

Johanne Bognøy

IG Coder: Enabling Visual Coding of Institutional Statements

Master's thesis in Applied Computer Science

Supervisor: Christopher Frantz

June 2021



Norwegian University of
Science and Technology

Johanne Bognøy

IG Coder: Enabling Visual Coding of Institutional Statements

Master's thesis in Applied Computer Science
Supervisor: Christopher Frantz
June 2021

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





Norwegian University of
Science and Technology

IG Coder: Enabling Visual Coding of Institutional Statements

Johanne Bognøy

01-06-2021

Master's Thesis

Master of Science in Applied Computer Science

30 ECTS

Department of Computer Science

Norwegian University of Science and Technology,

Supervisor: Associate Professor Christopher Frantz

Abstract

This thesis presents IG Coder, a web application for the interactive encoding of statements in the Institutional Grammar (IG) syntax.

Institutions are behavioral directives such as those found in policies and regulations. The Institutional Grammar is a device for analyzing institutions by decomposing them into their base components. The process of decomposing institutions is commonly referred to as encoding.

However, no software applications currently exist that facilitate the encoding process in a way suited to the Institutional Grammar and that produce output usable for analysis. Two general-purpose applications have been adapted for encoding, both of which have considerable shortcomings. A new encoding tool could be designed specifically for the needs of the Institutional Grammar, and this thesis does exactly that.

To investigate how the new tool should be designed, I conduct a study of the current encoding tools. This results in insights about the strengths and weaknesses of the current tools, as well as about what is needed in an encoding tool in general.

The IG Coder prototype is built on a new way of regarding institutional statements. It represents statements as trees, and this representation is visualized in an interactive, color coded tree graphic which serves as the basis of the coding interface. This is significant because institutional statements, like sentences in natural language, are complex and often have a hierarchical structure. The current encoding tools fail to visualize coded statements in such a way.

I evaluate the completed prototype via user testing and interviews with the participants, which yield in-depth insights about the prototype on several levels. This serves to guide its future development as well as show that there is interest in this tool in the IG research community. With IG Coder I have produced a tangible starting point for a brand new encoding tool for the Institutional Grammar.

Sammendrag

Denne oppgaven presenterer IG Coder, en webapplikasjon for interaktiv koding av institusjonelle setninger i Institutional Grammar-syntaksen.

Institusjoner er direktiver for oppførsel, slik som de som finnes i lovverk og regelverk. Institutional Grammar er et verktøy for analyse av institusjoner ved å oppdele dem i deres grunnleggende komponenter. Denne oppdelingsprosessen er ofte kalt koding.

Problemet er at det i dag ikke finnes noe programvare som gjør kodeprosessen enkel på en måte som er tilpasset Institutional Grammar og som produserer analyserbar utdata. I dag er to allsidige applikasjoner gjenbrukt til koding, og begge har store mangler for dette formålet. Ett nytt kodeverktøy kan designes spesielt med tanke på Institutional Grammar, og det er nøyaktig det denne oppgaven gjør.

For å undersøke hvordan dette verktøyet burde designes, gjennomfører jeg en studie av de to kodeverktøyene som brukes nå til dags. Dette resulterer i en forståelse av styrkene og svakhetene ved disse to verktøyene, samt av hva som egentlig trengs i et kodeverktøy.

Prototypen IG Coder bygger på en ny måte å se institusjonelle setninger. Den representerer setninger som trær, og denne representasjonen visualiseres i form av en interaktiv, fargekodet tregrafikk som fungerer som grunnlaget til kodegrensesnittet. Dette er betydningsfullt ettersom institusjonelle setninger, akkurat som setninger på naturlig språk, er komplekse og ofte har en hierarkisk struktur. De nåværende kodeverktøyene er ikke i stand til å visualisere kodede setninger på en slik måte.

Jeg evaluerer den ferdige prototypen ved hjelp av brukertesting og intervjuer med deltagerne, noe som gir innsikt om prototypen på flere nivåer. Dette tjener til å veilede dens fremtidige utvikling samt vise at det er interesse i dette verktøyet innen IG-forskningsmiljøet. Med IG Coder har jeg produsert et håndgripelig utgangspunkt for et helt nytt kodeverktøy for Institutional Grammar.

Acknowledgements

I would like to thank my supervisor, Christopher Frantz, for introducing me to the Institutional Grammar, guiding and advising me throughout this project, and helping me realize our idea of making the Institutional Grammar accessible to computing. We have had a great cooperation over these past two years.

I would like to thank the three participants who were kind enough to test my prototype and participate in interviews:

Angelo Baldado
Dr. Ute Brady
Dr. Bartosz Pielniński

Last but not least, I would like to thank my mother for her everlasting support and love.

Johanne Bognøy

Contents

Abstract	iii
Sammendrag	v
Acknowledgements	vii
Contents	ix
Figures	xi
Tables	xiii
Code Listings	xv
1 Introduction	1
1.1 Research Questions	2
1.2 Outline	3
2 Research Methods	5
2.1 Phase 1: Design	5
2.2 Phase 2: Development	6
2.3 Phase 3: Evaluation	7
3 Background	9
3.1 Content Analysis and Policy Coding	9
3.2 Prominent Coding Schemes	10
3.3 The Institutional Grammar	11
3.3.1 Regulative Statements	12
3.3.2 Constitutive Statements	13
3.3.3 Mapping and Order of Components	14
3.3.4 Nesting	15
3.3.5 IG Extended Features	16
3.3.6 IG Logico Features	17
3.4 Literature Review	18
3.5 Current Tools	21
3.5.1 Spreadsheets	21
3.5.2 Text Annotation Tools	24
3.5.3 Inline Coding	27
3.5.4 Automated Approaches	28
4 Review of Current Tools	29
4.1 Introduction	29
4.2 Method	30
4.3 Results	33

4.4	Discussion	42
5	Development of IG Coder	45
5.1	Initial State	45
5.2	Technical Design	46
5.3	Requirements	48
5.4	Development Process	52
5.5	User Interface Design	53
5.6	Implementation and Tests	57
5.6.1	Tools	57
5.6.2	Data Model	58
5.6.3	Tests	62
5.7	Deployment	63
6	Evaluation of IG Coder	65
6.1	Introduction	65
6.2	Method	65
6.2.1	Recruitment	66
6.2.2	User Testing	66
6.2.3	Interviews	67
6.3	Results	68
6.3.1	Interview 1: Angelo Baldado	68
6.3.2	Interview 2: Dr. Ute Brady	69
6.3.3	Interview 3: Dr. Bartosz Pielński	71
6.3.4	Overall Findings	73
7	Discussion	77
8	Conclusion	81
8.1	Summary	81
8.2	Limitations	82
8.3	Future Work	82
	Bibliography	83
A	Excel Questionnaire	87
B	Early Interview Questions	91
C	INCEpTION+Excel Questionnaire	93
D	IG Data Model	97
E	Test Statements	107
F	Interview Guide	109
G	IG Coder Public Repository	111

Figures

3.1	Spreadsheet template for IG 2.0 with example	23
3.2	INCEpTION user interface with examples	26
4.1	Questionnaire A: Level of experience with Microsoft Excel	33
4.2	Questionnaire B: Level of experience with Microsoft Excel	34
4.3	Questionnaire A: Microsoft Excel's suitability as a coding tool	34
4.4	Questionnaire B: Microsoft Excel's suitability as a coding tool	35
4.5	Questionnaire B: Level of experience with INCEpTION	35
4.6	Questionnaire B: INCEpTION's suitability as a coding tool	35
5.1	System architecture of IG Coder	47
5.2	IG Coder: List of test statements	54
5.3	IG Coder: Uncoded entry	54
5.4	IG Coder: After creating a root node	55
5.5	IG Coder: Statement editor	55
5.6	IG Coder: Component editor	56
5.7	IG Coder: Junction editor	56
5.8	IG Coder: Fully coded statement	57
5.9	Sample tree representation of a statement with the IG data model	61

Tables

3.1	Comparison of the TEI, Akoma Ntoso and the IG	12
3.2	Mapping of regulative and constitutive components	14
3.3	Selected papers on the IG	19
4.1	Table format for structuring the raw data	32
4.2	Identified advantages of Microsoft Excel as a coding tool	36
4.3	Identified advantages of INCEpTION as a coding tool	37
4.4	Identified disadvantages of Microsoft Excel as a coding tool	38
4.5	Identified disadvantages of INCEpTION as a coding tool	39
4.6	Identified needs in a coding tool for the IG (1/2)	40
4.6	Identified needs in a coding tool for the IG (2/2)	41

Code Listings

5.1	Interface for the elemental node in the IG data model	59
-----	---	----

Chapter 1

Introduction

Institutions are a fundamental object of analysis within the area of policy studies. The term encompasses behavioral directives ranging from social norms to public policies, which describe expectations for behavior under given circumstances and sanctions associated with given behaviors. In the Institutional Grammar (IG), individual such directives are called institutional statements.

The Institutional Grammar is a syntax for decomposing, or encoding, institutional statements. It defines a flexible syntactical structure that allows statements to be broken down to their base components, not unlike linguistic grammars. This in turn allows for the systematic analysis of policy structure and meaning. Most institutional statements prescribe expected actions for actors within given contexts and under given circumstances, often conveying either an obligation, prohibition or permission. The IG also defines another type of statement that describes the composition of institutional systems such as boards and committees.

The practice of encoding institutional statements using the IG is commonly referred to as policy coding. Researchers perform policy coding for various ends, though a common use case is the extraction and statistical treatment of syntactical components. The IG offers a native syntax that can be used to annotate statements inline but this method of coding does not lend itself to data extraction and analysis. To gain analytical capabilities, researchers have taken to using spreadsheets like Microsoft Excel as the primary policy coding tool. A spreadsheet template has been developed that separates IG components into columns, for one statement to be coded per row. This tool has served researchers well enough for over a decade but has notable limitations, most glaringly its incompatibility with complex statements, i.e., statements that contain inner statements or logical combinations.

One other tool has notably been used for policy coding, namely the text annotator INCEpTION. An altogether different approach to coding, this tool accommodates annotation of text selections in a source text. It allows for overlapping annotations, unlike inline coding in a text editor. INCEpTION's main limitations are the lack of visual overview when reviewing a coding and the reliance on exporting the coded data in order to perform analysis.

This brings us to the problem: current policy coding tools are inadequate. As

general-purpose applications, both spreadsheets and INCEpTION depend on customization to be used for policy coding, and their underlying data structures are unsuited to the domain-specific and graph-based nature of the IG. It is an institutional grammar, not a linguistic one. On top of this, neither has support for validation of correctness.

On account of the specific and unique needs of policy coding, researchers would be best served by a brand new coding tool. This thesis will provide exactly that: it will take you through the design, development and evaluation of IG Coder, a web application for policy coding. The thesis is a blend of development and research. IG Coder is relevant because the IG is gaining popularity among researchers and their students. The Institutional Grammar Research Initiative (IGRI) is a collective of researchers around the world engaged in policy analysis and was founded to stimulate the theoretical and methodological advancement of the Institutional Grammar¹. My supervisor, Christopher Frantz, is a member of this initiative. In fall 2020, IGRI researchers held an online course that taught policy coding in Microsoft Excel to students². Furthermore, a paper (Frantz & Siddiki, 2021) published in 2021 formally introduces IG 2.0, a large overhaul of the grammar with numerous syntactical refinements that is expected to see high uptake in the community. This overhaul is also available in an accompanying codebook (Frantz & Siddiki, 2020) which offers detailed reference and coding guidelines.

Important terminology The term *policy coding* is not unique to the IG, but in the remainder of this thesis it is used to mean encoding of institutional statements using the IG. Furthermore, I often abbreviate the term *policy coding tool* to *coding tool* but the two have the same meaning. Section 3.1 will explain these terms further.

Target audience This thesis is aimed at those familiar with or interested in the Institutional Grammar as well as computer scientists in general.

1.1 Research Questions

To cover the design and evaluation parts of this thesis, I pose the following research questions:

State of the art

RQ1 What are the strengths and weaknesses of current coding tools?

RQ1a What features from existing tools should the new coding tool retain?

RQ1b What features from existing tools should the new coding tool discard?

¹<https://institutionalgrammar.org>

²<https://institutionalgrammar.org/teaching-institutional-analysis/>

User needs

RQ2 What features do coders need in a coding tool for the IG?

RQ2a What are the essential features a coding tool for the IG should possess?

RQ2b What are the fringe features a coding tool for the IG should possess?

Evaluation of the new coding tool

RQ3 To what extent does IG Coder satisfy the needs identified in RQ1 and RQ2?

RQ3a To what extent is IG Coder aligned with coders' understanding of institutional statements?

RQ3b How satisfied are users with the coding interface of IG Coder?

RQ3c To what extent can IG Coder improve the coding workflow?

The purpose of RQ1 is to determine the state of the art in policy coding tools and identify ways the new tool can improve upon this state. It does so by finding strengths and weaknesses of the current tools that will guide the design of the new tool. I deem the strengths and weaknesses of current tools to translate directly to the respective sub-questions of RQ1.

RQ2 also seeks to guide the new tool's design but this time by focusing on user needs irrespective of the current tools. Its sub-questions separate essential and fringe system features. By essential features is meant features needed by many users, and fringe features are those suggested by at most two. The purpose of this is to help prioritize the features to implement.

RQ3 evaluates the completed prototype. By "needs identified in RQ1 and RQ2" I mean the features a new coding tool should and should not possess, whether they come from an existing tool or not. RQ3 evaluates the new tool against these needs and assesses its suitability as a policy coding tool, but does not compare it to the current tools. The primary reason for this is the very low number of people experienced with both current tools, of which a higher number would be needed for a reliable comparison.

The methods by which the research questions will be answered are described in Chapter 2.

1.2 Outline

Chapter 2 describes the phases of the research and how the research questions are mapped onto those phases, and gives an overview of the research methods used in each phase along with the reasoning behind them. Chapter 3 contextualizes the area of study, taking you from the high-level area of content analysis and policy coding to the Institutional Grammar itself. It furthermore offers a detailed explanation of the grammar, a brief history of its literature and descriptions of the

current coding tools. We then move on to the design phase in which Chapter 4 conducts a study of the current coding tools and answers RQ1 and RQ2. The development phase is covered by Chapter 5, detailing various aspects of IG Coder's development. In the evaluation phase, Chapter 6 conducts an evaluation of the IG Coder prototype via user testing and interview feedback, answering RQ3. All research questions answered, Chapter 7 discusses what I learned in this project and what it achieved. Finally, Chapter 8 concludes the thesis and suggests directions for future work.

The IG Coder prototype developed in this thesis is a web application, work on which was already started before the thesis began. Section 5.1 describes the state of the application at the outset of this thesis.

Chapter 2

Research Methods

The research project is divided into three phases: design, development and evaluation.

2.1 Phase 1: Design

The design phase is governed by RQ1 and RQ2, and as such investigates a) the state of the art in policy coding tools and b) the needs of coders. The resulting artifact of this phase will be a list of system features that will be considered and prioritized for implementation during the development phase.

The review of current tools conducted in Chapter 4 covers the design phase and answers RQ1 and RQ2. It selects Microsoft Excel and INCEpTION as the two current coding tools, with emphasis on the former since it is the most widely used tool. RQ1's sub-questions are answered for each of these tools.

The review of current tools employs a number of data sources, predominantly qualitative:

1. A questionnaire of students at the end of a coding course¹ asking about their perceived advantages and disadvantages with Microsoft Excel as a policy coding tool as well as their needs in a policy coding tool. This questionnaire employed both open, long-text questions and Likert scale (Robinson, 2014) rating tasks.
2. My notes from the aforementioned course where I noted the students' impressions, challenges and discussions
3. An older interview with my supervisor, where he took the role of a coder, on the policy coding experience in Microsoft Excel
4. A questionnaire of IG researchers. The questions were the same as those in the first questionnaire but also included equivalent questions on INCEpTION. This is the only data source on INCEpTION.

To answer the research questions, I extracted sentiments and ideas from these

¹<https://institutionalgrammar.org/teaching-institutional-analysis/>

samples in six categories: advantages and disadvantages with each of the two coding tools and essential and fringe features needed in a coding tool. I used semantic clustering to group and count similar sentiments. The questionnaire results (sources 1 and 4) are weighted more heavily than sources 2 and 3, because for source 2, the data is potentially biased as the notes were taken by a single person, and for source 3, the data is collected from a single participant as well as being partially outdated as the interview was conducted pre-IG 2.0.

RQ2 is divided into essential and fringe features. I classify identified features based on the number of people suggesting them, again giving particular weight to the questionnaire results. As source 2 lacks information connecting sentiments to participants, each of its results is counted only once.

I chose a combination of data sources to answer RQ1 and RQ2, primarily because a questionnaire alone would perhaps not be reliable enough due to the small population of researchers and students familiar with policy coding in spreadsheets. As of the time of writing, the IGRI has fewer than thirty members². The additional data sources, while limited, have the potential to bring out additional viewpoints because they represent different methods of data collection, i.e., observation and interviews as opposed to fixed-form questioning.

As the research questions pertain to software applications, reliability is also helped somewhat by the diversity of backgrounds in the population. Policy analysis attracts researchers from various disciplines and a minority of IG researchers are computer scientists. To put it bluntly, when asked about what a piece of software should and should not do, knowledge of how software is made might influence one's response.

Of the two current coding tools, Microsoft Excel is represented far more heavily than INCEpTION in the data sources. This reflects the former's popularity as a policy coding tool but also the difficulty of finding reliable data on the latter.

All in all, the limitations in these methods mean their results should not be completely relied upon. At the same time, the methods have the potential to give a general understanding of the advantages and disadvantages with current coding tools as well as needs in a coding tool.

2.2 Phase 2: Development

The development phase is not associated with any research questions. From a research perspective, it simply produces the artifact to be evaluated in Phase 3. This artifact, the IG Coder prototype, is to be a functional interface for policy coding and its development is detailed in Chapter 5. Nevertheless, the prototype is a vital part of the research because without it, Phase 3 is not possible. While the evaluation of the prototype is the primary contribution of this thesis, this evaluation depends on the prototype.

²"IGRI Personnel - Institutional Grammar Research Initiative (IGRI)", Institutional Grammar Research Initiative, <https://institutionalgrammar.org/igri-affiliates/>. Accessed 20 May 2021.

2.3 Phase 3: Evaluation

The evaluation phase is governed by RQ3 and described in Chapter 6. In this phase, the completed prototype is deployed to testers whose task is to code a given set of institutional statements using the tool. This is followed by a round of semi-structured interviews with the testers to gain insight into the suitability of IG Coder as a policy coding tool and answer RQ3.

The testers are selected from members of the IGRI³, including affiliates and interns.

The SAGE Encyclopedia of Qualitative Research Methods (Given, 2008) gives the following definition of semi-structured interviews: “The semi-structured interview is a qualitative data collection strategy in which the researcher asks informants a series of predetermined but open-ended questions.” (Given, 2008, p. 811) I chose this style of interview because of the need to gain a deeper understanding of the testers’ experience with the tool. With semi-structured interviews I have the ability to ask participants for clarification and elaboration where necessary while also having a core set of questions for all participants.

As part of this phase, I prepared an interview guide with questions sorted by topic. RQ3 and its sub-questions were incorporated into this list either directly or indirectly. To answer the research questions, I compare and synthesize the interview responses for each interview question linked to a research question.

The method of investigating RQ3 is qualitative. This means the research questions will be answered in words only, and no scales or other metrics will be used to measure extent. The primary reason for this is the limited number of participants, which is too low to perform reliable statistics on. To be able to participate in this study, participants must be experienced with policy coding in either Microsoft Excel or INCEPTION, resulting in a very small eligible population. Another factor is time constraints limiting the number of interviews I am able to conduct.

All in all, the evaluation of IG Coder is a small-scale study, serving as a first look at how well the IG Coder prototype functions as a policy coding interface and how it should be developed further.

³<https://institutionalgrammar.org/igri-affiliates/>

Chapter 3

Background

In this chapter, Sections 3.5.1 and 3.5.3 are based on the similar sections in my Advanced Project Work¹ report but have been modified.

3.1 Content Analysis and Policy Coding

Stepping away from the Institutional Grammar and looking at the greater context around it, content analysis is a research technique for analyzing qualitative text data (Hsieh & Shannon, 2005). The data source can be any instance of communicative language, and the method can be either qualitative or quantitative. The goal of content analysis is to understand, interpret and make inferences about the content of the text (Elo & Kyngäs, 2008; Hsieh & Shannon, 2005).

Content analysis is typically conceptual, focusing on words, themes and concepts within the text. It is a systematic and objective method of describing and quantifying such concepts (Elo & Kyngäs, 2008).

For conceptual analysis, the process is generally as follows. The researcher begins by deciding upon one or more concepts to examine. The level of analysis, i.e., whether to code on the level of words, phrases, sentences or themes must also be decided, as well as the level of implication, i.e., whether to only allow words that explicitly state the concept or also words that imply it to a set degree.

The text is then coded into categories. This means reducing the text into manageable categories that represent and describe the selected concept(s) (Elo et al., 2014). Each category will hold a set of words, themes or concepts occurring in the text that are deemed to have the same meaning, depending on the level of implication. While the term *category* is used here, the resulting set of categories could also be viewed as a taxonomy or classification scheme for the text.

The next step is to actually code the text according to the categories. Depending on the level of analysis, the text is split into fragments which are then categorized, a process which can be done by hand or with the help of software. If using software, the researcher need only input categories and the coding can be done

¹Course code: IMT4894

automatically, but the resulting coding is very sensitive to how the categories were defined. Coding by hand can be time-consuming but could be the only option if there is no software that suits the researcher’s needs.

Finally, the coding can be analyzed. The researcher makes inferences, identifies trends and patterns and draws conclusions. For instance, he or she might examine the language used in the text to search for bias. For quantitative analysis the results can be examined statistically, such as counting the number of occurrences in each category.

Content analysis uses coding to facilitate analysis of the content of a text. The coding process distils the text down to its core concepts, which the researcher has complete control of. The technique is very flexible; the researcher can code for whatever he or she wishes to investigate.

The term *coding* can mean both the process of creating a classification scheme and of applying the scheme to a text. However, the latter meaning is more aptly expressed by the term *encoding*.

This brings us to a more specific form of content analysis, namely policy coding. As the term implies, policy coding means encoding of policy documents or other legal texts in a predefined syntax. The Institutional Grammar is a specific example of policy coding with a well-developed classification scheme for individual directives. Another example can be found in Lane et al.’s coding framework for social distancing policies during the COVID-19 pandemic (Lane et al., 2020). The coding framework consists of a number of domains, i.e., categories, which are different community arenas such as gyms and movie theaters. It is a classification scheme for a specific type of policy.

Further terminology While the term *policy coding* is not unique to the IG, it is commonly used by the IG community to mean encoding of institutional statements with the IG and this thesis will do the same. Furthermore, in the remainder of this thesis the term *coding* is used to mean *encoding*, i.e., applying an existing classification scheme to a text. By *coder* is meant a person who codes institutional statements with the IG. However, *code* when used as a noun means computer code.

3.2 Prominent Coding Schemes

Before moving on to the Institutional Grammar itself, we will look at two notable text coding schemes and relate them to the IG to help contextualize the grammar. The Text Encoding Initiative (TEI) is an international consortium which maintains the TEI Guidelines (TEI Consortium, 2021), a recommended text markup standard (Cummings, 2013) which originated at a 1987 conference (Cummings, 2013; Vanhoutte, 2004). The Guidelines “apply to texts in any natural language, of any date, in any literary genre or text type, without restriction on form or content. They treat both continuous materials (‘running text’) and discontinuous materials such as dictionaries and linguistic corpora.” (TEI Consortium, 2021, iv. About These Guidelines). The Guidelines are expressed in the XML markup language but

are not restricted to it. Key characteristics of the Guidelines are their broad scope yet in-depth coverage (Cummings, 2013). The current version of the Guidelines is P5 (Wittern et al., 2009), which stands for Proposal 5.

Since the TEI Guidelines are designed for the immense scope of any text written in any natural language, it does not offer domain-specific schemas for fields such as political science. It does, however, offer a module for simple semantic and syntactic annotations of a linguistic nature. As we will see in Section 3.3, institutional statements are sentences in natural language and can thus be analyzed as such. However, this approach completely misses out on the institutional content of statements, which is what the IG was designed to be able to analyze. Therefore, the use cases of the TEI and the IG are mutually exclusive.

