Making sense of legal texts

de Maat, E.

Publication date
2012

Document Version
Final published version

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
UITNODIGING

voor het bijwonen van de verdediging van mijn proefschrift:

Making Sense of Legal Texts

op woensdag
12 september 2012
om 10:00 uur

in de Agnietenkapel,
Oudezijds Voorburgwal
231, in Amsterdam.

Emile de Maat
e.demaat@alumnus.utwente.nl
Making Sense of Legal Texts
MAKING SENSE OF LEGAL TEXTS

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
  prof. dr. D.C. van den Boom
ten overstaan van een door het college voor promoties ingestelde
commissie, in het openbaar te verdedigen in de Agnietenkapel
  op woensdag 12 september 2012, te 10:00 uur

door Emile de Maat
geboren te Almelo
Promotor: prof. dr. T.M. van Engers
Copromotor: dr. R.G.F. Winkels

Faculteit der Rechtsgeleerdheid
## Contents

1 Introduction ................................................................................................................................. 1

2 Parsing Structure ........................................................................................................................... 9

2.1 Structure of Dutch Laws ........................................................................................................ 10

2.2 Numbering ................................................................................................................................ 12

2.3 Automated Detection ............................................................................................................ 13

2.3.1 Separating the Different Parts of a Law ................................................................... 14

2.4 Splitting the Body of a Document in Parts and Articles ...................................................... 14

2.5 Splitting Articles ..................................................................................................................... 16

2.6 Recognising Quoted Text to be Added Elsewhere ............................................................... 17

2.7 Tables, Images and Other Content ....................................................................................... 19

2.8 Experiment ............................................................................................................................... 19

2.9 Conclusions .............................................................................................................................. 24

3 References ....................................................................................................................................... 27

3.1 References to Documents ....................................................................................................... 29

3.2 References to Parts of Documents ........................................................................................ 30

3.3 Resolving References ............................................................................................................. 31

3.4 Experiment ............................................................................................................................... 33

3.5 Generalisation to Other Sources .......................................................................................... 34

3.6 Conclusion ................................................................................................................................. 35

4 Classification of Sentences ............................................................................................................. 39

4.1 Norms ....................................................................................................................................... 46

4.1.1 Obligations ....................................................................................................................... 46

4.1.2 Rights ................................................................................................................................. 47

4.1.3 Application Provisions ....................................................................................................... 47

4.1.4 Penalisations ..................................................................................................................... 48

4.1.5 Calculations ....................................................................................................................... 48

4.1.6 Delegation ......................................................................................................................... 49

4.1.7 Publication provision ......................................................................................................... 50

4.2 Definitions .................................................................................................................................. 50

4.2.1 Definitions ........................................................................................................................ 50

4.2.2 Type Extensions ................................................................................................................. 52

4.3 Deeming Provisions .................................................................................................................. 52
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6.4</td>
<td>Example</td>
<td>107</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusion</td>
<td>109</td>
</tr>
<tr>
<td>6</td>
<td>Applications</td>
<td>111</td>
</tr>
<tr>
<td>6.1</td>
<td>Semantic Network for the Dutch Tax and Customs Administration</td>
<td>111</td>
</tr>
<tr>
<td>6.2</td>
<td>MetaVex</td>
<td>112</td>
</tr>
<tr>
<td>6.3</td>
<td>Conclusion</td>
<td>118</td>
</tr>
<tr>
<td>7</td>
<td>Conclusions and Future Work</td>
<td>119</td>
</tr>
<tr>
<td>7.1</td>
<td>Regularity of Legislative Sources</td>
<td>122</td>
</tr>
<tr>
<td>7.2</td>
<td>Architecture for an Integrated Application</td>
<td>123</td>
</tr>
<tr>
<td>7.3</td>
<td>Future Expansions</td>
<td>126</td>
</tr>
<tr>
<td>7.4</td>
<td>Text First or Model First?</td>
<td>128</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Detecting Structure Using JAPE</td>
<td>131</td>
</tr>
<tr>
<td>A.1</td>
<td>A Short Introduction to JAPE</td>
<td>131</td>
</tr>
<tr>
<td>A.2</td>
<td>Detecting Headings</td>
<td>132</td>
</tr>
<tr>
<td>A.3</td>
<td>Detecting Fixed Elements</td>
<td>133</td>
</tr>
<tr>
<td>Appendix B</td>
<td>Detecting References using JAPE</td>
<td>137</td>
</tr>
<tr>
<td>B.1</td>
<td>Basic References</td>
<td>137</td>
</tr>
<tr>
<td>B.2</td>
<td>References to Ranges</td>
<td>138</td>
</tr>
<tr>
<td>B.3</td>
<td>Multiple References</td>
<td>141</td>
</tr>
<tr>
<td>B.4</td>
<td>Zooming-In Layered References</td>
<td>142</td>
</tr>
<tr>
<td>B.5</td>
<td>Zooming-Out Layered References</td>
<td>143</td>
</tr>
<tr>
<td>B.6</td>
<td>Putting It All Together</td>
<td>143</td>
</tr>
<tr>
<td>Appendix C</td>
<td>Sentence Classification Patterns</td>
<td>145</td>
</tr>
<tr>
<td>C.1</td>
<td>Patterns for Obligations and Duties</td>
<td>145</td>
</tr>
<tr>
<td>C.2</td>
<td>Patterns for Short Titles</td>
<td>145</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>147</td>
</tr>
<tr>
<td>Summary</td>
<td></td>
<td>155</td>
</tr>
<tr>
<td>Samenvatting</td>
<td></td>
<td>161</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td></td>
<td>167</td>
</tr>
<tr>
<td>SIKS Dissertation Series</td>
<td></td>
<td>169</td>
</tr>
</tbody>
</table>
1 Introduction

A society is governed by its rules. Nowadays, many of those rules are codified in statutes, regulations and case law. In a sense, such rules provide a specification of how the society should be, how civilians in a society should behave, and how the society’s government and civil servants should behave. Though regulations differ from technical specifications, and are written in natural language, rather than specified in a formal language or schematics, they are nonetheless not that different. Just like technical specifications conform to certain guidelines, there are extensive manuals governing legislative drafting, based upon traditions and experiences. Just like engineers, legislative drafters base their work on established best practices and design patterns. Because of this, the legislation is actually far more regular than one might expect from natural language texts.

Within the field of Computer Science & Law, the branch of computer science that deals with law, people attempt to make legislation more easily accessible, and uses this regularity of the legislation as a basis to deal with it. One way they do this is by creating portals that allow users to search for legislation, using keywords, titles, etc. This kind of application is widespread, and many legislative bodies provide such a portal, such as legislation.gov.uk in the United Kingdom, wetten.nl in the Netherlands, Norme-In-Rete1 (Laws on Line) in Italy, LexML Brazil2, the online version of the United States Code3 and EUR-Lex4, the European Union’s database of regulations. In such portals, regulations are not stored as plain text, but in a more structured format, such as HTML, SGML or XML. This makes it possible to point users to a specific section of a document. Furthermore, there is often a lot of meta-data: information about the documents, such as date of enactment, date of repeal, jurisdiction, etc., each of which can help users to find the information they need.

However, for most users without a legal background, such applications are not helpful enough. Just like someone with a medical problem is looking for a diagnosis, and not just for medical information, a citizen is often looking for an answer to some legal problem, such as: Am I allowed to build a dormer on my house? For him, getting a link to the relevant regulations is an insufficient answer; he is looking for a simple yes or no. Thus, people within Computer Science & Law are also working on systems that provide these answers, by questioning the user about his situation and then applying the rules. These systems provide an answer to the question, as well as an explanation linked to the regulations.

A great variety of such systems have been created, covering a various topics and jurisdictions. A (random) selection includes a logic program covering the British Nationality Act (Sergot et al, 1986), the TAXMAN system, dealing with the United States’ corporate tax law (McCarty & Sridharan, 1980, 1981), KRIP (Nitta & Nagao, 1986) and its successor KRIP-2 (Nitta, Nagao & Mizutori, 1988) focusing on Japanese patent law, ESPLEX (Biagioli, Mariani & Tiscornia, 1987), a system on agricultural tenancies in Italy, CLIME (Winkels et al., 1998) dealing with

---

1 http://www.normeinrete.it/
2 http://projeto.lexml.gov.br/
3 http://uscode.house.gov/
4 http://eur-lex.europa.eu/
ship classification (for among others access to ports and insurance), the XML encoding of the Ontario Freedom of Information and Protection of Privacy Act (Powers, Adler & Wishart, 2004) and the system for the Dutch Legal Aid Service Counter (van Engers et al, 2006).

Though not all of the systems above have been used by the users for whom they are intended, interest for such systems grows. This is evident from the fact that several systems have been created on the request of a government or company. In the document *Automated Assistance in Administrative Decision-Making*, the Australian government (2007) notes “these systems have the potential to improve the accuracy, consistency and transparency of administrative decision-making”. Several companies make a living by developing and maintaining such systems, such as BeInformed (BeInformed Studio) in the Netherlands, Oracle (Ruleburst) in the United States and Knowledge Tools in Germany.

Each of these systems is based on a formal model of the law, though the precise formalism differs for each system. In the examples above, we already mentioned logic programming, XML, OWL and UML as possible modelling languages. We’ll continue with some examples of such models.

The first one is taken from Sergot et al. (1986), who model the British Nationality Act. Their program attempts to determine whether or not someone is a British citizen. This example is about the first subsection of the act, namely:

**British Nationality Act 1964, section 1.-l**

A person born in the United Kingdom after commencement shall be a British citizen if at the time of birth his father or mother is

(a) a British citizen; or
(b) settled in the United Kingdom.

Sergot et al. have formalised the act in Prolog. For this subsection, without items (a) and (b), the formal representation reads:

\[
x \text{ is a British citizen}
\text{ if } x \text{ was born in the U.K.}
\text{ and } x \text{ was born on date } y
\text{ and } y \text{ is after or on commencement}
\text{ and } x \text{ has a parent who qualifies under 1.1 on date } y
\]

The first line forms the conclusion that can be drawn, namely that someone (represented by \( x \)) is a British citizen. The other lines form the conditions that must be fulfilled in order to draw that conclusion. The time of his birth (labelled \( y \)) must be after commencement, and at that time, his parents should qualify for either item (a) or item (b) from the original text. Sergot et al. have added an intermediate concept (has a parent who qualifies under 1.1 on date \( y \)) to represent this. Item (a) and (b) are modelled separately. The rules representing those items have the intermediate concept as their conclusion.

---

\(^5\) This is not the final model used by Sergot et al., as it does not contain all the information they need for their complete program.
The representation for item (a) is:

\[
x \text{ has a parent who qualifies under 1.1 on date } y \\
\quad \text{if } z \text{ is a parent of } x \\
\quad \text{and } z \text{ is a British citizen on date } y
\]

As said, the conclusion is the intermediate concept, while the condition is that a parent of \( x \) (labelled \( z \)) should be a British citizen on the appropriate date. Note that Sergot et al. have applied some common sense knowledge, and have modelled father or mother as parent.

To complete the model of subsection I.(-I), the model of item (b) is added, which is similar to that of item (a):

\[
x \text{ has a parent who qualifies under 1.1 on date } y \\
\quad \text{if } z \text{ is a parent of } x \\
\quad \text{and } z \text{ is settled in the U.K. on date } y
\]

A representation that is somewhat further removed from natural language is given by Gordon (2008), who creates his models in the LKIF rule language. These rules can be used with the Carneades argument system to construct, evaluate and visualise arguments about a legal case. He gives an example from German family law:

\[
\text{§1601 BGB (Support Obligations)}
\]

Relatives in direct lineage are obligated to support each other.

Like the rules used by Sergot et al., the rules created by Gordon deal with a conclusion that follows from a set of conditions. Gordon’s representation of the sentence given above is:

\[
\text{(rule } \$-1601-\text{BGB)} \\
\text{ (if } (\text{direct-lineage } ?x \, ?y) \\
\text{ (obligated-to-support } ?x \, ?y))\]

The first line is merely a name for the rule (which is a reference to the section of the law that is modelled). The second line gives the conditions, in terms of two people labelled \( ?x \) and \( ?y \). The conditions states that if there is a direct-lineage relation between the two, the conclusion may be drawn, which is that there is also an obligated-to-support relation between them.

Another example, from van Engers et al., (2001), uses UML/OCL to model the Dutch income tax law. The example deals with the sentence:

\[
\text{Income Tax Act 2001, article 3.86, sub 2} \\
\text{The cycling deduction defined on the basis of the following subsections is reduced with the travelling expenses received for the travelling distance travelled by bicycle.}
\]
The model for this sentence is:

This model shows the different concepts from the sentence as a class: the *cycling deduction*, *travelling distance* and *travelling expenses*. The expenses and the deduction have a property *amount*, which indicates the amount of expenses and deduction. The distance has a property that indicates whether or not the distance was travelled by bicycle. There is a fourth concept, which is a reference to the *following subsections*, which need to be included separately, and which is represented in the model with dashed lines.

The distance and the expenses are related to each other, as the text deals with *travelling expenses received for the distance travelled*. This relation is indicated by a line between the two classes. Each distance can be associated with one amount of expenses, or not, which is indicated by a 0, 1 next to the line. Each amount of expenses must be associated with a distance travelled, which is indicated by a 1 next to the line.

The actual rule is given in OCL as a restriction related to the cycling deduction. It reads:

```ocl
{-- section 3.86 sub 2
    amount = amount@pre - TravellingDistance.allInstances->
        select(travelledByBicycle = true)->collect(travellingExpense.amount->sum
}
```

The first line is a reference to the text on which this rule is based. The text says that the cycling deduction is reduced with the travelling expenses that have been received, which is indicated by the first part of the rule: the (new) amount is the old amount (amount@pre) minus something. This something is collected as follows: take all travelling distances (i.e. all the trips the tax payer regularly makes), select from those distances the distances travelled by
bike, then select for those the amount of travelling expenses received, and finally sum these expenses.

Creating such complete, formal and executable representations of the law takes a lot of effort. This is a problem with any knowledge-based system, and has become known as the knowledge acquisition bottleneck (Feigenbaum, 1977). There is a lot of knowledge that has to be modelled, and often much of that knowledge is in the heads of experts. Moreover, the knowledge is often implicit, meaning that the experts themselves are not even aware of all that they know and use. Even when the information is documented or explained by the experts, much of it is left implicit, with the assumption that the receiver of the information is already aware of those parts. In addition, these experts are often valuable, and have limited availability. Likewise, the experts creating the model are valuable, making the creating of such model an expensive endeavour. We investigate ways to partially overcome this bottleneck for modelling the law.

In this thesis, we will focus on legislation. Legislation has the advantage that is has already been made explicit: it is already in text, rather than in the heads of experts. However, it is still a huge amount of knowledge that needs to be modelled. When it comes to legislation, two approaches are followed in order to reduce the effect of this bottleneck using computer applications: creating a model at the same time that the text is created and creating a model after the text has been created.

Creating a model of the text at the same time as the text requires specialised editing software. These editors require the user to provide information about the text they are writing at the same time as they are writing it. At the moment, editors that create a full, executable model of a text do not exist. There are, however, editors that help to create a document with sufficient metadata to allow for easy searching and linking. An example of such an editor is discussed in section 6.2.

Creating a model after the text is created can be done with automated methods of translating regulations, which concentrate on the recurring patterns that appear in the text. It has been attempted before in the POWER project (van Engers et al., 2000) and it successor (E)POWER. Within this project, the proposal for a new Dutch Income Tax law was modelled in UML with OCL rules (see the example given in figure 1). This model was then used to simulate the effects of the new law, and to detect any inconsistencies. Within this project, a tool called OPAL (Object-oriented Parsing and Analysis of Legislation) was developed which creates models for noun phrases that appear in legislative texts using specific patterns (van Gog & van Engers, 2001), thus performing a small step of the modelling process automatically.

This method may seem to become obsolete, now that editors are being developed to create a model at the same time as the text is being drafted. Still, this is a valid approach to pursue. First of all, the editors are certainly not yet perfect, and even when they are, it will be some

---

6 In theory, there is an obvious third approach, which is to create the model before the text is created, and then create the text based on the model instead of the other way around. Though there have been experiments with designing parts of legislation using decision tables (Overhoff and Molenaar, 1991), using such an approach for the entire law would require legislative drafters to acquire several new skills. Our focus is on methods that connect well to the current practice.
time before every legislative drafter has adopted them and each legislative document is drafted in this manner. Finally, these automated methods may actually be used within the editors, as import filters and as an additional component that recognises the text while the drafter is typing it, reducing the effort needed by the drafter to add all metadata to the document.

This second approach, automated methods of translating regulations, is the focus of the research presented here. The research is restricted to Dutch laws. Thus, the main question is:

To what extent can the creation of models of Dutch law be automated?

With Dutch law, we refer to Dutch law texts, not to the entire system of the law. In addition to relieving the acquisition bottleneck, a repeatable method for translation will also increase the inter-coder reliability of the modelling process. While the focus is on the (repeatable) automated translation, another aim of this research is to gain insight in the constructions used in legal texts.

For the translation process, we assume that the input is a Dutch legal text without any metadata or layout (which often introduces implicit metadata). The output should, as much as is possible, be isomorphic with the original texts, as recommended by Bench-Capon and Coenen (1992). For a clear definition of isomorphism, they refer to Karpf (1989), who formulates the following conditions that a representation must fulfil in order to be isomorphic:

1. Each legal source is represented separately.
2. The representation preserves the structure of each legal source.
3. The representation preserves the traditional mutual relations, references and connections between the legal sources.
4. The representation of the legal sources and their mutual relations is separate from all other parts of the model, notably representation of queries and facts management.
5. If procedural law is part of the domain of the model then the law module will have representation of material as well as procedural rules and it is demanded that the whole system functions in accordance with and in the order following the procedural rules.

The translation from natural language to formal models is not done in a single step. Broadly speaking, it takes part in two phases. First, we need to make the structure of the document explicit, and then we can continue to model the meaning of the document.

Detecting the structure means that we identify the different parts of the document, such as chapters, articles and paragraphs, and then add metadata to the document to mark those parts. This serves a number of purposes:

- Identifying the different parts of the document allows for referring to those specific parts rather than to the document as a whole;
- Identifying the sentences is the first step in processing the actual text (as this is based on the grammatical structure of sentences);
- Separating headers and indexes from the actual content makes it possible to skip those headers and indexes when creating models of the content.

As part of this research, the approach was only (systematically) tested using laws, though the same approach should also be valid for other legislative texts.
After the parts have been identified, we continue by searching for references in the text, which refer to the structure parts of the document itself and other documents. Like with the structure, metadata is added to the document to mark the references and to link them to their target. At this point, the document with metadata should be similar to the annotated documents used by portals such as wetten.nl (though without metadata on the history of the document). This means that the next steps can either be performed on documents that have been structured in this way, but also on (sufficiently) annotated documents from such portals.

For the creation of the actual model, we break the document apart into individual sentences, as identified when structuring the document. As we have seen in the examples above, many models are composed of partial models that represent individual sentences. This is also important in order to create an isomorphic model, as this means that the structure of the legislative source is preserved. As a preliminary step to the modelling, each sentence is classified. In the (E)-Power project this classification was left implicit and the step from the surface structure of sentences to a formal representation was typically too large to have useful automatic translations.

After a sentence has been classified, a model for that sentence is created, based on its classification, any signal words or language patterns that occur and (optionally) the grammatical parse of the sentence.

Finally, these individual models are combined into an integrated model. The different sentences in the law may refer to each other and to the same concepts. In this way, they are linked together. Thus, the models should be linked in the same way, by recognising when the sentences are linked and then creating a similar link between the individual models. This step involves a lot of common sense knowledge, which is not present in the text itself. Because of that, it cannot be (fully) automated.

Thus, by following these steps, we create a model of a legislative text. When such a model is coupled with (models of) knowledge from other sources, such as case law, legal tradition, juridical principles, etc., systems as mentioned above can be created. The entire process is depicted in figure 2.

In the next chapters, these steps will be further discussed.

Chapter 2 deals with the structure of Dutch laws, discussing the different elements that make up a law and the clues that the text contains to automatically detect these elements.

Chapter 3 discusses references in texts and the different textual forms they can take. Also discussed is a parser created to detect those references, and the way these references can be resolved, so that it is known to what they refer.

Chapter 4 proposes a classification of the provisions that occur in Dutch laws, and discusses two approaches on classifying provisions: a parser based on textual patterns, and an approach based upon machine-learning.
Chapter 5 shows how a frame-based representation for provisions of each type can be automatically created, by splitting the text using either textual patterns or a full-fledged grammar parser for the Dutch language.

After this discussion of the entire process, some applications that are based on partial results of this process are described in chapter 6, followed by the conclusions.

The Dutch government publishes its regulations in Dutch only. Sometimes, a translation is created by the Ministry of Foreign affairs, but for most laws, no such translation is available. Unless noted otherwise, the translations provided in this text are created by the author to illustrate the research done, and are not official. Moreover, the translation that has been chosen is not always the most elegant. Instead, the aim has been to create translations whose structure resembles the structure of the original Dutch sentences.

