was supported. This addressed part of the first sub-research question "Which NLP techniques could be used to help automate parts of the IG annotation process?". This research question was further answered through the literature review introducing further potentially relevant techniques, followed by the NLP chapter and development chapters narrowing down the relevant techniques. In the end, the techniques used were dependency parsing, POS-tags, uFeats, NER and coreference resolution. In the future the introduction of LLMs and some form of RE could also have potential. The second sub-question: "What are the current trends in the relevant papers on unstructured text processing with NLP?", was answered in the literature review. Where the most prominent techniques were RE and NER, many papers used the BERT model for tasks ranging from NER to causality detection and more. Further, there was a mix of machine learning based methods and manual mapping approaches. The field of NLP is active and spans many different domains, which in the review included the medical domain, cybersecurity, the judicial domain and others.

Looking at the first research question "To what degree can NLP tools be used to automate the annotation process of institutional statements to the IG Script notation?", this was partly answered through the sub-questions as outlined above. Further, the development section expanded on this with new NLP techniques and through supporting additional features of the IG. This updated feature set included handling of both constitutive and regulative statements, support for all the components of the IG Script notation, basic semantic annotations and extended nested statement handling. Through this development, the potential for automation of the annotation of institutional statements is exemplified. This shows that there is potential for supporting most of the annotation process. However, there

are some limitations to what is currently handled, this includes automating Attribute and Aim or Constituted Entity and Constitutive Function inference (section 7.2.5), handling more cases of nested statements, handling hybrid statements consisting of both constitutive and regulative sections, handling of more semantic annotations and certain statement structures.

These limitations are presented and discussed more in relation to the second literature question: "How accurate is automated parsing of text into IG Script notation using the solution?". Through the testing and evaluation chapter (chapter 7) the limitations of the software and the accuracy of the automated annotation are presented and discussed in detail. This also includes potential ways to address the shortcomings of the solution and other future work. The testing compared the automated annotations with a manual control set through four key metrics. First, the true positive rate which contained all components that were annotated accurately by the automated solution both in component type and in component content. The second category of partial positives contained components that were partially correct, either in scope, component type, or nesting. This would for example contain components where the automated solution annotated both the main component and its properties under the same component, while the manual annotation differentiated between the two. Finally, the third and fourth metrics were false positives and false negatives, which were the components not picked up by the true or partial positive metrics.

The automated parsing test results showed good accuracy on regulative statements for the Aim, Attribute and Activation condition components in the development data set. Where these components all had a true positive rate of 80% or above. Further, when including the partial positive rate the Aim, Direct Object and Execution Constraint also had detection rates at or above 80%. On the validation data set the results were quite a bit lower for the Aim, Attribute and Activation Condition components. This is due to a few reasons. First, the Activation conditions had more partial positives due to cases where the component was falsely detected as Execution constraints, signalling a need for a better method to differentiate between the two. The other main reason is due to a subset of these statements which the IG2NLP software struggled with. This was due to very long Activation condition components spanning multiple sentences or with internal semicolons, which the solution annotated as separate statements or was unable to scope properly. The second subset consisted of smaller highly edited statements where the Attribute was simply placed at the start of the statement. For these "phrase" statements the Attribute component was not properly integrated into the statement text, which caused issues in the dependency parsing based matching function used. When excluding these statement types mentioned the results on the two datasets were more comparable.

In addition to regulative statements, constitutive statements were also tested. For the constitutive statements, the performance was worse than for regulative statements, which is caused primarily by two factors. Firstly difficulty in differentiating between component types using dependency parsing, and second in the

amount of data available in the development phase of the solution. Due to a low amount of data to base the rules on the accuracy was also lower. The accuracy for Constituted Entities for example had a true positive rate of 50% in the development dataset, and 25% in the validation dataset. However, the Modal component detection rate was high with a True Positive rate of 100% in the development dataset, and 76.92% in the validation dataset. It is also worth noting that similar limitations to those described above in the regulative statements of the validation dataset also affected the constitutive statements.