More closely related to policy coding is the framework Akoma Ntoso. It originated from the United Nations Department for Economics and Social Affairs (UN/DESA)'s project "Strengthening Parliaments' Information Systems in Africa" in 2004 and 2005 (Barabucci et al., 2010; Vitali & Zeni, 2007). Akoma Ntoso is a set of XML schemas for representing parliamentary, legislative and judiciary documents, developed to enable open access to these materials. Open access means not only physical and online access but making the documents machine readable, opening the door for high value information services. Akoma Ntoso aims to be a Legal XML standard (Palmirani & Vitali, 2011). One of the pillars of Legal XML is to "[provide] a representation of the main structures of legal and legislative documents using a principled approach that provides the best combination of technological excellence and sophisticated juridical competency" (Palmirani & Vitali, 2011, p. 76).

The main difference between Akoma Ntoso and the IG is that the former aims to represent documents themselves while the latter is focused on analysis at a lower level. Whereas Akoma Ntoso supports the complete annotation of a number of different types of documents within the legal domain, the IG concentrates on policies and regulations only. Also, while the IG does provide guidelines for preprocessing such documents, the analysis of institutional statements (i.e., rules) is at its core. There has in fact been work on representing legal rules with XML markup, as exemplified by the Legal Knowledge Interchange Format (LKIF) (Gordon, 2008). LKIF is rooted in artificial intelligence, and the format was created to allow for computer reasoning with legal rules. Thus, the IG is different in that its fundamental goal is institutional analysis.

Table 3.1 gives an overview of the differences between the TEI, Akoma Ntoso and the IG discussed above.

3.3 The Institutional Grammar

This section will describe the as of writing most recent version of the Institutional Grammar, IG 2.0 (Frantz & Siddiki, 2021).

IG 2.0 offers three separate versions, or levels of expressiveness, of its syntax: IG Core, IG Extended and IG Logico. IG Core is the most fundamental version, suited

	TEI	Akoma Ntoso	IG
Scope of texts	Any texts in natural language	Legislative, parliamentary and judiciary documents	Institutional statements extracted from policies and regulations
Language	XML	XML	Specification only

Table 3.1: Comparison of the TEI, Akoma Ntoso and the IG

for coding simpler institutional statements and for analysis with a focus on the individual syntactical components. IG Extended, on the other hand, focuses on more fine-grained coding and capturing the structure of institutional statements more closely. At the highest level of expressiveness, IG Logico is intended to help achieve an understanding of the semantic relationships in and among institutional statements. IG Extended will be described further in Section 3.3.5 and IG Logico in Section 3.3.6.

The IG deals with institutional statements, which are written sentences that express a rule or norm, i.e., an institution. Frantz and Siddiki define institutional statements as follows: “Institutional statements *regulate* actions for actors within the presence or absence of particular constraints, or *constitute* or otherwise parameterize features of systems in which actors interact.” (Frantz & Siddiki, 2021, p. 2). The IG defines two types of institutional statements, regulative and constitutive, both of which are covered by this definition. However, when people think of institutional statements, they typically think of the regulative type, which we will begin with.

3.3.1 Regulative Statements

Regulative statements are behavioral directives, defined in the first half of Frantz and Siddiki’s definition of institutional statements. Following are two examples of regulative statements²:

Certified farmer must submit an organic system plan annually.

The Program Manager shall send a written notification of proposed suspension or revocation of certification to certified organic farmer.

In the IG, institutional components are classified as either mandatory (always present) or optional (may or may not be present) in a statement. Optional components allow for the construction of each of Crawford and Ostrom’s statement types: shared strategies, norms and rules (Crawford & Ostrom, 1995).

The following regulative components exist:

Attributes (A) The actor that carries out the action (i.e., the AIM), who may be an

²Examples of institutional statements in this chapter are taken from the IG 2.0 codebook (Frantz & Siddiki, 2020).

individual or a group. The actor may be described by their attributes, hence the name. This component is always present.

- Deontic (D) An operator that specifies whether the statement conveys an obligation (e.g., "must"), permission (e.g., "may"), prohibition (e.g., "must not") or some other type of prescription. This component may or may not be present.
 - Aim (I) The intent of the actor (i.e., the ATTRIBUTES), whether through an action or an intended outcome. This component is always present.
- Object (B) The recipient of the action carried out in the AIM, which may be animate or inanimate. An object may be *direct* (Bdir) or *indirect* (Bind), where the indirect object is the recipient of the direct object instead of the AIM. Each of the OBJECT components may or may not be present.
- Context (C) Defines the circumstances under which the statement applies or qualifies the action in the statement. It is divided into two: Activation Conditions (Cac) and Execution Constraints (Cex). If there are no explicit Activation Conditions in the statement, the default context clause is "under all conditions". If there are no explicit Execution Constraints in the statement, the default context clause is "no constraints". This component is always present, whether its content is explicit or implicit.
- Or else (O) A sanction for violating the action prescribed by the statement, which is an institutional statement of its own, i.e., a nested statement. This component may or may not be present.

3.3.2 Constitutive Statements

Defined in the latter half of Frantz and Siddiki's definition of institutional statements, constitutive statements describe features of institutional systems. Following are two examples of constitutive statements:

There is hereby established a public Food Security Advisory Board.

Commissioner of Agriculture and Markets shall be the Chairperson the Council.

Like regulative components, constitutive components are either mandatory or optional in a statement. The following constitutive components exist:

- Constituted Entity (E) The entity being constituted or directly affected in the system as specified by the CONSTITUTIVE FUNCTION. This component is always present.
 - Modal (M) An operator that specifies whether the system constitution is necessary, possible or impossible. This component may or may not be present.
- Constitutive Function (F) A verb specifying the role of the CONSTITUTED ENTITY in the system.

- If CONSTITUTING PROPERTIES are present, links the CONSTITUTED ENTITY to those. This component is always present.
- Constituting Properties (P) A physical or abstract object linked to the CONSTITUTED ENTITY by the CONSTITUTIVE FUNCTION. Provides parameters to the CONSTITUTED ENTITY. This component may or may not be present.
- Context (C) Identical to regulative CONTEXT, except it qualifies the CONSTITUTIVE FUNCTION instead of the action. Like its regulative counterpart, this component is always present, whether its content is explicit or implicit.
- Or else (O) The consequence of the CONSTITUTED ENTITY not being constituted or established, a consequence that is existential in kind. This component may or may not be present.

3.3.3 Mapping and Order of Components

There exists a syntactical correspondence between regulative and constitutive components as shown in Table 3.2. This is relevant for polymorphic statements, which are explained in Section 3.3.5.

The mapping implies that regulative and constitutive statements are structurally identical. While this is true on the level presented here, there are lower-level syntactical differences such as the OBJECT being divided into DIRECT OBJECT and INDIRECT OBJECT. The similarities are born of the fact that both types of statements use the same linguistic sentence structure, and as detailed above, there are notable semantic differences between the corresponding components.

Table 3.2: Mapping of regulative and constitutive components

Regulative		Constitutive
Attributes (A)	↔	Constituted Entity (E)
Deontic (D)	↔	Modal (M)
Aim (I)	↔	Constitutive Function (F)
Object (B)	↔	Constituting Properties (P)
Context (C)	↔	Context (C)
Or else (O)	↔	Or else (O)

On another note, readers familiar with the IG may have noted that the order of the regulative components presented here differs from the traditional ABDICO sequence (and similarly for constitutive components). I chose to present them in the order ADIBCO because in natural English, the sentence object (i.e., the regulative OBJECT) almost always takes place after the predicate (i.e., the regulative AIM), and institutional statements are almost always written in natural language. This order allows for easier reading of coded statements.

3.3.4 Nesting

IG 2.0 uses the term *atomic institutional statement* for the elementary form of an institutional statement. This is a regulative or constitutive statement that contains no more than one value for each component and has no inner statements. In practice, however, institutional statements are seldom expressed in atomic form; they might contain multiple actors, actions or objects and there may exist linkages between specific actors, actions and/or objects. Furthermore, specific syntactical components may take the form of a separate institutional statement.

IG 2.0 distinguishes between two forms of nesting: vertical nesting and horizontal nesting. Vertical nesting occurs when a syntactical component takes the form of a separate statement and the top-level statement thus contains an inner statement. The term *nested institutional statement* means a statement, which may or may not be atomic, that is contained within a component of another statement.

The OR ELSE component is a special case that always has an inner statement. It is an abstract component that actually is a separate institutional statement. Where an OR ELSE component exists, the top-level statement is referred to as the monitored statement, and the statement contained in the OR ELSE is known as the consequential statement. Following is an example of vertical nesting with the OR ELSE component:

"Organic farmers must comply with organic farming regulations",
OR ELSE
"Certifiers must revoke the organic farming certification"

Note in the example that there are two complete institutional statements, the second being embedded within the OR ELSE component of the first. The first statement is monitored for compliance (i.e., the monitored statement) and the second expresses a consequence of noncompliance with the first (i.e., the consequential statement). Furthermore, the first statement may be referred to as the top-level statement whereas the second is the nested statement.

In IG Core, vertical nesting is only allowed in the form of statement-level nesting. This is another term for nesting a statement within the OR ELSE component. However, IG Extended allows vertical nesting in a number of components, namely ATTRIBUTES, OBJECT, CONSTITUTING PROPERTIES, CONSTITUTED ENTITY and CONTEXT as well as OR ELSE. This is referred to as component-level nesting, and is here exemplified as follows:

"Organic farmers may sell their produce under the organic label {under the condition that organic farmers apply for certification}"

In the above example, the nested institutional statement is denoted by curly braces and belongs to the CONTEXT (EXECUTION CONSTRAINTS) component of the top-level institutional statement. The nested statement contains all mandatory components of a regulative statement.

Moving on to horizontal nesting, this is the side-by-side combination of syntactical components or entire statements. It occurs when there are multiple of the same

syntactical component in a statement. Such combinations are characterized by a logical operator, typically a conjunction (e.g., "and") or disjunction (e.g., "or"). Exclusive disjunctions (logical XOR, e.g., "either or") also occur in institutional statements but may be subject to interpretation if they are visually identical to an inclusive disjunction. Following is an example of horizontal nesting:

"Organic farmers must commit to their organic farming standards and accommodate regular reviews of their practices"

The above statement has two actions (i.e., AIMS), linked by a conjunction ("and"). Furthermore, each action is associated with a separate OBJECT.

An institutional statement that features horizontal nesting may be split into multiple atomic statements. The above example may be decomposed as follows:

*"Organic farmers must commit to their organic farming standards" AND
"Organic farmers must accommodate regular reviews of their practices"*

Since the original statement has multiple AIMS, splitting results in one statement for each AIM. Additionally, since the OBJECTS are dependent on their respective AIMS, the statements are not split further.

More complex statements with multiple independent combinations may also be normalized in this way. Following is an example of a statement with two independent combinations:

"Certified operations or handlers must accept and comply with organic farming regulations."

Containing two combinations of two components each, this statement may be decomposed into four atomic statements:

*"Certified operations must accept organic farming regulations"
AND
"Certified handlers must accept organic farming regulations"
AND
"Certified operations must comply with organic farming regulations"
AND
"Certified handlers must comply with organic farming regulations"*

3.3.5 IG Extended Features

IG 2.0 accommodates the decomposition of actors and objects into descriptors and properties. As an example, in the OBJECT "written notification" the descriptor is "notification" and "written" is a property of the "notification". IG Core allows only a simple property per syntactical component, but IG Extended offers the Object-Property Hierarchy for coding complex property configurations. It allows properties to have properties, where any property may be substituted by an object, and a property or object may be functionally dependent on or independent from their parent property or object.

IG Extended furthermore offers the Context Taxonomy for semantic annotation of CONTEXT components (ACTIVATION CONDITIONS and EXECUTION CONSTRAINTS). This allows for coding institutional context more closely, such as whether the context is of temporal, spatial or some other nature. The taxonomy is structured in a hierarchy with generic labels at the base level, e.g., "temporal" and more specific labels at deeper levels, e.g., "point in time".

Sometimes a complex institutional statement includes statements of both regulative and constitutive kinds. For example, the top-level or leading statement may be regulative and contain a nested statement of constitutive kind. This is referred to as a hybrid institutional statement; the aforementioned example is specifically a regulative-constitutive hybrid. The inverse form also exists. The resolution of hybrid institutional statements is optional in IG Core and a central feature of IG Extended.

For most institutional statements, it is not difficult to identify its kind. However, some statements can feasibly be coded as both regulative and constitutive, and these are referred to as polymorphic institutional statements. Refer to the mapping of components in Table 3.2; in a polymorphic institutional statement, each component may be interchangeably regulative or constitutive. Often, such statements are coded in both forms, i.e., generic form and the analyst may choose a form based on his or her preference.

3.3.6 IG Logico Features

One of IG Logico's central features is the annotation of references. Many institutional statements make reference to another section of a policy or a policy as a whole. This could be the policy the original statement belongs to or a different one, and the referenced section could be another institutional statement or a division at any level in a policy. IG Logico offers a syntax for annotating such references with an identifier of the referenced entity. References signal relationships between institutional statements and policies, and this allows those relationships to be coded.

Another feature of IG Logico is cross-component semantic annotations. Taxonomies are offered for annotating syntactical components with labels such as the actor, action or object's role in the institutional setting, whether it is animate or inanimate and whether it is concrete or abstract. There are also taxonomies for annotating commonly occurring types of regulative and constitutive functions.

Finally, IG Logico is concerned with making logical relationships explicit. Institutional statements often contain lists, with an implied logical conjunction between all the list items. As explained in Section 3.3.4, statements containing logical relationships can be decomposed into multiple atomic statements, which IG Logico emphasizes. Where there are multiple logical relationships, the coder may need to establish precedence, although for a list where all items have the same logical operator this is not necessary. Moreover, the CONTEXT component can always be regarded as a list of conditions and constraints, meaning there is an implicit

conjunction between ACTIVATION CONDITIONS and EXECUTION CONSTRAINTS.

3.4 Literature Review

This section is based on the literature review in my Research Project Planning³ report but the text has been modified. Since there is no published literature specifically on the coding tools of the IG, this literature review will simply give an overview of the Institutional Grammar's development. In light of this development, it will also make an argument for a specialized coding tool.

The concept of an Institutional Grammar was first proposed by Sue Crawford and Elinor Ostrom in 1995 (Crawford & Ostrom, 1995). Emerging in the field of political science, it was motivated by a need to define institutions in enough detail that they could be analyzed. Crawford and Ostrom presented a simple grammar with definitions for five basic components of institutions: ATTRIBUTES, DEONTIC, AIM, CONDITIONS, and OR ELSE. The sequence was given the acronym ADICO. Furthermore, they defined three types of institutions: shared strategies, norms and rules, where a shared strategy consisted of an ATTRIBUTES, AIM and CONDITIONS, a norm consisted of a shared strategy plus a DEONTIC and a rule consisted of a norm plus an OR ELSE, reflecting how the types of institutions were composed.

This sequence of components was mapped onto institutional statements, where CONDITIONS in practice often served as a catch-all for text that did not fit any other component. However, the fundamental idea of components had been conveyed and the grammar would gradually be refined to capture institutional statements more closely.

After that initial proposal, no new literature on the IG emerged until 2008 with Smajgl et al.'s simulation study applying the grammar (Smajgl et al., 2008). At this point the field started to gain momentum. In 2010, Basurto et al. proposed the first set of coding guidelines (Basurto et al., 2010) for applying the grammar which laid the foundation for a majority of later research on the IG.

The literature on the IG can be roughly divided into two types of contributions: a) those that apply the grammar and b) those that propose refinements to it (Siddiki et al., 2019). The former type is far more numerous but this discussion will focus on the latter because it studies the IG's syntactical structure, which is relevant to creating a data structure for a new coding tool.

Table 3.3 gives an overview of the papers discussed in this chapter and what they contribute to the IG. Three of these papers are marked with *Refinement* and are, strictly considered, the only papers that propose syntactical refinements to the grammar.

³Course code: IMT4205

Table 3.3: Selected papers on the IG

Title	Author	Year	Contribution	Ref.
A Grammar of Institutions	Crawford and Ostrom	1995	Original proposal of IG	Crawford and Ostrom, 1995
A Systematic Approach to Institutional Analysis: Applying Crawford and Ostrom's Grammar	Basurto et al.	2010	Application: Coding guidelines	Basurto et al., 2010
Dissecting Policy Designs: An Application of the Institutional Grammar Tool	Siddiki et al.	2011	Refinement: OBJECT component	Siddiki et al., 2011
nADICO: A Nested Grammar of Institutions	Frantz et al.	2013	Refinement: Nesting	Frantz et al., 2013
Institutional Grammar 2.0: A specification for encoding and analyzing institutional design	Frantz and Siddiki	2021	Refinement: IG 2.0 overhaul	Frantz and Siddiki, 2021

In the first syntactical refinement paper, Siddiki et al. propose the addition of a new syntactical component, the OBJECT, which addresses a challenge with the original grammar to differentiate between the AIM and the CONDITIONS (Siddiki et al., 2011). The new component closely corresponds with linguistic sentence objects and as such allows for more precise coding of statements that contain an object. The OBJECT component is given the symbol "B", turning ADICO into ABDICO.

Frantz et al. propose a powerful new feature to the grammar: a syntax for nesting a statement within another and combining statements side-by-side with logical operators (Frantz et al., 2013). These concepts are referred to as vertical and horizontal nesting, respectively. Vertical nesting is useful for coding complex statements where a component of the top-level statement contains a whole other statement. Most prominently, this is always the case for the OR ELSE component. Horizontal nesting allows for coding logical combinations between individual components or entire statements. This proposal solved several problems related to the coding of complex statements into the flat, uniform structure of the original grammar. It is an important contribution because in the real world, policies are written by humans in natural language and complicated statements are prevalent.

Frantz and Siddiki's paper presents IG 2.0, an overhaul of the grammar (Frantz & Siddiki, 2021). It is accompanied by a comprehensive codebook (Frantz & Siddiki, 2020) for the new grammar. IG 2.0 retains a simple version relatively close to the original grammar labelled IG Core and presents two new versions at different levels of complexity. All three versions encapsulate existing syntactical refinements over the original grammar, and IG Core includes some new concepts proposed in the paper that are considered fundamental. IG Extended covers concepts such as the Object-Property Hierarchy, Context Taxonomy and hybrid and polymorphic statements. IG Logico is a layer on top which emphasizes logical relationships, higher-level semantic annotations and inter- and intra-policy references. The three versions are kept separate on account of their different use cases. The paper notably also introduces constitutive statements as a new kind of institutional statement and names the traditional kind regulative. One of its minor changes is renaming the CONDITIONS component to CONTEXT.

As mentioned before, the first set of guidelines for policy coding were proposed by Basurto et al. These guidelines (Basurto et al., 2010) include step-by-step instructions for how to use the grammar to code institutional statements taken from policies. As part of introducing the OBJECT component, Siddiki et al. refine the aforementioned guidelines to accommodate their new syntactical component (Siddiki et al., 2011). The latest guidelines can be found in the IG 2.0 codebook (Frantz & Siddiki, 2020), a comprehensive manual with coding instructions for IG Core, Extended and Logico.

IG 2.0 introduces a plethora of features to the syntax. The new grammar is complex, powerful and no longer uniform, intended to accurately capture complex real-world statements. For example, an institutional statement may have two ATTRIBUTES combined by a logical operator such as "and". This statement may be

flattened into two statements, one for each `ATTRIBUTES`, with the "and" operator between the statements, as exemplified in Section 3.3.4. To code this in a tabular data structure such as a spreadsheet, one needs to use two rows. If the statement also has two independent `AIMS`, there are now four atomic statements requiring four rows. IG 2.0 is capable of coding this compactly in a hierarchical structure. Furthermore, vertically nested statements are equally unsuitable for a tabular structure. To code such a statement in a spreadsheet, one needs to use multiple rows as well as maintain a reference between the parent and child statements. There is no standard answer to whether the reference should be coded from the parent to the child, from the child to the parent or both ways, adding to the complexity. (This is explained further in Section 3.5.1.) A statement that features both horizontal and vertical nesting coded in a spreadsheet easily surpasses the limit for how much complexity a human coder can keep track of.

The Institutional Grammar has never been a linguistic grammar. Crawford and Ostrom intended for it to provide a definition of institutions by breaking them down to their core components. Even though some components have a strong correspondence to certain linguistic parts of sentence or parts of speech, the grammar belongs to a domain. This is an important argument for a specialized coding tool: while general-use tools can be repurposed to function as policy coding tools, they will always have shortcomings.

3.5 Current Tools

3.5.1 Spreadsheets

A spreadsheet is “a computer program that allows the entry, calculation, and storage of data in columns and rows”⁴. Spreadsheets like Microsoft Excel have been the primary coding tool for the Institutional Grammar since it was first applied. Even after the publication of the IG 2.0 codebook (Frantz & Siddiki, 2020), many IG researchers still prefer it over newer tools like INCEPTION. Note that while Microsoft Excel is probably the most well-known spreadsheet, any spreadsheet may be used in place of it.

Coding in spreadsheets uses a template which may be regarded as the IG customization layer. The template consists of a number of columns and the intent is to code one statement per row so that each cell holds the value of a syntactical component. There are columns for constitutive and regulative components, properties, logical operators, forward and backward reference, metadata such as section and statement identifiers, and more. Coding in a spreadsheet entails copying and pasting text into the appropriate columns, or alternatively typing, which is discouraged because of the potential for error.

Figure 3.1 shows the spreadsheet template in Microsoft Excel with an example of a coded statement. Due to the template’s length, the image has been wrapped. In

⁴“Spreadsheet”. Merriam-Webster.com Dictionary, Merriam-Webster, <https://www.merriam-webster.com/dictionary/spreadsheet>. Accessed 12 May 2021.

the template, the gray columns are fixed, meaning they are always visible when scrolling horizontally. The blue columns are for coding regulative statements, and past these (not pictured) are a set of corresponding green columns for constitutive statements.

As Figure 3.1 shows, references are made by assigning a unique identifier to each atomic statement. Such identifiers can often be taken from the policy; otherwise, they must be assigned manually. The example statement is complex, featuring two nested statements each containing logical combinations, the coding of which is explained below.

Institutional statements that contain multiple components of the same kind must be normalized to multiple atomic statements in order to code them in a spreadsheet. The logical operator column of either row is then filled to code the relationship between the statements. There is no standard answer to which row should hold the logical operator, which relies on consensus among coders. It has been decided that the logical operator columns in both rows are to be filled.

Vertical nesting, which is always encountered in the OR ELSE component, also takes multiple rows to code. The parent and child statements are coded in separate rows. To code their relationship, the child statement's identifier is filled into the parent's forward reference column, and vice versa with the child's backward reference column. Again, there is no standard answer to which row should hold the reference, so it has been decided that a double reference is to be used.

As we can see from Figure 3.1, spreadsheets are better suited to coding simple statements and coding in IG Core. However, once data has been coded in a spreadsheet it is very easy to move on to statistical analysis thanks to the statistical capabilities of software like Microsoft Excel and the R system⁵ as well as the fact that the standard spreadsheet format, comma-separated values (CSV), is so common.

⁵<https://www.r-project.org/>

	A	B	C	D	E	F	G	H	I	J	K	L	M
													Regulative Statements
Section Marker	Statement No.	Raw Statement	Attribute	Attribute Property	Deontic	Aim	Direct Object (Content)	Direct Object (Reference to statement)	Direct Object Property	Indirect Object (Content)	Indirect Object (Reference to statement)		
3	652	The Program Manager may initiate suspension or revocation proceedings against a certified operation: (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or	may initiate	may	has reason to believe	proceedings	has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or		suspension	operation			
4	652.1	The Program Manager may initiate suspension or revocation proceedings against a certified operation: (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or	may initiate	may	has reason to believe	proceedings	has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or		revocation	operation			
5	652.2	The Program Manager may initiate suspension or revocation proceedings against a certified operation: (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or	may initiate	may	has reason to believe	proceedings	has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or						
6	652.3	The Program Manager may initiate suspension or revocation proceedings against a certified operation: (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or	may initiate	may	has reason to believe	proceedings	has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or						
7	652.4	The Program Manager may initiate suspension or revocation proceedings against a certified operation: (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or	may initiate	may	has reason to believe	proceedings	has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or						
8	652.5	The Program Manager may initiate suspension or revocation proceedings against a certified operation: (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or	may initiate	may	has reason to believe	proceedings	has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or						
9	652.6	The Program Manager may initiate suspension or revocation proceedings against a certified operation: (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or	may initiate	may	has reason to believe	proceedings	has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or						
10	652.7	The Program Manager may initiate suspension or revocation proceedings against a certified operation: (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or	may initiate	may	has reason to believe	proceedings	has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or						
11	652.8	The Program Manager may initiate suspension or revocation proceedings against a certified operation: (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or	may initiate	may	has reason to believe	proceedings	has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or						
12	652.9	The Program Manager may initiate suspension or revocation proceedings against a certified operation: (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or	may initiate	may	has reason to believe	proceedings	has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or						
13													
14													
15													
1													
Section Marker	Statement No.	Raw Statement	Attribute	Attribute Property	Deontic	Aim	Direct Object (Content)	Direct Object (Reference to statement)	Direct Object Property	Indirect Object (Content)	Indirect Object (Reference to statement)		
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
1													
Section Marker	Statement No.	Raw Statement	Attribute	Attribute Property	Deontic	Aim	Direct Object (Content)	Direct Object (Reference to statement)	Direct Object Property	Indirect Object (Content)	Indirect Object (Reference to statement)		
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													

Figure 3.1: Spreadsheet template for IG 2.0 with example

3.5.2 Text Annotation Tools

Text annotation is the process and result of adding notes to a text without altering the text's content. It is a common method in Natural Language Processing (NLP), in which annotations are structured so as to be machine-readable. Many text annotation projects revolve around large knowledge bases or corpora and so require teams of annotators working together. Thus, several annotation platforms have been developed to facilitate these projects.