The research presented in this book has been conducted over period of several years, and many partial results have been published before. In particular, parts of chapter 3 have been published before in de Maat, Winkels and van Engers (2006) and de Maat, Winkels and van Engers (2009). Parts of chapter 4 have been published in de Maat and Winkels (2007, 2008, 2009, 2010), de Maat, Winkels and van Engers (2009) and de Maat, Krabben and Winkels (2010). A part of chapter 5 has been published in de Maat and Winkels (2011a, 2011b). Parts of chapter 6 have appeared in Winkels et al. (2005) and de Maat, van de Ven, Winkels and van Engers (2009).
2 Parsing Structure

Text is comprised of words, which combine into sentences. Sentences combine into paragraphs, and paragraphs become chapters. Aided by punctuation and layout, most humans do not have any problem distinguishing this structure. For automated processing, though, the implicit rules that people use to recognise this need to made explicit. This is a fairly straightforward task, though laws have a somewhat more complex structure than most texts.

Most laws follow a common structure. Legislators have tried to create uniform documents in order to increase legibility of the laws. This has led to explicit rules or guidelines, in for example, Africa\(^8\), the European Union\(^9\) and the Netherlands\(^10\). These documents discuss a wide variety of information related to so-called legislative drafting techniques. They include topics such as structure, numbering, preferred word use and grammar and template sentences.

In various projects, it has been attempted to make the structure of laws explicit, by modelling it in XML. In the Netherlands, the most notable attempts are BWB DTD, the format used on the official website of the government, and MetaLex (Boer et al., 2002), which later merged with several initiatives from other jurisdictions, such as the Italian Norme-in-Rete (NIR) XML schema and the pan-African Akoma Ntoso schema, to form the European standard CEN/MetaLex (Boer et al., 2009). In these formats, structure elements are marked with a tag that denotes its level (e.g. chapter) or a more generic tag, such as part or hierarchical container.

These projects give us useful insights into the structure of laws, and also point us to the fact that there is a lot of variation. Many of such XML schemas were initially too strict, and had to be made more flexible in order to accommodate the many variations that occur. What they do not give us, is some insight in what the language is that is used to denote the structure. Here, the official guidelines help, but since they focus on what should be, they give limited information on what actually is.

In this chapter, we will discuss the structure of Dutch laws, based upon the information of the guidelines and the study of several laws. After that, we will discuss how we can use this information to perform the first step of the proposed modelling process, as depicted in figure 3. This first step should take as its input a legislative text without any metadata or annotations, and should result in an annotated text in which the structure has been explicitly marked. We will discuss how we can detect the structure based on the information found, and we will present a parser that accomplishes this task.

---

\(^8\) Legislative Drafting Guidelines for persons involved in the drafting of legislation (2011)
\(^10\) Aanwijzingen voor de regelgeving (2005)
The interpretation of what a structure element is, and what should be recognised, follows closely the ideas behind MetaLex, expanded with insights from the official guidelines.

2.1 Structure of Dutch Laws

At top level, Dutch law (or any Dutch regulation) is divided into six parts:

1. title (opschrift);
2. preamble (aanhef);
3. body (lichaam);
4. conclusion (slotformulier);
5. signatures (ondertekening);
6. appendices.

The title of a law is usually rather elaborate, listing the date the law was signed, as well as its purpose, such as:

*Act of July 5th, 2006 on the merging of the municipalities of Medemblik, Noorder-Koggenland and Wognum, and on a modification of the borders of the municipality of Hoorn*\(^{11}\)

If the law has a short title, this short title is included in the title between parentheses:

*Act of October 31st, 2002, on rules concerning the exploration of and the excavation of minerals and concerning activities related to mining (Mining Act)*\(^{12}\)

The title is followed by the preamble, which starts with greetings from the king or queen. Currently, under Queen Beatrix, the preamble starts with:

---

\(^{11}\) Wet van 5 juli 2006 tot samenvoeging van de gemeenten Medemblik, Noorder-Koggenland en Wognum en een wijziging van de grenzen van de gemeente Hoorn

\(^{12}\) Wet van 31 oktober 2002, houdende regels met betrekking tot het onderzoek naar en het winnen van delfstoffen en met betrekking tot met de mijnbouw verwante activiteiten (Mijnbouwwet)
We Beatrix, by the grace of God, Queen of the Netherlands, Princess of Orange-Nassau, etc., etc., etc.
To all who see or hear this, greeting! Let it be known:13

For the previous monarchs, the preamble is of course slightly different, but the general structure remains the same14. After these greetings, some considerations and motivations for the law are mentioned in a sentence starting with Whereas We have considered.

Whereas We have considered that it is desirable to replace the existing system of legal regulations concerning the exploration for and production of minerals by new regulations, which fulfil current requirements, and to lay down a number of rules concerning certain activities.

The preamble ends with the sentence:

We, having heard the Council of State, and in having consulted the States General, have approved and understood, and we now approve and understand as follows:15

After the preamble, the body of law follows. The body is usually the largest part of the law, and contains all the rules. As such, it is the most important part for this research. The body of a law is divided into articles16, which may be grouped into several super levels. These levels are named (from smallest to largest):

1. Section (paragraaf)
2. Division (afdeling)
3. Title (titel)
4. Chapter (hoofdstuk)
5. Part (deel)
6. Book (boek)

This order (book being the largest part, section being the smallest part) has been established by the official guidelines17. However, laws and regulations do deviate from this order. For

---

13 Wij Beatrix, bij de gratie Gods, Koningin der Nederlanden, Prinses van Oranje-Nassau, enz. enz. enz.
Allen, die deze zullen zien of horen lezen, saluut! doen te weten:

14 There are two obvious changes to the first sentence of the preamble: first of all, instead of Beatrix, the name of the previous monarch is used, and secondly, if it is a King instead of a Queen, the titles will be King and Prince rather than Queen and Princess. William I, II and III also listed the title Grand Duke of Luxembourg.

15 The most different opening occurred during Emma’s regency. For laws from this period, the first line of the preamble is replaced with:

In the name of Her Majesty Wilhelmina, by the grace of God, Queen of the Netherlands, Princess of Orange-Nassau, etc., etc., etc.
We Emma, Queen-Widow, Regentess of the Kingdom,

In naam van Hare Majesteit Wilhelmina, bij de gratie Gods, Koningin der Nederlanden, Prinses van Oranje-Nassau, enz., enz., enz.
Wij Emma, Koningin-Weduwe, Regentes van het Koninkrijk,

16 If the law is very short, it may consist of a single article.

17 Aanwijzing 97, vierde lid
example, the Act on the Organisation of the Market in Health Care\textsuperscript{18} has three levels, chapter, section and division, with chapter being the largest part and division being the smallest. The guidelines also state that the level of “book” should only be used in a code of law.

In addition to these “named” levels, some older laws use one additional level, below section (and above the articles). These levels only have a title (indicated with a sub-header); they have no category (like section or chapter) or index.

The articles themselves can consist of the following:

- text;
- several numbered subparagraphs;
- text followed by several numbered subparagraphs (This usually occurs in amending laws, where the text is a sentence setting the scope for the changes that are described in the subparagraphs)(see section 4.4.3).

The subparagraphs can consist of text and subparagraphs. Most regular laws will not have subparagraphs within subparagraphs, but it is rather common for amending laws.

Within the text of a paragraph or subparagraph, one more structure element can occur: the list. A list consists of an introduction, several list items and optionally a conclusion. Though the list forms a single sentence (i.e. the individual list items are not complete sentences), the list items are often referenced separately (usually in conjunction with the introduction and the conclusion). Thus, it is necessary to identify the list items.

The body of the law is followed by a single, fixed sentence\textsuperscript{19}:

\textit{It is hereby ordered that this act will be published in the State Gazette, and that all ministries, authorities, bodies and officials shall ensure that this act is accurately implemented.}\textsuperscript{20}

This sentence is followed by the signatures of the reigning monarch and the ministers involved. After this sentence, there can be some appendices.

\textbf{2.2 Numbering}

With the exception of the level of the sub-header, the various parts of the regulations are numbered. The precise numbering used differs between regulations. The Guidelines advise to use Arabic numbering for all levels above article. Articles should also be numbered using Arabic numbers. This can either be done throughout the entire regulation or per chapter, with the chapter number added in front of the article number. If there is only one article, it is identified as \textit{Only article} (as opposed to \textit{Article 1}). List items are numbered \textit{a}, \textit{b}, \textit{c}, etcetera, with

\textsuperscript{18}Wet marktordening gezondheidszorg

\textsuperscript{19}There exists one variation on this sentence, for those laws that apply not only in the Netherlands, but also in the Netherlands Antilles and Aruba, in which case the sentence starts with: It is hereby ordered that this act will be published in the State Gazette, the Official Journal of the Netherlands Antilles and the Aruba Gazette

\textsuperscript{20}Lasten en bevelen dat deze in het Staatsblad zal worden geplaatst en dat alle ministeries, autoriteiten, colleges en ambtenaren wie zulks aangaat, aan de nauwkeurige uitvoering de hand zullen houden.
sub-lists numbered 1°, 2°, 3° etcetera. Change laws deviate from this numbering, and use
capitalised Roman numerals for articles and capitalised letters for subparagraphs.

Many regulations do not conform to the Guidelines (many have been written before the
guidelines were published). Hence, you may find structure elements that use a different
numbering than presented above, and you may also find numbering using ordinals and lower
case Roman numerals.

In addition, when pre-pending a chapter’s number to an article’s number, there are a several
different styles being used:

- Separate using a full stop: Article 3.15 (Income Tax Act 2001)
- Separate using a colon: Article 3:15 (General Administrative Act)
- Separate using a space: Article C 15 (Electoral Act)

When a law is expanded after its initial publication, newly inserted parts also need a number.
Paragraphs and list items are often (but not always) simply renumbered. For other levels, this
is not usually the case. Instead, the new part is given a number consisting of the number of the
last (regularly numbered) previous part, with some suffix to distinguish it. For example, a new
article inserted between articles 15 and 16 will often get assigned the number 15a.

Common suffixes include:

- Letters: Used with Roman and Arabic numbers: Article 212m (Bankruptcy Act)
- Arabic numbers: Used with letters: Article 1, item x1 (Higher Education and Scientific
  Research Act) or Article 1, item s.1. (Medication Act)
- Latin numerical adverbs: bis, ter, quater, quinies, seces, septies, octies: Article 12ter (Military
  Officials Act)

These suffixes can be combined if a new part is inserted between two parts that already have a
suffix in their number. For example, in the Penal Code, between articles 77h and 77i, an article
77h bis has been inserted. Likewise, between articles 77w and 77x, the articles 77wa, 77wb, 77we
and 77wd have been added.

Of course, these suffixes can also be used in combination with combined chapter/article
numbers, leading to more complex numbers like article 9.30a (Higher Education and Scientific
Research Act).

### 2.3 Automated Detection

In order to detect the structure of a law, we will have to detect all the divisions described
above. We will discuss the different “layers” separately, resulting in four steps:

---

21 Wet inkomstenbelasting 2001
22 Algemene wet bestuursrecht
23 Kieswet
24 Faillissementswet
25 Wet op het hoger onderwijs en wetenschappelijk onderzoek
26 Geneesmiddelenwet
27 Militaire Ambtenarenwet 1931
1. separate the document into its basic parts (title, preamble, body, conclusion, signatures and appendices);
2. separate the body of the document into its different levels (chapter, etc.) and articles;
3. separate articles in numbered paragraphs, when appropriate;
4. split article and paragraph text into sentences and lists.

2.3.1 Separating the Different Parts of a Law

Separating the different parts of the document is an easy task, due to the fixed formulas that are used for the preamble and the conclusion. These fixed sentences can easily be found in the text, and with that, the entire preamble has been found (being the first sentence, the last sentence and all text in between). This procedure can even be simplified by identifying the preamble as the piece of text starting with the first occurrence of \textit{We} and ending with the first occurrence of \textit{Let it be known}.\footnote{With the exception of Emma’s laws, this approach will cover all different first lines of the preamble, as they all start with \textit{We}. A separate rule will still be needed to identify the preamble of the laws drafted during Emma’s regency.} This of course assumes that \textit{We} never occurs in the title of the document, and \textit{Let it be known} does not occur in the preamble other than at the end.

As the conclusion starts with a fixed sentence (see above), the start of the conclusion can be found in a similar manner. A third part of the law that can be easily identified are the appendices, which all have a header starting with \textit{appendix} (\textit{bijlage} in Dutch). The detection of headers is discussed in more detail below.

So, with these fixed elements, the preamble, conclusion and the appendices can be identified. This also gives us enough information to establish the boundaries for the other parts:

- the title is the first part of the document, before the preamble;
- the body is the part between the preamble and the conclusion\footnote{Alternatively, the start of the body can be identified by the first header of an article or chapter, etc. that appears in the text. This may be useful for dealing with documents from which the pre-amble has been removed, and for dealing with the books from the Civil Code, which are often dealt with as if they were separate laws.};
- the signatures are the part between the conclusion and the appendices.

This approach can also be applied to other Dutch regulations. The overall structure is the same, and fixed formulas are used in those documents as well, though the precise text differs from those used in laws.

2.4 Splitting the Body of a Document in Parts and Articles

As mentioned above, the body of a document is split into several parts. These parts identify themselves by means of a header. Some examples of headers:

\textit{Article 1}

\textit{Title 18. Absence, disappearance and determination of death in certain situations}\footnote{Burgerlijk Wetboek Boek 1: Titel 18. Afwezigheid, vermissing en vaststelling van overlijden in bepaalde gevallen}

\textit{Title 1.1 Definitions and scope}\footnote{This approach will cover all different first lines of the preamble, as they all start with \textit{We}. A separate rule will still be needed to identify the preamble of the laws drafted during Emma’s regency.}
§ 1. Definitions

First division. On the declaration of bankruptcy

As you can see, there are three elements in such a header:

- a category label, such as Titel, Afdeling or §;
- an index, such as 1 or 1.1 (as discussed in section 2.1);
- optionally, a title.

So, in order to detect a header, we can search for patterns consisting of a category followed by an index followed by any other words (which form the title). Such a pattern can be written down in a grammar or regular expression (see appendix A) for an example of this.

Since a header always appears on a new line, we can also include the newline tokens in our patterns, including a newline both before and after the header. This will prevent several false positives, as otherwise we would often detect a header where in fact there was only a reference to a certain part.

Still, it is not guaranteed that we will not find false positives. With this general structure, we will still find any line in the document that happens to start with a category label followed by an index. There are two types of false positives we may encounter:

1. Sentences that happen to start with a category label followed by an index, such as:
   
   Article 12 does not apply.  
   
   With the suggested regular expressions, this sentence could be classified as a header with category article, index 12 and title does not apply.

2. Lines that start midway a sentence because newlines have been applied to control the flow of the text:
   
   … as meant in
d
   
   article 12 …

   As above, this would result into a heading denoting article 12.

Some of these errors can be avoided by adding the information that a header never ends with a full stop or a colon, whereas regular sentences do. This will reduce the number of type #1 false positives, as any sentences that end on the same line will have a full stop.

Though in general the number of false positives for headers is very small (as is confirmed in our experiment described in section 2.8), we can further reduce it by making a number of assumptions.

Errors of type #2 can be avoided if it is assumed that a header is always preceded by a blank line. As there will not be a blank line between two lines of the same sentence, this will prevent a label followed by an index inside a sentence to be classified. It may also prevent errors of

---

31 Algemene wet bestuursrecht: Titel 1.1 Definities en reikwijdte
32 Drank- en Horecawet: § 1. Begripsbepalingen
33 Faillissementswet, titel 1: Eerste afdeling. Van de faillietverklaring
34 In the remainder of this section, I'll discuss the issues in terms of a category label followed by an index, though all of it also applies to an ordinal followed by a category label.
type #1, as this will limit the chance of such an error occurring to the first sentence of a paragraph.

Another way to avoid errors of type #2 is to assume that a header always starts with capital letters, while a reference within a sentence does not.

These assumptions are likely to hold, unless the input text is derived from a source that had more layout information. For example, an HTML or Word document may have had the headers marked-up, using that information to use increased line spacing (rather than a blank line) to visually separate the headers. If such a source is stripped of its layout information, the result can be a document in which the headers are not preceded by blank lines.

Another way to avoid false positives is to run a post-processor afterwards to eliminate false positives by looking at the numbering. A heading for article 12 appearing between articles 17 and 18 is unlikely to be correct, especially if another article 12 header has been found before.

The regular expressions presented above, together with their improvements, are sufficient to detect most common headers. However, there are some headers they cannot deal with, all related to titles appearing without a category/index pair.

As mentioned earlier, some of the older laws include a level marked by a sub-header, which consists of only a title. For example, the first articles in the Disownment Act are grouped under the title General provisions. Where most (possible) headers can be recognised because they start with a category/index pair, such a clue is missing for these headers. These lines can be distinguished from regular sentences in the text because, like other headers, they do not end in a full stop. However, not every line that does not end with a full stop is a title; it can also be a sentence that is continued on the next line. In order to recognise these titles, we need to look at the context. An incomplete sentence will be followed by the remainder of the sentence, and eventually a full stop. A title will be followed by a blank line or an article header.

2.5 Splitting Articles

In order to split articles into sentences, existing tools can be used such as the ANNIE Sentence Splitter (Maynard et al., 2002). Such tools split text into sentences based on punctuation, capital letters and whitespace. However, next to sentences, articles can also contain two other elements: subparagraphs and lists.

Unlike chapters, articles, etc. subparagraphs do not have a full heading. Instead, they only have an index, followed by a full stop, with the contents of the subparagraph following directly. The line preceding the index is either the article heading, a scope declaration (ending with a colon)(see 4.2.1) or the last sentence of the previous subparagraph (ending in a full stop). Subparagraphs can be numbered using Arabic numbers (used in regular laws and for sub-subparagraphs in amending laws) or capital letters (in amending laws). Like headings, these

---

35 Onteigeningswet
36 Algemeene bepalingen
37 There exists a variation in which the index and the full stop are enclosed in square brackets, e.g. [1.] This is often used in older laws (those published around 1900).
indices can be detected using patterns. The start of a new subparagraph denotes the end of the next subparagraph.

List items are similar. They are numbered using lowercase letters (a, b, c, etc.), with sub-lists numbered 1°, 2°, 3° etc. The line preceding the index is a blank line or a line ending in a colon or a semicolon or a semicolon followed by and or or. If list items have been detected, the introduction of the list (and optionally, the conclusion of the list) should also be marked. The conclusion will start at the end of the last complete sentence (detected using the sentence splitter), and end at the start of the first list item. If the last list item ends with a semicolon, then the text that follows, until the start of the next sentence, is the conclusion. The items of a list and its sub-list(s) should be separated based upon the numbering used.

2.6 Recognising Quoted Text to be Added Elsewhere

Sentences that introduce new text in an existing law (insertions and replacements) contain this new text, which may include headings and indices, such as in the example below:

```
Act of December 17th, 2009 (Stb. 2010, 8), article I, sub D

After chapter 8, a chapter is inserted, reading:

CHAPTER 8A. PARTICIPANT AND PARENT REPRESENTATION; NATIONAL REPRESENTATION DISPUTES COMMITTEE

TITLE 1. GENERAL PROVISIONS

Article 8a.1.1 Definitions
In this chapter is understood by:
...
```

This new text is not explicitly marked, and the headers contained in the new text can disrupt the parsing of the document. Because of that, it is important to detect new text.

The new text starts after the colon. However, not every piece of text that follows a colon is new text. A colon can also indicate the start of a list, the start of several subparagraphs or the start of the actual definition in a defining sentence. Since a colon in these cases is followed by list items, subparagraphs or normal text (i.e. not a heading or index), the presence of a header for an article or higher-level structure element is a clear indication that it is the start of new text.

38 If a sub-list ends in a semicolon, it may be followed by the next item in the main list instead of a conclusion.
39 Wet van 17 december 2009 (Stb. 2010, 8), artikel I, lid D
40 Many sentences will also quote existing text. However, these quotes are always marked (using double angle quotation marks), and do not lead to complications in the way new text does.
text. If a colon is followed by an index for a list item or subparagraph, then there are some
heuristics that may apply:

- If there is only one list item or subparagraph, then the text is more likely to be new
text.
- If the first index is not 1 (or a or I, etc.), then the text is more likely to be new text.

However, these heuristics are insufficient to distinguish between all cases. A text replacing the
first two list items will have the same structure as a list with two list items (i.e. two list items,
one with index a. and one with index b.) and cannot be distinguished based on punctuation
and headers. The same is true for text that does not contain headers or indices. A better way
to determine whether the text following the colon is new text is to classify the sentence that
precedes the colon, using the methods and classification described in chapter 4. If the
sentence is a replacement or an insertion, the text following the colon must be new text;
otherwise, it is not.