Finally, looking at nested components in the development dataset seven out of twelve nested components were detected, and a single false positive was detected. In the validation dataset, two out of seven nested components were detected. Further, when introducing nesting the accuracy of the component detection is reduced because of the components nested within other components. This nesting adds another layer of complexity as each nested statement has its own components. For each case of nesting which is not detected all the internal components are then counted as false negatives, and for all false positive cases of nesting the internal components are counted as false positives. The results show that there are improvements to be made in the detection of nested components. This is primarily a limitation of inference, meaning for nested components where the Attribute and/or Aim, or the Constituted Entity and/or Constitutive Function are inferred from outside the direct component text. This inference process is currently not automated and is the primary cause of the false negatives. However, the introduction of automated inference has the potential of introducing additional false positives.

Some key takeaways from the exploration can be summed up as follows:

1. The methods used provide a good baseline to build on with further work and the introduction of additional techniques.
2. A limitation is the amount of data used to develop the rules of the matching function. With more data additional edge cases could be discovered, and additional patterns could emerge.
3. Constitutive statements need further work to increase annotation accuracy.
4. To further increase the detection rate of nested components a method for handling component inference is necessary.
5. A method for detecting whether a statement, or part of a statement is regulative or constitutive is necessary for hybrid statements.

Future work would include finding ways to address the limitations of the solution and to build further on it. Firstly, by obtaining more quality data additional rules could be created and more patterns could emerge. Secondly, the introduction of a dictionary based approach to distinguish between or detect certain component types could have merit as a solution. Thirdly, as mentioned above a method needs to be developed to distinguish between constitutive and regulative components, this could be through looking at keywords for deontics and modals, looking at the animacy of the entity(ies) in the statement, or through the use of an LLM. Fourth

the handling of component inference is necessary to increase nested component detection rates. Fifth, the introduction of further NLP techniques can be tested to see whether they can help increase the performance of the solution, this could for example be a form of RE or other relevant techniques. Finally, testing different models could be beneficial to see whether the performance can be increased through newly developed models, or domain-specific models for example.

8.1 Conclusion

The focus of this thesis was to test the potential of using NLP techniques for supporting the annotation of institutional statements of the IG 2.0 with the IG Script notation through automation. This goal was met by researching NLP techniques and through the development of IG2NLP, an automated annotation software for institutional statements. The annotations are based on a manual mapping of the output of NLP techniques to components of the IG using a custom "matching function". The NLP techniques used are NER, dependency parsing, POS-tagging, uFeats and coreference resolution. This thesis shows the potential of various NLP techniques for this task of automated annotation, and is supported by comparing the automated annotations with a manual control using a developed testing methodology.

The results show good performance for regulative statements, lower performance for constitutive statements and handling of nesting. However, these limitations are known and there are several potential ways to address them.

Through testing on the development dataset and a second validation dataset, certain limitations to the solution were discovered. These include limitations in the underlying methodology of basing the matching on dependency parsing and handling of certain statement structures. Another limitation is the size of the development dataset, where with more data further patterns could be discovered and handled. Further, the scoping of components could be improved to increase the true positive rate, and additional methods to distinguish between component types could help reduce false positives and partial positives. The detection of nested components could potentially be improved by developing a method for component inference of the Attribute and/or Aim or the Constituted Entity and/or Constitutive Function components. These components are often inferred from context in nested statements, and are not always directly part of the component contents originally. Finally, a method for distinguishing between regulative and constitutive statements would help the usefulness of the software and allow for handling hybrid statements containing both regulative and constitutive parts. However, beyond focusing on the results presented as part of this work, the exploratory nature a) provided a baseline for future exploration by drawing on the metrics developed as part of this work, and b) by building on the detailed documentation of the challenges in the encoding in order to arrive at solutions leading to higher accuracy.

As indicated before, future work would be to obtain more data to create and

refine rules for the matching function, the introduction of further NLP techniques, a dictionary based approach for certain component types or potentially the use of LLMs to expand the feature set, and potentially testing other models for the NLP techniques used.