One such platform is the BRAT Rapid Annotation Tool⁶, an online environment for collaborative, structured text annotation. While BRAT is still being maintained as of writing, another text annotation platform, WebAnno⁷, is based on BRAT (Yimam et al., 2013). WebAnno is a general-purpose linguistic annotation tool with support for project collaboration (Eckart de Castilho et al., 2016). However, around 2018 it was superseded by INCEpTION⁸, a larger platform with a broader scope including knowledge base population and fact linking (Klie et al., 2018). What is interesting is that INCEpTION has notably been used for policy coding with the IG.

INCEpTION is an open source text annotation platform developed by UKP Lab at TU Darmstadt. It facilitates a number of semantic annotation tasks and has machine learning capabilities to assist annotators. Moreover, it is a multi-user platform that allows users to collaborate on projects. One of its use cases is working with knowledge bases but its customizability and capabilities for text annotation make it an ostensibly viable tool for policy coding.

I underline that INCEpTION has only recently been taken up as a coding tool and has not gained much popularity yet. According to my supervisor, only around four or five researchers use it at the time of writing.

In INCEpTION, all annotations belong to a layer and layers are used to separate different kinds of annotations. IG researchers have therefore created a custom set of IG-specific layers for policy coding. There are separate layers for regulative and constitutive sets of components, and one for institutional statements which supports both monitored and consequential statements. Component layers support providing an inferred text value for the component and specifying whether the component contains an institutional statement and whether it implies negation. Whereas the spreadsheet template is the "customization layer" for spreadsheets, these layers make up the "customization layer" for INCEpTION.

Figure 3.2 shows INCEpTION's user interface with examples of coded statements. The current layer and the currently selected annotation are shown on the right pane. The figure shows the "IG Core Regulative Syntax" layer and what options it offers for each annotation. To add an annotation one selects the text to be annotated, and a new annotation is created using the current layer and is visualized above the marked text directly in the source, with color coding. Annotation fields

⁶<https://brat.nlplab.org/>

⁷<https://webanno.github.io/webanno/>

⁸<https://inception-project.github.io/>

may then be filled if necessary. Conveniently, the IG layers include keyboard shortcuts for marking syntactical components, e.g., while a component annotation is active, one can press the D key to mark it as a DEONTIC. The visualization shows all annotations on all layers but each layer can be hidden. Furthermore, annotations can overlap, which is useful in policy coding for annotating a whole institutional statement in addition to its syntactical components.

Even though layers can be individually disabled, a fully annotated, complex policy text tends to look cluttered in INCEpTION. It is often difficult to read a coding when revisiting or reviewing it due to the sheer amount of information presented all at once. Note that coder reviews are a reliability testing method commonly employed by IG researchers, and this need may not be reflected in the scope INCEpTION was designed for.

For the purposes of data analysis, INCEpTION offers exportation in around 20 formats including text annotation and natural language processing (NLP) formats. One such format is UIMA CAS. Unstructured Information Management Architecture (UIMA) is a middleware architecture for processing unstructured information (Ferrucci & Lally, 2004), and its standard format is the Common Analysis Structure (CAS), an object-based data structure. While UIMA is not an information management application itself, Apache UIMA⁹ is an open-source implementation of the UIMA specification (Ferrucci et al., 2009).

To my knowledge, it is not possible to automatically convert an INCEpTION coding to a spreadsheet format. Thus, to perform analysis of data coded in INCEpTION one needs additional tools and knowledge of unstructured information frameworks such as UIMA.

⁹<https://uima.apache.org/>

The screenshot displays the INCEpTION software interface. On the left, a dark sidebar contains navigation icons and a 'Log out' button. The main workspace is a text editor showing a document titled 'Test2/IGExampleStatements.txt' with 14 sentences. The text is annotated with various metadata tags such as (A) Attribute, (I) Aim, (D) Deontic, (C) Complex Component Values, (E) Entity, (P) Provisional, (T) Temporal, and (F) Frame. A right-hand sidebar provides configuration options for the current layer, including 'Layer', 'Annotation', 'Text', 'Component', 'Embeds Institutional Statement', 'Implies Negation', 'Inferred Component Value', and 'Physical Entity'. Each option has a 'Yes' or 'No' selection button.

INCEpTION Projects Dashboard Log out 027 min

IG Core Regulative Syntax
Create a IG Core Regulative Component Relationship relation by drawing an arc between annotations of this layer.

Layer: IG Core Regulative Syntax [Delete] [Clear]

Annotation: IG Core Regulative Syntax

Text: Persons
No links or relations connect to this annotation.

Additional Label(s):

Component: (A) Attribute [Show key bindings...]

Embeds Institutional Statement: Yes [No]

Implies Negation: Yes [No]

Inferred Component Value:

Physical Entity: Yes [No]

1-14 / 14 sentences [doc 1/1]

1 [Basic Examples].

2 Persons who qualify for membership in the Association shall be provisional members for a period of thirty (30) days.

3 Public employers (no matter how many employees) must provide employees with job protection for the duration of the order of quarantine.

4 [Complex Component Values].

5 An Annual General Meeting of the Association shall be held no sooner than thirty (30) days after the close of the fiscal year and no later than six (6) months after the close of the fiscal year.

6 [Statement Combinations].

7 All stall spaces must be swept clean and any refuse removed at the end of each market day.

8 Policy council - Provides input to City Council and staff about issues of concern; - Considers any matters which may be referred to the Policy Council by City Council or staff; - May take positions on policy initiatives from other levels of government within the mandate of the Policy Council; - Advises Council and staff on the Vancouver Food Strategy as it is developed, implemented and updated.

9 Untenured faculty members are ineligible to vote on the awarding of tenure in promotion and tenure review committees or unit meetings.

10 [Component-Level Nesting].

11 Each electricity provider shall establish an application form and procedures to enable eligible customers to participate in the net metering program offered by the electricity provider.

12 The Program Manager may initiate suspension or revocation proceedings against a certified operation: (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part; or (2) When a certifying agent or a State organic program's governing State official fails to take appropriate action to enforce the Act or regulations in this part.

13 [Vertical Nesting].

14 Certified organic farmers must not apply synthetic chemicals to crops at any time once organic certification is conferred, or else certifier will revoke certification from farmer.

Figure 3.2: INCEpTION user interface with examples

3.5.3 Inline Coding

IG 2.0 offers an inline syntax for coding institutional statements. It is not a coding tool but a noteworthy approach to coding nonetheless. Commonly referred to as *shorthand coding* or *inline coding*, the IG syntax allows for coding institutional statements directly inline. It is fully documented in the IG 2.0 codebook (Frantz & Siddiki, 2020). Following is an example of a regulative statement:

"Certifier must monitor farmers at all times."

In shorthand, this would be coded as

"Certifier (A) must (D) monitor (I) farmers (Bdir) at all times (Cex)."

As illustrated by this example, an annotation is inserted after each syntactical component. Every component in the IG is associated with an alphabetical symbol as specified in Sections 3.3.1 and 3.3.2, and in the above example the components are respectively ATTRIBUTES, DEONTIC, AIM, OBJECT (DIRECT) and CONTEXT (EXECUTION CONSTRAINTS).

As an inline syntax, shorthand coding can be used with any text processor or even written by hand. However, inline coding is by its nature greatly limited in its capabilities for data analysis. Rather, one of its use cases is helping the coder figure out a statement, not unlike a sketch before data entry in a coding tool.

The above example is straightforward, with simple values for each syntactical component. Now consider the following complex statement:

"When an inspection of an accredited certifying agent by the Program Manager reveals any noncompliance with the Act or regulations in this part, a written notification of noncompliance shall be sent to the certifying agent."

This statement is written in passive voice and should first be rephrased into active voice to make explicit the actors, actions and objects. This will make it easier to code. We rephrase the statement as follows:

"When Program Manager reveals any noncompliance by the accrediting certifying agent with the Act or regulations in this part under the condition that Program Manager performs inspection of an accredited certifying agent, Program Manager shall send a written notification of noncompliance to the certifying agent."

The statement is then coded as follows:

"{When [Program Manager (A)] reveals (I) any noncompliance (Bdir) [by the accrediting (Bind,prop1) certifying (Bind,prop2) agent (Bind)] with the Act or regulations in this part (Cex,eff) {[under the condition that] Program Manager (A) [performs] (I) inspection (Bdir) of an accredited (Bind,prop1) certifying (Bind,prop2) agent (Bind)} (Cac)} (Cac), [Program Manager (A)] shall (D) [send (I)] a written (Bdir,prop1) notification (Bdir) of noncompliance (Bdir,prop2) to the certifying (Bind,prop1) agent (Bind)."

To briefly explain the syntax, curly braces denote a nested statement and square

brackets explicitly specify an implied component. The IG 2.0 codebook contains the full documentation. As evidenced by this example, even shorthand coding can be hard to manage when used for complex statements. Even if a coder is able to annotate the text with relative ease, the resulting inline coding tends to be difficult to read when vertical and/or horizontal nesting is involved.

3.5.4 Automated Approaches

Manual policy coding is a tedious and time-consuming process, requiring that policy texts first be decomposed into institutional statements before those statements may be coded (Rice et al., 2021). Almost since the IG was first applied, researchers have looked for ways to make the work easier. Automating the coding process using computational tools has long been an attractive goal for the IG community.

As a step toward this goal, Rice et al. propose a supervised machine learning approach to automating policy coding with the IG (Rice et al., 2021). Their dataset comes from an existing body of manually coded policy texts (Siddiki, 2014). Rice et al. also attempt to use Stanford CoreNLP (Manning et al., 2014) to automatically identify IG syntactical components by leveraging the fact that components may be mapped to certain parts of speech, but this turned out to be too simplistic to produce useful results. However, they find the supervised learning approach to show promise as they attain a relatively high accuracy in applying it.

While automated approaches could one day be the primary method of data collection for researchers studying policy design, manually coded data is a prerequisite for developing those approaches. Siddiki's dataset (Siddiki, 2014) is coded using an older, more coarse-grained version of the IG and as such, a dataset using IG 2.0 would be required to train a machine learning model to code in IG 2.0. One of the motivations behind IG Coder is to make it easier to build such a dataset. Therefore, this thesis investigates the usability and workflow of current coding tools and evaluates IG Coder for the same.

Chapter 4

Review of Current Tools

This chapter is based on my Advanced Project Work¹ report but the study has been supplemented with a questionnaire on both Microsoft Excel and INCEpTION, and as such, the text has been heavily modified.

4.1 Introduction

This study will investigate RQ1 and RQ2 of this thesis. As a reminder, they are as follows:

State of the art

RQ1 What are the strengths and weaknesses of current coding tools?

RQ1a What features from existing tools should the new coding tool retain?

RQ1b What features from existing tools should the new coding tool discard?

User needs

RQ2 What features do coders need in a coding tool for the IG?

RQ2a What are the essential features a coding tool for the IG should possess?

RQ2b What are the fringe features a coding tool for the IG should possess?

As RQ1 refers to current coding tools, these must first be determined. By coding tools I mean software applications that can be used for manual policy coding, not inline coding or machine coding. Based on information given to me by my supervisor, who is an IGRI affiliate, the two most used software applications for policy coding are Microsoft Excel and INCEpTION. This has also been touched on in Chapter 3. Technically, any spreadsheet can be used in place of Microsoft

¹Course code: IMT4894

Excel but it is probably the most well-known spreadsheet and will therefore be investigated specifically in this study.

Of the current coding tools, spreadsheets are significantly more popular than INCEpTION at the time of writing. This discrepancy is reflected in the data sources for this study, which are described in Section 4.2. Furthermore, INCEpTION's low uptake impacts my ability to collect data on the tool, the implications of which have been mentioned in Chapter 2 and will be reiterated in this chapter.

4.2 Method

This study seeks answers in the following six categories derived from the research questions:

1. Advantages of Microsoft Excel as a coding tool (RQ1a)
2. Disadvantages of Microsoft Excel as a coding tool (RQ1b)
3. Advantages of INCEpTION as a coding tool (RQ1a)
4. Disadvantages of INCEpTION as a coding tool (RQ1b)
5. Essential features needed in a coding tool (RQ2a)
6. Fringe features desired in a coding tool (RQ2b)

The categories are mapped to research questions as shown in the parentheses. Here, I consider advantages to translate to strengths and disadvantages to weaknesses.

This study uses four different data sources, the reasoning behind which has been explained in Section 2.1. The sources are as follows, and are explained below:

1. Questionnaire of students on Microsoft Excel
2. Notes from IGRI coding course on Microsoft Excel
3. Interview conducted in spring 2020 on Microsoft Excel
4. Questionnaire of researchers on Microsoft Excel and INCEpTION

In fall 2020, researchers from the IGRI held an online coding course² for students at several universities. Students were taught the Institutional Grammar and how to code institutional statements in Microsoft Excel. My supervisor, Christopher Frantz, was among the professors. At the end of this course, a questionnaire was sent out asking the students for feedback on the course.

On this occasion, I sent out my own questionnaire to the students to inquire about their views of Microsoft Excel as a coding tool and needs of coders in a coding tool. This is data source 1. I asked about what they saw as advantages and disadvantages with Microsoft Excel as a coding tool as well as what features they thought were needed in a coding tool. Note that questions referred to Microsoft Excel specifically, not spreadsheets in general. Additionally, I asked some demographic questions regarding study level, study area and experience level with Microsoft Excel. Three questions were inserted by my supervisor which are irrelevant to this study, and the final question which was a catch-all for additional comments re-

²<https://institutionalgrammar.org/teaching-institutional-analysis/>

ceived no relevant responses. Different types of questions were used for different purposes: open long-text questions for advantages, disadvantages and needs, Likert scale (Robinson, 2014) rating tasks for experience level with Microsoft Excel and single choice questions for other demographic information. The course was attended by 12-14 students (varying from session to session) and I received 10 responses. The full questionnaire sheet is shown in Appendix A. In the remainder of this chapter, this questionnaire will be referred to as Questionnaire A.

I also attended the aforementioned Microsoft Excel coding course as an observer and noted down comments and discussions related to the coding experience in Microsoft Excel. The resulting notes make up data source 2. It should be noted that this data source lacks information connecting specific comments to specific people, and the group of students and professors in the course has significant overlap with the group I sent the aforementioned questionnaire to. Therefore, I treat this data source more carefully than the questionnaires, giving more weight to the latter as will be explained later in this section.

Data source 3 is an older interview conducted by my classmate and myself in spring 2020 as part of the course Integration Project³. Our objective was to gain insight about policy coding in Microsoft Excel, such as what the process is like, what the challenges are, why spreadsheets are used and how the coded data is used. The interviewee was Christopher Frantz, my supervisor. It is important to note, however, that this interview was conducted before IG 2.0 was in use. In this study I therefore filter out the sentiments that do not apply to IG 2.0. This interview is included as a data source for this study because it contains relevant insights, even though some of the information no longer applies. The interview questions are shown in Appendix B.

Finally, I conducted another questionnaire in spring 2021, which is data source 4. This questionnaire differed from the first in two ways: first, it was aimed at and sent out to researchers only, and second, it investigated both Microsoft Excel and INCEpTION. There is no overlap between the groups the two questionnaires were sent out to. In this questionnaire, the questions on Microsoft Excel were identical to those in Questionnaire A, and it furthermore asked equivalent questions on INCEpTION. I received five responses to this questionnaire. Also, as with Questionnaire A, questions referred to Microsoft Excel specifically and not spreadsheets in general. The full questionnaire sheet is shown in Appendix C. In the remainder of this chapter, this questionnaire will be referred to as Questionnaire B.

Data source 4 is the only source on INCEpTION in this study, reflecting its low uptake compared to Microsoft Excel. However, this also means the data on the former will be less reliable. It is yet early in the adoption of INCEpTION for policy coding, so results at this stage will have limited reliability.

Before we continue, I must clarify that part of this study was first conducted in the course Advanced Project Work⁴. That study covered categories 1-2 and 5-6, except categories 5-6 were combined as simply "needs of coders in a coding tool".

³Course code: IMT4807

⁴Course code: IMT4894

In other words, the study investigated Microsoft Excel only and did not cover INCEPTION. Furthermore, it was built on data sources 1-3 only. The study in this thesis supplements the aforementioned with data source 4 and merges the results from that data source directly into those of the original study. All six categories are thus covered.

Of these data sources, the questionnaires are relatively structured whereas the rest are largely unstructured. The approach taken to structure this information was semantic clustering. I set up five tables, one for each of the categories with the fifth table representing both categories 5 and 6. The fifth table will be split into two, separating essential and fringe features of a coding tool. The table structure is described in Table 4.1.

Cluster	Count	Sources
A grouping of similar data points	Total number of data points belonging to this cluster	A list of the sources the data points were extracted from, with a count for each source

Table 4.1: Table format for structuring the raw data

Next, I inspected each of the data sources and extracted data points from them. Here, a data point means a single sentiment, idea or opinion, formulated in a sentence (the sentence does not need to be complete). I placed each data point in an appropriate category. In the two questionnaires, I had formulated questions with the intent to map them directly to each of the categories, whereas with the other two data sources I had to use my discretion. The hardest part was distinguishing needs from advantages, as an advantage could often translate directly to a need and a disadvantage could simply be inverted to generate a need. To solve this, my rule of thumb was to look at the context around the sentiment and ask myself "is the context related to a specific coding tool or coding tools in general?". Moreover, the questionnaire responses often contained more than one data point as the respondent listed multiple ideas. These were treated as individual data points if they differed in meaning, otherwise as one.

Once I finished extracting data points, I inspected each table and formed clusters by merging all data points I deemed to have the same general meaning. To merge I first decided on a main sentence, preferably from a questionnaire response. I then slightly rephrased this sentence to include words from the sentences that would be merged into it, to capture the breadth of each cluster, as well as fixing grammar and spelling errors. Thus, each cluster was represented by one sentence. For each cluster, I counted the number of data points for each data source to populate the columns Count and Sources.

When counting, I observed the following rules. For the questionnaires, each questionnaire respondent counted no more than once in each cluster. For each of the other data sources, all data points in a cluster counted as a total of one even if multiple data points fit that cluster. The reasoning behind this is that firstly, the in-

interview was of a single person and secondly, the course notes lacked information connecting specific sentiments to specific people, as mentioned before. Furthermore, this somewhat reduces the potential for bias by weighting questionnaire responses more heavily than my notes which are potentially subjective and an interview which is in some ways outdated.

Finally, I sorted the tables, primarily by count in descending order, secondarily by cluster in alphabetical order. This resulted in five tables listing and ranking my findings in each category. However, categories 5 and 6 have not yet been separated, as the fifth table currently contains all identified needs in a coding tool. To answer RQ2 and distinguish between essential and fringe features, these needs must be prioritized. In Section 1.1, I defined these as “By essential features is meant features needed by many users, and fringe features are those suggested by at most two.” Therefore, I set a threshold at three: needs with a count of three or more are deemed essential features, whereas needs with a lower count are deemed fringe features. This will help me perform requirements engineering for IG Coder and prioritize the features to implement.

4.3 Results

Before presenting the main results and answering the research questions, I will present the demographics of the two questionnaires. It must be noted that with only 10 and 5 respondents, respectively, these results are not generalizable and are included to contextualize the main results.

In Figures 4.1 and 4.2 we can see how respondents in each of the questionnaires rated themselves in terms of experience level with Microsoft Excel. Both source questions are a 5-point Likert scale and all available options are shown. Overall, we see that respondents of Questionnaire B reported a slightly higher level of experience with Microsoft Excel.

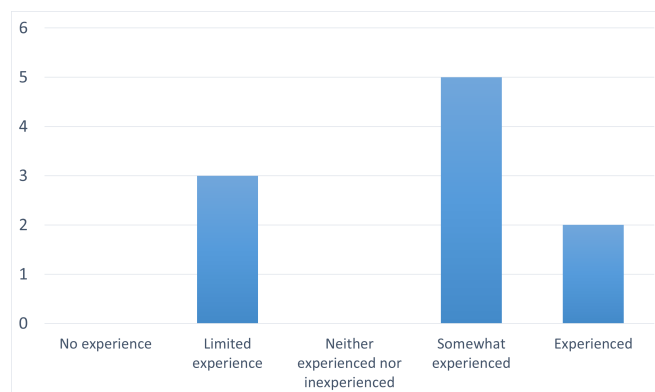


Figure 4.1: Questionnaire A: Level of experience with Microsoft Excel

Note that in all of the tables pertaining to demographics, I have adjusted the y-axis to one higher than the most popular option.

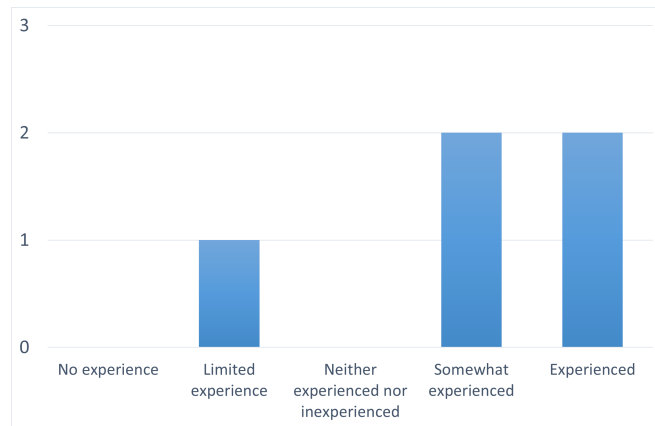


Figure 4.2: Questionnaire B: Level of experience with Microsoft Excel

Figures 4.3 and 4.4 show how respondents in the two questionnaires rated Microsoft Excel's suitability as a coding tool for the IG. Both source questions are a Likert scale of 7 points, and all available options are shown. Comparing the two questionnaires, the results are quite similar. Both groups of respondents reported a fairly strong positivity toward the tool.

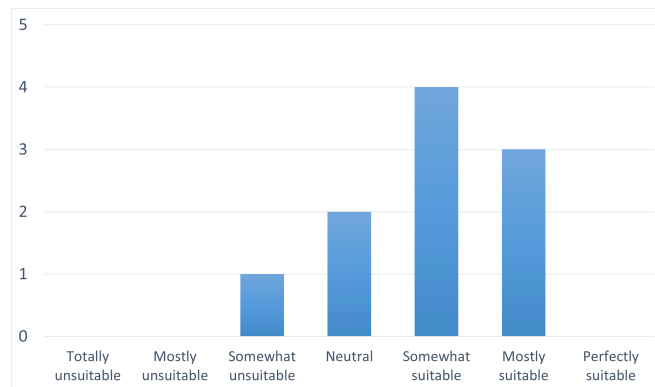


Figure 4.3: Questionnaire A: Microsoft Excel's suitability as a coding tool

Finally, Figures 4.5 and 4.6 show the Questionnaire B respondents' level of experience with and perceived suitability of INCEpTION. The source questions and options were identical to those on Microsoft Excel. Overall level of experience with INCEpTION in this group is slightly lower than that of Microsoft Excel but again the results are very similar. INCEpTION's reported suitability is also slightly lower than that of Microsoft Excel in the same questionnaire.