Now that we know where a new text starts, the next step is to determine where it ends. There
is no clear demarcation of the end of new text. Often, only one structure element (e.g. one
article or one list item, etc.) is inserted, but even if we assume that the new text contains only
one element, the end can be difficult to determine. For example, a subparagraph may contain
new text containing an article with subparagraphs. The article ends at the beginning of the
subparagraph following the original subparagraph, but based on structure alone, it is not
always possible to distinguish between that following subparagraph and a subparagraph that
belongs to the (new) article. In order to make the distinction, the index of the different
elements needs to be taken into account. First of all, the format of the index can already give
sufficient information to make the distinction. In an amending law, the articles and
subparagraphs use a different index format than in a normal law. Thus, when a amending law
modifies a normal law, new articles and subparagraphs can be distinguished from regular text
because of the format they use. However, there are a lot of situations that aren’t covered by
this: regular laws modifying regular laws, modification of different structure levels than articles
and subparagraphs and even sometimes amending laws modifying amending laws. Here, we
can apply a different heuristic: new text will always have contiguous numbering. So, as soon as
an index does not follow the previous one, that marks the end of the new text. Of course,
with this approach, if the indices in the new text by accident precede the index of the
following regular text, the end of the new text is still not detected correctly. This will occur
rarely, but if so desired, there are two more pieces of information that can be used to reduce
the number of errors:

41 New text that does not contain indices or headings will not disrupt the structure recognition of the entire
document if not detected, but to remain consistent, it is desirable to mark all new text as such.
42 Normal laws use Arabic numbering for both articles and subparagraphs; change laws use capital Roman
numerals for articles, and capital letters for subparagraphs.
43 There are other pieces of information that seem helpful, such as:
- A law will not contain two articles (or chapter, etc.) with the same index twice. Likewise, an article will
  (usually) not contain two subparagraphs or two list items with the same index. So if an there are e.g. two
  articles with the same index, one of them must belong to new text.
• If it can be assumed that the law is contiguously numbered (i.e. no indexed structure elements have been repealed or inserted) then this information can be used to determine which parts of the text does not belong to the new text (e.g. if a law contains the articles 1 to 30, and the heading article 12 occurs only once, then that article cannot belong to new text). Alternatively, one could go for the more relaxed assumption that if a structure element could belong to the regular text (i.e. it is not out of place there), it most likely belongs to the regular text and not to the new text.

• Insertions and replacements will often quantify the amount of elements that are inserted (e.g. one article is inserted). This can help in finding the end of the new text; if only one article is inserted, and the beginning of a new article is found, it is clear that that next article does not belong to the new text. However, this is no guaranteed method for finding the end of new text: if a subparagraph inserts an article, the next element may be another subparagraph and not an article.

2.7 Tables, Images and Other Content

Laws may contain tables, and could even contain images and other non-textual content. The way these are embedded in the document varies between document formats; thus, there is no generic way to detect them.

2.8 Experiment

The methods above for recognising the structure of a document have been (partially) implemented in a parser based upon the General Architecture for Text Engineering (GATE)(Maynard et al., 2002 and Cunningham, 2002).

GATE is an extensive, open-source tool for text processing. Among several other things, it allows the processing of text by means of the JAVA Annotations Pattern Engine (JAPE) (Cunningham, 2000). JAPE allows us to apply regular expressions to both strings and existing annotations (instead of just strings, as is common for regular expressions) and then apply new annotations.

We are using a pipeline of four GATE processing resources to recognise the starts and ends of different structure elements in a law.

First of all, we run a tokeniser, which splits the text into tokens such as words, numbers and punctuation. For this purpose, we use the GATE Unicode Tokeniser, which is part of the GATE library.

In general, this default tokeniser works fine for Dutch laws, though it is unable to recognise indexes that consist of a mix of numbers and letters, such as 13a, 1.1 or 2:1:1. It sees these as separate tokens, so:

• 13a is seen as the tokens “13” and “a”;
• 1.1 is seen as the tokens “1”, “.” and “1”;

• The elements of the law will appear in order, so any element that appears to be “out of order” must belong to new text.

This information is superseded by the information that the new text starts at the colon and is contiguously numbered.
Our next step is to apply a set of JAPE rules that recognise such structures and merge the separate tokens into one token, in order to simplify the steps that follow. 

The next step is to separate the sentences in the document. For this, we also use an existing GATE component, the ANNIE Sentence Splitter. In order to prevent the sentence splitter from incorrectly concluding that a full stop following an abbreviation actual marks the end of the sentence, it has to be provided with a list of abbreviations. As few abbreviations are used in Dutch laws, this list could be rather short (see table 1). For future applications, the list could be expanded with a complete list of Dutch abbreviations. The version of the sentence splitter that we used was case sensitive, thus the list did also contain variations for those abbreviations which were sometimes written with a capital letter and sometimes without. Furthermore, the sentence splitter would not recognise an abbreviation when it was preceded by a parenthesis. To circumvent this, the list of abbreviations also contained each abbreviation with an opening parenthesis pre-prended (e.g. not only *Trb.* was in the list, but *(Trb.* as well).

After running the sentence splitter, another series of JAPE rules is applied to recognise the headers in the document, the standard phrases which begin and end the preamble and the conclusion, and the indexes of subparts and list items.

After performing these four steps in GATE, the result is converted to XML using a postprocessor written in regular JAVA code rather than JAPE rules. This code copies most of the GATE annotations (such as the headings) and adds containers based on these headings. For example, for an article in the law that consists of two sentences, the GATE process will have detected:

- the header of the article, split in category and index;
- the end of both sentences;
- the header of the next article.

The postprocessor will generate XML based upon this information, in which it will also mark the complete header of the article as well as the category and index. It will mark the sentences, and it will mark the article as a whole (as derived from the start of this article and the start of the next article). The containers are added by a postprocessor instead of a JAPE transducer because of the varying order in which levels occur. For example, if a law follows the official guidelines, then a chapter is at a lower level than a division, so the start of a division will mark the end of a chapter (but the start of a chapter does not mark the end of a division). However, some laws deviate from these guidelines, so in order to determine the hierarchy, it is necessary

---

44 Rather than using the default tokeniser and then correcting the errors, we could also have made a specialised tokeniser, which will probably have a better performance. For this experiment, we went with the current scenario as it was simpler to execute.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Stands for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enz.</td>
<td>Enzovoort</td>
</tr>
<tr>
<td>No.</td>
<td>Numero</td>
</tr>
<tr>
<td>N°</td>
<td>Numero</td>
</tr>
<tr>
<td>Nr.</td>
<td>Nummer</td>
</tr>
<tr>
<td>Stb.</td>
<td>Staatsblad</td>
</tr>
<tr>
<td>Stcrt.</td>
<td>Staatscourant</td>
</tr>
<tr>
<td>Trb.</td>
<td>Tractatenblad</td>
</tr>
</tbody>
</table>

Table 1: Dutch abbreviations encountered in Dutch legislation
to keep track of the order in which the headings appear in the law, which cannot be done using JAPE regular expressions.

For our experiment, we have not implemented everything that has been described in this section. Most notably, the following is missing:

- recognition of the headers of appendices;
- recognition of quoted text;
- (full) distinction between the items of a list and its sub-list;
- recognition of the conclusion of a list.

The experiment was run on a dataset consisting of ten laws, randomly selected from a list of active non-modifying acts. These acts were downloaded from the portal of the Dutch government, wetten.nl. On wetten.nl, the acts are available in different formats: XML, RTF, HTML and ASCII. As the goal of this experiment is to see whether we can determine the structure in a document that does not yet have this structure marked, we used the acts in ASCII format rather than one of the other formats.

The acts from wetten.nl have line breaks dividing the texts in lines of approximately 80 characters. They contain a number of modifications compared to the official text. First of all, the header of a repealed article is still included in the text, excluding the title, followed by a note inside square brackets noting the fact that the article has been repealed, and when. Likewise, the header of a repealed chapter, department, etc. is also still included, excluding the title. These do not carry a note indicating their repealed status. A second modification is that any articles that contain modifications to other legislation have their content removed and replaced by a note inside square brackets noting the fact that the article includes changes to other legislation. Finally, some references have a number in square brackets added to them.

For the use in our experiment, these files were modified as follows. Each article in which the text had been replaced by a note (i.e. a repealed article or a modifying article) had the note converted to a sentence inside the article, by adding line breaks and a full stop, and removing the square brackets. The notes for references were removed, and the file was converted from ASCII to UTF-8 format.

By retaining the repealed and modifying articles, the test files still had the high-level structure of the complete laws, even though the articles themselves were rather simple. Table 2 shows the acts used, as well as their size measured in number of articles.
Table 2: Structure parsing test set

<table>
<thead>
<tr>
<th>Act</th>
<th>Year</th>
<th>Total Articles</th>
<th>Repealed</th>
<th>Modifying</th>
<th>High level structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Civil Registry Fees Act</td>
<td>1879</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Wrecks Act</td>
<td>1934</td>
<td>14</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Act of June 27th, 1963 (Stb. 1963, 344)</td>
<td>1963</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>Student Grant Act</td>
<td>1986</td>
<td>207</td>
<td>197</td>
<td>0</td>
<td>15 chapters, 21 titles</td>
</tr>
<tr>
<td>Act of February 1st, 1990 (Stb. 1990, 60)</td>
<td>1990</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Act on the Marking of Plastic Explosives</td>
<td>1998</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Act on the State Committee for International Private Law</td>
<td>1998</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Working Conditions Act</td>
<td>1999</td>
<td>70</td>
<td>22</td>
<td>0</td>
<td>8 chapters</td>
</tr>
<tr>
<td>Cableway Installations Act</td>
<td>2004</td>
<td>44</td>
<td>0</td>
<td>4</td>
<td>10 chapters</td>
</tr>
<tr>
<td>International Child Protection Implementation Act</td>
<td>2006</td>
<td>32</td>
<td>0</td>
<td>3</td>
<td>11 chapters</td>
</tr>
</tbody>
</table>

These files were processed by the parser, and the results were checked by hand for errors. For all acts, the parser detected the high-level structure (introduction and conclusion, chapters, titles and articles) without errors. The errors that occurred all occurred within an article (and did not propagate outside of that article). Table 3 shows the number of articles that contain errors for each act: as an absolute number, as a percentage of the number of articles in that act and as a percentage of the real articles in that act (i.e. not counting the repealed and modifying articles with dummy content).

<table>
<thead>
<tr>
<th>Act</th>
<th>Articles with Errors</th>
<th>% Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Articles</td>
<td>All articles</td>
<td>Real articles</td>
</tr>
<tr>
<td>Civil Registry Fees Act</td>
<td>1</td>
<td>86%</td>
</tr>
<tr>
<td>Wrecks Act</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Act of June 27th, 1963 (Stb. 1963, 344)</td>
<td>1</td>
<td>89%</td>
</tr>
<tr>
<td>Student Grant Act</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Act of February 1st, 1990 (Stb. 1990, 60)</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Act on the Marking of Plastic Explosives</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Act on the State Committee for International Private Law</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Working Conditions Act</td>
<td>4</td>
<td>94%</td>
</tr>
<tr>
<td>Cableway Installations Act</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>International Child Protection Implementation Act</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Total</td>
<td>7</td>
<td>98%</td>
</tr>
</tbody>
</table>

Table 3: Structure parsing errors

The assumption that the word *We* does not appear in the title and that therefore the first occurrence of *We* in the document may be seen as the start of the pre-amble (see section 2.3.1) held for these test files.

As was expected in section 2.4, the recognition of headers does not pose a problem. In fact, there were no errors with regards to the headers; all the errors found were related to lists. The Working Conditions Act contained five articles in which the parser had failed to properly process a list. In one article there was a sub-list that ends in a full stop followed by a
semicolon (i.e. “;”). This pattern was not implemented in the parser, which concluded that the list had ended based on the full stop.

In another list, the introduction does not end with a colon. The parser was based on the assumption that each introduction does end with a colon, and failed to recognise the list.

In a third list, there is an abbreviation using dashes, in which the parser mistakenly identified one of the dashes as the start of a new item. This is in fact a bug in the parser, which does not check whether a new list item starts at a new line (but instead fully relies on the output of the sentence splitter to recognise the end of a sentence).

The fourth error occurred in a list where the last item ends in a comma instead of a semicolon. The parser is based on the assumption that any item will end with either a semi-colon or a full stop, and missed the end of this item.

The errors in the other two laws were both also a list in which the introduction did not end in a colon. All three laws in which this error occurred did include other lists in which the introduction did end with a colon, so the use of the colon is not consistent within each law.

Bacci, Spinosa, Marchetti, and Battistoni (2009) have created a similar parser for Italian regulations. Their parser consists of two elements. For the preamble and the conclusion of a regulation, they have opted for a machine-learning approach, using Hidden Markov Models. This is because the patterns in the header and footer are not very precise. Multiple models are available; if the specific subtype (e.g. Act, Bill, Regional Act) of the document is known, a model specific for that subtype is used, if not a generic model is used. The second part of the parser targets the body of the document. It is a non-deterministic finite automaton that tries to parse the body of the document according to the restrictions set out by the NIR XML schema (see Biagioli, Francesconi, Spinosa & Taddei, 2003). It would seem that the preamble and conclusion of a Dutch law are both less informative and more structured than the preamble and conclusion of an Italian law. As shown in this chapter, the preamble and conclusion of a Dutch law consist mainly of fixed sentences, and are easily recognised. In addition, they have little content we wish to mark-up. Hence, for our parser for Dutch laws, we have no need of the fuzzy methods that Bacci et al. needed to handle Italian laws.

As far as the body is concerned, our methods do not differ much. A finite automata, as used by Bacci et al., has the advantage of knowing what is allowed at a certain point in the text, which makes it easier to interpret the text. However, few elements in Dutch laws are ambiguous in this respect; almost any header has only one interpretation. Using a finite automata might handle lists a bit better (as the index of a list is easily confused with that of a sub-list or a paragraph), but our method seems to perform fine on lists as well. On the other hand, finite automata have the disadvantage that they can get stuck if the input text does not conform to the prescribed structure. In this respect, our method is a bit more forgiving as it will start a new element whenever such a header is encountered, which means that an error is almost always contained within a paragraph or article.

Bacci et al. also indicate that in Italian law, like in Dutch law, the inclusion of quoted text makes the process of parsing the document much harder. The exact impact is somewhat
different, though. In Italian law, the entire quoted structure is marked with quotes, which means that they do not have problems determining where a quoted text starts and ends. However, the fact that the quoted text can start at any level of a legal document does increase difficulty, as does the fact that they need to determine whether a modification merely modifies text or also includes structure information. This problem does not occur in Dutch text as there is a difference in the markings of the two: quoted text consisting of a few words (no structure) is marked with double angle quotes, whereas bigger modifications are unmarked, but start on a new line.

2.9 Conclusions

In this chapter, we discussed the structure of Dutch laws and how the text of these laws gives us handles to automatically detect this structure. An experimental parser, implementing most of the ideas, performed very well, recognising the high level structure of law without errors, and correctly recognising the structure of 96% of the articles.

There are a few difficulties with automatic recognition of the structure of laws. Though most of them are solvable, they do increase the amount of effort required. First, there is the variation of indexes used, especially those indexes that contain spaces, and the occurrence of headers without a category. When a law has been drafted using the official guidelines, these difficulties should not occur. The guidelines specify what indexes to use, and do not allow headers without a category. This means that a structure recogniser for laws that are drafted under the guidelines can be simpler than presented above.

Two other issues are not covered by the guidelines. First of all, it is difficult to determine the end of a sub-list without checking the indexes. The guidelines specify that all elements of a list should end with a semicolon or a colon, with the exception of the last item. There is no specific rule for the last item of a sub-list, which sometimes ends with a semicolon (as it is the end of the item in the main list) and sometimes in a full stop (as it is the last item in the sub-list). Specifying a clear way to end a sub-list, such as a full stop followed by a semicolon, such as used in the Working Conditions Act (see section 2.8), will make it easier to make the distinction.

Another problem occurs with large blocks of quoted texts (i.e. quoted text containing entire structure elements rather than a few words). Determining the end of such texts is a complicated task, which would help if there was a clear marker indicating the end the quote. A disadvantage of this is that reliance on such a marker means that any errors introduced because the marker is missing will affect large parts of the document. In general, though, it should give better results with less effort. An alternative is to mark the beginning of each structure element that is part of the quoted text with a marker. This method is used in bills in the United States, which have an opening quote character at the start of each structure element, and a closing quote character at the end of the entire block. This method would also reduce the effort needed to recognise blocks of quoted text, and the impact of a missing marker would be less.

Though our experiment was aimed at Dutch laws, we expect that this method will also work for many other jurisdictions. Other Dutch regulations, such as royal decrees, also have to
follow the official guidelines, and do resemble the laws. Usually, the fixed formulas differ, so patterns have to be added to accommodate those. Also, such regulations may contain text that has not been split into numbered paragraphs, but consists of unnumbered paragraphs, which will also require some slightly different approach. In many other countries, similar structure for legislation is used, which suggests that a similar method will also apply on those regulations (which has been confirmed by the parser used by Bacci, Spinosa, Marchetti, and Battistoni, 2009). Thus, with stricter adherence to the guidelines, and some additions to those guidelines, the legislator can create documents that are easier to parse, which will result in better handling of new laws by a parser. It should be noted, however, that quoted structures of older laws inside new laws will still require the more complicated approach.
3 References

References are the counterpart of document structure. All sources of law are glued together by means of references, and their position in the legal framework is also made clear through references. The next step in our proposed modelling process, depicted in figure 4, is to detect and annotate any references found in a legislative document. This step starts out with a legislative document in which the structure of the text has been made explicit, and annotates any references in that document. This means marking any text that forms a reference, and adding the target to that reference.

Legal texts refer to other legal texts (which may be part of the same document) for a variety of reasons. First of all, lower regulations that are promulgated under the authority of a higher regulation will refer to that regulation to establish their legal ground. Next, regulations that change other regulations need to refer to these regulations in order to make clear where the changes are applied.

Within the rules themselves, references are used for a variety of reasons, such as:

- To set the scope for a definition (indicating that the definition is valid throughout that scope): *for the application of article 12, it is understood by ...*;
- To re-use definitions and descriptions that have been made in other texts: *one of the persons meant in sub 2, item a*;
- To indicate that certain rules only apply if other rules have already been applied: *if article 14 is applied*;
- To indicate that a certain rule is an exception to another rule, and that the other rule does not apply in this case: *in exception of article 17 or article 17 does not apply*;

---

45 Such a higher regulation will have delegated power to a lower regulation by means of a phrase like *By royal decree, specific rules will be set to ...*. In a sense, this is a reference to a future regulation (the royal decree). An important difference with the other references is that these references cannot be resolved based on the information found in the source document, but require the target document (with it reference back) in order to be resolved. These references will not be discussed further in this thesis.
To explicitly indicate that a rule is not an exception to another rule: *Without prejudice to article 17 or article 17 does apply.*

Other documents may include other purposes of references; case law, for example, will indicate which rules have applied and which have been deemed not applicable. Linked together by means of these references, the sources of law form a network.

In Boer et al. (2009), different bibliographic entities are discussed that may be the target of a reference. These are inspired by the functional requirements for bibliographic records (International Federation of Library Associations and Institutions, 1998). The bibliographic entities are:

- A *work*, which is some regulation, e.g. the Coin Act 2002.
- An *expression*, which is a specific version of a regulation. This could be the original version (e.g. the Coin Act 2002 as it was originally published) or some later version (e.g. the Coin Act 2002 as modified on November 22\textsuperscript{nd}, 2006). Each work has at least one expression; many works will have only one. An expression is said to *realise* a work.
- A *manifestation*, which is an expression in a specific formatting and layout, e.g. the PDF created by SDU of the Coin Act 2002 as modified on March 1\textsuperscript{st}, 2007. A manifestation is said to *embody* an expression.
- An *item*, which is a specific instance of a manifestation, i.e. the copy of the aforementioned PDF file that resides in the documents folder of my computer. An item is said to exemplify a manifestation.

Legislation refers to a work. When applying the legislation, all references should be seen as references to the current version of that work (or, when applying the legislation on something that happened in the past, all references should be seen as references to the version that applied at the time being considered). Even the references in amending laws refer to a work; the difference with a regular law is that it will only be applied once. However, at the time the amending law is made, it is unknown to what expression it will be applied, and the textual references are made at the level of a work. Case law, on the other hand, usually refers to a specific version of a work (i.e. the expression that was relevant for the case). It may even happen that they refer to page numbers, in which case they refer to a manifestation. Strictly speaking, doctrine also refers to a specific version, though it may be valid through a specific range of versions, and could be seen as referring to a specific subset of expressions (whereas a work encompasses all expressions). When a new version of a law appears, doctrine may also be upgraded, leading to a new version of the doctrine to go with the new version of the law.
Many existing applications and formats for legal content have an option to mark references inside the text; often, these take the form of HTML hyperlinks or something similar. These are very generic, and do not give us much insight into the structure of the references.

Also, in existing systems, this can lead to problems when the hyperlink leads to a specific item (instead of an expression). This is often the case when a collection comes from one publisher, who has added hyperlinks to other files inside the collection. Such a collection will not connect to items outside the collection and material from other publishers. In this case, it may be useful to detect the references anew to create a complete network that is no longer confined to the collection (see Winkels et al., 2005).

Like the headings that indicate the structures, references follow fixed format, obviously facilitated by the fact that the documents being referred to all have a similar structure. These structures enable the automated detection of references. After a reference has been detected, it should be resolved, meaning establishing the identity of the work or expression that is being referred to.