All in all, the results show promise for applying NLP techniques to automate institutional statement annotation. Furthermore, the field of NLP is active, and new and updated methods and models could improve these results further in the future. There are limitations to the approach, but with more data and potentially through introducing further techniques these limitations can potentially be addressed. At the same time, recognizing the emerging role and relevance of LLMs, this thesis can serve as a starting point to provide a testbed for comparative studies that span across different branches of NLP techniques and their application to automate the encoding of institutional language.

Bibliography

- [1] S. E. S. Crawford and E. Ostrom, 'A grammar of institutions,' *The American Political Science Review*, vol. 89, no. 3, pp. 582–600, 1995, ISSN: 00030554, 15375943. DOI: 10.2307/2082975.
- [2] E. Ostrom, *Understanding institutional diversity*, eng, Princeton, N.J, 2005.
- [3] X. Basurto, G. Kingsley, K. McQueen, M. Smith and C. Weible, 'A systematic approach to institutional analysis: Applying Crawford and Ostrom's grammar,' *Political Research Quarterly*, vol. 63, pp. 523–537, Sep. 2010. DOI: 10.1177/1065912909334430.
- [4] S. Siddiki, C. M. Weible, X. Basurto and J. Calanni, 'Dissecting policy designs: An application of the institutional grammar tool,' *Policy Studies Journal*, vol. 39, no. 1, pp. 79–103, 2011. DOI: 10.1111/j.1541-0072.2010.00397.x.
- [5] S. Siddiki, C. M. Weible, X. Basurto and J. Calanni, 'Dissecting policy designs: An application of the institutional grammar tool,' *Policy Studies Journal*, vol. 39, no. 1, p. 86, 2011. DOI: 10.1111/j.1541-0072.2010.00397.x.
- [6] C. Frantz, M. K. Purvis, M. Nowostawski and B. T. R. Savarimuthu, 'Nadico: A nested grammar of institutions,' in *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, G. Boella, E. Elkind, B. T. R. Savarimuthu, F. Dignum and M. K. Purvis, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 429–436, ISBN: 978-3-642-44927-7.
- [7] C. K. Frantz and S. Siddiki, 'Institutional grammar 2.0: A specification for encoding and analyzing institutional design,' *Public Administration*, vol. 99, no. 2, pp. 222–247, 2021. DOI: 10.1111/padm.12719.
- [8] C. K. Frantz and S. N. Siddiki, 'Institutional grammar 2.0 codebook,' *CoRR*, vol. abs/2008.08937, pp. 114–115, 2020. DOI: 10.48550/arXiv.2008.08937.
- [9] C. K. Frantz and S. Saba, *Institutional Grammar - Foundations and Applications for Institutional Analysis*. Palgrave Macmillan, 2022, pp. 256–260.
- [10] C. K. Frantz and S. Saba, *Institutional Grammar - Foundations and Applications for Institutional Analysis*. Palgrave Macmillan, 2022, p. 79.