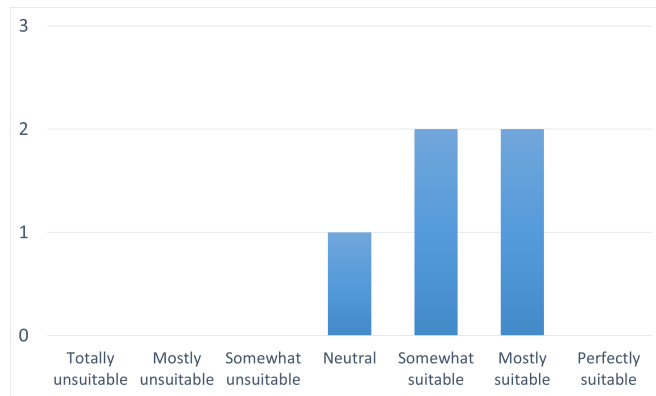


Figure 4.4: Questionnaire B: Microsoft Excel’s suitability as a coding tool

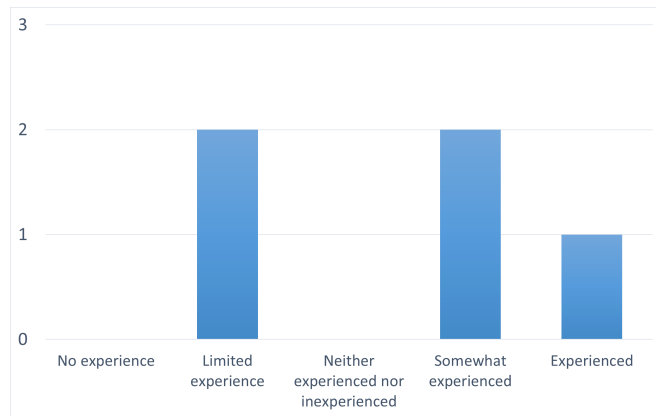


Figure 4.5: Questionnaire B: Level of experience with INCEPTION

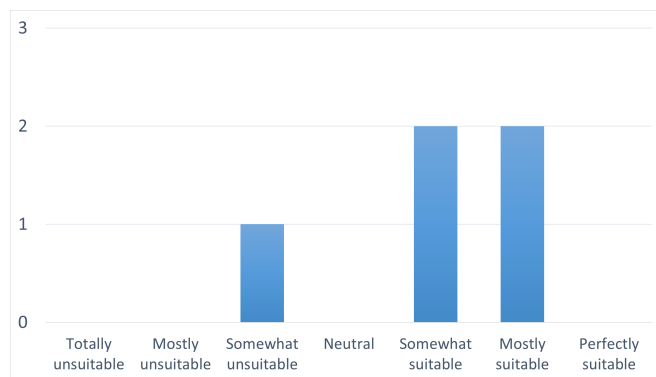


Figure 4.6: Questionnaire B: INCEPTION’s suitability as a coding tool

Moving on, the ranked findings in each of the six categories, without distinguishing between categories 5 and 6, are shown in Tables 4.2 to 4.6. According to the weighting method described in Section 4.2, they are weighted roughly by the num-

ber of people who reported a sentiment or idea in each cluster, giving emphasis to the questionnaires. In the source lists, "Quest" is short for questionnaire. Since the categories outlined in Section 4.2 map to research questions, we can now answer the research questions.

RQ1: What are the strengths and weaknesses of current coding tools? Since I in Section 1.1 deemed strengths and weaknesses to directly translate to RQ1's respective sub-questions, this question is answered by those sub-questions.

RQ1a: What features from existing tools should the new coding tool retain?

This research question is answered by Tables 4.2 and 4.3. In the two questionnaires, I asked participants "In your opinion what makes Excel / INCEpTION a useful coding tool?" I believe this formulation helps participants think about the advantages of the tools, which is what I am after. This question came after asking participants to rate their level of experience with the tool and to rate it as a policy coding tool.

As we can see, there is a higher number of clusters related to Microsoft Excel than to INCEpTION. I remind the reader that the data pertaining to Microsoft Excel is significantly richer than that on INCEpTION, and that this applies to the next research question as well (RQ1b).

Cluster	Count	Sources
Represents data in an organized fashion	8	QuestA (6), Notes (1), Interview (1)
Easy to navigate, simple	4	QuestA (4)
The parsed information is easy to see, interpret, compare, and manipulate	4	QuestB (2), Interview (1), Notes (1)
Coded data can almost immediately be used for analysis	3	Interview (1), Notes (1), QuestB (1)
"Everyone" knows how to use a spreadsheet, easy to share	2	Interview (1), QuestA (1)
Quick and easy to populate cells that contain the same information by copy-pasting	2	QuestA (1), QuestB (1)
Ability to freeze cells is useful	1	QuestA (1)
The tabular format enforces thinking about each category, which is good for less experienced coders	1	QuestB (1)
Very good educational tool for the IG, easy to understand for beginners	1	QuestB (1)

Table 4.2: Identified advantages of Microsoft Excel as a coding tool

Cluster	Count	Sources
Allows cross-statement annotation	2	QuestB (2)
Allows duplicate annotations	1	QuestB (1)
Allows sharing and collaboration	1	QuestB (1)
Coding directly on the document is intuitive	1	QuestB (1)
Easy to create new layers/tag sets	1	QuestB (1)
Free and open source	1	QuestB (1)
Layers/tag sets are specifically designed for coding in the IG	1	QuestB (1)

Table 4.3: Identified advantages of INCEpTION as a coding tool

RQ1b: What features from existing tools should the new coding tool discard?

This research question is answered by Tables 4.4 and 4.5. In the two questionnaires, participants were asked "What challenges have you encountered coding in Excel / INCEpTION?" I chose to use the word "challenges" because it helps the participants think about their own experience with the tools, as opposed to asking something akin to "What are the weaknesses of the tools?". This question immediately followed the question asking about advantages with the tool.

RQ2: What features do coders need in a coding tool for the IG?

This overarching research question is answered by Table 4.6. In both of the questionnaires, I asked participants "If you could design a new coding tool for IG 2.0 (i.e., new software), what three capabilities should that tool possess?" My intent with asking for three features was to avoid "intimidating" the respondents, because without the "three" qualifier, this question might as well have asked for a complete requirements document for the application. Several respondents thus listed multiple ideas, which I extracted and organized as explained in Section 4.2. One response was hard to split up: "Accountability, accuracy, efficiency". This was left as is because it describes non-functional requirements, which are also useful to me in requirements engineering for IG Coder.

In both of the questionnaires, the above question immediately followed the one asking about disadvantages with the tool (INCEpTION in the case of Questionnaire B). This may have had an effect on the responses, as respondents by this point had thought about advantages and disadvantages with the tool and were now asked to come up with features a coding tool should have. This should be taken into consideration when inspecting the identified needs.

Note that the interview (data source 3) yielded no data points pertaining to needs. This is because the interview focused on the coding experience in Microsoft Excel, and coding tools in general were not discussed.

Cluster	Count	Sources
Time-consuming and tedious to code by copy-pasting	5	QuestB (3), Notes (1), QuestA (1)
Confusion and loss of overview in larger tables	4	QuestA (2), Interview (1), Notes (1)
Difficult and cumbersome to code and keep track of nested statements	4	Interview (1), Notes (1), QuestA (1), QuestB (1)
Annoying to move back and forth between far apart columns	2	QuestA (2)
No ability to capture inter-statement or inter-document linkages	2	QuestB (2)
No coding completeness or quality controls	2	Notes (1), QuestB (1)
Hard to rephrase a statement	1	Notes (1)
Hierarchies between components and properties cannot be represented	1	QuestB (1)
Limited cell space, words are often cut off	1	QuestA (1)
Metadata has to be manually populated	1	QuestA (1)
No standard way of indicating "not yet coded"	1	Interview (1)
Not a good tool for big projects with several documents and several team members	1	QuestB (1)
Not computationally tractable	1	Notes (1)
Rows full of "n/a" are hard to read	1	Interview (1)

Table 4.4: Identified disadvantages of Microsoft Excel as a coding tool

RQ2a: What are the essential features a coding tool for the IG should possess?

According to the threshold defined in Section 4.2, needs with a count of three or higher are deemed essential features. As we can see in Table 4.6, three needs were identified that belong to this category:

- Ability to export coded text in different formats (e.g. Excel CSV)
- An overview area underneath each statement with organized fields for IG components to visually double-check one's coding
- Color coding for components distinguishing between regulative and constitutive

All of these had a count of three and are sorted alphabetically. They will have priority in defining the requirements for IG Coder.

RQ2b: What are the fringe features a coding tool for the IG should possess?

According to the threshold defined in Section 4.2, needs with a count of two or

Cluster	Count	Sources
Editing an annotation requires deleting and recreating it	2	QuestB (2)
Graphic representation of coded data is unreadable	2	QuestB (2)
Missing output/export of coded text into a usable format	2	QuestB (2)
Missing validation of correctness and completeness	2	QuestB (2)
Changing layers/tag sets during coding is impractical	1	QuestB (1)
Coding in the IG requires the use of several layers/tag sets	1	QuestB (1)

Table 4.5: Identified disadvantages of INCEpTION as a coding tool

lower are deemed fringe features. Refer to Table 4.6; 27 needs were identified that belong to this category. Eight of these had a count of two, meaning they were reported by two people. All of the needs will be considered during requirements engineering for IG Coder but the ranking will help me prioritize what features to invest in.

Table 4.6: Identified needs in a coding tool for the IG (1/2)

Cluster	Count	Sources
Ability to export coded text in different formats (e.g. Excel CSV)	3	QuestB (3)
An overview area underneath each statement with organized fields for IG components to visually double-check one's coding	3	QuestB (2), QuestA (1)
Color coding for components distinguishing between regulative and constitutive	3	QuestA (2), Notes (1)
Ability to directly connect related institutional statements within the same document, e.g. using section markers	2	Notes (1), QuestB (1)
Ability to drag components from original statement into category field (e.g., select the actor and drag it into Attribute field)	2	QuestA (1), QuestB (1)
Ability to flag external policy documents referenced in an institutional statement, e.g. using section markers	2	Notes (1), QuestB (1)
Ability to generate an output file for accessible viewing of coded data	2	QuestB (2)
Accommodate implicit (tacit) components	2	Notes (1), QuestB (1)
Clear representation of coded statements for review	2	QuestB (2)
Fast non-redundant handling of nested/multiple statements and logical relationships	2	QuestA (2)
More space for each component	2	QuestA (2)
Ability to change nesting levels without losing annotation content (e.g., if a property is shifted to level 3, it should not lose its content)	1	QuestB (1)
Ability to review earlier coded statements	1	QuestB (1)
Ability to tag IG components directly within the document (without having to copy and paste)	1	QuestB (1)
Ability to upload text documents for coding, and ability to manipulate uploaded text	1	QuestB (1)
Accountability, accuracy, efficiency	1	QuestA (1)

Table 4.6: Identified needs in a coding tool for the IG (2/2)

Cluster	Count	Sources
Context-sensitive "code completion" (e.g., Attributes field should draw on previously entered Attributes and possibly even Objects)	1	QuestB (1)
Detection of stop words for removal in annotation process (e.g., "The farmer" should be coded as "Farmer")	1	QuestB (1)
Discourage manual typing and offer alternatives	1	Notes (1)
Facilities to support inter-coder reliability assessments (or at least preparation of data to facilitate this)	1	QuestB (1)
IG is already quite complicated, so a coding tool must not add to the confusion	1	Notes (1)
Not force coders to make an upfront decision about whether a statement is regulative or constitutive	1	Notes (1)
Prepopulated categories/tags for capturing IG syntactic components	1	QuestB (1)
Quizzes for self-study	1	QuestA (1)
Reminder to export data before closing browser window	1	QuestB (1)
Separate regulative and constitutive statements	1	QuestA (1)
Support all three levels of IG coding	1	QuestB (1)
Validation of coded statements	1	QuestB (1)
Visualize hierarchy of statement and components	1	QuestA (1)
Visually link similar categories	1	QuestA (1)

4.4 Discussion

With this study I have answered RQ1 and RQ2 of this thesis, to the extent possible at this stage in the development of the Institutional Grammar. My findings give a picture of the state of the art in policy coding tools as well as the needs of IG students and researchers in a new policy coding tool.

The ranking of the findings gives a rough idea of what features and needs are the most wanted and what disadvantages are the most reported among coders. However, given the small population of the study and crude weighting method, this ranking should not be relied upon completely. All of the findings should be given consideration in the design of IG Coder.

As touched on before, advantages with existing tools can be regarded as features a new coding tool should retain, i.e., needs. Furthermore, disadvantages can simply be inverted to generate features or needs. Most of these features manifest as functional requirements, whereas a few are non-functional (e.g., the response "Accountability, accuracy, efficiency"). However, this does not mean I will directly adopt these findings as requirements for IG Coder. I must first and foremost design IG Coder according to RQ3, i.e., in a way such that it can be tested as a coding interface, and I must prioritize its requirements according to the time frame of this thesis. The requirements engineering process is described in Section 5.3.

Three needs were identified that were categorized as essential. It should be noted that they were deemed essential only because they had a count of 3, and that all the needs should be given consideration because of the negligible difference in count. The first, "Ability to export coded text in different formats (e.g. Excel CSV)" was reported exclusively by researchers in Questionnaire B. This is related to a disadvantage with INCEpTION, "Missing output/export of coded text into a usable format", which was reported by two researchers in Questionnaire B. For researchers to make use of their coded data it must be in a usable format, such as CSV (the standard spreadsheet format).

The second of the essential needs is "An overview area underneath each statement with organized fields for IG components to visually double-check one's coding". This was reported in both of the questionnaires. Microsoft Excel fulfills this need to an extent, as a coded statement is organized into separate fields for syntactical components. INCEpTION does not offer such a visualization of the coded data, as supported by the disadvantage is "Graphic representation of coded data is unreadable" (count of 2). As institutional statements can be quite complex, I agree that a readable visual representation is important in a coding tool.

The final essential need, "Color coding for components distinguishing between regulative and constitutive" was reported in Questionnaire A and my coding course notes. In other words, it was reported mostly by students who were learning the IG. This is related to the previous need as color coding is a powerful tool for visualization. Given a coding interface in IG Coder, this would be a low-hanging fruit, easy to implement and to great effect.

Note that some of the findings contradict each other to an extent, such as the ad-

vantage "Quick and easy to populate cells that contain the same information by copy-pasting" (count of 2) and the disadvantage "Time-consuming and tedious to code by copy-pasting" (count of 5), both with Microsoft Excel. These two clusters are not directly equivalent, as I interpret the first to mean copying cells that are intended to hold the same information, and the second to mean coding new content. This shows that coders can disagree on what is the best way to code, which is important because it means there is no gold standard, no single solution that will please everyone. I am under no illusion that IG Coder will be the end-all be-all of policy coding tools, but I believe I can prototype a viable alternative to the current tools.

On another note, the top reported advantage with Microsoft Excel is that it "Represents data in an organized fashion". With a count of 8, this cluster is far more popular than most other clusters. It appears that the students from Questionnaire A value the organized nature of Microsoft Excel, as they reported this 6 times. However, none of the researchers from Questionnaire B reported this.

Compare the above with the Microsoft Excel disadvantage "Difficult and cumbersome to code and keep track of nested statements". These two findings conflict because tabular and nested data structures are fundamentally incompatible. Microsoft Excel and INCEPTION are designed for their respective data structures, and as touched on above, coders have different personal preferences. Therefore, in IG Coder I should find a way to capture the best of both worlds of tabular and nested data structures as well as to make it flexible with regards to coding preferences.

In summary, many interesting and useful ideas about policy coding tools were identified in this study. I am unable to point out any one finding as more important than the rest, in part because of the limitations of the ranking method and also because I believe all the findings are worth considering. In the next chapter I describe the development of IG Coder, including its system requirements which take the results of this study into consideration.

Chapter 5

Development of IG Coder

5.1 Initial State

The web application IG Coder was first started as a student project in the course Applied Computer Science Project¹. Christopher Frantz's (my thesis supervisor) project was to prototype a web application for policy coding, which two fellow students and I took on. We made several technical and architectural decisions at this stage which will be elaborated on later in this section. The first version of IG Coder featured a rudimentary implementation of the Institutional Grammar pre-IG 2.0, a user interface with a tree visualization of what a coded statement might look like and a simple backend set up with a graph database. While its functionality was very limited, it was the first step toward a brand new policy coding tool.

As a side note, the naming of the application was not a conscious choice; IG Coder came naturally to us.

The implementation of the Institutional Grammar in IG Coder has been largely my responsibility and is a significant part of my contribution to the application. It is hereafter referred to as the *data model*. IG 2.0 was a work in progress at the time of the aforementioned course, and the first version of the data model reflected this. Moreover, this version had several flaws in its design as I was learning about the IG at the same time. The current version of the data model is discussed in Section 5.6.2.

The IG Coder project was carried over to the course Integration Project². Here I also worked with two other students, one the same as in the first course. This course emphasized integration of technologies and innovation, and we further explored the graph database as well as implementing some simple add functionality on the backend. On the frontend we further developed the tree visualization and made simple dialogue boxes for editing the different nodes, each of which represented a different element of the tree. The data model saw minor improvements only.

¹Course code: IMT4886

²Course code: IMT4807

In 2020, one other student and I worked on IG Coder as a summer project. The goal of this project was not to develop the coding interface but build a user management system around it with coding projects and levels of permission; it was thus dubbed Management Project. As part of this, we designed and built an SQL database and implemented a backend API for it. Our intent was to store the coded statements themselves in the graph database and everything else in the SQL database. On the frontend we implemented user sign-up and sign-in, and designed (but did not fully implement) pages for project and document management.

Finally, during fall 2020 I worked some more on IG Coder outside of course work. First and foremost, since an early version of the IG 2.0 codebook (Frantz & Siddiki, 2020) had been published at this point, I set out to overhaul the design of the data model. This was both to update it to IG 2.0 and fix the many mistakes I saw. I refer throughout to the new data model as version 2, in line with the naming of IG 2.0. This work was primarily writing requirements for the data model but I also started implementing it. This consisted of creating stubs for new classes that would replace old ones, moving the class that held raw statement text higher up in the hierarchical data structure and creating a stub class for constitutive statements which the old data model did not support. The rest of the implementation I left to my thesis project. Other than that, I made a small addition to the Management Project by writing unit tests for the API endpoints.

This brings us to the start of this thesis. The initial state of IG Coder in this thesis was a largely unfinished application, yet conveniently an existing codebase I was intimately familiar with.

5.2 Technical Design

In this section I detail IG Coder’s technology stack and the decisions that were made before the start of this thesis. I will focus on the frontend because the IG Coder prototype developed in this thesis does not use a backend, a decision which is explained in Section 5.3. I use the term *code* in noun form to mean software code, whereas the term *coding* still means encoding and has nothing to do with software development.

IG Coder is a React³ app bootstrapped using Facebook’s Create React App⁴ scripts. The decision to use React as a frontend framework was made in the Applied Computer Science Project course and the primary reason was its popularity. React was ranked the number one frontend framework in the State of JavaScript 2019 survey (State of JavaScript, 2019). I also note that both of my teammates had worked in the development industry, so being less experienced myself, I had a certain level of trust in their suggested technology choices.

Originally, IG Coder’s frontend was written primarily in JavaScript and only the data model was written in TypeScript⁵. Over the years the frontend was gradually

³<https://reactjs.org/>

⁴<https://github.com/facebook/create-react-app/>

⁵<https://www.typescriptlang.org/>

converted to TypeScript, and at the start of this thesis only a few remaining files had the .js extension as opposed to .ts. However, even at the end of this thesis the frontend source code is not fully typed using TypeScript, though mostly.

TypeScript adds optional, static type definitions to JavaScript and is compiled to JavaScript. The developer annotates their code with types, which are then validated during compilation. In the Applied Computer Science Project course, we decided to use it to make the data model more robust. Part of what TypeScript offers is interfaces, which are blueprints for JavaScript objects with type annotations and which we believed would make it easier for us to work with the data model throughout the application. Type annotations generally improve the clarity of the code and its intent as well as making debugging easier, and for these reasons we decided to make use of TypeScript more and more in our frontend code. In developing the prototype for this thesis I have continued to type annotate my frontend code because I value predictability and consistency in my code.

IG Coder’s frontend uses the Redux library for managing application state. Specifically, it uses the React-Redux⁶ bindings. Redux offers a centralized state container and mechanisms for updating immutable state, and React-Redux offers APIs that enable React components to interact with Redux state. The main benefits of immutable and centralized state are improved predictability and consistency in the application’s behavior as well as making it easier to test. The Create React App scripts support a Redux option which automatically generates stub code for Redux in the new application.

Using Redux was another decision we made during the Applied Computer Science Project course. During my bachelor’s project, which also revolved around creating a web application, my group also used Redux for state management and I had a positive experience with it, which is why I agreed to use it for this project.

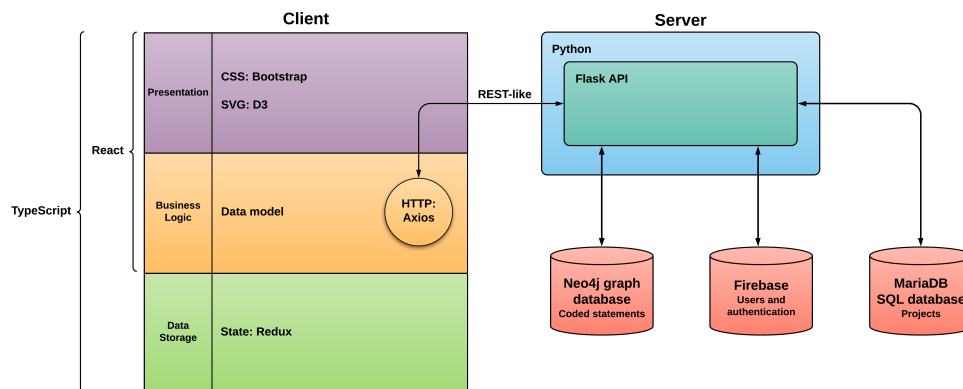


Figure 5.1: System architecture of IG Coder

Figure 5.1 gives an overview of IG Coder’s system architecture with key technologies, including the databases that are part of the Management Project. The back-

⁶<https://react-redux.js.org/>

end server is written in Python, and Flask⁷ is used to build the backend API. Communication between the client and server is based on the REST architectural style (Fielding, 2000), but since it does not follow all REST principles it is strictly an HTTP API. The Management Project uses Google Firebase⁸ for user management and authentication, while it has a MariaDB relational database to store projects and documents as well as associate these with users and coded statements in the other databases. Not part of the Management Project is the graph database Neo4j⁹ for storing coded statements, though this component of the system is the most incomplete.

To describe the technologies used in the frontend, I separate it by presentation, business logic and data storage layers. The client application is written in TypeScript, and both the presentation and business logic layers are built using React. To explain more closely, the frontend consists of React web components, each of which contains code for both presentation and application logic. On the presentation layer generally, the Data-Driven Documents (D3) library¹⁰ for manipulating Scalable Vector Graphics (SVG) is used for visualizing coded statements in graph form. For styling the user interface IG Coder uses Bootstrap CSS via the React-Bootstrap¹¹ library. The user interface design will be explained further in Section 5.5. Next, the business logic layer features the data model which is a set of classes for coding institutional statements and which will be explained in depth in Section 5.6.2. Also part of business logic is the Axios HTTP client¹², used for HTTP communication with the backend server. Finally, the data storage layer is built with React-Redux.

Ever since the first course, the IG Coder project has been housed in a private GitLab repository, including the Management Project which previously resided in its own branch, but now has been merged into the development branch. Thus, all team members have been using Git¹³ for version control. At the end of this thesis, the repository was cloned to a public GitHub repository which is linked in Appendix G.

5.3 Requirements

The system requirements for the IG Coder prototype in this thesis were determined first and foremost to allow me to investigate and answer RQ3. At the same time, they were guided by my findings on RQ1 and RQ2, the method of which will be described in this section. As the first step of the requirements engineering process, I formulated the following objective for the prototype:

⁷<https://flask.palletsprojects.com/>

⁸<https://firebase.google.com/>

⁹<https://neo4j.com/>

¹⁰<https://d3js.org/>

¹¹<https://react-bootstrap.github.io/>

¹²<https://github.com/axios/axios/>

¹³<https://git-scm.com/>

Provide a rich user interface that allows for the coding of raw institutional statements with IG 2.0.

Note that I do not refer to this prototype as a Minimum Viable Product (MVP) even though the term fits a prototype made for a specific objective. I did not have an MVP mindset when developing the system requirements because both my supervisor and I wish to take IG Coder further as a policy coding application in the future. For this reason and the fact that IG Coder already had both a frontend and backend it did not make sense to impose strict limitations on it.

A set of requirements for the prototype were still necessary, of course, so next I made a number of decisions based on the aforementioned objective. This was done during the design phase of this thesis but before I had completed my review of current tools. These decisions can be regarded as high-level requirements, whereas I defined lower-level requirements following the aforementioned review. My decisions at this stage, along with my reasoning behind them, were as follows:

The Management Project will be excluded from this prototype. User, project and document management are not necessary in a prototype user interface.