### 3.1 References to Documents

Laws and treaties are often referred to by their short title, such as the *Customs Act*\(^{46}\) or *Treaty establishing the European Economic Community*\(^{47}\). If no short title exists, the guidelines prescribe a format including the date, subject and publication number, such as:

*the Act of November 4\(^{th}\), 1950 for the establishment of additional regulations regarding military pensions that have been given out during hostile occupation, as well as modification of several laws, which provide rules regarding military personnel (Stb. 1950, K 479)*\(^{48}\)

Often, this format is somewhat modified, leaving out the subject:

*the Act of November 4\(^{th}\), 1950 (Stb. 1950, K 479)*

For treaties, the guidelines prescribe a similar format, including date, place, subject and publication number\(^{49}\).

*the on June 20\(^{th}\), 1956, in New York negotiated Convention on the Recovery Abroad of Maintenance (Trb. 1957, 121)*

Finally, there is also a prescribed format for European directives:

---

\(^{46}\) *Douanewet*

\(^{47}\) *Verdrag tot oprichting van de Europese Gemeenschap*

\(^{48}\) *de Wet van 4 november 1950 tot nadere vaststelling van de regelingen op het gebeid van militaire pensioenen, welke gedurende de vijandelijke bezetting zijn uitgevaardigd, zomede nadere wijziging van verschillende wetten, welke regelen geven inzake militair personeel (Stb. 1950, K 479) (Verwijzing uit de Wet privatisering ABP, artikel 77)*

\(^{49}\) *Aanwijzing 88, eerste lid*

Een verdrag wordt aangehaald overeenkomstig het volgende voorbeeld:

*het op 20 juni 1956 te New York tot stand gekomen Verdrag inzake verhaal in het buitenland van uitkeringen tot onderhoud (Trb. 1957, 121).*

In addition to references by full title or short title, some more formats are used. A law can refer to itself using the anaphor this law. Sometimes, other anaphors like that law or the law mentioned before are also used. Furthermore, a law can define an abbreviation for a document (often the law or the treaty) which can then also be used as a reference. For example:

<table>
<thead>
<tr>
<th>Immovable Property Valuation Act, article 2, definition of the law</th>
</tr>
</thead>
<tbody>
<tr>
<td>In this law, it is understood by the law: the Immovable Property Valuation Act.</td>
</tr>
</tbody>
</table>

So, in this law, any reference to the law refers to the Immovable Property Valuation Act (which happens to be the law itself).

### 3.2 References to Parts of Documents

A part of a document, such as a chapter or article, is referred to by means of its category or index (as discussed in section 2.1 and 2.2), e.g. chapter II or article 24. The category article is sometimes abbreviated to art.: art. 24. Subparagraphs and list items do not include a category in the header, but they do use a category in a reference, e.g. item a. For numbered subparagraphs, it is common to use an ordinal instead of the actual index, e.g. first subparagraph instead of subparagraph 1.

In order to refer to a part of a specific document, the reference to the part is combined with the reference to the document: article 2 of the Mining Act. Likewise, a reference to a subpart can be combined with the reference to a containing part: article 2, first subparagraph (which again can be combined with the reference to the containing document).

For such a “layered reference”, there are three ways to arrange the layers:

- Zooming in: The reference starts at the top level and travels down to the lowest level: Mining act, article 2, first subparagraph.
- Zooming out: The reference starts at the lowest level and travels up to the top level: first subparagraph, article 2, Mining Act. Between two levels, the word of can appear, so first subparagraph of article 2 of the Mining Act or first subparagraph, article 2 of the Mining Act can also occur.
- Zooming in, then zooming out: The reference starts at some level (usually the article) then “zooms in” and at the end “zooms out” again: article 2, first subparagraph, of the Mining Act. The “zooming out” part usually consists of one step, sometimes two, but seldom more.

It is also possible for a reference to refer to more than one part of the law. This is most often done by providing two or more indices. The category can be singular or plural (i.e. both article and articles are used in a reference to multiple articles). Some examples: articles 12 and 13, article

---

50 Wet waardering onroerende zaken, artikel 2, aanhef en negende onderdeel
In deze wet wordt verstaan onder de wet: de Wet waardering onroerende zaken.

51 In fact, there are several different categories used to denote list items. The guidelines prescribe the use of onderdeel, but older laws also use onder, sub or letter.
Alternatively, a range of indices is specified by specifying the first and the last index of the range. They are either separated by means of a hyphen (though this is discouraged in the guidelines\textsuperscript{52}) or the words up to and including: article 12 - 15, items a - f, articles 12 up to and including 15.

As with references to a single part of a document, references to multiple parts can be combined with references to containing parts of documents: \textit{first and second subparagraph, article 2 of the Mining Act}. References to different levels can also be combined \textit{articles 7, second subparagraph and 8 up to and including 12}.

If the reference contains a lot of repetition, because the same numbered subparagraphs or items from several articles are referenced, then this can be shortened by \textit{each time}. For example, a reference to \textit{articles 8d, first subparagraph, 53d, first subparagraph and 133d, first subparagraph} can be written as \textit{articles 8d, 53d and 133d, each time the first subparagraph}.

It is also possible that a reference contains exceptions on a given range or super part: \textit{chapter 3, except article 17}.

Lists have an introduction and, optionally, a conclusion, which are referred to as \textit{introduction} and \textit{conclusion}. They are seldom referred to on their own; usually they are quoted together with some of the list items: \textit{article 12, introduction and items i and j}.

Finally, like full documents, parts of documents are sometimes referred to using anaphors like \textit{this article} or \textit{the previous article}.

In appendix B, it is shown how these references can be detected using patterns in GATE.

### 3.3 Resolving References

After we have found a reference, it should be resolved, meaning that the identity of the document being referred to is determined. This identity will usually take the form of a Universal Resource Identifier (URI). Such an identifier can be meaningful or opaque.

If the identifier is meaningful, then the identifier is somehow linked to its meaning. For example, within the NIR project, it is prescribed that the identifier for \textit{article 2} should be \textit{art2}, and that the complete identifier for \textit{article 50 of the Customs Act} should be \textit{#art2} appended to the URI for the Customs Act (see Spinosa, 2001). Using patterns, references and parts of references can be identified, so it is possible to recognise the text \textit{article 50, first subparagraph of the Customs Act} as a reference that refers to \textit{subparagraph 1 of article 50 of the Customs Act}. Using the NIR rules, we can use this information to create the correct URI.

In a system that uses identifiers that relate directly to the names or indexes of the elements of the document, it is easy to determine the right URI. However, there are some disadvantages to such a system. First of all, it is does occur that laws have two articles that have the same

\textsuperscript{52} \textbf{Aanwijzing 65}

Het einde van een periode of reeks wordt aangeduid met de uitdrukking "tot en met".
number, or two subparagraphs within an article to have the same number, etc. In such cases, a meaningful identifier is still possible, but will have to rely on more than just the names and indexes by including position: the first article 12 and the second article 12. A second problem occurs when a part gets renumbered or renamed. In such cases, the identifier should change too, which means that some metadata is needed to link the part to its previous incarnations.

So, many systems use meaningless identifiers, which have the obvious disadvantage that the identifier does not relate in some way to the name or index of the text. This means that no URI can be constructed; it must be retrieved from some list.

It is also possible that the identifier is partly meaningful, partly meaningless, in which case a part of the URI can be constructed, and the other part needs to be retrieved.

The method described above only works when a reference is complete; that is, it contains the identity of the document that is being referred to. Many references encountered in a law do not contain that information. In such cases, we need to determine the work being referred to before we can determine the correct URI.

In most cases, an incomplete reference to a document part refers to that part within the same document (or within the same part of a document). So:

- A reference to article 72 is a reference to article 72 of the law it is encountered in.
- A reference to subparagraph 2 is a reference to subparagraph 2 of the article of the law it is encountered in.

A common exception to this is found in sentences that describe changes in existing laws. Such sentences are often preceded by a scope declaration (see section 4.4.3). Such a scope declaration sets the location of any changes being made. For example:

<table>
<thead>
<tr>
<th>Customs Act Introduction Act, article XIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>To the Aviation Act, the following changes are made.</td>
</tr>
</tbody>
</table>

Any incomplete references following this scope declaration do not refer to the containing document but instead to the document being set in the scope declaration. So, if this scope declaration is being followed by a subparagraph that refers to article 37a, then this is a reference to article 37a of the Aviation Act. This does not apply to any reference contained in quoted text or text to be inserted, as references in those texts should be resolved as part of the document that they are quoted from or are to be inserted in. In order to correctly resolve these references, these scope definitions need to be known.

Another group of references that do not contain the identity of the document that is being referred to are the anaphors. There are a number of different groups of anaphors that may be encountered.

---

53 Such mistakes are seldom present in a new law, but are usually introduced when changes are made later on.
54 Invoeringswet Douanewet, artikel XIX
In de Luchtvaartwet worden de volgende wijzigingen aangebracht.
1. References to *this law, this article*, etc.: These references refer to the law (or the article, etc.) in which they are found, and are easily resolved if the structure and identity of the document are known.

2. References to *that law, that article*, etc.: These references refer to an earlier reference (most likely the previous). Providing that previous reference has been found, all the information needed to resolve such a reference is available.

3. Reference to *the previous article, the previous subparagraph*, etc.: References to a previous structure part are easily resolved if the structure and identity of the document are known.

4. Nested references, such as *the articles mentioned in the previous article* are a reference to a reference that can be found in a specific location. This requires that the references to that location (in the example: *the previous article*) is first resolved. Provided that the reference in that location has already been found, the complete reference can now be resolved.

### 3.4 Experiment

In order to test this approach, a parser for recognising references to laws has been constructed based on the patterns described above, excluding the patterns for exceptions and repetitions. These results have been published before in de Maat, Winkels and van Engers (2006). Similar systems have been built before for other jurisdictions. Bolioli, Dini, Mercatali, and Romano (2002) and Palmirani, Brighi and Massini (2003) have created reference detection systems for Italian law. The system built by Palmirani et al. was capable of (partially) detecting 93.6% of all references in a set of Italian IT laws. Martínez-González, de la Fuente and Vicente (2005) have created a system for Spanish legal texts, which had a recall of 54%.

The Dutch parser was applied to six randomly selected Dutch laws. These were selected to include one law from before 1900 and another law from between 1900 and 1950, since the language used in those older texts may differ from the language used in modern text.

The results are presented in table 4. The references have been split into two groups: Simple references, which are non-layered references to a single element, and complex references, which are references that are layered and/or refer to multiple elements. A complex reference was considered to be partly found if part of the text was recognised as a reference, but the complete reference was not recognised. The skipped column shows references to documents other than laws, which were not included in this experiment.

---

55 In addition to this formal test, the parser has also been informally tested on several other laws and decrees, in which it performed very well, with few errors.
Table 4: Test results for detecting references

<table>
<thead>
<tr>
<th>Reference</th>
<th>Simple Found</th>
<th>Simple Missed</th>
<th>Complex Found</th>
<th>Complex Partly Missed</th>
<th>Complex Skipped</th>
<th>Complex False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wet tarieven in burgelijke zaken</td>
<td>1843</td>
<td>24</td>
<td>0</td>
<td>27</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Natuurschoonwet</td>
<td>1928</td>
<td>40</td>
<td>0</td>
<td>46</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Wet aansprakelijkheid olietankers</td>
<td>1975</td>
<td>38</td>
<td>0</td>
<td>35</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Wet op de lijkbezorging</td>
<td>1991</td>
<td>69</td>
<td>0</td>
<td>47</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Wet gemeentelijke basisadministratie persoonsgegevens</td>
<td>1994</td>
<td>251</td>
<td>1</td>
<td>156</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Wet op het notarisambt</td>
<td>1999</td>
<td>118</td>
<td>1</td>
<td>127</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>540 (99%)</td>
<td>3 (1%)</td>
<td>438 (95%)</td>
<td>24 (5%)</td>
<td>0 (0%)</td>
<td>11 (2%)</td>
</tr>
</tbody>
</table>

The parser achieved good results, finding 99% of all simple references, and 95% of all complex references. The few misses were caused by missing labels, names or patterns from the parser. If such labels and patterns occur more often, they can be included as well\(^56\) (though there will always remain some patterns that are too rare to include).

False positives occur if one of the labels used for the categories is used in a different meaning. For example, a subparagraph is called *lid* in Dutch (which also means *member*). Hence, confusion can occur when the text uses the text *the first member*, it may be a reference to either the first subparagraph or the first member of e.g. a committee. Such a false positive can sometimes be detected during the resolving of the reference, if the perceived target does not exist.

An adapted version of the parser has been used by Opsomer et al. (2009), who expanded the parser for Flemish legal texts. It was used on 1600 text fragments, ranging in size from very small (part of an article) to large (an entire chapter). Each fragment was part of Flemish, Belgian or European environmental and energy legislation, all written in Dutch. After the parser had recognised a reference, it was resolved using a module that searches a legislation database (EMIS Navigator, a database with environmental legislation applicable in Flanders) for the text being referenced. This procedure was also used to identify false positives: if a (perceived) reference referred to a non-existing text, it was deemed to be false. Opsomer et al. estimate that the parser found 85% of all references, with a precision of 95%.

### 3.5 Generalisation to Other Sources

The method described above works well for finding references in law texts to other laws, treaties and European directives.

Other (national) regulations follow the same writing style as laws, and are subject to the same guidelines. As such, the same patterns used to find references in laws will be effective in finding references in regulations. However, laws refer only to other laws, treaties and European legislation, whereas lower level legislation will also refer to other lower level legislation. For those references, the patterns will have to be expanded. Just as for the

\(^{56}\) The missing names have been included in any case; the parser used a list based on current laws, and some of the references found referred to retracted laws.
documents already described above, the Guidelines for Legal Drafting prescribe a format for references to lower level legislation, which can be used a basis for expansion of the patterns.

In addition to lower level legislation, commentaries and case law form an important source of legal information as well. These documents do not have to abide by the official guidelines, and hence are less uniform. Van Opijnen (2010) found that when referring to secondary EU legislation, Dutch judgments employed a great variety of formats (he gives examples of ten different formats used). Still, he found that a number of patterns would suffice to detect the references in all these formats. In general, the writing style used for these references remains close to the style used in legislation, and as such, the patterns used to find references in legislation will likely form a good starting point for finding references in jurisprudence and commentaries. Some expansions will be needed:

1. Patterns for references to new categories: Regulations do not refer to case law and commentaries, but case law and commentaries do contain such references. Patterns will be needed to cover these additional documents.
2. Unofficial names: Commentaries and jurisprudence will often refer to documents using an unofficial name. Just as official names, such names are difficult to detect using patterns, and should be collected in a list, to be used in the same way as the list of official names.
3. Variant patterns: As Van Opijnen found, commentaries and jurisprudence are somewhat less strict, and apply more different formats; hence, more patterns are likely necessary in addition to those already used for laws.

Next to these difficulties in detecting references in the text of commentaries and case law, there is also an additional issue with regards to the resolving of those references.

As mentioned above, regulations refer to a work. As such, we only need to know the identity of the document being referred to in order to resolve the reference. Commentaries and jurisprudence refer to a specific expression of a work\(^57\). Ideally, to resolve the reference, we need to know the identity of the document but also the version of the document. Commentaries will often refer to the version of a regulation as it was valid when the commentary was published, or, alternatively, they refer to some future version of the regulation. Case law usually refers to that version of the law that was applicable to the facts being judged. In both cases, the text will contain some clues (dates) relating to the relevant version\(^58\). The question is whether it is possible to automatically extract these dates from the documents.

3.6 Conclusion

References in Dutch laws are very well structured, and can be easily detected using patterns. This is confirmed by a test on six very diverse Dutch laws, which showed an accuracy of 97% and barely any false positives. A similar test on a more diverse Flemish corpus, which included

\(^{57}\) Though it is quite possible that a commentary is applicable to a certain range of expressions, or even to the entire work.

\(^{58}\) In addition to the date of the version, the date of publication of the referring document may also be relevant. The future version of a law of January 1st, 2012 that is being referred to in a commentary may very well be a different version than the actual version of that law on January 1st, 2012.
documents from different jurisdictions, gave an accuracy of 85%, with 95% precision. Furthermore, such parsing gives us most of the information needed to resolve the references, though for a complete approach it is also necessary to scan the law for definitions defining abbreviations of document names and for scope definitions.

The study of the structure of references and the different forms that references can take has led to the identification of some less-common structures. These structures have been included in the European XML standard for marking up legal sources, which includes mark-up for exception constructions and each-time constructions. Such support was not present in the predecessors of CEN/MetaLex, meaning that these references could not properly be annotated.

As the results mentioned show, there are a few troubles with the recognition of references. The variety of indexes that are used do increase the number of rules needed to parse everything, but it does significantly increase the difficult of the task. But when it comes to resolving references, those references that do not explicitly refer to their target(s) by means of an index pose something of a problem.

The first type of such references is the range. When a range is referenced, we can identify the start and the end of the range, but this does not give us full knowledge of what parts of the document are referenced. For example, a reference to articles 12 to 15 could be, depending on the structure of the target document:

- a reference to articles 12, 13, 14 and 15;
- a reference to articles 12, 13, 13a, 14 and 15;
- a reference to articles 12, 13 and 15.

This set of targets may differ from one version of a law to another version of the law. This means that for each version, we need to calculate the exact set of items referred, if that information is required for our purpose.

A related question that arises is: Is this effect intended by the legislator? When he refers to articles 12 to 15, does he intend for a future article 13a to be included in that reference? One could argue that this is always an issue when a law is changed, as any part of the law is a set similar to a range. Chapter 2 is a range including several articles, and when a new article is inserted into Chapter 2, this means that any references to Chapter 2 now also include this new article. Still, named sections like Chapter 2 have a clear theme, and it seems less likely that a new article inserted into a chapter should not be included in a (earlier) reference to that chapter.

Another group of references that do not include a specific index are those that refer to the previous article or subparagraph, etc. Like ranges, these require a derivation to determine the target. Also like ranges, modifications may unintentionally change the target of the reference. For example, article 61 of the Military Penal Code refers to the previous article, meaning article 60. In 2000, a new article 60a was inserted between articles 60 and 61, without modifying article 6159. As a result, the (literal) target of the reference in article 61 has changed to article 60a. In both situations (ranges and the use of previous), it may be that the resulting situation is

---

in fact desired and correct, but it seems that using explicit references (i.e. those that contain the index of each target) is less prone to produce errors.

A possible next step for the detection of references is to detect the type of reference as well. As mentioned in the introduction, references may serve various goals, such as referencing a target in order some definition or to modify it. Adding such a type to each reference will make it more valuable for navigating these links between documents.

Next to improving access to legislation, an application that can automatically detect references in legal sources opens up the possibility to perform network analysis of the corpus of sources of law. Such research has already been performed on the French legal codes by Mazzega, Bourcier and Boulet (2009), on court decisions of the United States Supreme Court, by Smith (2005) and Bommarito, Katz and Zelner (2009), on the Estonian Law of Obligations by Liiv, Vedeshin, and Täks (2007) and Dutch Supreme Court cases by Winkels and de Ruyter (2011). Such research can help identifying important sections of the law and key cases, as well as detect clusters of related sources. This, in turn, may lead to improved search techniques for legal sources and new ways to measure complexity of laws (see Bourcier and Mazzega, 2007).
4 Classification of Sentences

Most laws are too large to model in a single step. Therefore, we will first model parts of the law, and then integrate those parts in order to come to a complete model. As a unit for those partial models we chose the sentence, as this is the basic building block for the text.

In order to choose the right approach for modelling the sentence, we first want to know its general meaning: is it a normative sentence, or a sentence that changes an existing law? Depending on this general meaning, it is likely that a different sort of model should be created.

![Figure 6: Classifying sentences step](image)

Figure 6 shows the position of this step in the entire process. The input for this step is a document in which the structure has been marked up, so that we can target the individual sentences. After we have classified the sentences in this step, we will continue to create the models of the individual sentences. As we take this step before analysing the sentences in detail, we wish to make the classification using a shallow approach, looking at the surface structure of the sentence only, without doing an elaborate analysis. We have tried two different approaches: a knowledge engineering approach using patterns, and a machine learning approach using bags of words (i.e. all the individual words in a sentence). Deschamps (2011) found that laws in the Dutch language lack uniformity, which makes classification more difficult. Still, our research shows that classification is still feasible (see section 4.5).

However, before we classify the sentences, we need to know what kinds of sentences exist. Legal theory often discusses the kind of rules or norms that may occur in a law, and such classifications have been used for formal models as well (see for example Sartor, 2006). However, these classifications seldom address the actual text. The official guidelines for legislative drafting and legislative drafting literature describe some sentence types (which we will encounter later in this chapter), but do not provide an overall categorisation.

Still, there are two categorisations we wish to present here. First of all, our own broad categorisation (published before in de Maat and Winkels, 2007 and de Maat, Winkels and van...
Engers, 2009), and two more detailed categorisations from Tiscornia and Turchi (1997) and Atienza and Manero (1998).