- [11] M. Vannoni, 'A political economy approach to the grammar of institutions: Theory and methods,' *Policy Studies Journal*, vol. 50, no. 2, pp. 453–471, 2022. DOI: 10.1111/psj.12427.
- [12] M. Vannoni, 'A political economy approach to the grammar of institutions: Theory and methods,' *Policy Studies Journal*, vol. 50, no. 2, p. 469, 2022. DOI: 10.1111/psj.12427.
- [13] D. Rice, S. Siddiki, S. Frey, J. H. Kwon and A. Sawyer, 'Machine coding of policy texts with the institutional grammar,' *Public Administration*, vol. 99, no. 2, pp. 248–262, 2021. DOI: 10.1111/padm.12711.
- [14] S. Huang, L. J. Liang and H. C. Choi, 'How we failed in context: A text-mining approach to understanding hotel service failures,' *Sustainability (Switzerland)*, vol. 14, no. 5, 2022. DOI: 10.3390/su14052675.
- [15] K. A. Ogudo and D. M. J. Nestor, 'Sentiment analysis application and natural language processing for mobile network operators' support on social media,' 2019. DOI: 10.1109/ICABCD.2019.8851052.
- [16] S. Scannapieco and C. Tomazzoli, 'Shoo the spectre of ignorance with qa2spr an open domain question answering architecture with semantic prioritisation of roles,' vol. 1959, 2017. [Online]. Available: <http://ceur-ws.org/Vol-1959/paper-05.pdf>.
- [17] R. Nokhbeh Zaeem, M. Manoharan, Y. Yang and K. S. Barber, 'Modeling and analysis of identity threat behaviors through text mining of identity theft stories,' *Computers and Security*, vol. 65, pp. 50–63, 2017. DOI: 10.1016/j.cose.2016.11.002.
- [18] O. Gurbuz and O. Demirors, 'From organizational guidelines to business process models: Exploratory case for an ontology based methodology,' vol. 1, 2017, pp. 320–329. DOI: 10.1109/CBI.2017.22.
- [19] J. L. Martinez-Rodriguez, I. Lopez-Arevalo and A. B. Rios-Alvarado, 'Openie-based approach for knowledge graph construction from text,' *Expert Systems with Applications*, vol. 113, pp. 339–355, 2018. DOI: 10.1016/j.eswa.2018.07.017.
- [20] M. Štravs and J. Zupančič, 'Named entity recognition using gazetteer of hierarchical entities,' *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11606 LNAI, pp. 768–776, 2019. DOI: 10.1007/978-3-030-22999-3_65.
- [21] A. Iftikhar, S. W. U. Q. Jaffry and M. K. Malik, 'Information mining from criminal judgments of lahore high court,' *IEEE Access*, vol. 7, pp. 59 539–59 547, 2019. DOI: 10.1109/ACCESS.2019.2915352.

- [22] J. J. Nolan, M. Stevens and P. David, 'Understanding and exploring competitive technical data from large repositories of unstructured text,' vol. 2327, 2019. [Online]. Available: <https://ceur-ws.org/Vol-2327/IUI19WS-ESIDA-7.pdf>.
- [23] K. Khadilkar, S. Kulkarni and S. Venkatraman, 'A knowledge graph based approach for automatic speech and essay summarization,' 2019. DOI: 10.1109/I2CT45611.2019.9033908.
- [24] G. Husari, E. Al-Shaer, B. Chu and R. F. Rahman, 'Learning apt chains from cyber threat intelligence,' 2019. DOI: 10.1145/3314058.3317728.
- [25] P. Chocron, A. Abella and G. De Maeztu, 'Contextmel: Classifying contextual modifiers in clinical text; [contextmel: Un clasificador de modificadores contextuales en texto clínico],' *Procesamiento del Lenguaje Natural*, vol. 65, pp. 45–52, 2020. DOI: 10.26342/2020-65-5.
- [26] H. Liu, Z. Li, D. Sheng, H.-T. Zheng and Y. Shen, 'Multi-entity collaborative relation extraction,' vol. 2021-June, 2021, pp. 7678–7682. DOI: 10.1109/ICASSP39728.2021.9413673.
- [27] N. Boudjellal, H. Zhang, A. Khan, A. Ahmad, R. Naseem, J. Shang and L. Dai, 'Abioner: A bert-based model for arabic biomedical named-entity recognition,' *Complexity*, vol. 2021, 2021. DOI: 10.1155/2021/6633213.
- [28] A. L. Olex and B. T. McInnes, 'Review of temporal reasoning in the clinical domain for timeline extraction: Where we are and where we need to be,' *Journal of Biomedical Informatics*, vol. 118, 2021. DOI: 10.1016/j.jbi.2021.103784.
- [29] J. Santoso, E. I. Setiawan, C. N. Purwanto, E. M. Yuniarno, M. Hariadi and M. H. Purnomo, 'Named entity recognition for extracting concept in ontology building on indonesian language using end-to-end bidirectional long short term memory,' *Expert Systems with Applications*, vol. 176, 2021. DOI: 10.1016/j.eswa.2021.114856.
- [30] C.-C. Huang, P.-Y. Huang, Y.-R. Kuo, G.-W. Wong, Y.-T. Huang, Y. S. Sun and M. Chang Chen, 'Building cybersecurity ontology for understanding and reasoning adversary tactics and techniques,' 2022, pp. 4266–4274. DOI: 10.1109/BigData55660.2022.10021134.
- [31] P. Zhang, 'A numerical fact extraction method for chinese text,' 2022, pp. 97–103. DOI: 10.1109/SmartCloud55982.2022.00021.
- [32] H. U. Haq, V. Kocaman and D. Talby, 'Deeper clinical document understanding using relation extraction,' vol. 3164, 2022. [Online]. Available: <https://ceur-ws.org/Vol-3164/paper21.pdf>.