The prototype should work without a backend server. I have limited time for development in this project. A prototype user interface can function without a backend, and thus I can save a great deal of time I would otherwise have spent on a component of the application that users never see anyway. This will furthermore allow me to safely skip TLS encryption of the web server (see Section 5.7). For the purposes of the prototype, I know I can rely on the browser for simple data storage. I am aware that the lack of a backend means NLP tools will not be available, which means the prototype cannot process coded data using NLP technologies.

Support all of IG Core, IG Extended and IG Logico. They are all part of IG 2.0, have different use cases and are all expected to have high uptake. However, I prioritized IG Logico lower than the rest in case I was unable to cover them all due to time constraints.

- IG Core – *must have*
- IG Extended – *must have*
- IG Logico – *should have*

Support constitutive as well as regulative statements. As a major addition in IG 2.0, constitutive statements are crucial for a coding tool to support.

Implement export and import of coded statements. In addition to browser storage, this serves as a save/load mechanism and allows coded data to be shared. Support the following formats:

- JSON (native format; export and import) – *must have*
 - A JSON schema for the native format should be offered.

- CSV (for spreadsheets; export only) – *should have*
- UIMA CAS (for text annotators; export only) – *should have*
- Shorthand (for reading; export only) – *nice to have*

Support viewing of all entries in a document in addition to coding of each entry.

For the user tests, a list of entries should be given for the users to code.

Do not support creation nor deletion of documents. A document with entries should be pre-made for the user tests.

Some of the decisions above are labelled with one of "must have", "should have" and "nice to have". This indicates the decision's priority from highest to lowest. Decisions on the top level were not prioritized, as all were considered necessary. The decisions were in part decided based on responses to Questionnaire B, all of which I had received at the time. Specifically, the list of export formats came from these responses. Additionally, my supervisor advised me on what would be important for the new tool, such as full support for IG 2.0. It was expressly my decision to leave out the backend, a result of concerns about the time frame for development because my past experiences have taught me to be conservative in estimating development times.

Before this thesis and in the very beginning of it, my supervisor and I discussed a few possible features of the coding interface that would assist the user when coding text. One was auto-completion, i.e., when coding an `ATTRIBUTES` component the interface would suggest previously coded `ATTRIBUTES`. The other was automatic detection and removal of stop words, i.e., to code "The Program Manager" as "Program Manager". The latter required NLP tools to implement and thus needed to be performed on the backend. When I decided to leave out the backend from the scope of the prototype, I therefore also had to leave out this feature. The former feature was brought into the lower-level requirements, only as suggestions instead of auto-completion, as described later in this section.

Two practical concerns related to development emerged from these decisions. First, in the codebase, the Management Project branch had been merged into the development branch because I at one point before this thesis believed I would need the backend API as part of this prototype, and the former branch held the newest version of the API. Therefore, all functionality related to the Management Project needed to be disabled for this prototype. I solved this by adding switches, or flags, in the application's central state and implemented checks of these switches in all the relevant code, allowing the Management Project to be turned on and off as necessary. I ensured that switching it off would disable all backend server requests, because that was

Second, the coding interface for this prototype had to be rebuilt from scratch, for several reasons: the data model had changed too much in version 2, the original node editors (dialogue boxes for editing each element of a tree) were inconsistent and poorly planned, and to make it easier to accommodate the new requirements. This brings us to the lower-level requirements coming from the review of current

tools. They were not comprehensive, only a list of features I decided to include in the prototype. These requirements were as follows:

Visualize coded statements as trees. This feature already existed since before this thesis but was to be updated to version 2 of the data model as well as enhanced in two major ways. The first was color coding for the different elements of the tree (different types of nodes) but not for different syntactical components nor differentiating between regulative and constitutive statements, as all of those would be labelled with text. The second was text labels for each node showing the coded text. This tree would fulfill the need for an overview of the coded statement with organized "fields" (nodes) for syntactical components as well as for visualizing the hierarchy between statements, components and properties. It would make it easier to revisit as well as review coded statements.

Each node in the tree should be clickable, bringing up a node editor. The idea of node editors has existed since the very first version of IG Coder. The different types of nodes are explained in Section 5.6.2 but the basic idea is each node editor allows the manipulation of the node's own fields as well as controlling its child nodes. This system dictates the coding workflow, and while I realized at an early point it might not be more time-efficient than the current coding tools, I decided to move ahead with it as an experiment for this thesis.

Offer alternatives to manual typing. Since manual typing as a coding method is prone to error, the interface should allow selecting and dragging text from the full statement to components. As an alternative to dragging, text should also be able to be copied and pasted into components. However, I decided not to support direct annotation of the text at this stage. (I still wanted to support this in the future, just not for the prototype.)

Suggestions of previously entered component values. Since manual typing was to be discouraged as a coding method, I decided it would be unnecessary to implement auto-completion but the interface should still show a list of suggestions. I furthermore decided that since an actor in one statement can be an object in another, ATTRIBUTES and OBJECT as well as their constitutive counterparts should all draw on the same pool of previously entered component values.

Allow rephrasing of components and entire statements. The interface should allow the coder to rephrase the entire statement and code that version, while also allowing them to rephrase individual components.

Accommodate implicit components. The interface should allow explicitly specifying the values of implicit components. This feature should be able to be used in combination with the above (rephrasing of components).

Offer help text in each node editor. This feature did not come from the review of current tools but is something I came up with around the same time. I realized it would be necessary to explain the different node types (i.e., different elements of a tree) to the users. Therefore, each node editor should display a short help text which briefly explains the node type and serves as quick reference for the available actions.

As is apparent, the system requirements for the prototype were not all that detailed. I did not write low-level use cases because I felt it was not necessary as the sole developer on this project. I had decided on the requirements myself and was going to implement them myself, so I had my own ideas about what exactly I would implement. I know detailed specifications are necessary for communicating requirements in projects with multiple people.

5.4 Development Process

As the sole developer on this project, my development process was rather straightforward. To manage the work, I took the requirements described in Section 5.3 and split them into manageable issues, which I placed in my backlog. I kept a personal implementation log which was divided into backlog, current issues and completed issues. Underneath each issue I kept all my personal notes about its implementation, including lists of possible approaches to a solution and a brief summary of my final solution for future reference. I also noted down bugs with reproduction steps, causes and solutions. This way, all the information I discovered during my work on each issue was accessible for future reference.

I worked on the issues sequentially, for the most part. As a rough outline, I began by updating the data model implementation to version 2, then updated and refined the tree visualization, and finally built the node editors. However, I ended up revisiting a small number of issues because I later found better (i.e., more efficient or elegant) solutions. When I got stuck on an issue I moved on to the next to give myself time to think about the former.

During the development phase, I had a two-hour meeting with my supervisor every two weeks. No other people were involved in the development process. In these meetings I demonstrated the features I had implemented since last time and he gave me feedback, most of the time from the perspective of a user. Even though he is a computer scientist, we spent very little time discussing technical solutions. After each meeting, I implemented his suggestions before resuming work on issues, showing I was able to respond to and adapt to feedback in an agile way. Granted, the feedback was for the most part lists of small or even minute enhancements, but a handful of items required several hours each to implement. The largest feature that was introduced because of feedback from my supervisor was a preferences panel with three configurable settings pertaining to the coding interface. These were not part of the system requirements but were added during development of the user interface, after my supervisor suggested them.

I used Git¹⁴ for version control, like I had been doing since the very beginning of IG Coder. The IG Coder repository has two main branches: master and development, where master is for the production build while development is for work. There is also a branch for the Management Project but it was merged into development and has not been in use since. It is kept in case that specific project is taken up again.

When I was in a team working on IG Coder, we used a branching model in which a new branch is created for every issue that is being worked on. After the issue is completed, a merge request (also known as pull request) is submitted, another team member reviews the code and when it is accepted, the branch is merged into development and deleted. This method allows team members to coordinate their changes. However, when working on a project alone such coordination is not necessary. Therefore, during the development phase in this thesis, I did not create a new branch for every single issue. I did, however, work on a branch separate from development, which I merged into the latter every 2-3 issues. I was aware that this meant multiple issues were bundled into each commit, so I made an effort to make each commit message descriptive of the changes in that commit.

Since IG Coder was only a prototype and a work in progress, I did not write comprehensive documentation for it in this thesis. The data model was and is the most well documented component of the application. The project's GitLab repository has a wiki which contains documentation of the backend API and the Management Project but very little of the client, except for a page about the data model. This page specifies the design of the data model in detail, including class descriptions with fields but not data types, as well as rules for how the classes should be used to comply with the IG 2.0 specification. It is shown in Appendix D but I discuss it further in Section 5.6.2.

While there is limited formal documentation of the client, all the code I wrote in this thesis is tidy and fairly well commented, in my opinion. All classes and nearly all functions are documented with JSDoc¹⁵.

5.5 User Interface Design

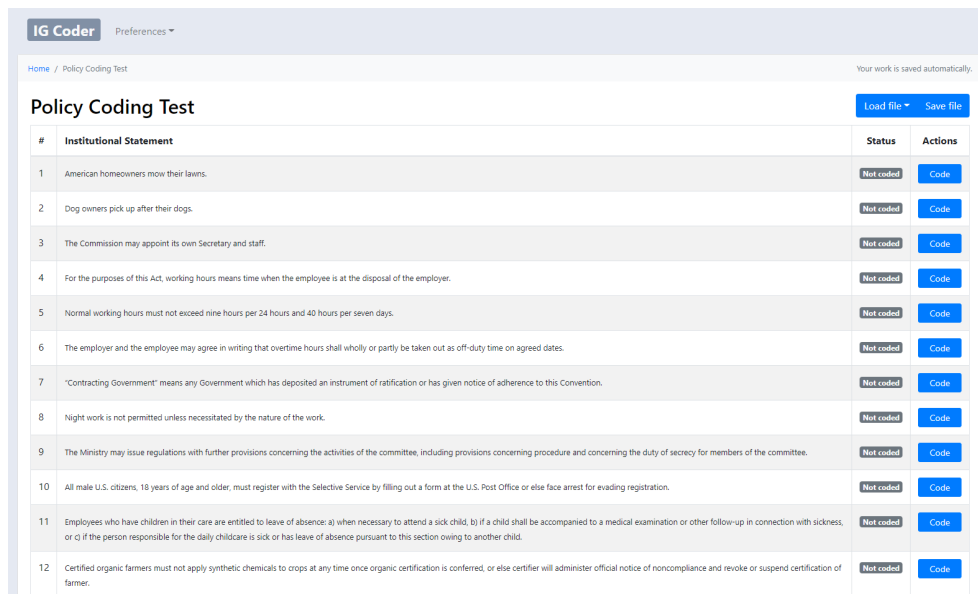
Since the very first version of IG Coder, there has been a base user interface (UI) with a background, a navbar along the top and a body container. As I mentioned in Section 5.2, IG Coder uses Bootstrap for styling the interface (specifically, React-Bootstrap which wraps Bootstrap components in React components). These reusable components make it easy to compose a user interface with a consistent style, so that I can focus on functionality.

Thus, I created UI for each feature as part of implementing it. This means I never made drawings of what the user interface for the prototype should look like. While I had some ideas about the user interface in mind, I only made some small notes

¹⁴<https://git-scm.com/>

¹⁵<https://jsdoc.app/>

in my implementation log and spent very little time planning the user interface. Figure 5.2 shows the page listing all statements. For this page I thought the most practical solution to display the list would be a table. This would allow me to add columns if more information needed to be displayed for each statement, and this turned out to be useful since I added the "Status" column much later. While this page shows the list of test statements, it is intended to represent a specific document (i.e., policy or regulation) containing a chronological list of statements. This page also contains the "Load file" and "Save file" buttons, since they pertain to the document as a whole.



#	Institutional Statement	Status	Actions
1	American homeowners mow their lawns.	Not coded	Code
2	Dog owners pick up after their dogs.	Not coded	Code
3	The Commission may appoint its own Secretary and staff.	Not coded	Code
4	For the purposes of this Act, working hours means time when the employee is at the disposal of the employer.	Not coded	Code
5	Normal working hours must not exceed nine hours per 24 hours and 40 hours per seven days.	Not coded	Code
6	The employer and the employee may agree in writing that overtime hours shall wholly or partly be taken out as off-duty time on agreed dates.	Not coded	Code
7	"Contracting Government" means any Government which has deposited an instrument of ratification or has given notice of adherence to this Convention.	Not coded	Code
8	Night work is not permitted unless necessitated by the nature of the work.	Not coded	Code
9	The Ministry may issue regulations with further provisions concerning the activities of the committee, including provisions concerning procedure and concerning the duty of secrecy for members of the committee.	Not coded	Code
10	All male U.S. citizens, 18 years of age and older, must register with the Selective Service by filling out a form at the U.S. Post Office or else face arrest for evading registration.	Not coded	Code
11	Employees who have children in their care are entitled to leave of absence: a) when necessary to attend a sick child; b) if a child shall be accompanied to a medical examination or other follow-up in connection with sickness; or c) if the person responsible for the daily childcare is sick or has leave of absence pursuant to this section owing to another child.	Not coded	Code
12	Certified organic farmers must not apply synthetic chemicals to crops at any time once organic certification is conferred, or else certifier will administer official notice of noncompliance and revoke or suspend certification of farmer.	Not coded	Code

Figure 5.2: IG Coder: List of test statements



IG Coder Preferences

Home / Policy Coding Test / Entry 1

Your work is saved automatically.

Raw

American homeowners mow their lawns.

Rephrased

To start coding, choose a statement type.

Regulative Constitutive Statement combination

Figure 5.3: IG Coder: Uncoded entry

In Figure 5.3 we see the entry page (in IG Coder, an entry represents an institutional statement, as described in the data model specification). It shows the raw statement, which is read-only but allows the statement to be rephrased after clicking the "Rephrased" bar. When no coding exists for the entry, a new coding can be started by creating a new root node, which has three possible types.

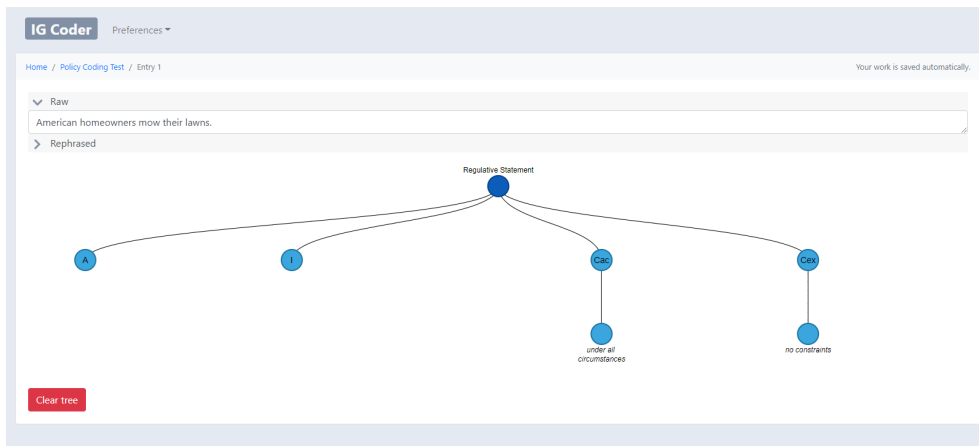


Figure 5.4: IG Coder: After creating a root node

Figure 5.4 shows the entry page after clicking the "Regulative" button in the previous figure. Creating a Statement node of regulative or constitutive kind automatically creates Component child nodes for its mandatory components. This is where the D3 tree visualization comes in. The tree is an SVG graphic made of SVG circles, lines and text boxes, and is generated as well as styled by D3 code. This means Bootstrap cannot be relied on for styling the tree. This page also contains a button "Clear tree", which upon confirmation deletes the coding.

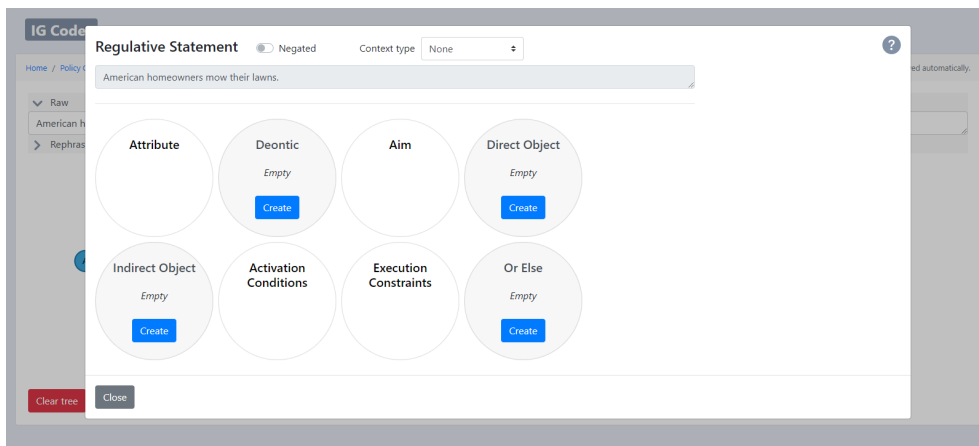


Figure 5.5: IG Coder: Statement editor

The next figure, Figure 5.5, shows the result of clicking on the Regulative Statement node in the previous figure. When clicking on any node, this dialogue box (hereafter referred to as a node editor) appears on top of the entry page. In the editor, both kinds of Statement node display a grid of their child nodes, where optional components are absent by default but can be created and deleted. Clicking on a child node takes you directly to that node's editor.

All node editors have a question mark in the upper half, which when hovered over displays a block of help text in the blank area below it. Furthermore, the upper bar shows the node type and component type if applicable as well as a number of controls depending on the node type. Nodes of all types can be marked as Negated, and most node types can be labelled with a Context type (of IG 2.0's Context Taxonomy).

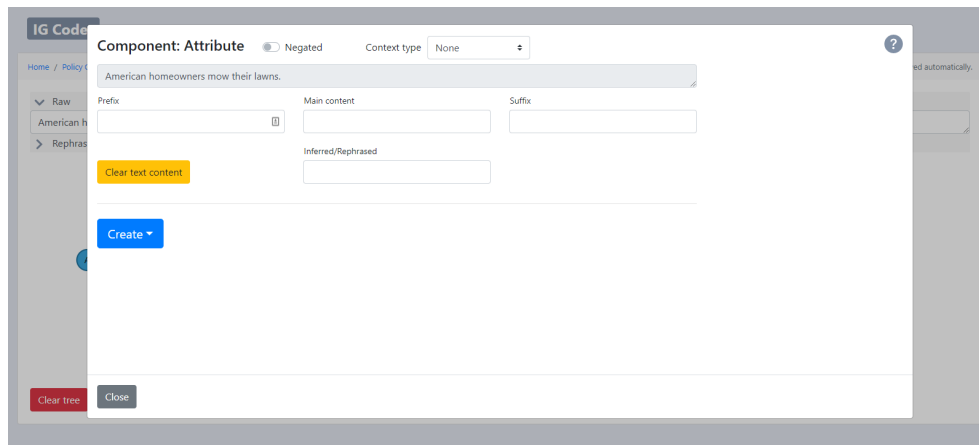


Figure 5.6: IG Coder: Component editor

Figure 5.6 shows the node editor of the Component node of type Attributes in the tree from Figure 5.4. Component, Junction and Property nodes can all have text content, which the four text fields are for. The full statement is shown in a read-only field, and text can be selected and then dragged or copied into the text fields.

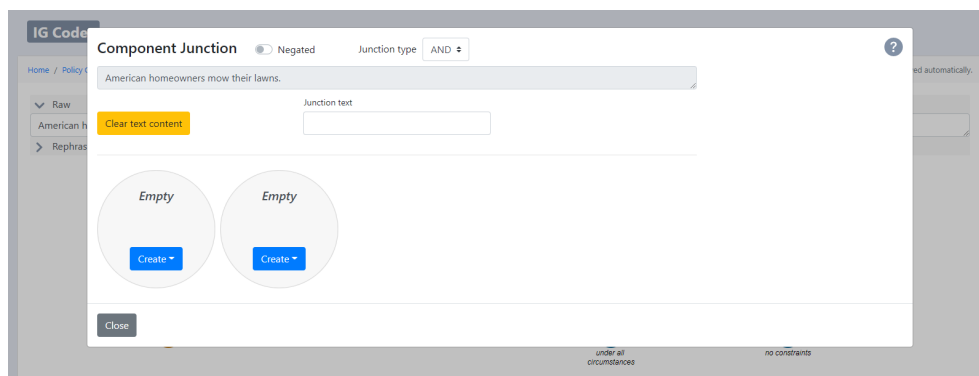


Figure 5.7: IG Coder: Junction editor

Moving on, Figure 5.7 shows the node editor of a Junction node, which is not pictured in Figure 5.4. A Junction node represents the logical combination of its two children, and thus two empty slots for child nodes are shown. Only one text field is available, and the upper bar allows its logical operator to be set.

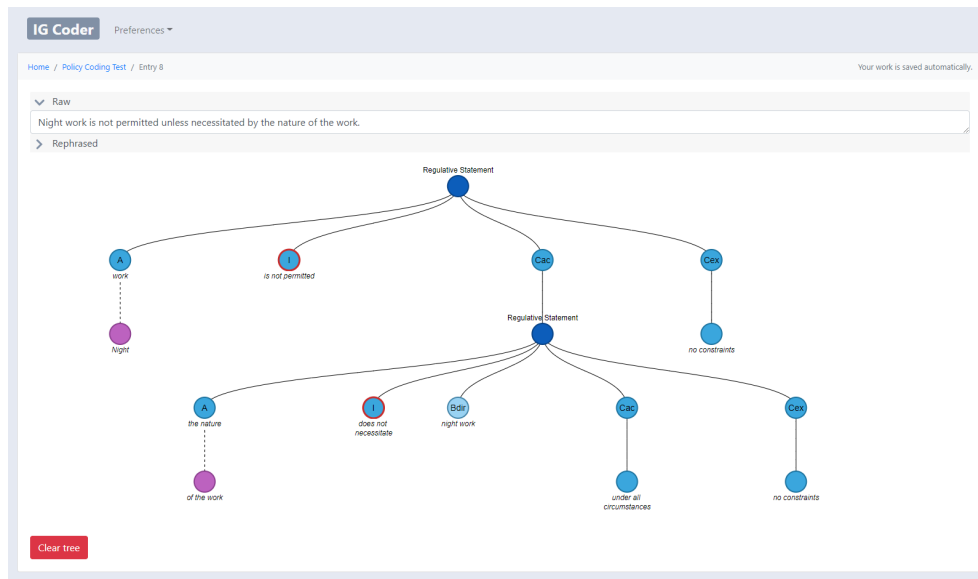


Figure 5.8: IG Coder: Fully coded statement

Finally, Figure 5.8 shows the tree representation of the fully coded example statement. Activation Conditions (Cac) and Execution Constraints (Cex) have text labels by default when no constraints have been coded. Nodes outlined in red are marked as Negated. Hovering over any node in the tree displays a tooltip containing node type, component type if applicable and text content if applicable. For Property nodes (purple), dashed lines mean the Property is functionally independent from its parent node. A Property node can be set to functionally dependent on its parent node, in which case the line to its parent node becomes solid, and back. The Property editor is not shown among these images, because it is very similar to the Component editor.

I reiterate that using Bootstrap made it easy to have a consistent style across the app. I only had to ensure the D3 tree visualization fit in as well as being easy to read. The user experience (UX) could certainly be improved as I am not a UX designer, but that is a matter for the future.

5.6 Implementation and Tests

5.6.1 Tools

For developing IG Coder in this thesis, I used JetBrains' WebStorm¹⁶ as my IDE. It is a professional IDE with helpful facilities for working with both React and TypeScript, and is free for students.

As described in Section 5.2, IG Coder is a React app with built-in React scripts.

¹⁶<https://www.jetbrains.com/webstorm/>

During development, I used the `start` script to run a development version of the app, which compiles the TypeScript code and serves the app locally. Changes to the code automatically trigger a new compilation of the app and refresh of the page. For production (i.e., deployment), however, the `build` script should be used. This produces a `build` folder containing a bundled and optimized version of the app, ready for deployment.

I used the Node package manager (NPM)¹⁷ for dependency management. NPM is the largest software registry for JavaScript packages, and practically all the client-side technologies mentioned in Section 5.2 were installed into IG Coder as NPM packages. NPM offers a command-line interface (CLI) for managing packages, and furthermore lists all the package dependencies for an application in a file called `package.json`. This list is divided into main dependencies, which are required to build the application for production, and "dev" dependencies, which are only required during development (i.e., testing-related packages).

5.6.2 Data Model

In this section, I will primarily discuss the data model but also bring up some components of the prototype that are related to it. I see the data model as my main contribution to the IG Coder project, and therefore I do not detail the implementation of the rest of the prototype in this chapter.