In our own vision, the law consists of a set of core rules and several layers of supporting rules. The goal of regulations is (or perhaps: should be) to set rules for the people living in a country and the organisations established in that country. The regulations tell them what they can do and what they cannot do, and what their rights and obligations are. So, we could expect a regulation to mainly consist of statements like Everybody has the right to freedom of speech and If you take care of a child less than eighteen years of age, you have a right to child subsidy. Such statements do indeed appear in the law, for example:

**General Child Benefit Law, article 7, sub 1**
Conform the stipulations of this law, the insured has a right to child benefit for an own child, a stepchild and a foster child which:
- a. is younger than 16 years of age and belongs to his household; or
- b. is younger than 18 years of age and is maintained by him for a significant amount.

However, in addition to such core rules, there are additional rules that help support these core rules. By merely specifying that people have a right to child benefit, these benefits are not automatically distributed. A system needs to be set up for this. This leads to two layers of additional procedural overhead. A first layer, directed at the behaviour of citizens (or their organisations), tells citizens what procedure they have to follow to achieve certain goals. An example of this is:

**General Child Benefit Act, article 14, sub 2**
A request is made by means of an application form, which is provided by the Social Insurance Bank.

The second layer is aimed at civil servants, and deals with their side of the procedures, for example:

**General Child Benefit Act, article 17d, sub 3**
The Attorney General will inform the Social Insurance Bank of any circumstances as meant under sub 1 or 2.

In addition to such norms, there are other types of sentences as well. Hart (1961) distinguishes two types of rules in a law: primary and secondary rules. The primary rules are the rules that

---

60 Or province, municipality, etc., depending on the jurisdiction of that regulation.
61 Algemene kinderbijslagwet, artikel 7, eerste lid
De verzekerde heeft overeenkomstig de bepalingen van deze wet recht op kinderbijslag voor een eigen kind, een aangehoud kind en een pleegkind dat
- a. jonger is dan 16 jaar en tot zijn huishouden behoort, of
- b. jonger is dan 18 jaar en door hem in belangrijke mate wordt onderhouden.

62 Algemene kinderbijslagwet, artikel 14, tweede lid
Een aanvraag wordt ingediend door middel van een door de Sociale verzekeringsbank beschikbaar gesteld aanvraagformulier.

63 Algemene kinderbijslagwet, artikel 17d, derde lid (vervallen)
Het openbaar ministerie doet van een omstandigheid als bedoeld in het eerste en het tweede lid mededeling aan de Sociale Verzekeringsbank.
refer to human behaviour. Secondary rules are actually rules about primary rules, and form a meta-level. Three types of secondary rules are given by Hart: rules of recognition, rules of change and rules of adjudication. Rules of recognition determine which rules are ‘official’, rules of change allow for the changing of rules and rules of adjudication empower individuals to judge whether a rule has been broken.

The norms given above are all sentences that represent primary rules, but a law text also contains sentences that represent such secondary rules. For example, this is a sentence containing a rule of change, a sentence that allows others to set new rules:

**General Child Benefit Act, article 24b**

By Ministerial Decree additional rules can be set regarding the articles 24, sub 1, 2, 3, 4, 5 and 6, and 24a.

A law may also contain sentences that describe actual changes to other legislation. These do not correspond to Hart’s rules.

So, now we have a four-layered model of the kind of rules we can encounter in the law, consisting of the core rules and three types of supporting rules.

In addition to these rules, laws contain definitions, sentences that define concepts used elsewhere in the law. These definitions support the rules, both the core rules and the procedures. Together with rules, the definitions make up the body of the law. Add to that the introduction, conclusion and appendices, and we come to a more complete model of a legislative text:

Expert systems that deal with the law are often limited to one or two levels of this model. For example, a system that advises people whether or not they have a right to child benefit will only address the core rules. A somewhat more elaborate system, which also advise them on

---

64 Algemene kinderbijslagwet, artikel 24b
Bij ministeriële regeling kunnen nadere regels worden gesteld met betrekking tot de artikelen 24, eerste, tweede, derde, vierde, vijfde en zesde lid, en 24a.
what they need to do to obtain them will contain rules from the second layer as well, but is unlikely to include rules from the other layers. On the other hand, a workflow system for the child benefit procedure would include rules from both the second and third layer, and a system that keeps track of the history of a law text would track the rules from the fourth layer. The contents of the introduction and the conclusion are seldom relevant for expert systems, so we not consider those further in this chapter. The contents of the appendices are often relevant, but their contents can vary wildly between laws, so they do not share many common patterns, and we will disregard those as well.

Though the model sketched above does line up neatly with the different expert systems that exist, it does not give us much insight in the types of sentences that we can encounter in a law. A more detailed view is given by Tiscornia and Turchi (1997), who have made a categorisation to use for their *Lexsearch* project. They present two models. The first one is based on the position of the components, and is shown in table 5.

<table>
<thead>
<tr>
<th>Identifying Elements</th>
<th>Introductory Part</th>
<th>Main Part</th>
<th>Final Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of document</td>
<td>Preamble</td>
<td>Definitions</td>
<td>Organising financial coordination</td>
</tr>
<tr>
<td>Date and number of document</td>
<td>General</td>
<td>Attributing</td>
<td>Transitional, temporal and/or territorial force</td>
</tr>
<tr>
<td>Title</td>
<td>Citations</td>
<td>competence</td>
<td>Date and time of promulgation</td>
</tr>
<tr>
<td></td>
<td>Formula of promulgation</td>
<td>Constitutive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scope</td>
<td>Interpretative</td>
<td></td>
</tr>
<tr>
<td></td>
<td>General Principles</td>
<td>Instituting</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Procedural</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sanctioning</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Derogation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extensions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Abrogation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Substitution</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prorogation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Suspension</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Tiscornia and Turchi's model of provisions, based on their position

This model is similar to that of de Maat and Winkels above, in that it divides the law into an introduction, body and conclusion. This model disregards the appendices, and separates the identifying elements from the introduction. Also, it is more detailed with regard to the contents of these parts.

The second model re-arranges the provision types according to their function, called “basic components”. It is shown in table 6.

<table>
<thead>
<tr>
<th>Identifying Elements</th>
<th>Fixed Parts</th>
<th>Provisions</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of document</td>
<td>Formula of promulgation</td>
<td>Definitions</td>
<td>Amendments to the text:</td>
</tr>
<tr>
<td>Date and number of document</td>
<td>Date and place of promulgation</td>
<td>Sanctions</td>
<td>Substitutions, Abrogations and Additions</td>
</tr>
<tr>
<td>Title</td>
<td></td>
<td>Prescriptions</td>
<td>Temporal amendments:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Prorogations and Suspensions.</td>
</tr>
</tbody>
</table>

Table 6: Tiscornia and Turchi's model of provisions, based on their function
Tiscornia and Turchi consider several groups of provisions. The first group is that of the norms that concern the text under examination, which includes fixed parts, scope, general principles and financial provisions.

The second group are the norms that describe the relation to the legislative system: links, coordinating norms and transitional norms.

The third group is formed by the actual norms: statements that prescribe some behaviour (prescriptions) and the statements that describe the sanctions that are imposed when the prescriptions are not complied with.

The fourth group is formed by the constitutive norms, which Tiscornia and Turchi describe as a non-uniform category, which contains:

- procedural norms;
- classificatory norms;
- constitutive norms (norms that create a body or an office that did not exist before);
- norms attributing competence;
- definitions;
- (true) constitutive norms: norms that create a legal effect.

The classification made by Atienza and Manero (1998) features more sub-classifications. In total, they distinguish nineteen different kinds of legal sentences, divided in several groups, as depicted in figure 9.

The first distinction they make is between legal and meta-legal sentences. Legal sentences are those sentences that belong to some legal system, whereas meta-legal sentences are those sentences that are about such a legal system. These meta-legal sentences correspond to Hart’s rule of recognition.

The legal sentences are then divided between sentences of a practical nature and of a non-practical nature. With practical nature, Atienza and Manero mean that the sentences have the function of guiding or evaluating behaviour. The group of sentences of a non-practical nature consists of definitions, which do not guide behaviour, but that instead identify the meaning of other sentences.

The sentences of a practical nature are divided in normative and evaluative sentences. Evaluative sentences are sentences that evaluate behaviour, and thus provide a motivation for the normative rules that guide behaviour.

The normative sentences fall apart in actual norms and sentences that express the use of powers that have been conferred by norms, such as the enacting or repealing of a law. The sentences in this second group are called normative acts.

Within the norms, there is a distinction between regulative and constitutive norms (or deontic and non-deontic norms). Regulative norms are the norms that actually guide behaviour, whereas constitutive norms state how institutional results and normative changes are brought about. The regulative norms are further divided based on three criteria:
Figure 9: Atienza and Manero's sentence classification
1. whether the conditions for their application are indicated in an open or a closed form
   (principles versus rules);
2. whether the norm prescribes an action to be taken or an end state that has to be
   reached (strict principles and action rules vs. policies and end rules);
3. whether the norm indicates obligatory behaviour or facultative behaviour.

Two different groups of constitutive norms are recognised. Power-conferring rules are rules that
stipulate what one must do to produce an institutional result or normative change. They can
make the exercise of those powers obligatory or facultative. Furthermore, Atienza and Manero
also distinguish between whether or not executing the action is optional or non-optional.

Purely constitutive rules are rules that stipulate that if a certain state of affairs comes to pass,
some institutional result or normative change is produced.

The meta-legal sentences are further divided into three groups. First, there is the distinction
between sentences of a practical nature and of a non-practical nature, as it has also been
applied to legal sentences. The sentences of a practical nature are then divided into mandatory
rules and criteria for evaluation.

The different models suggest that the categories of provisions can be grouped in different
ways. This is most clearly illustrated by Tiscornia and Turchi, who group the same provision
categories in two different ways. Atienza and Manero consider the motivations for the rules
(evaluative sentences) to be closer to the normative sentences than definitions, grouping the
first to as sentences of the practical kind, while labelling definitions as sentences of the non-
practical kind. In the other models, the definitions are considered to be closer to the
normative sentences, as those two together form the actual rules.

The models also suggest that there are many ways to divide the normative sentences. Atienzo
and Manero divide the deontic norms along three axis: openness of the condition, describing
an action or an end result, mandatory or permissive. De Maat and Winkels distinguish norms
by the “distance” to the citizen. Tiscornia and Turchi divide deontic norms in prescriptions
and sanctions.

Constitutive norms are divided into purely constitutive norms and power-conferring rules by
Atienzo and Manero. Tiscornia and Turchi distinguish six different types: procedural norms,
classificatory norms, institutive norms, power-conferring norms, definitions and purely
constitutive norms. As mentioned above, Atienzo and Manero do not consider definitions as
norms (nor do de Maat and Winkels). The other subdivisions are not named by Atienzo and
Manero, and likely fall under their category of purely constitutive norms.

Tiscornia and Turchi also label several amending sentences (which they group together as
“links”), which are grouped by Atienzo and Manero under normative acts and by De Maat and
Winkels as rule management.

---

65 Even though it may be obligatory to obtain a certain result, performing the action to obtain that result may be
optional, if there are other actions available to achieve that same result.
Our focus is on the classification of sentences that appear in the body of Dutch laws. This means that we can ignore categories that belong to the introduction or conclusion, such as the fixed parts. We also need not look at Atienza and Manero’s meta-legal sentences or evaluative legal sentences, as these do not seem to appear in the body of Dutch laws. This leaves us with three broad categories:

1. Normative sentences, which set the rules for people and organisations that live in a country, and which correspond to Hart’s primary rules and Atienza and Manero’s express norms.
2. Definitions, which clarify the terms used by normative sentences and correspond to Atienza and Manero’s sentences of non-practical kind.
3. Lifecycle and maintenance sentences, which are rules about the law, and which enact, modify and repeal existing regulations, corresponding to Atienza’s and Manero’s purely normative acts.

In the following sections, the different sentences types found for each of these categories are discussed. These have been published before in de Maat and Winkels (2008, 2010). Following the classification, two experiments are presented that deal with the automatic classification of sentences in these categories.

4.1 Norms

The actual content of (original) legal sources is formed by norms. Normative sentences may confer rights and permissions or impose duties and obligations. Procedural rules are expressed as norms as well (usually, each step of a procedure is formulated as an obligation).

Research has been performed on how to distinguish these different types of norms (Franssen, 2007). During this research, it became clear that it was difficult to separate rights and permissions, and that it was likewise difficult to separate duties and obligations. Because of this, the norms have been grouped together in two large groups (obligations and rights) rather than separated further, as this will not benefit automated processing of the law.

4.1.1 Obligations

Obligations are the sentences that express a situation that must (or must not) occur, such as:

<table>
<thead>
<tr>
<th>Working Conditions Act, article 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>In his actions on the workplace, the employee is obliged to take care of his own safety and health and that of others, to his ability, and in accordance with his education and the instructions given by the employer.</td>
</tr>
</tbody>
</table>

Though words like *must* and *is obliged* are used in Dutch law, the guidelines recommend not to use these. Instead, expressions are used that describe a desired situation as if it is a fact. An example of such norm is:

66 This research was supervised by the author of this thesis.
67 Arbeidsomstandighedenwet, artikel 11
De werknemer is verplicht om in zijn doen en laten op de arbeidsplaats, overeenkomstig zijn opleiding en de door de werkgever gegeven instructies, naar vermogen zorg te dragen voor zijn eigen veiligheid en gezondheid en die van de andere betrokkene personen.
Funeral Act, article 46, sub 1
No bodies are buried on a closed cemetery.

The individual steps of a procedure are also expressed in a similar way, for example:

Voting Act, article J 25, sub 1
The voter hands the polling card to the chairman of the polling station.

Procedural norms followed the same sentence formats as regular norms, but have a slightly different meaning, as they are part of a larger procedure. For example, the different steps of such a procedure have to be applied in order. This is often not made explicit in the law, and needs to be gathered from the context.

4.1.2 Rights
Rights are sentences that describe a situation that may occur, or that confer a specific right to someone:

Passport Act, article 9
Within the limits as determined in this law, every Dutchman has a right to a national passport, valid for five years and for all countries.

4.1.3 Application Provisions
Application provisions specify situations in which other legislation (usually an article or subsection of an article) does apply. In this way, the application domain of a norm can be extended or restricted (effectively creating an exception to a rule). Eijlander and Voermans (1999) present application provisions as a way to prevent repetitions in the law.

The official guidelines mention three different phrases that can be used:

1. The word applies is used if the provision that is being referred to can be applied literally.
2. The phrase applies correspondingly is used if the provision that is being referred to cannot be applied literally, but the meaning is still clear.
3. The phrase applies, with the understanding that … is used if the provision that is being referred to cannot be applied literally, and some modifications have to be made for this application.

---

68 Wet op de lijkbezorging, artikel 46, eerste lid
Op een gesloten begraafplaats worden geen lijken begraven.

69 Kieswet, artikel J 25, eerste lid
De kiezer overhandigt aan de voorzitter van het stembureau de oproepingskaart.

70 Paspoortwet, artikel 9
Iedere Nederlander heeft binnen de grenzen bij deze wet bepaald, recht op een nationaal paspoort, geldig voor vijf jaren en voor alle landen.

71 Aanwijzing 83
1. De uitdrukking "is van toepassing" wordt gebruikt, indien de bepaling waarnaar wordt verwezen, letterlijk kan worden toegepast.
2. De uitdrukking "is van overeenkomstige toepassing" wordt gebruikt, indien de bepaling waarnaar wordt verwezen, niet geheel letterlijk kan worden toegepast, maar misverstand over de toe te passen tekst uitgesloten is.
3. De uitdrukking "is van toepassing, met dien verstande dat ……." wordt gebruikt, indien de bepaling waarnaar wordt verwezen, gedeeltelijk of met wijziging van bepaalde onderdelen moet worden toegepast.
An example of such an application provision is:

**Constitution, article 7, sub 4**
The previous members do not apply to making commercial advertisements.

In addition to these sentences which indicate that some other legislation does apply, there are also statements that indicate that some other legislation do not apply, using the phrase *does not apply.* Often, an application provision that states that another piece of legislation does apply seems to be included to take away any doubts as to whether it ought to apply or not.

### 4.1.4 Penalisations

The violation of some norms will carry punishment in the form of a fine or imprisonment. If this is the case, the law will specify the penalties. In Dutch law, this is usually done through the phrase *is punished with/by.* In the Penal Code, the behaviour that is punished is usually specified in the same sentence:

**Penal Code, article 365**
The civil servant that, by abuse of power, forces someone to do something, not to do something or to allow something, is punished by imprisonment for at most two years or a monetary fine of the fourth category.

In most other laws, the behaviour is specified in some other article that is being referred to:

**Mining Act, article 133, sub 1**
Breaking article 43, sub 2, is punished with a monetary fine of the second category.

The Penal Code is divided into separate sections for crimes and misdemeanours. Other laws will explicitly specify whether the punishable fact is a crime or a misdemeanour by adding a sentence such as:

**Mining Act, article 133, sub 2**
The fact marked as punishable by this article is a misdemeanour.

These sentences always follow this same structure.

### 4.1.5 Calculations

Calculations are sentences determining some value. Often, several of these sentences together form a procedure that should be followed in order to arrive at the correct result. There are many patterns that indicate calculation. For simple assignments, the verb *to amount* is commonly used:

---

72 *Grondwet, artikel 7, vierde lid*
De voorgaande leden zijn niet van toepassing op het maken van handelsreclame.

73 *Wetboek van strafrecht, artikel 365*
De ambtenaar die door misbruik van gezag iemand dwingt iets te doen, niet te doen of te dulden, wordt gestraft met gevangenisstraf van ten hoogste twee jaren of geldboete van de vierde categorie.

74 *Mijnbouwwet, artikel 133, eerste lid*
Overtreding van artikel 43, tweede lid, wordt gestraft met geldboete van de tweede categorie.

75 *Mijnbouwwet, artikel 133, tweede lid*
Het in dit artikel strafbaar gestelde feit is een overtreding.
The elderly discount amounts to € 739.

Other than that, almost any verb indicating a mathematical operation can occur, and indicates a calculation, such as:

The travel deduction as established based on the next subparagraphs is reduced by the compensation received for distances travelled with public transport.

4.1.6 Delegation

Delegations confer the power to create additional rules to some legal entity. Most often, this power is conferred onto a minister, for the creation of rules that do not require (immediate) involvement of the parliament.

The delegation can allow for the creation of rules:

By Order-in-council, rules may be set regarding the obligation of the owner of a water company to do research into the state of the water that is used by him to prepare tap water.

Alternatively, it can be an order to create rules to arrange for something, like:

By or based upon Order-in-council, specific rules are set regarding the voting in other ways than by ballots.

Sometimes, a delegation is followed by additional sentences that set additional restrictions or guidelines for the rules that may or must be created. For example, the previous sentence is followed by:

As much as possible, these rules are set conform the provisions of this law regarding the voting by ballots.

---

76 Wet inkomstenbelasting 2001, artikel 8.17, tweede lid
De ouderenkorting bedraagt € 739.

77 Wet inkomstenbelasting 2001, artikel 5.87, derde lid
De op basis van de volgende leden bepaalde reisaftrek wordt verminderd met de voor de per openbaar vervoer afgelegde reisafstand ontvangen reiskostenvergoedingen.

78 Waterleidingwet, artikel 4, derde lid
Bij algemene maatregel van bestuur kunnen regels worden vastgesteld, inhoudende de verplichting van de eigenaar van een waterleidingbedrijf onderzoekingen te verrichten met betrekking tot de hoedanigheid van het water dat door hem gebruikt wordt voor de bereiding van leidingwater.

79 Kieswet, Artikel J 34, eerste lid, eerste volzin
Bij of krachtens algemene maatregel van bestuur worden nadere regels gesteld betreffende het stemmen anders dan door middel van stembiljetten.

80 Kieswet, artikel J 34, eerste lid, tweede volzin
Deze regels worden zoveel mogelijk vastgesteld overeenkomstig de bepalingen van deze wet betreffende het stemmen door middel van stembiljetten.
4.1.7 Publication provision

A publication provision orders the publication of certain information. It usually accompanies a delegation, ordering the announcement of anything that has been decided based on the delegation.

Animal Feed General Act, article 1, sub 3
Our Minister announces the enactment of modification of a community measure in the State Gazette, insofar these have to be acted upon, mentioning the articles of this law impacted by the communal regulation.

4.2 Definitions

For each term used in a legal text, a clear meaning needs to be established. According to Eijlander and Voermans (2000), there are four techniques to define a term in legal texts:

a. No definition: the legislator does not define the term at all but uses the common meaning of the word;
b. Definition by context: a given term may be ambiguous in itself, but put into the context of given legislative text, its meaning is clear and precise;
c. Definition: a new meaning is given to a term;
d. Definition by reference: a term is defined by reference, that is, a reference is included to some other text where the term is defined.

This section deals with the definitions: sentences that explicitly define terms that occur in a legal source. Such sentences define terms using other terms, which in turn may be explicitly defined, not defined, defined by context or defined by reference.

4.2.1 Definitions

Definitions are used to describe the terms that occur in a legal source. A definition mentions both the term being defined as well as the actual definition. In many cases, such a definition is formed by a description of what is being defined:

General Administrative Law Act, article 1:4, sub 1
By administrative judge is understood: an impartial body that is appointed by law and charged with administrative judicial settlement.