- [33] M. Tissera and R. Weerasinghe, 'Grammatical structure oriented automated approach for surface knowledge extraction from open domain unstructured text,' *Journal of Information and Communication Convergence Engineering*, vol. 20, no. 2, pp. 113–124, 2022, Cited by: 2. DOI: 10.6109/jicce.2022.20.2.113.
- [34] K. Lai, J. R. Porter, M. Amodeo, D. Miller, M. Marston and S. Armal, 'A natural language processing approach to understanding context in the extraction and geocoding of historical floods, storms, and adaptation measures,' *Information Processing and Management*, vol. 59, no. 1, 2022. DOI: 10.1016/j.ipm.2021.102735.
- [35] V. Khetan, R. Ramnani, M. Anand, S. Sengupta and A. E. Fano, 'Causal bert: Language models for causality detection between events expressed in text,' *Lecture Notes in Networks and Systems*, vol. 283, pp. 965–980, 2022. DOI: 10.1007/978-3-030-80119-9_64.
- [36] S. Acharya, S. Mandal and R. Basak, 'Solving arithmetic word problems using natural language processing and rule-based classification,' *International Journal of Intelligent Systems and Applications in Engineering*, vol. 10, no. 1, pp. 87–97, 2022. DOI: 10.18201/ijisae.2022.271.
- [37] R. Yan, X. Jiang, W. Wang, D. Dang and Y. Su, 'Materials information extraction via automatically generated corpus,' *Scientific Data*, vol. 9, no. 1, 2022. DOI: 10.1038/s41597-022-01492-2.
- [38] P. Beslin Pajila, K. Sudha, D. Kalai Selvi, V. Nivas Kumar, S. Gayathri and R. Siva Subramanian, 'A survey on natural language processing and its applications,' 2023, pp. 996–1001. DOI: 10.1109/ICESC57686.2023.10193469.
- [39] S. Mehta, G. Jain and S. Mala, 'Natural language processing approach and geospatial clustering to explore the unexplored geotags using media,' 2023, pp. 672–675. DOI: 10.1109/Confluence56041.2023.10048848.
- [40] K. Detroja, C. Bhensdadia and B. S. Bhatt, 'A survey on relation extraction,' *Intelligent Systems with Applications*, vol. 19, 2023. DOI: 10.1016/j.iswa.2023.200244.
- [41] K. Manas and A. Paschke, 'Semantic role assisted natural language rule formalization for intelligent vehicle,' *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 14244 LNCS, pp. 175–189, 2023. DOI: 10.1007/978-3-031-45072-3_13.
- [42] F. Dong, S. Yang, C. Zeng, Y. Zhang and D. Shi, 'Chinese medical named entity recognition based on pre-training model,' in *Green, Pervasive, and Cloud Computing*, H. Jin, Z. Yu, C. Yu, X. Zhou, Z. Lu and X. Song, Eds., Singapore: Springer Nature Singapore, 2024, pp. 139–155, ISBN: 978-981-99-9893-7. DOI: 10.1007/978-981-99-9893-7_11.