The data model is my class-based implementation of IG 2.0 in IG Coder, and has a unique format for representing institutional statements. It is implemented after my specification, which is shown in Appendix D. I have been developing this specification since the very beginning of IG Coder, and its current version is v2, in line with the naming of IG 2.0.

The data model was created because IG Coder needed to facilitate coding of institutional statements, and in the very first course, my teammates and I explored the representation of statements in a tree structure. Therefore, the data model is built on a few fundamental ideas and concepts:

Institutional statements are represented as trees. My supervisor, who was our Product Owner in the very first project, advised us that this was a good choice in a new coding tool, presumably because he was familiar with the disadvantages of coding in a tabular structure. The primary advantage of such a tree structure is the ease of representing vertical and horizontal nesting. Therefore, the data model has a hierarchical structure, consisting of nodes which contain child nodes. The elemental node structure is described below.

Different node types represent different elements of the tree. This was a consequence of representing statements as trees: we needed to determine exactly how the trees are structured. Note that node types do not correspond to syntactical components of the IG. The Component node type represents

¹⁷<https://www.npmjs.com/>

components, and has a field specifying the type of component. The node types and their roles are described below.

Code listing 5.1 shows the TypeScript interface for the base Node class. Note the final field, `children`, which is an array of Node objects. This is the building block for a hierarchical structure: a Node points to its child Nodes, which again point to their child Nodes. To clarify, in JavaScript and thus TypeScript objects are passed by reference, meaning that in memory, child Nodes are not actually contained within their parent nodes. However, if printing the tree structure as a JavaScript Object Notation (JSON) string, they are.

Code listing 5.1: Interface for the elemental node in the IG data model

```
/**
 * The base contract for all Nodes
 */
export interface INode {
  /* ID of this Node, unique within its Document */
  id: number,
  /* ID of the Document this node belongs to */
  document: number,
  /* This Node's type/archetype/role in the statement tree */
  nodeType: NodeType,
  /* Whether this Node's meaning is negated */
  isNegated: boolean,
  /* Optional context type for using the Context Taxonomy on this Node */
  contextType?: ContextType;
  /* ID of the node this node is a child of (undefined if root) */
  parent?: number,
  /* The time and date this Node was created */
  createdAt: Date,
  /* The time and date this Node was last changed */
  updatedAt: Date,
  /* Array of child nodes of this Node */
  children: INode[]
}
```

The data model contains classes for different node types, all of which extend the base Node class. Summarizing the node type specification in Appendix D, the node types and their roles are as follows:

Statement Common base class for Regulative Statement and Constitutive Statement, not to be used directly.

Regulative Statement Represents a regulative statement, with a child Component node for each regulative component.

Constitutive Statement Represents a constitutive statement, with a child Component node for each constitutive component.

Junction Represents the logical combination of its two child nodes. Has a field specifying its logical operator and a field for text content. Common base class for the following three classes, not to be used directly.

Statement Junction Represents a side-by-side combination of two Statement nodes.

Component Junction Represents a side-by-side combination of two Component nodes.

Property Junction Represents a side-by-side combination of two Property nodes.

Component Represents a regulative or constitutive component. Has a field specifying its component type and a field for text content. Depending on its component type, has a variety of rules for what child nodes it can have. For instance, the OR ELSE component type can have a Statement node as its child.

Property Represents an object or property in the Object-Property Hierarchy of IG 2.0. Has a field for text content. Can show up in the tree only as child of a Component node or of another Property node.

To build statement trees, i.e., code institutional statements in the data model, one first needs a Document object that contains a list of Entries. An Entry is a wrapper class for an institutional statement containing the root node as well as fields for the full statement text and rephrased text. The Entry class has a function to create a root node, and from there, all node classes have functions to create child nodes. All nodes have an ID as one of its fields, which are unique within each Document. As a temporary solution before fully implementing the backend, my original team wrote a simple counter class to increment a numeric ID for each node that was created in each Document. This works well as long as the application is client-only but will of course have to be replaced in a multi-user situation, since numeric IDs will conflict.

To ensure the tree is syntactically correct in IG 2.0, all node creator functions have built-in checks that validate the desired action in the current context. In other words, correctness is enforced while coding, not validated afterward. The various data model-specific errors that can be produced are enumerated in a custom error class.

Figure 5.9 illustrates how a complex statement can be represented (i.e., coded) in the current version of the data model. It exemplifies most features of the data model, excluding only constitutive statements, the OR ELSE component and context types. The original statement is given as part of the figure. Note that I built this figure before starting development in this thesis, so this illustration was an important step for me to figure out what IG Coder should do as well as look like. For instance, the figure shows the color coding I decided on for the different node types, which I then emulated when implementing the user interface. To a high extent, this illustration served as a plan for the tree visualization in the user interface, but I note that a very basic D3 tree existed in IG Coder before this thesis, which I built the illustration on.

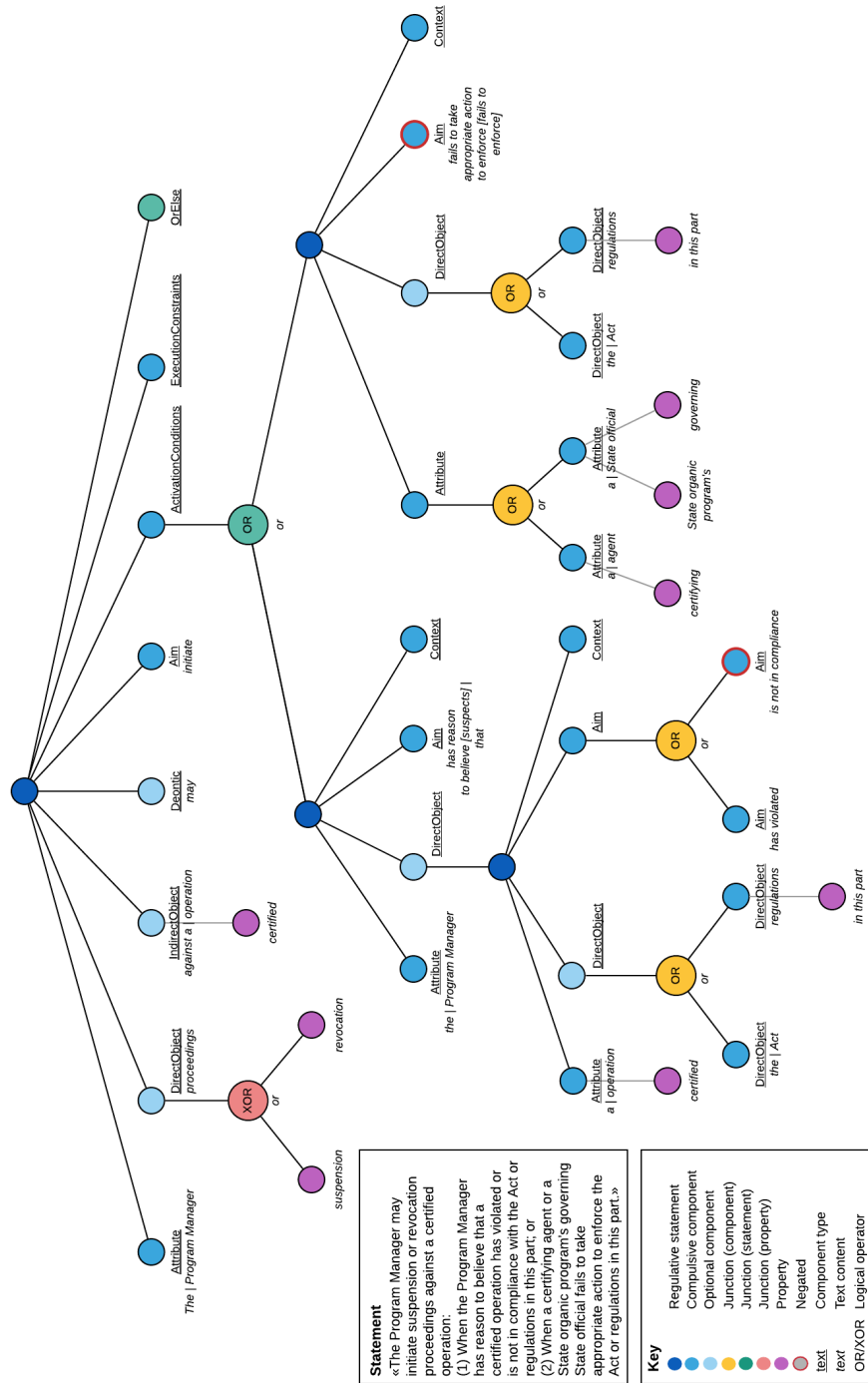


Figure 5.9: Sample tree representation of a statement with the IG data model

The data model is at the core of IG Coder, providing the data structure with which coded institutional statements are manipulated, stored and exchanged. The central state of the application stores the working document in its current state, and this state is furthermore persisted to browser storage upon every change. Thus, the user's work is saved automatically in the browser, which I find very convenient. However, since the user might lose this data, e.g., if they have to reinstall their browser or use a different machine, I needed to offer an alternative storage method.

Therefore, one of the system requirements given in Section 5.3 was to implement export and import of coded statements. To implement these functions, I simply print the tree structure to a JSON file and download the file, and conversely let a file be uploaded and build a tree structure from it. For the purposes of the prototype, the entire Document is exported and imported. My intent with these was as a save/load mechanism in the absence of a backend database, as well as an alternative solution to the browser storage on the client's machine. While work is saved automatically in the browser, the save/load mechanism backs it up.

Furthermore, the export/import functions allow documents to be shared between users. After the user tests, which are detailed in Chapter 6, I was able to import my participants' coded documents into IG Coder and inspect how they coded the sample statements. I elaborate on this in Section 6.2.2.

Unfortunately, due to a lack of time I was not able to implement all the requirements listed in Section 5.3. Most importantly, I did not have the time to implement export in any other formats than the native data structure. I bring up and discuss these missing requirements in Chapter 7.

5.6.3 Tests

This section will describe the unit testing of the prototype. User testing is conducted in a separate study, described in Chapter 6.

To facilitate unit testing in IG Coder, I used `ts-jest`¹⁸, a library building on `Jest`¹⁹, a JavaScript testing framework, providing full TypeScript support to it. I focused on writing automated unit tests for the data model, and did not have the time to write such tests for the user interface. I instead tested the user interface manually, and my supervisor tried it out at the end as well. The prototype was small enough that this worked for us. I tested the prototype in Google Chrome, Mozilla Firefox and Microsoft Edge, all of which I had access to. I did not have access to Safari, however, so I used an online service to test in this browser.

The unit tests covered practically all the functions in the data model. However, the child node creators were numerous, so they were tested by simply calling them, storing the returned child node in a variable and then calling a function on the child node. If something was wrong, a TypeScript error or data model error would be produced and the test would fail, showing me this error. Other functions were

¹⁸<https://github.com/kulshekhar/ts-jest>

¹⁹<https://jestjs.io/>

more thoroughly tested, verifying their output with different inputs. There was furthermore a large test which ran dozens of data model functions in sequence, building a statement tree using all possible functionalities, the way it would be done in the coding interface.

While the unit tests in combination called all the data model functions, they did not check every possible case with every possible input, as there are simply too many possibilities. Again, I found that this level of testing was sufficient for the size of the prototype and the time frame in this thesis. In the future, however, it will be advantageous to expand the unit tests prior to expanding the functionality of the application.

5.7 Deployment

To deploy IG Coder to users for the evaluation described in Chapter 6, I used the virtualization platform OpenStack via NTNU Gjøvik's installation, SkyHiGh²⁰. Among other services, it offers virtual machines that can be connected to NTNU's internal as well as external networks. My supervisor and I have a space in SkyHiGh for the IG Coder project.

In this space, I set up a virtual machine and pulled IG Coder's Git repository. Once I had finished the prototype, I built the newest version of the app as described in Section 5.6, producing a `build` folder. I then used the open source web server NGINX²¹ to statically serve the contents of the `build` folder on the machine's HTTP port and set this as the default access port. Finally, I claimed a public IP address from SkyHiGh's pool and allocated this to the machine, enabling public access to the app. I shared this IP address with the test participants.

The virtual machine had the following ports open for ingress (incoming packets): 80 (HTTP), 443 (HTTPS) and 22 (SSH). Both my supervisor and I had SSH access to the machine.

Note that I did not set up TLS or SSL security on this web server. This is because the app runs entirely on the client and does not communicate with any remote server, meaning there is no network communication to encrypt. The web server only hosts the bundled and minified²² JavaScript code, the base HTML document in which the app's content is dynamically displayed and the accompanying CSS stylesheets. The JavaScript code runs on the client's machine. TLS encryption, even via free certificate authorities such as Let's Encrypt²³, requires a domain name which is rarely free. For all the aforementioned reasons, I saw no need to go to the trouble of this.

²⁰<https://www.ntnu.no/wiki/display/skyhigh>

²¹<http://nginx.org/>

²²Minification compresses code and markup by removing all unnecessary characters (i.e., whitespace and comments) and shortening variable names without changing its functionality, dramatically reducing its file size. This improves load times and bandwidth usage, resulting in a better user experience.

²³<https://letsencrypt.org/>

Chapter 6

Evaluation of IG Coder

6.1 Introduction

This chapter will conduct an evaluation of IG Coder, investigating RQ3 of this thesis. As a reminder, RQ3 is as follows:

Evaluation of the new coding tool

RQ3 To what extent does IG Coder satisfy the needs identified in RQ1 and RQ2?

RQ3a To what extent is IG Coder aligned with coders' understanding of institutional statements?

RQ3b How satisfied are users with the coding interface of IG Coder?

RQ3c To what extent can IG Coder improve the coding workflow?

As was done with RQ1 and RQ2, RQ3 and each of its sub-questions will be answered separately. Furthermore, while questions are phrased using "To what extent" and "How satisfied", I will answer them entirely in words and not using scales or other metrics, as mentioned in Section 2.3.

I also reiterate that by "the needs identified in RQ1 and RQ2" I mean the identified features a new coding tool should and should not possess, whether they come from an existing tool or not.

6.2 Method

The evaluation consists of two major activities. First, the IG Coder prototype is deployed to a group of testers whose task is to code a given set of institutional statements using the prototype. Second, I conduct a semi-structured interview with each of these testers to gain insight into their experience with the prototype and gather information with which I can answer the research questions. Section 6.2.1 describes how I recruited participants for the evaluation, and the two activities are detailed in Sections 6.2.2 and 6.2.3, respectively.

6.2.1 Recruitment

Before I could begin the evaluation of IG Coder, I needed a group of testers. Therefore, going back to Questionnaire B, there was a final question at the end (not shown in Appendix C) where I asked participants if they would be interested in testing a new policy coding tool. At this stage, I informed them that participation would consist of testing the tool with sample statements and participating in an interview with me. The question linked to a separate, small questionnaire for sign-up.

Additionally, my supervisor took it upon himself to contact a few people within IGRI who might be interested, and one other researcher contacted some of his students who had been working with the IG.

At the beginning of the evaluation phase I sent out an email to all prospective participants mentioned above. It included all the information they needed to participate: an information letter with detailed information on the evaluation activities and steps to ensure their privacy, an interview guide with possible questions for the interview, a link to the IG Coder prototype, a video tutorial of the prototype and instructions for the testing activity. I collected participants' consent in a separate questionnaire linked from this email. Asking for consent was necessary because I needed to record and transcribe the interviews, being on my own and unable to take sufficient notes.

The interview guide is shown in Appendix F and is used as a guideline for the semi-structured interviews, described in Section 6.2.3.

As mentioned above, I recorded a video tutorial of the completed IG Coder prototype. I populated the prototype with five sample statements, and in the tutorial I demonstrated how to code them. However, to save time, I only coded the first statement completely; for the rest, I coded their unique features only. Altogether, the video covered most features of the prototype but due to the small number of sample statements and time constraints it did not cover some of the more complex features. In terms of a technical solution, the video was a simple recording of my screen and microphone as I did not have the time nor knowledge to edit it.

The final list of participants would come from the responses to the aforementioned consent questionnaire. Having recruited participants, I could begin the user testing.

6.2.2 User Testing

To test the IG Coder prototype, I deployed it to the testers as described in Section 5.7, sending them the link by email as mentioned in Section 6.2.1. In the email I also gave the following instructions:

“Past the welcome screen you will see a table of statements. Your task is to code as many of these statements as possible. Following this, please download your coding using the "Save file" button and send the file to me.”

Prior to sending the email, I populated the prototype with 14 institutional statements which the testers were to code. These statements are shown in Appendix E.

I started with simple, atomic statements and gradually introduced IG 2.0 features such as the OR ELSE component, simple and complex properties, and vertical and horizontal nesting. A majority of the statements were regulative but I included a few constitutive statements as well. The last four statements were highly complicated, and because of this and the relatively high number of statements I formulated the instructions as "code as many of these statements as possible" instead of "code all statements".

After the testers had finished coding, they had been instructed to download the coding and send it to me. The "Save file" functionality of IG Coder was to be used for this (while I used the term "download", this functionality simply generates a file on the client and saves it). I could then load their coding into IG Coder myself using the "Load file" functionality and inspect it. This was not strictly part of the evaluation as I did not analyze their coded data in any way but having the codings available helped me discuss their experience with them during the interviews. This brings us to the interview part of the evaluation.

6.2.3 Interviews

As part of the email correspondence between each of the testers and I, we scheduled an interview following the testing activity. I encouraged them to make time as soon after the testing activity as possible, as far as their schedules allowed, so that their experience using the prototype would still be fresh in their memories. As mentioned in Section 2.3, these interviews were semi-structured. I used an interview guide, shown in Appendix F, as a starting point for asking questions. Participants had been given this guide in advance. The interview guide included RQ3 and its sub-questions either directly or indirectly but they were not highlighted or otherwise called out as particularly important. Having the interview guide close at hand during the interviews allowed me to be flexible, adapting my questions to the participant and steering the discussion in a useful direction while ensuring I covered the most important topics.

In practice, I ended up asking only around half of the questions in the interview guide. Furthermore, in all the interviews I skipped the questions labelled "Reviewing your tool of choice" because I feared they would take up too much time and because the participants' tool of choice came up in the discussions anyway. I also skipped the question "How did you code multiple properties on the same component?" because it turned out difficult to explain in a remote interview setting, even with examples. Outside of the interview guide I asked participants about their area of research.

In terms of practicalities, I hosted each interview in a Zoom online meeting except one which was hosted by the participant, also using Zoom. To collect data from the interviews I recorded them locally on my computer and wrote a transcription afterward, which participants had been informed of in advance and had consented to. In the next section I give a detailed summary of each interview conducted in this evaluation.

6.3 Results

6.3.1 Interview 1: Angelo Baldado

My first interview was with Angelo Baldado, an IGRI intern studying toward a Master of Public Administration at Syracuse University. As an intern, he has been part of research projects as well as internal discussions about the development of the Institutional Grammar.

I asked him some questions about his policy coding background, such as what tool he prefers. While Angelo has coded in both spreadsheets and INCEpTION, he prefers the latter. He also said he has not really performed analysis of coded statements yet, rather focused on testing and comparing the different methodologies, i.e., coding tools. For instance, he has taken part in an experiment to code the same policy in spreadsheets and INCEpTION, then compare the processes.

In addition to INCEpTION, he likes inline coding because it allows him to play with the statement as a piece of text as opposed to copying and pasting into boxes. He used the word *organic* to describe inline coding, and I can see that INCEpTION offers a similar kind of freedom as a text annotation tool. In terms of complexity level, Angelo tries to decompose as much as possible using IG Extended.

Moving on to IG Coder itself, I first asked him how he approached the coding exercises and he said he coded them all in one go. He later added he tried to do them in the frame of 30-40 minutes, as he had exams coming up at the time.

We then started discussing the workflow in IG Coder and comparing it to that in INCEpTION. Angelo came to the conclusion that INCEpTION is faster. I believe this is because essentially, one is tagging the statement directly and immediately as opposed to having to explicitly create a new statement node to code a nested statement, for instance. Thus, IG Coder requires more clicks than INCEpTION to perform the same action. When coding nested statements in IG Coder, such as *ACTIVATION CONDITIONS*, Angelo described it as a different “mental mode” because one has to explicitly make a new statement node, whereas in INCEpTION he only needs to “click and highlight”. He had some trouble figuring out how to code nesting and logical combinations in IG Coder as he did not know what it would look like in the tree nor how to make it the way he wanted.

What he liked about IG Coder is that it has built-in defaults, such as it shows that a regulative statement can have a Direct Object but does not have to. This reminds the coder about his or her options, which is helpful. He also really liked the tree visualization, saying he loved seeing the sentence structure that way and that it added some depth to his understanding of institutional statements.

One aspect Angelo found confusing was the placement of Junction nodes and their labels. As a reminder, a Junction node represents the logical combination of its two child nodes, meaning the two nodes being combined are shown below the Junction node. While he saw how it made sense, he found this difficult to read because he wanted to read the left child first and then the Junction, but in the tree the Junction came first. He later added that “it would be great to see it flip-flopped”, with the Junction shown below the two elements being combined.

Furthermore, Angelo suggested a minor enhancement to IG Coder. This was a button to go backward, i.e., to move directly from a node to its parent node. While from my perspective as a developer, this would require traversing the tree since a node does not have access to its parent, I can see how convenient it would be for users. This would make it easy to go back to a Junction node, for instance. I next asked him how he used the Prefix and Suffix fields. He liked the idea but ended up skipping them and lumping everything together. He again pointed out that he has not really performed analysis yet and could see how coding Prefix and Suffix could help filter out the noise. Bringing up a conversation in his research team, he pointed out that the decision whether to code prepositions and other stop words is often a matter of personal preference as well as time constraints. When asked about my choice of the terms Prefix and Suffix, Angelo said they were intuitive to understand but that thinking about it, there might be linguistic terms for them that could be even more appropriate.

In general, Angelo thought there was a lot to keep track of in IG Coder. He said this about creating Junction nodes, and that Prefix and Suffix were just another layer on top. I did not bring this up, but having to tick a box to mark a node as Negated is in the same category.

Near the end, I asked Angelo what INCEpTION did better than IG Coder. He reiterated that he had only spent 30-40 minutes on IG Coder and so was still learning it, but also that INCEpTION is decidedly faster. Contrasting the two, he said IG Coder “makes you think harder about structure” whereas INCEpTION lets you just code. He saw this as a good thing about IG Coder and came to the conclusion that it is a great tool for teaching and practicing the Institutional Grammar because of its focus on structure. It has a lot to keep track of inherently because of this focus. On the other hand, he was unsure whether it was a good tool for research analysis. He again raised the visual representation as a strength of IG Coder that INCEpTION does not have, since in INCEpTION the numerous lines tend to get confusing.

I followed up on this by asking whether if I could make IG Coder’s workflow faster, it would be an overall better tool. Angelo pointed out to me that this might not be the right way to look at it because in his experience, “better” is subjective. He said, “it’s about recognizing what is the strength and maximizing that”, and that IG Coder could be optimized more for teaching. He furthermore said IG Coder has a different way of seeing the IG and institutional statements.

Overall, Angelo was excited about the new tool and said he would love to see it taken further.

6.3.2 Interview 2: Dr. Ute Brady

My second interviewee was Dr. Ute Brady, a postdoctoral scholar at Syracuse University. As an environmental social scientist, her area of interest within institutional analysis is assessing policy design in international conservation treaties and governance.

I first asked about her policy coding background. Dr. Brady primarily uses spread-

sheets for policy coding but has tried INCEpTION as well. As part of her dissertation, she coded a large dataset of institutional statements, nearly 4,000, in Microsoft Excel because it was the only tool available at the time. She is thus very experienced with and accustomed to spreadsheet coding. INCEpTION, on the other hand, she finds hard to adjust to and finds its visual representation of coded statements hard to read.

When coding the dataset for her dissertation, due to the sheer number of statements she coded at a level of complexity akin to IG Core. She focused on identifying the constraints and did not decompose them unless absolutely necessary. Having performed spreadsheet coding for so long, Dr. Brady has developed her own approach for efficient coding. She told me she cuts the whole institutional statement, pastes it into each column and deletes all the text that does not fit in the columns. I imagine this is useful because in Microsoft Excel, selecting only part of the text in a cell to copy into another requires double-clicking the cell first, which is inconvenient.

We then moved on to discussing IG Coder. When doing the coding exercises in IG Coder, Dr. Brady tells me she tried to code them all in one go but got stuck in several places and so ended up jumping around the statements, coding intermittently over around five days. She had to consult the video tutorial to check out how to perform some functions.