Another option is a listing of possibilities:

---

81 Kaderwet diervoeder, artikel 1, derde lid
Onze Minister doet mededeling in de Staatscourant van de vaststelling of wijziging van een communautaire maatregel voorzover daaraan uitvoering moet worden gegeven, onder vermelding van de artikelen van deze wet waarop de communautaire maatregel betrekking heeft.

82 Algemene wet bestuursrecht, artikel 1:4, eerste lid
Onder administratieve rechter wordt verstaan: een onafhankelijk, bij de wet ingesteld orgaan dat met administratieve rechtspraak is belast.
Some other forms of definitions can also be identified, such as the abbreviation, which simply abbreviates a longer word or term:

**Tobacco Act, article 1, introduction and item k**
In this law and the stipulations based on it, it is understood by nicotine: nicotine alkaloids.

The most common abbreviation in Dutch law is probably the abbreviation of a specific minister to *Our Minister*:

**Tobacco Act, article 1, introduction and item b**
In this law and the stipulations based on it, it is understood by Our Minister: Our Minister of Health, Welfare and Sport.

Some definitions do not give a description themselves, but instead give a reference to a location where the description is provided (i.e. the term used for definition used is defined by reference). This can either be another source, or a location within the same source:

**Higher Education and Scientific Research Act, article 1.1, introduction and item f**
In this law, it is understood by institutions: an institution as meant in article 1.2.

Next to the term being defined and its definition, many definitions will also contain a scope declaration, which states for which documents the definition is valid. Most often, the scope is *this law or this law and the stipulations based on it*. However, more restricting scopes are also possible. For example, the previous example, which defines the meaning of *institutions* within the Higher Education and Scientific Research Act is later overruled for chapter 7 of that act:

**Higher Education and Scientific Research Act, article 7a.1, introduction and item a**
In this chapter, it is understood by institutions: an institution as meant in article 7a.2.

---

83 Telecommunicatiewet, artikel 1.1, aanhef en onderdeel i
In deze wet en de daarop berustende bepalingen wordt verstaan onder aanbieden van een elektronisch communicatienetwerk: het bouwen, exploiteren, beheren of beschikbaar stellen van een elektronisch communicatienetwerk.

84 Of course, any definition can be seen as a short-hand notation of a longer description.

85 Tabakswet, artikel 1, aanhef en onderdeel k
In deze wet en de daarop berustende bepalingen wordt verstaan onder nicotine: nicotinealkaloïden.

86 Tabakswet, artikel 1, aanhef en onderdeel b
In deze wet en de daarop berustende bepalingen wordt verstaan onder Onze Minister: Onze Minister van Volksgezondheid, Welzijn en Sport.

87 Wet op het hoger onderwijs en wetenschappelijk onderzoek, artikel 1.1, aanhef en onderdeel f
In deze wet wordt verstaan onder instelling: een instelling als bedoeld in artikel 1.2.

88 Wet op het hoger onderwijs en wetenschappelijk onderzoek, artikel 7a.1, aanhef en onderdeel a
In dit hoofdstuk wordt verstaan onder instelling: een instelling als bedoeld in artikel 7a.2.
4.2.2 Type Extensions

Type extensions are very similar to definitions. However, instead of completely defining a new term, they expand or limit an earlier definition. The most common use of type extensions is to expand a common sense definition. In these cases, the law source does not define the term, but instead uses the common meaning of the word, and expands upon that meaning. Sometimes, this is done through a “regular” definition, by allowing the term itself to appear in the actual definition:

**Animal Health and Welfare Act, article 1, definition of keeper**

In this law and the stipulations based on it, it is understood by keeper: owner, keeper or herder.

In this case, the second *keeper* does refer to the common sense definition of *keeper*, and not (recursively) to the term *keeper* as defined here. An alternate method for expanding an earlier definition is by using a type extension, which explicitly extends an earlier definition.

**Equal Treatment General Act, article 1, sub 2**

By direct distinction based on gender is also understood distinction based on pregnancy, childbirth and motherhood.

The term that is being expanded can be explicitly defined in an earlier definition, but it is also possible that this is not the case. In this case, the type extension extends the common sense definition of the term.

Next to extending a definition, it is also possible to restrict a definition:

**Automobile and Motorbike Tax Act 1992, article 4, sub 1**

In this law and the stipulations based on it, it is understood by motorbike a motorised vehicle on two wheels, as well as such a motorised vehicle which is attached to a sidecar. By motorbike is not understood a moped as meant in article 1, sub 1, item e, of the Road Traffic Act 1994.

4.3 Deeming Provisions

Related to the definitions are the deeming provisions, which introduce a legal fiction. The deeming provision declares one situation to be equal to another situation, in a certain context. If a situation is deemed equal to another situation, then any rules that apply to the latter also apply to the first. For example:

---

89 *Gezondheids- en welzijnswet voor dieren, artikel 1, definitie van houder*

In deze wet en de daarop berustende bepalingen wordt verstaan onder houder: eigenaar, houder of hoeder.

90 *Algemene wet gelijke behandeling, artikel 1, tweede lid*

Onder direct onderscheid op grond van geslacht wordt mede verstaan onderscheid op grond van zwangerschap, bevalling en moederschap.

91 *Wet op de belasting van personenauto's en motorrijwielen 1992, artikel 4, eerste lid*

In deze wet en in de daarop gebaseerde regelingen wordt verstaan onder motorrijwielen een motorrijtuig op twee wielen, alsmede een dergelijk motorrijtuig dat is verbonden met een zijspanwagen. Onder motorrijwielen wordt niet verstaan een bromfiets in de zin van artikel 1, eerste lid, onderdeel e, van de Wegenverkeerswet 1994.
A Dutchman who is employed by the State of the Netherlands is always deemed to live in the Netherlands if he is posted as a member of a diplomatic, permanent or consular representation of the Kingdom of the Netherlands in foreign countries.

The effect of this statement is that someone is considered to live in the Netherlands, even though he actually lives outside of the Netherlands.

4.4 Lifecycle and maintenance

4.4.1 Enactment Date

These are sentences that set the enactment date for (part of) a regulation, or that arrange for the enactment date to be set.

Each law includes a sentence that set its enactment date, or arranges for its enactment to be set. The most straightforward of these simply set a date for the enactment of the law:

**Income Tax Act 2001, article 2.2, sub 2, introduction and item a**
A Dutchman who is employed by the State of the Netherlands is always deemed to live in the Netherlands if he is posted as a member of a diplomatic, permanent or consular representation of the Kingdom of the Netherlands in foreign countries.

However, this is not very common. The guidelines suggest three common formats, the first of which is to defer the setting of the date to a Royal Decree:

**Fuel Taxes Environment Tariffs Act 1991, article IV, sub 1, first sentence**
This law is enacted starting on January 1st, 1991.

However, this is not very common. The guidelines suggest three common formats, the first of which is to defer the setting of the date to a Royal Decree:

**Exception Situations Coordinating Act, article 11**
This law is enacted on a date to be set by Royal Decree.

The others link the enactment date to the date of publication, either following it directly:

**Act of July 7th, 2010 (Stb. 2010/305), article 9**
This law is enacted starting on the day after the date of publication

Or with some delay:

---

92 *Wet inkomstenbelasting 2001, artikel 2.2, tweede lid, aanhef en onderdeel a*
Een Nederlander die in dienstbetrekking staat tot de Staat der Nederlanden, wordt steeds geacht in Nederland te wonen indien hij is uitgezonden als lid van een diplomatieke, permanente of consulaire vertegenwoordiging van het Koninkrijk der Nederlanden in het buitenland.

93 *Tarievenwet brandstofheffingen milieu 1991, artikel IV, eerste lid, eerste zin*
Deze wet treedt in werking met ingang van 1 januari 1991.

94 In such cases, the Royal Decree will contain a sentence of a similar format as described here to set the enactment date of the law. A Royal Decree which sets an enactment date for a law does not need to set an enactment date for itself; it is automatically enacted at its publication date.

95 *Coördinatiewet uitzonderingstoestanden, artikel 11*
Deze wet treedt in werking op een bij koninklijk besluit te bepalen tijdstip.

96 *Wet van 7 juli 2010 (Stb. 2010/305), artikel 9*
Deze wet treedt in werking met ingang van de dag na de datum van uitgifte van het Staatsblad waarin zij wordt geplaatst.
Wrongful Act Conflict of Laws Act, article 10
This law is enacted starting on the first day of the second calendar month after the date of publication in the State Gazette in which it is included.

More complicated situations exist in which the date differs for different parts of the law:

Notaries Act, article 134
This law is enacted on a date to be set by Royal Decree, which may differ for separate parts and articles.

Or:

Artificial Insemination Donor Data Act, article 14
This law is enacted on a date to be set by Royal Decree, which may differ for separate parts and articles, with the exception of article 3, sub 2, second sentence, and sub 3 up to and including 5, which provisions are enacted starting on the date of the first calendar month after two years after the date of publication in the State Gazette in which this law is included.

4.4.2 Short Title
If it is thought necessary, a law will also define a short title which can be used to refer to it.

Notaries Act, article 135
This act may be referred to as: Notaries act.

It is also possible that a law will modify the short title of another law. This is usually done to avoid confusion, when a new law has the same name as he predecessor.

Sometimes, the short title may be abbreviated, which is indicated by a provision like:

Income Tax Act 2001, article 11.4, sub 2
The short title may be abbreviated to: IT Act 2001.

4.4.3 Change Provisions
Change provisions are modifications in existing legislation. Most laws are amending laws, consisting mostly of such changes in other laws (instead of new rules). In these laws, change provisions make up the bulk of the text.

97 Wet conflictenrecht onrechtmatige daad, artikel 10
Deze wet treedt in werking met ingang van de eerste dag van de tweede kalendermaand na de datum van uitgifte van het Staatsblad waarin zij wordt geplaatst.

98 Wet op het notarisambt, artikel 134
Deze wet treedt in werking op een bij koninklijk besluit te bepalen tijdstip, dat voor de verschillende onderdelen en artikelen verschillend kan zijn.

99 Wet donorgegevens kunstmatige bevruchting, artikel 14
Deze wet treedt in werking op een bij koninklijk besluit te bepalen tijdstip dat voor de verschillende artikelen of onderdelen daarvan verschillend kan luiden met uitzondering van artikel 3, tweede lid, tweede volzin, en derde tot en met vijfde lid, welke bepalingen in werking treden met ingang van de eerste kalendermaand na verloop van twee jaren na de datum van uitgifte van het Staatsblad waarin deze wet wordt geplaatst.

100 Wet op het notarisambt, artikel 135
Deze wet wordt aangehaald als: Wet op het notarisambt.

101 Wet Inkomstenbelasting 2001, artikel 11.4, tweede lid
De citeertitel kan worden afgekort tot: Wet IB 2001.
There are four types of changes: insertion of new text, replacing of text, deletion of text and renumbering of sections.

An insertion adds new text to the document. The text describes this as appending text if the text is added to the end of a structure element; otherwise it is described as inserting text.

Act of June 6th, 2002 (Stb. 303), article I, sub IIa
To article 7.36, a new sentence is appended, to read as follows: Article 7.34, sub 5, applies correspondingly.

When replacing text, some text is removed and new text is added instead. This can be done at the level of a few words within a sentence:

Act of June 6th, 2002 (Stb. 303), article III, sub V
In article 7.12, sub 1, second sentence, «article 7.3b» is replaced by: article 7.3c.

Alternatively, if an entire sentence, section or article is replaced, the modifying provision will simply refer to that element and quote the new text:

Act of June 6th, 2002 (Stb. 303), article IV, sub B
Article 2.8 will read: …

The deletion of text, a repeal, can affect an entire law, an element of a law or only a few words.

Act of June 6th, 2002 (Stb. 303), article I, sub QQ
Article 17.2 is repealed.

The last change is the renumbering (or relettering) of structure elements. Because renumbering an element requires the modification of all text referring to that element, it is somewhat uncommon for articles to be renumbered. On the other hand, anything below the level of article (subsections and lists) is almost always renumbered to keep a continuous numbering.

Act of June 6th, 2002 (Stb. 303), article IIIc, sub C
The articles 17a.1 to 17a.25 are renumbered to the articles 17.20 to 17.54.

Often, renumbering is related to the insertion or deletion of text. In the modifying text, the operations are often combined in one sentence:

---

102 Wet van 6 juni 2002 (Stb. 303), artikel I, lid IIa
Aan artikel 7.36 wordt een volzin toegevoegd, luidende: Artikel 7.34, vijfde lid, is van overeenkomstige toepassing.

103 Wet van 6 juni 2002 (Stb. 303), artikel III, lid V
In artikel 7.12, eerste lid, tweede volzin, wordt «artikel 7.3b» vervangen door: artikel 7.3c.

104 Wet van 6 juni 2002 (Stb. 303), artikel IV, lid B
Artikel 2.8 komt te luiden: …

105 Wet van 6 juni 2002 (Stb. 303), artikel I, lid QQ
Artikel 17.2 vervalt.

106 Wet van 6 juni 2002 (Stb. 303), artikel IIIc, lid C
De artikelen 17a.1 tot en met 17a.25 worden vernummerd tot de artikelen 17.20 tot en met 17.54.
Renumbering sub 2 to 5 to sub 3 to 6, a new sub 2 is inserted, reading: …

Another renumbering operation is the adding of an index to a paragraph that did not have one before:

**Act of July 12th, 2009 (Stb. 2009/245), article I, sub Da, sub 1**

Reumbering is not always done explicitly; sometimes the header of a structure element is modified in a way that also affects the index, implicitly renumbering the element:

**Act of June 6th, 2002 (Stb. 303), article IIIa, sub A**

The heading of chapter 5a will read: Chapter 5. Accreditation in higher education.

None of the sentences above have a complete reference. They refer to articles, but not to articles in a specific law. As described in section 3.3, such an incomplete reference normally refers to another element of the same text, but this is usually not the case in amending laws. If many changes are made in the same text, then they are grouped and preceded by a sentence that sets the scope, such as:

**Act of June 6th, 2002 (Stb. 303), article I, introduction**

To the Higher Education and Academic Research Act, the following modifications are made:

### 4.5 Experiment: Pattern-based Approach

We built a classifier (in Java) that takes well-structured legal sources as input and tries to classify their sentences according to their type based on typical patterns associated with these types. For each sentence type, the classifier includes several patterns, as we have found that there were many different ways in which each type of sentence can be expressed. This variation in expressions has also been found by other researchers, such as Deschamps (2011).

Deschamps also discusses semasiological variation. This means that a single expression is sometimes used to for different types of sentences. She gives the example of an obligation, which turns out to be a permission, due to the existence of exceptions to that obligation. We do not take context in consideration when classifying the sentences, so a sentence is classified based on its own pattern and meaning. Thus, the sentence form Deschamps example would be classified as an obligation. Just as with the actual text, the meaning of the entire law only becomes clear when the different sentences are combined.

---

107 Aanpassingswet geregistreerd partnerschap, artikel 4, lid A 1
Onder vernummering van het tweede tot en met vijfde lid tot derde tot en met zesde lid wordt een nieuw tweede lid toegevoegd, luidende: …

108 Wet van 12 juni 2009 (Stb. 2009/245), artikel I, lid Da, lid 1
Voor de tekst wordt de aanduiding «1» geplaatst.

109 Wet van 6 juni 2002 (Stb. 303), artikel IIIa, lid A
Het opschrift van hoofdstuk 5a komt te luiden: Hoofdstuk 5. Accreditatie in het hoger onderwijs.

110 Wet van 6 juni 2002 (Stb. 303), artikel I, aanhef
In de Wet op het hoger onderwijs en wetenschappelijk onderzoek worden de volgende wijzigingen aangebracht:
The classifier assumes that the input is structured using MetaLex XML. In MetaLex, sentences and lists are marked, as well as the separate list items within each list. This enables the classifier to treat each sentence separately.

The classifier is a simple pattern matcher. We used 88 patterns from about twenty Dutch laws. Most patterns consist of only a verb phrase, like *mogen* (may) for a right/permission or *wordt aangehaald als* (is referred to as) for the defining of a short title. Sometimes, additional keywords have been added, as in *kan regels stellen* (may create rules). As additional examples, the patterns for obligations and sentences setting a short title are given in appendix C.

The patterns are stored in a format for the Java pattern matcher (*java.util.regex*). The patterns mentioned above become:

```java
\s+(mag|mogen)\b
\s+wordt\s+aangehaald\s+als(:)?\s+
\s+kan\s+regels\s+stellen\s+
```

In this format, \s+ denotes one or more whitespace characters, \b denotes a word boundary, and (:)? is an optional colon. The first pattern allows for either the singular or the plural form of the verb.

The classifier will attempt to match a sentence to each available pattern. If the sentence matches several patterns, the classifier will prefer the longest of the matches. (This does not happen often; however, some of the patterns overlap, such as *kan* for a right and *kan regels stellen* for a delegation.)

As mentioned in section 4.1.1, the official guidelines recommend that words like *must* are not used in obligations. Instead, they are formulated using the normative indicative (Šarčević, 1997), a description of the desired situation. For such sentences, no patterns could be identified. However, as this is the only category for which no patterns exist, the parser assumes that whenever a sentence does not contain any pattern, it is a sentence using the normative indicative (and therefore an obligation).

A different approach was tried to classify sentences with an embedded list, such as:
Two different approaches were made. The first approach (de Maat & Winkels, 2008) assumed that most lists, like the example above, included the pattern needed for classification in the introduction of the sentence.

In the second approach (de Maat & Winkels, 2009), the classifier searches for a pattern in the introduction first. If one is found, the entire list is classified according to that pattern. If no pattern is found, the individual list items are searched for patterns and classified as if they were separate sentences.

The classifier was tested on eighteen different Dutch regulations, four of which were completely new laws. The others were amending laws, mostly containing changes in existing laws. With the exception of a single Royal Decree, these were all bills, pending at parliament. The length of the laws varied from very short (three sentences) to quite long (166 sentences on 23 pages A4); most were quite recent (patterns in the past have been different).

In each document, all sentences belonging to the body of the text were parsed, including any sentences that were quoted from or to be inserted in other documents. To check whether clauses were classified correctly, all sentences and lists in all laws were also classified manually. This manual classification was performed by several persons, who classified the sentences based on the description given in de Maat and Winkels (2007).
4.5.1 Results

Table 7 shows the results for sentences (not lists) in the different sources.

<table>
<thead>
<tr>
<th>Source</th>
<th>Sentence Type</th>
<th>Total</th>
<th>Correct</th>
<th>% Correct</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Royal Decree Stb. 1945, F 214 (as modified per 01/01/2002)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bill 20 585 nr. 2</td>
<td>New</td>
<td>31</td>
<td>30</td>
<td>97%</td>
<td></td>
</tr>
<tr>
<td>Bill 22 139 nr. 2</td>
<td>New</td>
<td>22</td>
<td>20</td>
<td>91%</td>
<td></td>
</tr>
<tr>
<td>Bill 27 570 nr. 4</td>
<td>76%</td>
<td>21</td>
<td>16</td>
<td>Change</td>
<td></td>
</tr>
<tr>
<td>Bill 27 611 nr. 2</td>
<td>100%</td>
<td>11</td>
<td>11</td>
<td>Change</td>
<td></td>
</tr>
<tr>
<td>Bill 30 411 nr. 2</td>
<td>New</td>
<td>141</td>
<td>128</td>
<td>91%</td>
<td></td>
</tr>
<tr>
<td>Bill 30 435 nr. 2</td>
<td>Change</td>
<td>40</td>
<td>39</td>
<td>98%</td>
<td></td>
</tr>
<tr>
<td>Bill 30 583 nr. A</td>
<td>Change</td>
<td>27</td>
<td>27</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Bill 31 531 nr. 2</td>
<td>Change</td>
<td>3</td>
<td>3</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Bill 31 537 nr. 2</td>
<td>Change</td>
<td>29</td>
<td>29</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Bill 31 540 nr. 2</td>
<td>Change</td>
<td>7</td>
<td>7</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Bill 31 541 nr. 2</td>
<td>Change</td>
<td>8</td>
<td>8</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Bill 31 713 nr. 2</td>
<td>Change</td>
<td>7</td>
<td>6</td>
<td>86%</td>
<td></td>
</tr>
<tr>
<td>Bill 31 722 nr. 2</td>
<td>Change</td>
<td>31</td>
<td>22</td>
<td>71%</td>
<td></td>
</tr>
<tr>
<td>Bill 31 726 nr. 2</td>
<td>Change</td>
<td>78</td>
<td>67</td>
<td>86%</td>
<td></td>
</tr>
<tr>
<td>Bill 31 832 nr. 2</td>
<td>Change</td>
<td>7</td>
<td>7</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Bill 31 833 nr. 2</td>
<td>Change</td>
<td>4</td>
<td>4</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Bill 31 835 nr. 2</td>
<td>Change</td>
<td>99</td>
<td>90</td>
<td>91%</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>591</strong></td>
<td><strong>537</strong></td>
<td><strong>91%</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Sentence classification results per law

The first thing to notice is that the classifier performs well, classifying 90% of all sentences correctly. Out of the 591 sentences, 537 were classified correctly. Table 8 presents the results for the different types of sentences. The column *In corpus* shows the number of sentences present in the test set for each type, both as an absolute number and as a percentage. The column *Missed* shows how many of these sentences were not correctly identified. For example, the test set contained 35 application provisions, but one was incorrectly classified (meaning that 34 were correctly classified). The column *False* presents the amount of sentences that were incorrectly classified as a particular type, e.g. eight sentences were incorrectly classified as an application provision. Each false positive corresponds to a *Missed* somewhere else.