- [43] A. K. Sasidharan and R. Rahulnath, 'Structured approach for relation extraction in legal documents,' 2023. DOI: 10.1109/GCAT59970.2023.10353444.
- [44] W. Shi, H. Wang and X. Lou, 'Multi-modal graph reasoning for structured video text extraction,' *Computers and Electrical Engineering*, vol. 107, 2023. DOI: 10.1016/j.compeleceng.2023.108641.
- [45] S. Hochreiter and J. Schmidhuber, 'Long short-term memory,' eng, *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997, ISSN: 0899-7667.
- [46] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, 'Bert: Pre-training of deep bidirectional transformers for language understanding,' 2019. DOI: 10.48550/arXiv.1810.04805.
- [47] N. K. A. H. Rupasinghe and K. Panuwatwanich, 'Understanding construction site safety hazards through open data: Text mining approach,' *ASEAN Engineering Journal*, vol. 11, no. 4, pp. 160–178, 2021. DOI: 10.11113/AEJ.V11.17871.
- [48] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov, *Roberta: A robustly optimized bert pretraining approach*, 2019. DOI: 10.48550/arXiv.1907.11692.
- [49] F. Elia. 'Constituency parsing vs dependency parsing.' (), [Online]. Available: <https://www.baeldung.com/cs/constituency-vs-dependency-parsing> (visited on 01/12/2023).
- [50] L. Màrquez, X. Carreras, K. C. Litkowski and S. Stevenson, 'Semantic Role Labeling: An Introduction to the Special Issue,' *Computational Linguistics*, vol. 34, no. 2, pp. 145–159, Jun. 2008, ISSN: 0891-2017. DOI: 10.1162/coli.2008.34.2.145.
- [51] M. Vannoni, 'A political economy approach to the grammar of institutions: Theory and methods,' *Policy Studies Journal*, vol. 50, no. 2, pp. 453–471, 2022. DOI: 10.1111/psj.12427.
- [52] P. Qi, Y. Zhang, Y. Zhang, J. Bolton and C. D. Manning, *Stanza: A python natural language processing toolkit for many human languages*, 2020. DOI: 10.48550/arXiv.2003.07082.
- [53] V. Dobrovolskii, 'Word-level coreference resolution,' in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, M.-F. Moens, X. Huang, L. Specia and S. W.-t. Yih, Eds., Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 7670–7675. DOI: 10.18653/v1/2021.emnlp-main.605.
- [54] K. D'Oosterlinck, S. K. Bitew, B. Papineau, C. Potts, T. Demeester and C. D'velder, 'Caw-coref: Conjunction-aware word-level coreference resolution,' 2023. DOI: 10.48550/arXiv.2310.06165.
- [55] K. D'Oosterlinck, S. K. Bitew, B. Papineau, C. Potts, T. Demeester and C. D'velder, 'Caw-coref: Conjunction-aware word-level coreference resolution,' 2023. DOI: 10.48550/arXiv.2310.06165.

- [56] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes and A. Mian, 'A comprehensive overview of large language models,' 2024. DOI: 10.48550/arXiv.2307.06435.
- [57] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes and A. Mian, 'A comprehensive overview of large language models,' p. 31, 2024. DOI: 10.48550/arXiv.2307.06435.
- [58] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes and A. Mian, 'A comprehensive overview of large language models,' pp. 33–34, 2024. DOI: 10.48550/arXiv.2307.06435.
- [59] M. Benaida, 'Developing and extending usability heuristics evaluation for user interface design via ahp,' *Soft Computing*, 2023. DOI: 10.1007/s00500-022-07803-4.
- [60] U. D. contributors. 'Obl: Oblique nominal.' (), [Online]. Available: <https://universaldependencies.org/u/dep/obl.html> (visited on 29/04/2024).
- [61] U. D. contributors. 'Advcl: Adverbial clause modifier.' (), [Online]. Available: <https://universaldependencies.org/docs/en/dep/advcl.html> (visited on 24/04/2024).



 **NTNU**

Norwegian University of
Science and Technology