I then asked her to compare her workflow in IG Coder to that in spreadsheets. Dr. Brady pointed out to me that this might not be a fair comparison since she has been coding in spreadsheets for a long time and any new tool has a learning curve. When we discussed the speed of coding in IG Coder, spreadsheets and INCEpTION, we agreed that INCEpTION is probably the fastest way to code. For atomic statements, IG Coder and spreadsheets are much the same but as statements become more complex, she wondered if IG Coder might not be more time efficient than spreadsheets, provided the coder has overcome the hurdle of figuring out IG Coder.

Dr. Brady tells me she found the tree visualization of statements very appealing, being a visual person. The tree made much more sense to her visually than a spreadsheet of multiple rows for one complex statement. It allowed her to easily see what she had coded and double-check her coding in a way that neither INCEpTION nor spreadsheets offer. She brought up that she has two very similar treaties where one is modelled after the other, and she pictured the coded trees side by side which would allow them to be compared visually. While I did not tell her this, it spurred me to think of program functionality that could compare two trees and quantify their differences, e.g., for assessing inter-coder reliability. This could be an interesting future direction.

We next discussed the output of the different coding tools, i.e., what file formats the coded data can be exported to. Dr. Brady is familiar with running statistics on spreadsheets using the R system¹. She furthermore said INCEpTION has no usable output for her, and we agreed that IG Coder needs some sort of output and that

¹<https://www.r-project.org/>

the most useful format would probably be CSV.

When I asked about the Prefix and Suffix fields in IG Coder, she told me she first mixed them with properties but understood them when she went back to the video tutorial. We agreed that the term "stop words" is useful for communicating the intent with the Prefix and Suffix fields.

Overall, Dr. Brady found IG Coder to be very intuitive for simple statements but struggled more with complex ones. In discussing her specific challenges, a deep-rooted issue with IG Coder came to light. I made a very early design decision in IG Coder that logical relationships are not to be decomposed and statements are to be coded as compactly as possible. This is because I wanted to keep the data structure, i.e., the representation of a coding, as small as possible. It is still possible to decompose relationships but not at all intuitive, which I realized when Dr. Brady pointed out there is no way to copy a statement. She was attempting to decompose a logical relationship into two statements that were identical save for the one component that had two values combined by a logical operator. It is not the first time I have encountered this issue, so this will be an important problem to solve moving forward.

Another issue she struggled with was deleting specific elements, or nodes. Unfortunately, for her it meant she had to delete the entire statement on a few occasions, which I do not think a user should have to put up with. This was because it was not intuitive to her that a node can only be deleted from its parent node. From a computer scientist's perspective it makes total sense but most users are not computer scientists, so this kind of feedback is very useful to me and appreciated. On a similar note, Dr. Brady pointed out to me that it might be helpful if I explain terms like "child node" further for most people.

Toward the end, we discussed two possible minor enhancements to IG Coder. First, Dr. Brady found that when she used the Rephrased field to rephrase a statement, in the node editor she could not tell if she was looking at the original statement or the rephrasing, and therefore she manually added brackets to the rephrasing. IG Coder could add these automatically to help the user. Second, she suggested it would be great if there was also functionality to export a tree as a graphic, such as PDF or JPEG. This would be a way to compare similar statements, for instance. All in all, Dr. Brady thinks IG Coder addressed some of the weaknesses with the spreadsheet, especially since she is a visual person. She further said I took the feedback from the questionnaire (Questionnaire B) and did something great with IG Coder.

6.3.3 Interview 3: Dr. Bartosz Pielniński

For my third and final interview I sat down with Dr. Bartosz Pielniński, an Associate Professor at the University of Warsaw's Faculty of Political Science and International Studies. While he usually specializes in analyzing non-profit organizations, he currently focuses on methodological issues, which is where the Institutional Grammar comes in. He has worked on implementing the IG in social policies and

currently works with colleagues from other fields of political science.

While I previously believed Dr. Pielinski is a computer scientist, he is not. He tells me he found people in the Warsaw computer science community who are also interested in the Institutional Grammar and has been working together with them for two years.

When asked about his policy coding background, Dr. Pielinski told me he started with INCEPTION and attempted spreadsheets later, so he has experience coding in both. While there are numerous reasons he does not find Microsoft Excel appropriate for annotation, he concedes that it is currently the best tool for analysis. I followed up on this by asking how coded data in INCEPTION can be exported for analysis. Similarly to what Dr. Brady told me, Dr. Pielinski said it is very difficult and that he has not quite reached that level.

Dr. Pielinski's long-term goal is to create a public policy observatory that uses the IG. He imagines a system that collects a huge amount of policy documents worldwide and lets all of these documents be coded with the IG and furthermore analyzed so that policies from different countries and cultures may be compared. In the meantime, he practices his skills with the IG and cooperates with colleagues in multiple research projects as an IG advisor of sorts. This also gives him opportunities to apply the grammar in different fields, broadening his knowledge.

After this, we moved on to discussing IG Coder. I began by asking, as before, how he approached the coding exercises, to which he said he approached IG Coder at least five to six times. He also said he showed it to other people to help teach them the Institutional Grammar and how to code, and he thinks it is a very good tool for this purpose. Pursuing this, I asked the admittedly leading question of whether he thinks INCEPTION is a good tool for teaching. Dr. Pielinski asserted that INCEPTION is only for experienced users of the IG, and even then it is not user friendly when used for the IG. Therefore, it is particularly ill-suited for teaching the grammar.

I then asked the same question about IG Coder. Dr. Pielinski thinks it is a very good educational tool because one has to think about the decision before one makes it, and he described IG Coder as "clean" and its interface as "very user friendly". He pointed out it is easy to present the Institutional Grammar as something very complex and stated that IG Coder is not over-complicated. I believe he also liked the fact that I offered an instructional video of how to use it.

One issue he found was that when coding the last few statements of the exercises, which were highly complicated, the tree visualization became very crowded and text labels overlapped. This was due to the high number of nested statements. In fact, Angelo Baldado also told me this. To that, I told Dr. Pielinski of my idea to add functionality that would allow nodes to be collapsed, hiding their child nodes. We agreed that this would be a good solution, though he pointed out that this was an edge case since few statements reach that level of complexity.

When we discussed the coding workflow in IG Coder, Dr. Pielinski brought up a possible optimization to the process in the form of a shortcut to move directly between nodes without having to leave the node editor. He suggested that keyboard

shortcuts for this would be great, which prompted me to mention INCEpTION's keyboard shortcuts. However, Dr. Pieliński thought it was important to limit the number of keyboard shortcuts in the program because many people use a lot of different text editors, each of which have their own set of shortcuts, so another one might be too much for some people.

We then discussed the terms Prefix and Suffix. Dr. Pieliński pointed out that different users and teams will have to decide on their own approach for how to use them, because every team uses their own subset of the IG's features, according to their research goals. He then brought up spreadsheets, saying one could always add more columns to the IG template, including columns for Prefix and Suffix, but he thinks there are already too many.

When I mentioned the Inferred / Rephrased field, he told me he used it sparingly. He furthermore told me his computer science colleagues prefer not to rephrase because "then it's easier for the system to learn how to [automate] the IG coding", from which I presume they are working on machine learning with the IG. However, many other teams have a habit of rephrasing a lot, so he thinks this functionality is important for a universal tool.

On the topic of a universal tool, Dr. Pieliński stated that IG Coder is the first functional tool that accommodates all or nearly all IG 2.0 features. We agreed that its flexibility is a strength, because again, teams use different subsets of features.

He raised one minor point of annoyance, namely the fact that the default relationship between an entity and its property is functionally independent. He believed the default should be functionally dependent because he does not use the former relationship in his research, and suggested that the relationship could even be disabled entirely for those that do not use it. We agreed that from a logical standpoint, functionally independent is the weaker relationship and I pointed out that the default should be whichever occurs most often, resulting in fewer extra clicks. At the end, Dr. Pieliński reiterated that IG Coder is a great educational tool and expressed his interest in seeing the tool taken further.

6.3.4 Overall Findings

With all interviews summarized, we can now compare them and answer the research questions.

RQ3: To what extent does IG Coder satisfy the needs identified in RQ1 and RQ2? IG Coder addresses several of the identified weaknesses with spreadsheets and INCEpTION, particularly the lack of a readable graphic visualization of coded statements. The tree visualization satisfies multiple needs: providing an overview of the coded statement; visualizing the hierarchy between statements, components and properties; allowing the coder to visually double-check their coding; and helping a reviewer quickly understand the coding.

Furthermore, the tree visualization is color coded, which helps to quickly identify the different elements of the tree. The color coding does not differentiate

between regulative and constitutive components, which was one of the top identified needs, but this is very easy to change.

However, a considerable limitation of the IG Coder prototype is the lack of export formats, another top need. The bare minimum of export and import in a native format is met but without the ability to convert to common formats such as CSV, statements that have been coded in IG Coder cannot be analyzed. This greatly limits IG Coder's usefulness as a coding tool.

User needs pertaining to coding workflow were less clear-cut, meaning it is difficult to say whether IG Coder satisfied these needs. From my interviews with the participants who had tested IG Coder, I understand that coding workflow is subject to personal preference above all. I believe the best way to respond to this is flexibility. To meet the needs of a variety of users, IG Coder should offer multiple methods of coding.

RQ3a: To what extent is IG Coder aligned with coders' understanding of institutional statements? I identified one major issue with the implementation of the IG 2.0 specification, stemming from my design decision to avoid decomposition of logical relationships as much as possible. Two of my three interviewees reported confusion with this. In fact, I realized at a late point that this decision was based on an incorrect assumption of mine. Take the following example statement:

Organic farmers must commit to their organic farming standards and accommodate regular reviews of their practices.

This statement has two AIMS, each associated with a separate OBJECT. The first AIM is directly linked to the first OBJECT, and likewise with the two others. To make these logical relationships explicit, the statement must be decomposed as follows:

Organic farmers must commit to their organic farming standards AND Organic farmers must accommodate regular reviews of their practices.

If keeping the logical combinations compact, as was my intent, the exact meaning of the institutional statement is unclear. This means IG Coder should accommodate explicit decomposition of statements after all. My reasoning behind the original design decision was to save space in the data structure and avoid redundancy, but perhaps there is a way to get the best of both worlds. This will be an important future direction to investigate.

Other than that, my interviewees seemed to agree with my design decisions pertaining to IG 2.0. The tree representation may take some getting used to, especially for more complex statements, but one interviewee who was relatively new to the IG reported that it actually added to his understanding of institutional statements.

RQ3b: How satisfied are users with the coding interface of IG Coder? First of all, my interviewees seemed very satisfied with the tree visualization – it is so far proving to be a success, and is perhaps the greatest strength of the prototype.

Minor enhancements can always be made to the tree but all in all, its fundamentally unique way of seeing institutional statements brings something new to the table for policy coding tools.

Multiple interviewees commented that the tree visualization could be a great tool for teaching the Institutional Grammar. They reported that this is because it has a strong focus on the structure of institutional statements and because the coder has to think about their decision before they make it. On the other hand, this has an impact on the coding workflow, which the next question addresses.

This research question has some overlap with the next (RQ3c), and outside of the tree visualization, I did not speak in depth with my interviewees about parts of the user interface not related to the coding workflow. We will therefore move on to the final research question.

RQ3c: To what extent can IG Coder improve the coding workflow? After conducting and summarizing the interviews, I realized that this question might not have been the right one to ask at this stage. This is because users have to understand the tree structure before the coding workflow can be reliably assessed, and the user tests were not sufficient for my participants to completely figure out IG Coder. Also, this question was based on an assumption of mine that IG Coder would actually be comparable to the current coding tools as a tool for research analysis. This evaluation shows that it is too early to make such a comparison.

However, most of the interviewees as well as I agreed on one point: out of all the coding tools including IG Coder, INCEpTION is the fastest way to code.

I originally posed this question out of an aspiration to make a complete coding tool that could be used for research analysis. There are two criteria for a coding tool to be suitable for research analysis: first, the coded data must be exportable into a useful format and second, the coding workflow needs to be efficient enough for coding a large dataset of statements as painlessly as possible.

My interviewees pointed out a few ways in which IG Coder's workflow may be optimized over its current state. For instance, the ability to move directly from one node to another without closing and reopening the node editor would go a long way. I believe this applies to both parent and sibling nodes in addition to child nodes, the latter of which is currently supported. Keyboard shortcuts would be convenient as well, though they add to the learning curve.

On the topic of the learning curve, it will in any case take time for users to understand and become fluent in building statement trees in IG Coder. The tree representation is a powerful new approach to policy coding but it remains to be seen to what extent users prefer this method over traditional annotation. However (please note that this is based on only one interviewee's response), given that coders are fluent in building statement trees, this method of coding may be more efficient for complex statements than coding in spreadsheets.

The evaluation is discussed as part of the Chapter 7 discussion.

Chapter 7

Discussion

In this section I discuss various aspects of this thesis, including what I learned from the research and development experiences in this project and what I achieved with IG Coder.

First of all, there are a few features listed in the system requirements I ended up not having the time to implement. I did my best to prioritize requirements so that if I had to leave anything out, it would be the least impactful features.

Unfortunately, the most noticeable shortcoming in the IG Coder prototype is the missing export formats, i.e., CSV, UIMA CAS and shorthand. These turned out to be more time-consuming to implement than I first thought, since they all require converting the native data structure to completely different formats. In the case of UIMA CAS, I was not even familiar with this format to begin with, meaning I would have to read up on it first. I will need to verify that this format is indeed useful for people before I embark on implementing it. CSV, on the other hand, I am confident is useful as long as I can implement a conversion to the template spreadsheet. Finally, while the shorthand syntax had the lowest priority in my requirements, it would be convenient for printing and reading simple statements. Also, I believe it would be the easiest to implement out of the three, because it is the native IG 2.0 syntax which my data model is designed after, and because it is a simple text format.

On a related note, I did not create a JSON schema for the data model. This is because I was planning to update the data model after this thesis anyway, so I did not want to spend time on something I would discard before long. However, I am aware that if the native data structure is to be used by other developers, a machine-readable schema is a huge benefit to them.

Furthermore, the IG Coder prototype has limited IG Logico support. This was prioritized slightly lower than IG Core and IG Extended because it has a narrower set of use cases. Admittedly, due to this prioritization I spent less time reading up on and understanding IG Logico, meaning I did not have a clear enough idea of what to implement. At the same time, the uptake of IG Logico should be verified before spending time implementing support for it.

The final requirement I left out was user assistance via suggestions of previously

entered component values. This feature would have been interesting to test with users and find out how useful it is to them. While it would have enhanced the coding interface and made it slightly "smarter", I deemed it a non-essential feature and had to leave it behind in favor of more critical ones.

It was difficult for me to estimate the time required to implement many of the requirements, because for several of them, I did not have an idea beforehand of how to solve the problems they posed. Thus, I did not know whether I would be able to solve it in two hours or two days, for instance. I know that as I gain more experience as a developer, I will have solved a variety of problems which will make it easier to make such estimates in the future.

One experience I will remember from this project is that as the sole developer, I had to drastically prioritize what features to implement despite all the interesting and compelling ideas given to me in the two questionnaires I conducted. Reading through the responses was very inspiring, motivating me to make IG Coder an engaging application. To put it simply, creation is fun. Fortunately, nothing is stopping me from continuing to work on IG Coder in the future.

As a software developer, I have little experience designing user interfaces. I also did not emphasize graphical design nor user experience in the evaluation of IG Coder, and when I built the user interface, I focused on implementing a functional interface over finding the optimal user experience. Therefore, my supervisor's feedback was very helpful to tell me what aspects were difficult from a user perspective. A majority of his feedback to me during development pertained to the user interface, and as a result, the UI was gradually improved rather than planned out in advance. In a project with a single developer, I believe this approach worked well.

Moving on to the user tests and following interviews, I learned memorable lessons from this project phase as well. In my experience, the semi-structured interview format worked very well for my purpose. I had the interview guide at hand while conducting the interviews and thus had the opportunity to skip or rephrase a question on the spot. Also, not having to note down responses was very freeing and enabled me to listen to the participant. Listening to what the respondent is saying a prerequisite for being able to adapt the interview to them and ask the most productive questions.

In retrospect, however, I could have kept the research questions separately when conducting the interviews, e.g., as part of the interview guide and clearly highlighted. This would have helped me made sure my discussions with the participants covered the areas necessary for me to answer the research questions. The way I did it, they were interwoven among the rest of the questions, and I could not remember which ones were the research questions.

As mentioned before, my original aspiration was and still is to make a complete coding tool that can be used for research analysis. The IG Coder prototype I made in this thesis is a significant step toward such a tool. I made a draft of a functional coding interface, and while it may not have been the optimal solution in every way, it served to start a cycle of user feedback and improvement, producing real

results. I believe that is more important than having a flawless plan.

One of my key findings in the evaluation of IG Coder was that its flexibility is a strength. As I have touched on before, I believe flexibility is the way to go in the future, because the evaluation has shown that different coders have different preferences. I imagine that if the application offered multiple alternatives for coding methods, including methods similar to those of the current coding tools, and was also capable of exporting coded data into useful formats, then it would be a good tool for research analysis.

The aspect of IG Coder I am the most proud of, and that I would argue is my most important contribution to the Institutional Grammar, is the tree structure and its graphic visualization. Its success is in part thanks to my supervisor, who originally came up with the idea of representing institutional statements as trees. In this thesis I realized that idea.

Chapter 8

Conclusion

8.1 Summary

In this thesis I have taken you through the design, development and evaluation of the web application IG Coder. This application facilitates the interactive coding of statements in the Institutional Grammar syntax. The IG is at the core of this thesis.

After giving an overview of the project phases and the research methods used in each, the thesis began in earnest with a contextualization of the Institutional Grammar within the bigger picture. Following this, I gave a primer of the IG along with an overview of its literature. I then described the current state of policy coding tools for the IG, setting the stage for what was to be created in this thesis.

The design phase followed, conducting a review of the current coding tools. This review was primarily based on two questionnaires aimed at two different groups of coders. It resulted in insights about the strengths and weaknesses of the current tools but also about what is needed in a coding tool, all of which would help me design the new tool.

I began the development chapter by telling the history of IG Coder. The chapter otherwise described all important aspects of IG Coder's development, including its system requirements which built on the review of current tools, as well as the IG data model and tree visualization, which I believe are the two most noteworthy aspects of the IG Coder prototype.

In the evaluation phase, the IG Coder prototype was tested by users familiar with the Institutional Grammar. Following the user tests, I interviewed each participant to gain insights about their experience with the tool and determine the extent to which it responded to the needs identified earlier in the thesis. The evaluation uncovered a few shortcomings in the prototype which I might not have noticed on my own, as well as suggestions for large and small improvements to the application, helping to guide future work on it.

The IG Coder prototype developed in this thesis is a big step toward a new coding tool. A great deal of work remains, but it has already received attention from and garnered interest in a few people in the IGRI.

8.2 Limitations

This thesis had a few considerable limitations as a research project. First and foremost, the small population of people familiar with the Institutional Grammar means the two studies (i.e., the review of current tools and the evaluation of IG Coder) are built on sparse amounts of data. This negatively impacts the reliability of their results. All of this only goes to show that it is yet early in the development and uptake of the Institutional Grammar.

Furthermore, the amount of software development I could do in this thesis was limited because I am only one person and there were research objectives to be investigated both before and after the development phase. There are several things I wanted to do with IG Coder that I did not have the time for, but I hope to continue working on the tool in the future.

8.3 Future Work

To live up to its aspirations as a complete policy coding tool, IG Coder must be developed further. On a high level, flexibility should be valued and prioritized in the eventual development of the coding interface.

On a lower level, I believe the two most important features to be implemented next are as follows:

1. Accommodate decomposition of logical combinations while keeping the data structure compact
2. Ability to export coded data in common formats, primarily CSV but also other formats

However, it may be prudent to strengthen unit testing of the existing features before adding new ones.

To make the coding interface more flexible, future work should look into adopting direct text annotation as an alternate coding method. This would require finding a way to interface text annotation against the IG data model.

In addition, an interesting research direction emerged from my interviews about IG Coder. Future work could look into functionality that compares two statement trees and quantifies their differences, which could enable automatic assessment of inter-coder reliability.

Stepping back, the contribution of this thesis to the Institutional Grammar is a new way of regarding institutional statements which is also actualized in a new coding interface. I hope this is the start of a complete coding application which will make it easy to apply the IG in all kinds of research.

Bibliography

- Barabucci, G., Cervone, L., Palmirani, M., Peroni, S., & Vitali, F. (2010). Multi-layer Markup and Ontological Structures in Akoma Ntoso (P. Casanovas, U. Pagallo, G. Sartor, & G. Ajani, Eds.). In P. Casanovas, U. Pagallo, G. Sartor, & G. Ajani (Eds.), *AI Approaches to the Complexity of Legal Systems. Complex Systems, the Semantic Web, Ontologies, Argumentation, and Dialogue*, Berlin, Heidelberg, Springer Berlin Heidelberg.
- Basurto, X., Kingsley, G., McQueen, K., Smith, M., & Weible, C. M. (2010). A Systematic Approach to Institutional Analysis: Applying Crawford and Ostrom's Grammar. *Political Research Quarterly*, 63(3), 523–537. <https://doi.org/10.1177/1065912909334430>
- Crawford, S. E. S., & Ostrom, E. (1995). A Grammar of Institutions. *American Political Science Review*, 89(3), 582–600. <https://doi.org/10.2307/2082975>
- Cummings, J. (2013). The Text Encoding Initiative and the Study of Literature. In *A Companion to Digital Literary Studies* (pp. 451–476). John Wiley & Sons, Ltd. <https://doi.org/10.1002/9781405177504.ch25>
- Eckart de Castilho, R., Mújdricza-Maydt, É., Yimam, S. M., Hartmann, S., Gurevych, I., Frank, A., & Biemann, C. (2016, December). A Web-based Tool for the Integrated Annotation of Semantic and Syntactic Structures, In *Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH)*, Osaka, Japan, The COLING 2016 Organizing Committee.
- Elo, S., Kääriäinen, M., Kanste, O., Pölkki, T., Utriainen, K., & Kyngäs, H. (2014). Qualitative Content Analysis: A Focus on Trustworthiness. *SAGE Open*, 4(1). <https://doi.org/10.1177/2158244014522633>
- Elo, S., & Kyngäs, H. (2008). The qualitative content analysis process. *Journal of Advanced Nursing*, 62(1), 107–115. <https://doi.org/10.1111/j.1365-2648.2007.04569.x>
- Ferrucci, D., & Lally, A. (2004). UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, 10(3-4), 327–348. <https://doi.org/10.1017/S1351324904003523>
- Ferrucci, D., Lally, A., Verspoor, K., & Nyberg, E. (2009). Unstructured Information Management Architecture (UIMA) Version 1.0. <https://docs.oasis-open.org/uima/v1.0/uima-v1.0.html>

- Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* (Doctoral dissertation). University of California, Irvine.
- Frantz, C. K., Purvis, M. K., Nowostawski, M., & Savarimuthu, B. T. R. (2013). nADICO: A Nested Grammar of Institutions (G. Boella, E. Elkind, B. T. R. Savarimuthu, F. Dignum, & M. K. Purvis, Eds.). In G. Boella, E. Elkind, B. T. R. Savarimuthu, F. Dignum, & M. K. Purvis (Eds.), *Prima 2013: Principles and practice of multi-agent systems*, Berlin, Heidelberg, Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-44927-7_31
- Frantz, C. K., & Siddiki, S. N. (2020). *Institutional Grammar 2.0 Codebook* [arXiv preprint arXiv:2008.08937]. arXiv preprint arXiv:2008.08937. <https://doi.org/10.1111/padm.12719>
- Frantz, C. K., & Siddiki, S. N. (2021). Institutional Grammar 2.0: A specification for encoding and analyzing institutional design. *Public Administration, n/a(n/a)*, 1–21. <https://doi.org/10.1111/padm.12719>
- Given, L. (2008). The SAGE Encyclopedia of Qualitative Research Methods. *Thousand Oaks, California*. <https://doi.org/10.4135/9781412963909>
- Gordon, T. F. (2008). Constructing Legal Arguments with Rules in the Legal Knowledge Interchange Format (LKIF) (P. Casanovas, G. Sartor, N. Casellas, & R. Rubino, Eds.). In P. Casanovas, G. Sartor, N. Casellas, & R. Rubino (Eds.), *Computable Models of the Law*, Berlin, Heidelberg, Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-85569-9_11
- Hsieh, H.-F., & Shannon, S. E. (2005). Three Approaches to Qualitative Content Analysis [PMID: 16204405]. *Qualitative Health Research, 15*(9), 1277–1288. <https://doi.org/10.1177/1049732305276687>
- Klie, J.-C., Bugert, M., Boullosa, B., de Castilho, R. E., & Gurevych, I. (2018). The INCEpTION Platform: Machine-Assisted and Knowledge-Oriented Interactive Annotation, In *Proceedings of the 27th international conference on computational linguistics: System demonstrations*, Association for Computational Linguistics. <http://tubiblio.ulb.tu-darmstadt.de/106270/>
- Lane, J., Garrison, M. M., Kelley, J., Sarma, P., & Katz, A. (2020). Strengthening policy coding methodologies to improve COVID-19 disease modeling and policy responses: a proposed coding framework and recommendations. *BMC Medical Research Methodology, 20*(1), 298. <https://doi.org/10.1186/s12874-020-01174-w>
- Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., & McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit, In *Proceedings of 52nd annual meeting of the association for computational linguistics: System demonstrations*, Baltimore, Maryland, Association for Computational Linguistics. <https://doi.org/10.3115/v1/P14-5010>
- Palmirani, M., & Vitali, F. (2011). Akoma-Ntoso for Legal Documents. In G. Sartor, M. Palmirani, E. Francesconi, & M. A. Biasiotti (Eds.), *Legislative XML for the Semantic Web: Principles, Models, Standards for Document Management* (pp. 75–100). Dordrecht, Springer Netherlands. https://doi.org/10.1007/978-94-007-1887-6_6