The biggest part of the sentences is formed by the norms. 44% of all sentences belong to one of the norm categories. The next biggest category consists of the change provisions, with 37%. Some sentences were a concatenation of two sentences. For example, one sentence contained two changes: renumbering and a repeal. These sentences are listed in table 8 as 'Mixed type'.

About half of the misses were caused by patterns that were unknown to the classifier; these sentences were incorrectly classified as the default (normative indicative), and sometimes as a norm of the type right/permission. Two notable patterns were missing: a renumbering pattern dealing with re-lettering rather than renumbering, and a new pattern for delegations.
Those misclassifications that were not caused by missing patterns were instead caused by patterns that were somehow too broad. For example, most false positives of the “repealed” type sentences were provisions concerning the repeal of fines instead of articles. This will require more sophisticated patterns or dedicated ‘anti-patterns’ (i.e. not applicable when it contains the word ‘fine’).

Both false penalisations were in fact a right; the pattern that triggered this classification was part of a qualification of a legal body that was given certain rights. Such a qualification is given in an auxiliary sentence. This means that classifier will find two (or even more) patterns: one in the auxiliary sentence, and one in the principal sentence. As it does not have the option to distinguish between the two, it will pick the longest match (which will not always be the correct one).

If the principal sentence does not contain any pattern (because it uses the normative indicative), the classifier will only find the pattern in the auxiliary sentence, and will automatically arrive at the wrong conclusion. This is the cause of almost all false rights and false application statements.

Table 9 shows the distribution of the patterns that were actually encountered in the test set. For each type, the number of patterns known is shown, as well as the number of patterns encountered. The results column shows for each encountered pattern how often it has been correctly applied, and how often it has been incorrectly applied, causing a false positive result.
<table>
<thead>
<tr>
<th>Type</th>
<th>Patterns Known</th>
<th>Patterns Used</th>
<th>Results per pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Correct</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Definition</td>
<td>14</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>Norm – Right/Permission</td>
<td>17</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Norm – Obligation/Duty(^{112})</td>
<td>15</td>
<td>8</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Delegation</td>
<td>7</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Publication Provision</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>Application Provision</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Enactment Date</td>
<td>1</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>Short Title</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Value Assignment</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Penalisation</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Change - Scope</td>
<td>2</td>
<td>2</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>Change - Insertion</td>
<td>4</td>
<td>4</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Change - Replacement</td>
<td>3</td>
<td>3</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Change - Repeal</td>
<td>2</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>Change - Renumbering</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>44</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>32</td>
</tr>
</tbody>
</table>

Table 9: Patterns used

\(^{112}\) Eight obligations that do not follow the “normative indicative” format were not classified using a pattern (but they were still correctly classified as an obligation). Hence, this table shows only 21 correctly applied patterns for obligations, even though 29 obligations were correctly classified according to table 8.
The numbers suggest that there are a couple of main patterns that account for a majority of the sentences identified. For example, 60 rights were correctly identified. Of those, 55 used the pattern *may* and four used the pattern *is qualified*. This corresponds to the result of Franssen (2007), who concluded that the majority of right could be identified with those two patterns. This distribution suggests that some of the other patterns may be superfluous.

Table 9 also shows that the most false positives are caused by a small set of patterns as well, with one pattern for rights, one for repeal and one pattern for application provisions being the biggest offenders. However, these patterns are also responsible for many of the correct classifications, so removing them will not improve the results. It may be possible to narrow them down instead, but this also carries the risk of reducing the number of correct results.

As said, the results discussed above only referred to the sentences encountered in the regulations. For lists, the performance is influenced by the manner in which they are handled. Table 10 shows the results for the two methods that were tried: classification based on the introduction, and classification based on the introduction followed by items if the introduction contained no pattern.

<table>
<thead>
<tr>
<th>Introduction only</th>
<th>Introduction, then Items</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
</tr>
<tr>
<td>Royal Decree Stb. 1945, F 214 (as modified per 01/01/2002)</td>
<td>4</td>
</tr>
<tr>
<td>Bill 20 585 nr. 2</td>
<td>4</td>
</tr>
<tr>
<td>Bill 22 139 nr. 2</td>
<td>2</td>
</tr>
<tr>
<td>Bill 27 611 nr. 2</td>
<td>1</td>
</tr>
<tr>
<td>Bill 30 411 nr. 2</td>
<td>25</td>
</tr>
<tr>
<td>Bill 30 435 nr. 2</td>
<td>4</td>
</tr>
<tr>
<td>Bill 31 537 nr. 2</td>
<td>2</td>
</tr>
<tr>
<td>Bill 31 713 nr. 2</td>
<td>2</td>
</tr>
<tr>
<td>Bill 31 722 nr. 2</td>
<td>6</td>
</tr>
<tr>
<td>Bill 31 726 nr. 2</td>
<td>2</td>
</tr>
<tr>
<td>Bill 31 832 nr. 2</td>
<td>3</td>
</tr>
<tr>
<td>Bill 31 835 nr. 2</td>
<td>7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>42</strong></td>
</tr>
</tbody>
</table>

Table 10: List classification results per law

Classification on introduction only gives an accuracy of 90%, which is close to the result achieved for sentences. Classification of introduction followed by items performs a lot worse, with an accuracy of only 81%. However, this disregards the lists that were partially correct, i.e. some of the items were classified correctly, while others were not. Using classification by introduction, followed by items, only 5% of the lists were fully classified incorrectly.

4.6 Experiment: Machine Learning Approach

As an alternative to the pattern-based approach for classification, we have attempted a machine learning approach, as literature on general text classification (such as Sebastiani, 2002)
suggests that machine learning is superior to knowledge based approached. The results presented in this section have been published before in de Maat, Krabben and Winkels (2010).

Machine learning techniques build a classifier by learning the characteristics of each category from a set of already classified examples. In general, these techniques are more flexible and less domain dependant. They also require less expert knowledge, as experts often find it easier to classify something than to come up with reasons for the classification. A disadvantage of machine learning is that it works like a black box; the classifier cannot provide reasons for the classifications it makes. Furthermore, machine learning requires a large dataset to be available, while a knowledge-based approach does not need any formal dataset.

In the legal domain, Francesconi and Passerini (2007) used ML to perform a classification task on provisions in Italian laws. They arrive at similar results as we did (93% accuracy). Gonçalves and Quaresma (2005) used machine learning for classifying documents from the Portuguese Supreme Court and Attorney General’s Office and also achieve accuracy rates above 90%. Opsomer et al. (2009), however, report results no higher than 65% for classifying Belgian environmental laws. These studies are difficult to compare to our own research, as the jurisdictions, fields of law and languages differ. Also, the grain size of the “documents” to be classified ranges from sentences (in our case) to entire chapters of laws (in Opsomer’s case). Gonçalves and Quaresma classify court decisions instead of laws.

For the classification of sentences, we use Support Vector Machines (Cortes & Vapnik, 1995). SVMs have been shown to perform at least as good and usually much better in text categorization than other popular algorithms such as decision trees or naive Bayes classifiers (see Yang & Liu, 1999 and Sebastiani, 2002). SVMs were also used by Francesconi and Passerini, Opsomer et al., and Gonçalves and Quaresma mentioned above.

We have done two sets of experiments, using the libsvm toolkit for WEKA (Hall et al., 2009), with linear kernel and default settings. The first experiments were aimed at finding the optimal settings for data representation and pre-processing. After those optimal settings had been selected, we evaluated whether a classifier trained using these settings could generalise to texts that were not included in the training set.

We will first discuss the data representation, followed by a description of the two experiments.

4.6.1 Data representation

Each sentence needs to be represented in a way that can be handled by the automated classifier. A common approach is to represent a document (in our case: a sentence) by the words it contains, disregarding the order in which they appear. This is called a bag of words model. The basic approach is to select all the words as they appear in the sentence, though some changes can be made to the selection process that may lead to better results.

A stop list contains those words that are deemed unlikely to be useful in the classification process. It contains words like the, from and him. When using the stop list, words appearing on the list are not included in the representations of the sentences.
**Stemming** is the conversion of a word to its morphological root\(^{113}\). When using stemming, the morphological root of words is added to the representations instead of the words themselves.

**Grouping of numbers** is the replacing of numbers by a special character. So, rather than indicating that a sentence includes the number 12 and that another sentence includes the number 182, it is merely noted that both include a number. This may help classification if it is the presence of a number that is an indication of a certain type, rather than its exact value.

**Conversion to lower case characters** means that all upper case characters in a word are replaced by their lower case equivalent.

A **minimal term frequency** adds a threshold for word frequency. When applying a minimal term frequency, words are only included in the representations if they apply at least a certain number of times in the training sentences of one class.

The selected words form a vocabulary \( \mathcal{V} \). This vocabulary contains all the words that appear in the dataset. Each sentence is represented as a vector \( \mathbf{w}_1, \ldots, \mathbf{w}_n \) (where \( n \) is the number of words in the database). Each weight corresponds to a word from the vocabulary. The value of the weight depends on the weighing method used. We tested three commonly used methods:

- A **binary** weight that indicates whether or not a word appears in the sentence. In this case, the weight is one if the word appears in the sentence, and zero if not.
- A **term frequency** (TF) weight that is equal to the number of times that a word appears in a sentence.
- An **inverse document frequency weight** (TFIDF), which incorporates information on how often a word appears in the entire corpus. The weight is equal to:

\[
\text{term frequency} \times \log \frac{\text{number of sentences}}{\text{number of sentences containing } w}
\]

This weight takes into consideration that words appearing in many sentences are less discriminating.

### 4.6.2 First Experiment: Pre-processing

In this experiment, different configurations were tested in order to compare the effect of the different settings. We used the same dataset as the one used in testing our pattern-based classifier\(^{114}\) (see section 4.5), using only the sentences. Moreover, sentences of classes that were too small were left out, as were sentences with mixed types, i.e. two or more classifications because of auxiliary sentences. This left 584 sentences for the experiment.

Because of the relatively small data set, we used cross-validation for evaluating our classifiers, i.e. use the same data for training and testing. We used a special form of cross-validation: the Leave-One-Out (LOO) procedure. In this procedure, each instance is selected for testing once and evaluated on a classifier based from the training set of all other instances. The results of these tests are shown in table 11. The LOO Accuracy is calculated using the results of all tests combined, and is a predictor for the accuracy of a classifier based on the entire set.

\(^{113}\) We used the Dutch version of the Snowball stemmer (http://snowball.tartarus.org/).

\(^{114}\) In this section, we will refer to the pattern-based classifier as the knowledge engineering or KE classifier.
Table 11: LOO Accuracy for different data representation settings

<table>
<thead>
<tr>
<th></th>
<th>Term weight</th>
<th>Stop list</th>
<th>Group numbers</th>
<th>Stemming</th>
<th>Min. term frequency</th>
<th>Lower case</th>
<th>LOO accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>binary</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>1</td>
<td>no</td>
<td>93.32</td>
</tr>
<tr>
<td>baseline + TF</td>
<td>TF</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>1</td>
<td>no</td>
<td>92.29</td>
</tr>
<tr>
<td>baseline + TFIDF</td>
<td>TFIDF</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>1</td>
<td>no</td>
<td>93.32</td>
</tr>
<tr>
<td>baseline + stop list</td>
<td>binary</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>1</td>
<td>no</td>
<td>94.01</td>
</tr>
<tr>
<td>baseline + grouping</td>
<td>binary</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>1</td>
<td>no</td>
<td>92.81</td>
</tr>
<tr>
<td>baseline + stemming</td>
<td>binary</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>1</td>
<td>no</td>
<td>92.47</td>
</tr>
<tr>
<td>baseline + min. term frequency 2</td>
<td>binary</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>2</td>
<td>no</td>
<td>93.15</td>
</tr>
<tr>
<td>baseline + min. term frequency 3</td>
<td>binary</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>3</td>
<td>no</td>
<td>92.47</td>
</tr>
<tr>
<td>baseline + lowercase</td>
<td>binary</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>1</td>
<td>yes</td>
<td>93.15</td>
</tr>
<tr>
<td><strong>Optimal</strong></td>
<td>binary</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>2</td>
<td>no</td>
<td>94.69</td>
</tr>
</tbody>
</table>

Table 11: LOO Accuracy for different data representation settings

Even without any pre-processing, the (predicted) accuracy is already quite high, above 93%. The different forms of pre-processing only improve this by small amounts, and often even decrease the performance (when used on their own). Nevertheless, using a stop list has a positive effect on the LOO accuracy.

The highest LOO accuracy as achieved by using binary weight, a stop list and a minimum term frequency of ‘2’, which resulted in a LOO accuracy of 94.69%\textsuperscript{115}. Table 12 shows the results for the different classes when using these settings. It shows that the classifier has trouble with definitions, misclassifying 6 out of 14. Also, more than half of the misclassified sentences is classified as an obligation (17 out of 31).

<table>
<thead>
<tr>
<th></th>
<th>In corpus</th>
<th>Missed</th>
<th>False</th>
<th>Recall</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>14</td>
<td>2%</td>
<td>6</td>
<td>4</td>
<td>57.14%</td>
</tr>
<tr>
<td>Permission</td>
<td>59</td>
<td>10%</td>
<td>5</td>
<td>7</td>
<td>91.53%</td>
</tr>
<tr>
<td>Obligation</td>
<td>181</td>
<td>31%</td>
<td>9</td>
<td>17</td>
<td>95.01%</td>
</tr>
<tr>
<td>Delegation</td>
<td>19</td>
<td>3%</td>
<td>2</td>
<td>0</td>
<td>89.47%</td>
</tr>
<tr>
<td>Publication provision</td>
<td>6</td>
<td>1%</td>
<td>1</td>
<td>0</td>
<td>83.33%</td>
</tr>
<tr>
<td>Application provision</td>
<td>41</td>
<td>7%</td>
<td>4</td>
<td>2</td>
<td>90.24%</td>
</tr>
<tr>
<td>Enactment date</td>
<td>18</td>
<td>3%</td>
<td>1</td>
<td>1</td>
<td>94.44%</td>
</tr>
<tr>
<td>Short title</td>
<td>4</td>
<td>1%</td>
<td>1</td>
<td>0</td>
<td>75.00%</td>
</tr>
<tr>
<td>Change – Scope</td>
<td>55</td>
<td>9%</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Change – Insertion</td>
<td>44</td>
<td>7%</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Change – Replacement</td>
<td>111</td>
<td>19%</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Change – Repeal</td>
<td>23</td>
<td>4%</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Change – Renumbering</td>
<td>9</td>
<td>2%</td>
<td>2</td>
<td>0</td>
<td>77.78%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>584</td>
<td>31</td>
<td>31</td>
<td>94.69%</td>
<td>94.69%</td>
</tr>
</tbody>
</table>

Table 12: Results per class when using optimal configuration

A more detailed analysis is made using the confusion matrix, shown in table 13. There we can see that the problem with definitions resides in an inability to distinguish between definitions\textsuperscript{115} Francesconi and Passerini (2007) report similar best settings: replacing digits and non-alphanumeric characters, stemming, binary weighting and a minimum term frequency of 2, leading to a LOO accuracy of 92.44%.
and obligations: all six misclassified definitions are classified as an obligation, while three of 
ine misclassified obligations are classified as definitions. A similar problem exists between 
obligations and permissions, with four out of five misclassified permissions classified as 
obligations, and four out of nine misclassified obligations classified as permissions.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Permission</th>
<th>Obligation</th>
<th>Delegation</th>
<th>Publication provision</th>
<th>Application provision</th>
<th>Enactment date</th>
<th>Short title</th>
<th>Change – Scope</th>
<th>Change – Insertion</th>
<th>Change – Replacement</th>
<th>Change – Repeal</th>
<th>Change – Renumbering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>8</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Permission</td>
<td></td>
<td></td>
<td></td>
<td>54</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obligation</td>
<td>3</td>
<td>4</td>
<td></td>
<td>172</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delegation</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Publication provision</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application provision</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enactment date</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Short title</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change – Scope</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change – Insertion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change – Replacement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change – Repeal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change – Renumbering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 13: Confusion matrix when using optimal configuration

4.6.3 Second Experiment: Generalisation Across Laws

Different laws often have a (slightly) different way of phrasing certain sentences. This means 
that, although within a certain law all definitions follow the same patterns, another law may 
use different patterns. Often, these patterns are slight variations of each other. This means 
that a classifier, when faced with a new law, may not perform as well.

Our first experiment does not give a good indication for the generalisation across laws. In the 
LOO setup, each sentence was tested using a classifier based on all other sentences in the 
corpus. This means that the training set also contained sentences from the same law as the 
sentence being tested. In order to test the classifier’s generalisation across laws, another 
experiment was conducted, this time testing all the sentences in one law using a classifier 
based on sentences from other laws in the corpus.

For each law, table 14 shows the number of sentences (size), the number of misclassifications 
in the original LOO experiment (Original LOO) and the number of misclassifications when 
testing the law using a classifier based on the other laws (Train/Test). Since the dataset for the 
Train/Test condition is smaller, the results are expected to be a bit worse. However, if the 
results are a lot worse, this may be because the law used phrasings that did not occur in the 
training set. Bill 30 411 is an example of a document that does a lot worse with a classifier 
trained on all other laws (28 misclassified versus 7 in original setup).

To confirm this, we conducted a separate LOO experiment for each law, using only the 
sentences of that law as dataset (shown in the last column of table 14). For two bills (30 411
and 31 722) this classifier is predicted to perform better than the classifier based on all other laws, suggesting that these two laws do indeed have some phrases that are unique to them. These seem different enough that a classifier trained on other laws will not automatically generalise to include them.

<table>
<thead>
<tr>
<th>Law</th>
<th>Size</th>
<th>Original LOO</th>
<th>Train/Test</th>
<th>One law LOO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Royal Decree Stb.1945, F 214</td>
<td>28</td>
<td>6</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>Bill 20 585 nr. 2</td>
<td>31</td>
<td>4</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>Bill 22 139 nr. 2</td>
<td>22</td>
<td>1</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Bill 27 570 nr. 4</td>
<td>21</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Bill 27 611 nr. 2</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Bill 30 411 nr. 2</td>
<td>141</td>
<td>7</td>
<td>28</td>
<td>22</td>
</tr>
<tr>
<td>Bill 30 435 nr. 2</td>
<td>40</td>
<td>3</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Bill 30 583 nr. A</td>
<td>26</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Bill 31 531 nr. 2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Bill 31 537 nr. 2</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Bill 31 540 nr. 2</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Bill 31 541 nr. 2</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Bill 31 713 nr. 2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Bill 31 722 nr. 2</td>
<td>32</td>
<td>1</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>Bill 31 726 nr. 2</td>
<td>78</td>
<td>3</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Bill 31 832 nr. 2</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Bill 31 833 nr. 2</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Bill 31 835 nr. 2</td>
<td>99</td>
<td>0</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 14: Number of misclassified sentences when testing each law in the dataset separately

In a third experiment, we built a classifier using the entire data set of the previous experiments, and two new laws were used to test the classifier. Both are recent bills, pending at Parliament. One bill (32 393, from May 2010) is an amending law, containing only changes in other laws. The second bill (32 398, from June 2010) is a new law, though it also describes many changes in existing laws.

In this experiment, we also included sentences of mixed type (which cannot be classified correctly) and lists, which were classified based on their headers.

Both new laws were classified by the ML classifier as well as the KE classifier. The results are shown in table 15. Both classifiers performed well, with the ML classifier scoring slightly better on Bill 32 393, but quite a bit worse on Bill 32 398.

<table>
<thead>
<tr>
<th>Test set</th>
<th>ML approach</th>
<th></th>
<th>KE approach</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nr. misclassified</td>
<td>Accuracy</td>
<td>Nr. misclassified</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Bill 32 393 nr. 2</td>
<td>71</td>
<td>3</td>
<td>95.77%</td>
<td>4</td>
</tr>
<tr>
<td>Lists</td>
<td>18</td>
<td>3</td>
<td>83.33%</td>
<td>1</td>
</tr>
<tr>
<td>Bill 32 398 nr. 2</td>
<td>sentences</td>
<td>205</td>
<td>23</td>
<td>88.78%</td>
</tr>
<tr>
<td>Lists</td>
<td>9</td>
<td>0</td>
<td>100%</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 15: Accuracy in the second experiment

Table 16 shows the confusion matrix for Bill 32 398. The results are quite similar to those of the first experiment, showing problems with the classification of definitions and with the distinction between permissions and obligations.
4.6.4 Analysis

As a trained SVM is in essence a black box, it is not possible to determine with absolute certainty why certain misclassifications occur. However, there are some shared features between the misclassifications that make for possible and plausible explanations of the errors. First of all, the ML classifier seems to suffer from two problems that also led to problems for the KE classifier:

- **Keywords appearing in subordinate sentences**: sometimes, keywords that are strongly linked to one of the classes appear in an auxiliary sentence (where it bears no influence on the type of the main sentence). For example, the sentence:

  *In that case, he notifies Our Ministers of his plans in advance, so that he can hear their opinion.*

  This sentence forms an obligation, but the word *can* in the auxiliary sentence is strongly related to a permission. For the KE approach, we already suggested that filtering out subordinate sentences could help improve the results; it seems this may also help the ML approach.