- Rice, D., Siddiki, S., Frey, S., Kwon, J. H., & Sawyer, A. (2021). Machine coding of policy texts with the Institutional Grammar. *Public Administration*, *n/a(n/a)*, 1–15. <https://doi.org/10.1111/padm.12711>
- Robinson, J. (2014). Likert Scale. In A. C. Michalos (Ed.), *Encyclopedia of Quality of Life and Well-Being Research* (pp. 3620–3621). Dordrecht, Springer Netherlands. https://doi.org/10.1007/978-94-007-0753-5_1654
- Siddiki, S. N. (2014). Assessing Policy Design and Interpretation: An Institutions-Based Analysis in the Context of Aquaculture in Florida and Virginia, United States. *Review of Policy Research*, *31*(4), 281–303. <https://doi.org/10.1111/ropr.12075>
- Siddiki, S. N., Heikkila, T., Weible, C. M., Pacheco-Vega, R., Carter, D., Curley, C., Deslatte, A., & Bennett, A. (2019). Institutional Analysis with the Institutional Grammar. *Policy Studies Journal*. <https://doi.org/10.1111/psj.12361>
- Siddiki, S. N., Weible, C. M., Basurto, X., & Calanni, J. (2011). Dissecting Policy Designs: An Application of the Institutional Grammar Tool. *Policy Studies Journal*, *39*(1), 79–103. <https://doi.org/10.1111/j.1541-0072.2010.00397.x>
- Smajgl, A., Izquierdo, L. R., & Huigen, M. (2008). Modeling Endogenous Rule Changes in an Institutional Context: The ADICO Sequence. *Advances in Complex Systems*, *11*(02), 199–215. <https://doi.org/10.1142/S021952590800157X>
- State of JavaScript. (2019). The State of JavaScript 2019: Front End Frameworks [Accessed 18 May 2021]. <https://2019.stateofjs.com/front-end-frameworks/>
- TEI Consortium. (2021, April). TEI P5: Guidelines for Electronic Text Encoding and Interchange (TEI Consortium, Ed.) [Version 4.2.2. Accessed 26 May 2021]. <http://www.tei-c.org/Guidelines/P5/>
- Vanhoutte, E. (2004). An Introduction to the TEI and the TEI Consortium. *Literary and Linguistic Computing*, *19*(1), 9–16. <https://doi.org/10.1093/llc/19.1.9>
- Vitali, F., & Zeni, F. (2007). Towards a country-independent data format: the Akoma Ntoso experience, In *Proceedings of the V Legislative XML Workshop*.
- Wittern, C., Ciula, A., & Tuohy, C. (2009). The making of TEI P5. *Literary and Linguistic Computing*, *24*(3), 281–296. <https://doi.org/10.1093/llc/fqp017>
- Yimam, S. M., Gurevych, I., de Castilho, R. E., & Biemann, C. (2013). Webanno: A flexible, web-based and visually supported system for distributed annotations, In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Appendix A

Excel Questionnaire

This is also referred to as Questionnaire A.

Survey on Excel Coding of Institutional Grammar Statements

* Obligatorisk

1. First off, what level are you currently studying at? *

- Undergraduate
- Graduate (master's)
- Graduate (doctoral)
- I'm not currently studying

Annet

2. Since you answered "I'm not currently studying", what is your occupation?

3. Since you answered "Graduate (doctoral)", what is your area of research (e.g., water management)?

4. You've been coding institutional statements in Excel. What is your level of experience with Microsoft Excel?

	No experience	Limited experience	Neither experienced nor inexperienced	Somewhat experienced	Experienced
Experience level	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

5. How would you rate Microsoft Excel as a coding tool for Institutional Grammar 2.0?

	Totally unsuitable	Mostly unsuitable	Somewhat unsuitable	Neutral	Somewhat suitable	Mostly suitable	Perfectly suitable
Excel as a coding tool	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

6. In your opinion what makes Excel a useful coding tool?

7. What challenges have you encountered coding in Excel?

8. If you could design a new coding tool for IG 2.0 (i.e., new software), what three capabilities should that tool possess?

9. Thinking about the instructions you received for coding IG 2.0 in Excel, what additional instructions would have been useful for you to clarify the coding process in Excel (e.g., types of statements, complexity)?

10. Were the associated resources (e.g., coding manual, cheat sheet) provided to you useful in coding IG 2.0 in Excel?

- Yes
- No

11. If no, please list one thing that would have been more useful.

12. If you have any additional thoughts, please leave those below.

Appendix B

Early Interview Questions

These questions were asked in the policy coder interview in the course Integration Project¹. At the time, Microsoft Excel was the only policy coding tool in use.

- How does your typical day look like as a policy coder?
- What are your biggest struggles as a policy coder?
- Why do you use the spreadsheet?
- Have you used or do you use any other tools than the spreadsheet?
- What are the pros and cons of the spreadsheet?
- When using the spreadsheet, how do you know you've coded a statement correctly? Do you need someone to review it?
- After you've filled out the spreadsheet, what do you do with it?

¹Course code: IMT4807

Appendix C

INCEpTION+Excel Questionnaire

This is also referred to as Questionnaire B.

Survey on Coding of Institutional Statements in INCEpTION and Excel

INCEpTION Coding

First, let's start with your experience and impression of INCEpTION as an IG 2.0 coding tool.

1. What is your level of experience with the text annotation tool INCEpTION in general? 

	No experience	Limited experience	Neither experienced nor inexperienced	Somewhat experienced	Experienced
Experience level with INCEpTION	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

2. How would you rate INCEpTION as a coding tool for Institutional Grammar 2.0?

	Totally unsuitable	Mostly unsuitable	Somewhat unsuitable	Neutral	Somewhat suitable	Mostly suitable	Perfectly suitable
INCEpTION as a coding tool	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

3. In your opinion what makes INCEpTION a useful coding tool?

4. Which challenges have you encountered when coding in INCEpTION?

Excel Coding

In the following, we will have a look at Microsoft Excel as an IG 2.0 coding tool. If you are using a tabular coding tool equivalent to Microsoft Excel (e.g., LibreOffice), please relate to this experience when responding to the questions.

5. What is your level of experience with Microsoft Excel in general?

	No experience	Limited experience	Neither experienced nor inexperienced	Somewhat experienced	Experienced
Experience level with Excel	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

6. How would you rate Microsoft Excel as a coding tool for IG 2.0?

	Totally unsuitable	Mostly unsuitable	Somewhat unsuitable	Neutral	Somewhat suitable	Mostly suitable	Perfectly suitable
Excel as a coding tool	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

7. In your opinion what makes Excel a useful coding tool?

8. Which challenges have you encountered when coding in Excel?

Appendix D

IG Data Model

Version 2

Version 2 of the data model makes several major changes:

- Supports constitutive as well as regulative statements
- Supports IG Extended features: component-level nesting, Object-Property Hierarchy and Context Taxonomy
- Declarative way of differentiating between statement and component level for the purposes of horizontal nesting: the Junction node type has statement and component level variants
- The Or else component is now a Component type rather than a special node type
- Negation is supported on all nodes, and is now a field on the base node class instead of its own node type
- Object and Context subtypes (direct and indirect object, activation conditions and execution constraints) are unpacked to the same level as the other components, eliminating the Subcomponent node type for the sake of simplicity

Example Tree

Figure 5.9 shows a coding of the below institutional statement according to version 2 of the data model. The coding follows IG Extended. The statement exemplifies most features of the data model.

Furthermore, the figure makes the following simplifications:

- In the top-level statement, all optional components are shown, even though the original statement does not have an Or else component.
- Nested statements that have no Activation Conditions or Execution Constraints have those children truncated to a single node labelled Context. The top-level statement shows the correct way.

Node types are color coded according to the key in the figure. The Or else component type is treated specially, having the same color as a Statement Junction, because this component can be seen as a junction between two statements, the monitored statement on the left side and the consequential on the right, joined by an implicit XOR operator. However, the data model finds it more practical to implement as a Component.

Example Statement

«The Program Manager may initiate suspension or revocation proceedings against a certified operation:

- (1) When the Program Manager has reason to believe that a certified operation has violated or is not in compliance with the Act or regulations in this part;
or
- (2) When a certifying agent or a State organic program’s governing State official fails to take appropriate action to enforce the Act or regulations in this part.»

Glossary

policy = a written legal document or document of rules

document = the class that represents a policy

statement = a sentence in a policy that ends with a period

entry = the class that represents a statement

text content = a field on a node that holds text data extracted from a statement

Statement = umbrella term for RegulativeStatement and ConstitutiveStatement (Node types)

Junction = umbrella term for StatementJunction, ComponentJunction and PropertyJunction (Node types)

Context = umbrella term for ActivationConditions and ExecutionConstraints (Component types)

Deviations from the IG 2.0 Specification and Limitations

- When a statement has e.g. multiple Aims, IG 2.0 splits the original statement into two or more statements. However, because such duplication of data is inefficient in a computer system, this data model stores logically combined components within the same statement. Such statements can be split at the time of exporting to shorthand or other formats.
- IG Logico features are not yet supported.
- Polymorphic institutional statements are not yet supported.

Document

- A complete policy is represented by a Document. It has a field `entries` which is an array of Entries, each representing a statement in that policy. The Entries have a fixed chronological order.
- It holds an `entryMap` which is a mapping of entry IDs to their respective indices in the `entries` array.
- Other fields on a Document are `name`, `description` and `id`.
- All Entry and Node IDs are unique within a Document.

Entry

- A statement in a policy is represented by an Entry. It has a field `root` which contains the root node of the tree representing that statement.
- It further has a field `original` which contains the raw, unedited text of the statement, and an optional field `rephrased` which holds a rewritten version of the statement.
- Also has a field `document` to hold the ID of the Document it belongs to.

Node

- Nodes represent and compartmentalize the various elements of a statement: the root statement, nested statements, regulative and constitutive components of those statements, properties of Attributes and Objects as well as logical combinations of those elements.
- All Nodes have the following common fields:
 - `id`: A number unique to the Node within its Document
 - `document`: The ID of the Node's Document
 - `nodeType`: The Node's role in the tree
 - `isNegated`: Whether the Node's meaning is negated (default: false)
 - `contextType`: [Optional] A Context Taxonomy label for the node
 - `parent`: [Optional] The ID of the Node's parent Node, if it has one
 - `createdAt`: The date and time this Node was created - for debugging
 - `updatedAt`: The date and time this Node was last changed - for debugging
 - `children`: An array of Nodes that are this Node's children

The `updatedAt` field is set to the current date when a node's own fields are changed, when a child node is added to it and when a child node is deleted from it.

Trees

- Valid node types for the root of a tree are `Statement` and `StatementJunction`.

- Invalid node types for the root of a tree are Component, ComponentJunction, Property and PropertyJunction.
- Ignoring the existence of Property nodes, all leaf nodes in a tree must be Component nodes and have text content, and only leaf nodes can have text content.
- To read the coded statement, the tree should be traversed depth-first. According to the IG specification, the Object comes directly after the Attribute but you can read the Object after the Aim for the statement to make better sense in English.

Text Content

- If a Component or Property node has text content, it represents a primitive.
- The text content of Junction nodes contains the string from the raw statement that constitutes a logical conjunction or disjunction, usually "and" or "or".
- Component, Property and Junction nodes have a text field which must always be defined.
- The text field is an object consisting of:
 - `main` - the text that most narrowly fits the Component, Property or Junction, taken directly from the raw statement
 - `prefix` - text from the statement that precedes the main part, e.g. prepositions and articles
 - `suffix` - text from the statement that succeeds the main part, e.g. "suspects (I) *that*"
 - `inferredOrRephrased` - an explicit specification of an inferred Component or Property, a rephrased version of the text in the `main` field, or a combination of both
 - An inferred Component or Property can be empty in the raw text, and can thus be specified explicitly.
 - A common use case for rephrasing text is if the raw text is written in passive voice and the coder rewrites it to be active.
- The existence of text content is determined by whether at least one of the `main` and `inferredOrRephrased` fields is non-empty. This is because an inferred component may be completely absent in the source text, so that `main` is empty and `inferredOrRephrased` is not.
- The text field supports two coding states.
 - The `main` OR `inferredOrRephrased` fields are a non-empty string, e.g. "Program Manager" for nodes whose content has been coded.
 - The `main` AND `inferredOrRephrased` fields are both an empty string "" for nodes whose content has not yet been coded.

- The previous version of the data model supported a third state, "intentionally empty". The empty string was reserved for this state, and the "not yet coded" state was signified by an undefined main field. This is no longer the case, as there is no need for this state due to optional components.
- The following exceptions apply: The main and inferredOrRephrased fields are always empty strings for Component nodes of type ActivationConditions/ExecutionConstraints and Property nodes whose child is a Statement/StatementJunction.
- If the main and inferredOrRephrased fields are both empty strings, the prefix and suffix fields are to be ignored.

Negation

- A Component, Property or Junction node should be marked as negated if its text source in the raw statement indicates a logical negation, or in other words, the text source has a negative meaning.
- The text content of a negated node should retain any negation strings (e.g. "not", "no") from the raw statement.

Node Types

- **Statement**
 - Common base class for RegulativeStatement and ConstitutiveStatement; not to be used directly
 - Can be a root
 - **RegulativeStatement**
 - Represents an regulative statement where DirectObject, IndirectObject, Deontic and Or else are optional
 - Must have exactly eight Component children: Attribute, DirectObject (optional), IndirectObject (optional), Deontic (optional), Aim, ActivationConditions, ExecutionConstraints, OrElse (optional)
 - Optional children must exist but can be empty
 - **ConstitutiveStatement**
 - Represents a constitutive statement where ConstitutingProperties, Modal and Or else are optional
 - Must have exactly seven Component children: ConstitutingProperties (optional), Modal (optional), ConstitutiveFunction, ConstitutedEntity, ActivationConditions, ExecutionConstraints, OrElse (optional)
 - Optional children must exist but can be empty

- **Junction**

- Represents a logical combination of two elements (nodes)
- Common base class for StatementJunction, ComponentJunction and PropertyJunction; not to be used directly
- Has a field `junctionType` that holds a logical operator out of [AND, OR, XOR]
- Has a field `text` that holds text content
- Must have exactly two children
- **StatementJunction**
 - Used for horizontal nesting of statements
 - Each of its children must be of type Statement or StatementJunction, independently of each other
 - Can be a root
- **ComponentJunction**
 - Used for horizontal nesting of components within a statement
 - Has a field `componentType` that holds one of those listed below; used to pass down component type from ancestor Component node
 - Each of its children must be of type Component or ComponentJunction, independently of each other
 - Cannot be a root
- **PropertyJunction**
 - Used for horizontal nesting of Property nodes
 - Has a field `isFunctionallyDependent` that is used to pass descendant Property node's state to ancestor Component or Property node
 - Each of its children must be of type Property or PropertyJunction
 - Cannot be a root

- **Component**

- Represents a component of a regulative or constitutive statement
- Has a field `text` that holds text content
- Has a field `componentType` that holds one of those listed below; its component type gives it additional rules that override those on this level
- If it has text content, it must not have children
- If it has children, it must not have text content
- If it has children, all its descendant leaf nodes that do not come from another Statement node or a Property node must be of a component type matching its own. For the purposes of this, ActivationConditions and ExecutionConstraints are not valid component types.
- Cannot be a root

- **Property**

- Represents a property or object in the Object-Property Hierarchy
- Has a field `text` that holds text content
- Has a field `isFunctionallyDependent` that holds whether or not the property or object this node represents is functionally dependent on its parent
- Must have either one child or none
- Its child, if it has one, must be of type `Statement`, `StatementJunction`, `Property` or `PropertyJunction`
- If it has text content and a child, the child must be of type `Property` or `PropertyJunction`
- If its child is of another type than `Property` and `PropertyJunction`, it must not have text content
- Cannot be a root

Component Types

Common:

- **ActivationConditions**

- Can have any number of children, including 0
- Cannot have text content
- Each of its children must be of type `Statement`, `StatementJunction`, `Component` or `ComponentJunction`
- If its child is of type `Component`, that child must have the component type `SimpleContext`

- **ExecutionConstraints**

- Can have any number of children, including 0
- Cannot have text content
- Each of its children must be of type `Statement`, `StatementJunction`, `Component` or `ComponentJunction`
- If its child is of type `Component`, that child must have the component type `SimpleContext`

- **OrElse**

- Must have either one child or none
- Cannot have text content
 - EXCEPTION: The above rule is not implemented in the codebase, which treats Or else as able to have text content.
- Its child, if it has one, must be of type `Statement` or `StatementJunction`

Regulative:

- **Attribute**

- Must have either one child or none
- Its child, if it has one, must be of type Statement, StatementJunction or ComponentJunction
- If it has text content, it can have any number of children if each of them is of type Property or PropertyJunction

- **DirectObject**

- Must have either one child or none
- Its child, if it has one, must be of type Statement, StatementJunction or ComponentJunction
- If it has text content, it can have any number of children if each of them is of type Property or PropertyJunction

- **IndirectObject**

- Must have either one child or none
- Its child, if it has one, must be of type Statement, StatementJunction or ComponentJunction
- If it has text content, it can have any number of children if each of them is of type Property or PropertyJunction

- **Deontic**

- Cannot have children
- Must have text content

- **Aim**

- Must have either one child or none
- Its child, if it has one, must be of type ComponentJunction

Constitutive:

- **ConstitutingProperties**

- Must have either one child or none
- Its child, if it has one, must be of type Statement, StatementJunction or ComponentJunction
- If it has text content, it can have any number of children if each of them is of type Property or PropertyJunction

- **Modal**

- Cannot have children
- Must have text content

- **ConstitutiveFunction**

- Must have either one child or none
- Its child, if it has one, must be of type ComponentJunction

- **ConstitutedEntity**

- Must have either one child or none

- Its child, if it has one, must be of type Statement, StatementJunction or ComponentJunction
- If it has text content, it can have any number of children if each of them is of type Property or PropertyJunction

Special:

- **SimpleContext**
 - Matches the ActivationConditions and ExecutionConstraints types
 - Cannot have children
 - Must have text content

Context Types

Context types are the same as in the most recent version of the IG 2.0 codebook. They are in a hierarchy, although the hierarchy is syntactically inconsequential. Also, they are numerically indexed, and the recurring "Beginning" and "End" types are prefixed with either "t" (for temporal) or "sp" (for spatial) to make them distinct.

Lists of Context Components and Properties

Component nodes of types Attribute, DirectObject, IndirectObject, Constituting-Properties and ConstitutedEntity can have any number of Property children. Likewise, Component nodes of types ActivationConditions and ExecutionConstraints can have any number of children. These children are implicitly combined with the logical AND operator. Optionally, properties can be coded using PropertyJunction nodes to make the logical combination explicit.

To find the index of such a child, all nodes have a function that takes a node ID and returns the children array index of the child node if it is a child of the current node. This method is preferred to keeping a mapping of IDs to indices on the node itself, because the latter method adds unnecessary data to the document tree.

Validity of a Tree

While a tree is saveable and loadable in any state, in order to export it to another format, the tree's completeness must be validated. Below are the conditions that must be checked for during validation. The rules system is built into the data model and enforces all but the below rules.

- For every node, each of its children must have the node's ID in their parent field
- Each leaf node that is not BaseNode must have non-empty text content

- Each Component node whose first child is Property/PropertyJunction must have non-empty text content
- For each Junction node, neither of its children can be BaseNode
- Each TextContent object with empty main AND inferredOrRephrased fields must also have prefix, and suffix empty

Appendix E

Test Statements

1. American homeowners mow their lawns.
2. Dog owners pick up after their dogs.
3. The Commission may appoint its own Secretary and staff.
4. For the purposes of this Act, working hours means time when the employee is at the disposal of the employer.
5. Normal working hours must not exceed nine hours per 24 hours and 40 hours per seven days.
6. The employer and the employee may agree in writing that overtime hours shall wholly or partly be taken out as off-duty time on agreed dates.
7. "Contracting Government" means any Government which has deposited an instrument of ratification or has given notice of adherence to this Convention.
8. Night work is not permitted unless necessitated by the nature of the work.
9. The Ministry may issue regulations with further provisions concerning the activities of the committee, including provisions concerning procedure and concerning the duty of secrecy for members of the committee.
10. All male U.S. citizens, 18 years of age and older, must register with the Selective Service by filling out a form at the U.S. Post Office or else face arrest for evading registration.
11. Employees who have children in their care are entitled to leave of absence:
 - a) when necessary to attend a sick child, b) if a child shall be accompanied to a medical examination or other follow-up in connection with sickness, or c) if the person responsible for the daily childcare is sick or has leave of absence pursuant to this section owing to another child.
12. Certified organic farmers must not apply synthetic chemicals to crops at any time once organic certification is conferred, or else certifier will administer official notice of noncompliance and revoke or suspend certification of farmer.
13. The notification shall provide: (1) A description of each noncompliance; (2) The facts upon which the notification of noncompliance is based; and (3) The date by which the certifying agent must rebut or correct each non-

compliance and submit supporting documentation of each correction when correction is possible.

14. Drivers must hand their driver's license to the police officer when stopped in traffic control, or else the police officer must enforce this under any circumstances and, depending on severity, must either fine the driver or arrest him, or else internal investigators must follow up on this issue in any case.

Appendix F

Interview Guide

The below document was sent out to participants of the IG Coder evaluation as part of the invitation to participate. It was then used as a guide for conducting the semi-structured interviews for evaluation.

Interview Guide

Following are the topics and questions we will discuss in a semi-structured interview held remotely. This implies that any response to the below questions can lead to follow-up questions (e.g., for clarification).

Topics:

- Some demographic questions
 - Are you a student or researcher?
 - What is your preferred tool for policy coding? Further questions will refer to this tool as well as IG Coder.
- Your use of IG in your research
 - What data structure do you usually use for your analysis?
 - What kinds of statements are you usually coding?
 - In what level of detail are you coding institutional statements? For example, are you coding combinations of statements, nested statement structures?
 - Which aspects of institutional statements do you find more useful in your research?
 - Which export formats would you like to be available in a coding tool?
- Reviewing your tool of choice
 - What are the main strengths of your tool of choice?
 - What are the main challenges with your tool of choice?
- Your coding workflow using IG Coder
 - When you performed the coding exercises in IG Coder, did you code

- them all in one go or intermittently?
- How does your workflow in IG Coder compare to that in your tool of choice?
 - How would you rate your efficiency with IG Coder compared to that with your tool of choice?
- Your coding preferences
 - When coding in IG Coder, how did you use the "prefix" and "suffix" fields, if at all?
 - Regarding the terms "prefix", "main content", "suffix" and "inferred / rephrased", would other terms have been easier to understand for you?
 - How did you code multiple properties on the same component? (An example will be given.)
 - When coding nested statements in IG Coder, did you code breadth-first or depth-first?
 - Your understanding of institutional statements
 - How well does IG Coder align with your understanding of institutional statements?
 - Your evaluation of IG Coder
 - To what extent does IG Coder retain the strengths you see in your tool of choice?
 - To what extent does IG Coder respond to the challenges you see in your tool of choice?
 - To what extent does IG Coder respond to your needs when coding institutional statements?
 - Can you think of aspects that your tool of choice does better than IG Coder?

Appendix G

IG Coder Public Repository

The source code for IG Coder is public in a GitHub repository available at the following link:

<https://github.com/bjohanne/ig-coder>