- **Missing standard phrases**: As was noted in earlier research on the KE classifier, the accuracy of the classifier when dealing with a new law can drop quickly if that law uses a phrase that was not encountered before.

A third problem that occurred with the KE classifier was caused by variations on known patterns that used the same words but a different word order, or that insert other words between the words that make up the actual pattern. It is generally acknowledged that a ML approach is more flexible and is therefore better when dealing with such variations. In our test, the ML classifier did not have problems with these variations.

116 Hij stelt in dat geval Onze Ministers vooraf in kennis van zijn plannen, ten einde hun mening daaromtrent te kunnen vernemen.
• **Keywords linked to different classes:** Certain keywords are linked to different classes. For example, *may* is an indicator of a permission, but *may not* indicates an obligation. The KE classifier recognises both patterns, and chooses between them based on which pattern is the longest. The ML seems to make more errors in this situation. This may improve with more training data.

• **Normative indicative:** In Dutch law, many obligations are written in the normative indicative, a sentence without any clear keywords. The KE classifier deals with statements of fact by using them as a “default” outcome, assigned to any sentences that do not contain a pattern that signifies another class. It would seem that the ML classifier links the signal words for a passive sentence (the Dutch verb *worden*) to obligations, causing non-obligation sentences that use that verb to be classified as obligations.

• **Not enough data:** In some cases, a standard phrase was present in our training set, but was filtered out due to the minimal term frequency of two. For example, in our first experiment, the only two misclassifications within all ‘changes’ were two renumberings. These sentences were actually reletterings and therefore the only two sentences that contained the Dutch word *verletterd*. Since a minimal term frequency of 2 was used, whenever one of the two reletterings was used as test item the word *verletterd* appeared only once in the training set and therefore was not extracted, and hence not used to classify the sentence.

It also seems that classes with fewer documents, and therefore poorer statistics, lead to more errors (but not always). Francesoni and Passerini report similar results.

• **Sparse data:** Typically, statistical text classification methods not only work better when trained on more data, but also on larger documents (although Opsomer et al. blame the large and therefore heterogeneous documents in his case for the poor classification performance). Sentences are small units to be classified and therefore the document vectors are almost empty.

• **Focus on the wrong keywords:** The KE classifier will only focus on keywords within the patterns provided. A ML classifier ‘picks’ its own keywords. This is usually an advantage, since it can discover new correlations. However, in our experiments, the classifier sometimes discovered correlations within the test set that do not generalise to new data. For example, one training set might by coincidence contain a lot of permissions for some advisory board. In this case, the ML classifier could link the words *advisory* and *board* to a permission, causing future non-permissions containing these words to be wrongfully classified as a permission. It is possible that this problem is caused by overfitting. SVMs are supposed to be resistant to overfitting (Joachims, 1998), but they are not immune.

In all the above experiments, no limit was set to the number of extracted terms from the training set; between 700 and 1700 terms were extracted in the above experiments. To investigate the question whether overfitting took place, some of the experiments were rerun with a maximum number of selected terms set to 50 words per class, leading to extraction of 250 up to 500 words. The results show a decrease in train accuracy and increase in test accuracy from the original experiments, suggesting that overfitting indeed took place when no limit was set to the amount of selected terms.
Franseconi and Passerini and Gonçalves and Quaresma used the same data set for training and testing (through cross-validation or LOO testing). None of these studies investigated the classification on completely new test sets that were not involved in the building process of the classifier. Franseconi and Passerini reported LOO testing with usually perfect accuracy on the training set, which could indicate that the classifier is overfit to the data. Gonçalves and Quaresma (2005) reported a reduced cross-validation accuracy due to the reduction of selected features from nearly perfect (up to 99.5%) to levels between 93 and 96%. Although this is taken to be a loss in general accuracy, it can also be an indication that the high accuracy levels were a result of overfitting to characteristics specific to the data set that was used. Further research is required to test this hypothesis.

- **Keywords outside of a standard phrase:** Many sentences can be classified based on some standard phrase. Since the ML classifier does not consider the order in which words appear, it is possible that it classifies based on one word of the phrase, rather than the complete phrase. Here, the rigidness of the KE classifier seems to be an advantage rather than a disadvantage.

- **Skewness:** Machine learners tend to favour bigger classes and prevalent patterns over the more uncommon patterns or smaller classes, which has a general positive effect if the distribution of classes and used patterns in the train data represents the real-world distribution. A negative effect is that uncommon patterns or classes are sometimes misclassified because of their small prior chance. This may be an explanation for the many sentences that are misclassified as an obligation.

The problems mentioned above can be used to explain most errors found, but there remain some errors for which we do not have an explanation yet.

### 4.7 Conclusion

The sentences in Dutch laws can be divided in different categories that each perform a distinct type of knowledge or operation. This classification can be applied automatically, using a classifier that is based on either a pattern-based approach or a machine-learning approach. In our experiments, both achieve an accuracy of over 90%, and it seems possible to improve this, by filtering out auxiliary sentences and by using a larger training set.

At first glance, the SVM classifier seems to perform better than the pattern-based approach, achieving a LOO accuracy of 94.7% on the test set, while the pattern-based classifier scores only 90.7%. However, the result of the pattern-based classifier was heavily impacted by some missing patterns: patterns that were used by a law in the test set that were not used by any laws in the training set. In the LOO setup, this was no (great) issue for the SVM classifier, as the training set would always contain sentences from the same law as the sentence being tested (and was therefore more likely to include the necessary pattern).

When it comes to generalisation to laws outside the training set, the SVM classifier seems to score worse than the pattern-based approach. However, this may improve when a bigger training set is reached.
At the moment, it seems that both methods can achieve rather similar results. We expect that, if we filter the subordinate sentences out of the input, the main difference will be in the classification of those sentences that are misclassified: the KE approach tends to misclassify as an obligation (using the default in case of a missing pattern), whereas the ML approach has more diverse misclassifications (though still focusing on obligations, likely due to skewness).

Our classification contains fifteen different classes. Compared to classifications such as that of Sartor (2007), Tiscornia and Turchi (1997) and Atienza and Manero (1998), the classification distinguishes few subclasses of the actual norms, though a comparison with Francesconi and Passerini (2007) suggests that this may be as detailed a classification as can be made using only surface features of the sentence.

Francesconi and Passerini classify provisions (which may be composed of several sentences) in ten categories, which fall in the same broad categories as we have found, though there are differences in different sub-types distinguished.

In our classification, we only detect two types of basic norms: permission and obligation. Francesconi and Passerini are more detailed and detect permission, duty, prohibition and exception. Due to the normative indicative construction, duty and prohibition are difficult to distinguish in Dutch law. With regard to exceptions, we have chosen to focus on the deontic nature of the exception (i.e. permission or obligation) rather than its status as an (explicit) exception.

As for the rule management class of norms, Francesconi and Passerini distinguish four modifications: repeal, insertion and substitution, which means that they do not have the renumbering operation or the scope declaration. The scope declaration is not an actual operation, but merely a sentence that reduces repetition. It does not form a provision all by itself, so it makes sense that it does not appear in Francesconi and Passerini’s classification of provisions. As for renumbering, this operation apparently does not exist in Italian laws. Likewise, there are no provisions to set a short title or the enactment date. On the other hand, Italian law uses delegification provisions, which are part of a process to simplify Italian law, and which make a law susceptible to change by lower sources. This is an operation that does not occur in Dutch law.

Another example of the differences between jurisdictions can be taken from Ogawa, Inagaki and Toyama (2008), who classify modifying sentences for an automated consolidation system. They recognise three types of renumbering operations: renumbering, attaching and shifting. Renumbering and attaching (adding an index to previously unnumbered text) correspond to our renumbering category. Shifting, however, does not occur in Dutch laws. It is a renumbering operation that does not explicitly specifies the target numbering, but instead specifies how many positions the articles are “shifted”, e.g. *Articles from 22 to 27 shall be shifted down two articles at a time.*

When attempting to classify the actual norms in the law, we initially focussed on distinguishing the different deontic norms and power-conferring norms. We found only patterns to distinguish between obligations and rights; no patterns were found that indicated power-conferring norms. We did find patterns for one subcategory of power-conferring norms,
namely delegation of legislative powers, and the related publishing provision. It may be possible that more patterns can be identified by focusing on such sub-category rather than the entire category. This may be one method to improve the current classifier.

Moreover, the ML classifier has been trained using a dataset classified using the classes for which patterns had been found. It may be that a ML approach can detect more classes than the KE approach, so it may be interesting to train a ML classifier based on a dataset that is classified using a more detailed classification.

A way to increase the performance of the classifiers may be to combine the two approaches. For example, rather than choosing one approach over the other, both classifiers can be run, with an additional module that decides which classification to use in case the outcome of the classifiers does not match. Another way to combine the approaches is by using the presence of the patterns in a sentence as additional features for the SVM, which may improve its performance.

If we consider the two methods as being more or less equal in terms of accuracy, then the pattern based approach is preferable, as it gives us more information on the reason for a given classification. This information is useful if when we proceed to make a model of the sentence (see the next chapter). However, if in the future, we know more exactly what information we need for this modelling, it may be possible to create a SVM that gives this information in addition to the classification, creating a level playing field.
5 Creating Model Fragments

After each sentence has been classified, an actual model for each sentence can be constructed.

The input for this step consists of classified individual sentences, with references marked-up. We also know the pattern that was used to classify each sentence. For several types of sentences, these patterns, together with some added features, are sufficient to extract all information needed to create a model of the sentence. This is usually the case with sentences that are about the law itself, instead of the subject matter of the law. These sentences are discussed in section 2. Other sentences, such as obligations, do focus on the subject matter, and can vary wildly. Simple patterns will not suffice to deal with these sentences, and to extract information from these types of sentences, we use a Dutch grammar parser. These sentences are discussed in section 3. Several of the ideas in this chapter have been published before (de Maat & Winkels, 2010a, 2011a, 2011b).

There have been earlier attempts to convert parse trees to models. Bos et al. (2004) have converted the output of a parser for the English language to first order logic using lambda calculus. An example of such a conversion, given by Bos et al., is:

\[
\exists a (\text{school-board}(a) \land \text{hearing}(a)) \land \exists b (\text{female}(b) \land \exists c (\text{dismiss}(c) \land \text{patient}(c, b) \land \text{at}(a, c) \land \exists d (\text{crowd}(d) \land \text{patient}(d, a) \land ((\exists e (\text{student}(e) \land \text{with}(d, e)) \land \exists f (\text{teacher}(f) \land \text{with}(d, f))) \land \text{event}(d))))))
\]

A similar approach has been followed by McCarthy (2007), who uses unification to transform the parse tree to a quasi-logical form. He has applied this technique on judicial opinions from
appellate courts in the United States. As an example, figure 11 shows the transformation of the sentence:

She has also brought this ADA suit in which she claims that her former employer, Policy Management Systems Corporation, discriminated against her on account of her disability.

McCarthy’s goal is different than ours, though. He states his goal is “to produce a structured case note, which is a computational version of the traditional "brief" that first-year law students are taught to write as a summary of each case in their casebooks.”; the transformation shown in figure 11 is an intermediate step towards attaining that goal.

For our goal, supporting the creation of knowledge based systems, such transformations seem less helpful. They are very detailed, and a far cry from the models used in current knowledge based systems. Most knowledge based systems choose predicates that correspond to some question that can be communicated to the user. For example, in POWER, the model for calculating cycling deduction contains the following class attributes (taken from van Engers et al., 2000):

- isInTheHabitOfTravellingToAPlaceOfWorkAtLeastThreeDaysAWeekAndForThat-PurposeMainlyRidesByBike
- regularCommuterTraffic
• travelsAtLeastOnceAWeekBetweenResidenceAndPlaceOrPlacesOfWork
• travellingBackAndForthBetweenResidenceAndPlaceOfWorkIsDoneWithin24Hours

In IBM’s translation of The Freedom of Information and Protection of Privacy Act (Powers, Adler & Wishart, 2004) we find similar coarse-grained attributes:

• proper-exceptions-to-notice-obligation
• disclosure-by-law-enforcement-institution
• authorized-by-a-constituent-next-of-kin-or-legal-representative
• for-the-purpose-for-which-it-was-obtained-or-compiled-or-for-a-consistent-purpose

These attributes all correspond with a single question that can be communicated to the user. Ideally, we want to generate models that are more in line with these attributes and conditions. Sarwa Bajwa, Samad and Mumtaz (2009) use an approach that creates UML models by (broadly speaking) mapping nouns to classes, verbs to methods and adjectives to attributes. Thus, the sentence:

\textit{Customer has purchased a red ball.}

Results in the following model:

![Figure 12: Samand and Mumtaz' UML model for Customer has purchased a red ball.]

A similar approach has been attempted as part of the POWER project, where partial UML models were generated by creating classes based upon nouns found in the text and adding adjectives and adjective phrases as attributes. Several “fixed juridical constructs” (many of which correspond with the patterns used by us to classify sentences) can also be recognised and are translated as a specific rule attached to the class (van Gog & van Engers, 2001). For example, the sentence:

\textit{a house is considered an owned house if the house is owned by the tax payer}

Results in the model:

![Figure 13: Van Gog and van Engers' UML model for a house is considered an owned house if the house is owned by the tax payer]
Other methods do not generate UML models, but mark-up relevant sections of the text. Kiyavitskaya et al. (2008) use similar patterns as presented in section 4.1 to identify rights and obligations in case law and to identify certain basic entities: actor, policy, event, date and information. These were then marked in the text, yielding results such as:

<Right>A <Actor>covered entity</Actor> may deny an <Actor>individual</Actor>’s request for amendment, </Right> if it determines that the <Information>protected health information</Information> or record that is the subject of the request:

<Index>(i)</Index> Was not created by the <Actor>covered entity</Actor>,

<Exception>unless the <Actor>individual</Actor> provides a reasonable basis to believe that the originator of <Information>protected health information</Information> is no longer available to <Policy>act</Policy> on the requested amendment </Exception> ... 

For statute law, a similar approach has been followed by Biagioli, Francesconi, Passerini, Montemagni and Soria (2005), who attempt to fill in frames for provisions in Italian laws. For example, for an obligation, this approach attempts to fill the slots addressee, action and third-party. It does so by assigning specific elements of the sentence to specific slots. As an example of this, they say: “the addressee of an obligation typically corresponds to the syntactic subject of the sentence, while the action (s)he is obliged to carry out is usually expressed as an infinitival clause”. This leads to mark-up such as:

[[Il comitato misto]subj]addressee] `e tenuto [[a raccomandare modifiche degli allegati secondo le modalit`a previste dal presente accordo]i_clause]action]

[[The Joint Committee]subj]addressee] shall [[be responsible for recommending amendments to the Annex as foreseen in this Agreement]i_clause]action].

Brighi, Lesmo, Mazzei, Palmirani and Radicioni (2008) and Palmirani and Brighi (2010) likewise try to fill semantic frames by mapping parts of a parse tree to roles in the frame. They confine themselves to amending clauses in Italian law. A similar system was made by Spinosa et al. (2009), though they used a chunker, not a parser, as a pre-processor for their system117.

For many sentences, though, the use of a parser to recognise the different elements in a sentence does not seem to be necessary. In particular, sentences that deal with the law itself (rather than with the application domain of the law) follow such strict patterns, that the information can be extracted based upon these patterns. This is demonstrated by Ogawa, Inagaki and Toyama (2008), who obtain a formal representation of amending clauses by means of regular expressions.

In this chapter, we will attempt to create semantic frames for Dutch law, similar to those introduced by van Kralingen (1995) en used by Biagioli et al. (2005) and Brighi et al. (2008). In order to fill these frames, we will use textual patterns for those sentences that deal with the law (discussed in section 5.1, followed by an experiment in section 5.2). For those sentences that deal with the application domain of the law, we will use a parser to fill the slots (discussed

117 A chunker splits a sentence into groups of related words, such as a noun phrase. It provides less information than a full parser.
in section 5.3, followed by a discussion of our experiences with this method in section 5.4). The next step in the process, the integration of the models, is discussed in section 5.5.

5.1 Sentences Dealing With the Law

There are several types of sentences that are about the law itself. These are sentences that change the text of a law or set the enactment date of a law. Compared to sentences which deal with the subject of the law (which could be anything: oil tankers, accountancy, book prices, etc.) they have very limited variation. In general, they follow clear patterns, from which it is easy to extract the information we need.

5.1.1 Scope Declaration

A scope declaration always contains one relevant piece of information: the scope. This scope is identified by a reference to the relevant text. If we assume that the reference parser has successfully identified all references during pre-processing, then the scope is easily found: it is the reference at the start of the sentence, or more easily: the one reference that appears in the sentence.

<table>
<thead>
<tr>
<th>Act of May 20th, 2010 (Stb. 2010/200), article I, introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Penal Code is modified as follows:</td>
</tr>
<tr>
<td>Scope</td>
</tr>
<tr>
<td>Scope</td>
</tr>
</tbody>
</table>

If a scope is set for an article or subpart, then all changes made within that article or subpart refer to that scope.

5.1.2 Repeal

A repeal removes an entire law, or part of that law. Other than the knowledge that we are dealing with a repeal, only one bit of information needs to be extracted: the reference to the text that is to be repealed. This reference is directly at the beginning of the main sentence (though it is also the only reference).

<table>
<thead>
<tr>
<th>Act of June 12th, 2008 (Stb. 2008/229), article I, sub B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article 4 is repealed.</td>
</tr>
<tr>
<td>Repeal</td>
</tr>
<tr>
<td>Repeal</td>
</tr>
</tbody>
</table>

Alternatively, only part of the text is repealed, i.e. a number of words are deleted. In that case, we need to extract the location where the text to be modified is located, and the text that should be removed. The location will be a reference at the start of the sentence, following the word *In* (though again, it will be the only reference in the sentence). The text to be removed will be marked with double angle quotes.

---

118 Wet van 20 mei 2010 (Stb. 2010/200), artikel I, aanhef
Het Wetboek van Strafrecht wordt als volgt gewijzigd:
119 Wet van 12 juni 2008 (Stb. 2008/229), artikel I, lid B
Artikel 4 vervalt.
Act of May 20th, 2010 (Stb. 2010/205), article III\textsuperscript{120}

In article 438, sub 1, 2\textsuperscript{nd} item, of the Penal Code, «profession or appointment, » is repealed.

<table>
<thead>
<tr>
<th>Repeal</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>Penal Code, article 438, sub 1, 2\textsuperscript{nd} item</td>
</tr>
<tr>
<td>Text</td>
<td>profession or appointment,</td>
</tr>
</tbody>
</table>

5.1.3 Insertion

An insertion inserts some new text into the document. There are several different scenarios for insertion. The easiest situation exists when an entire new (numbered) element is inserted. In this case, sentence will be something like this:

Act of May 13\textsuperscript{th}, 2004 (Stb. 2004/220), article I, sub A\textsuperscript{121}

After article 7:1 a new article is inserted in section 7.1, reading: …

We need to model two pieces of information. Firstly: the text to be inserted. This can be found after the colon. Then, we need to know where to insert it. The (part of the) document is denoted by a reference that is preceded by in (in this case: in section 7.1). If the insertion has been preceded by a scope declaration, then this reference may be incomplete, and needs to be combined with the scope declaration to get a complete reference. The location is given by another reference, preceded by either before or after\textsuperscript{122}.

<table>
<thead>
<tr>
<th>Insertion</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>Section 7.1</td>
</tr>
<tr>
<td>Location</td>
<td>Article 7:1</td>
</tr>
<tr>
<td>Position</td>
<td>after</td>
</tr>
<tr>
<td>Text</td>
<td>…</td>
</tr>
</tbody>
</table>

If the target is entirely defined by the scope definition, it will be missing from the sentence. Similarly, if the location is the end of the document, no reference for the position will be given. Instead, the text is said to be appended instead of inserted. For example, the following sentence is missing both references:

Act of May 20\textsuperscript{th}, 2010 (Stb. 2010/200), article I, sub A, sub 2\textsuperscript{123}

A paragraph is appended, reading: …

<table>
<thead>
<tr>
<th>Insertion</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>end</td>
</tr>
<tr>
<td>Text</td>
<td>…</td>
</tr>
</tbody>
</table>

If a modification is made within a paragraph, it is not possible to point to the position using a reference (as sentences and words are not numbered). In these cases, instead of a reference, quoted text is used to describe the location:

\textsuperscript{120} Wet van 20 mei 2010 (Stb. 2010/200), artikel III
In artikel 438, eerste lid, onder ten 2e, van het Wetboek van Strafrecht vervalt «beroep of betrekking,».

\textsuperscript{121} Wet van 13 mei 2004 (Stb. 2004/220), artikel I, lid A
Na artikel 7:1 wordt in afdeling 7.1 een artikel ingevoegd, luidende: …

\textsuperscript{122} More complex positioning sometimes occurs, but will not be discussed here.

\textsuperscript{123} Wet van 20 mei 2010 (Stb. 2010/200), artikel I, lid A, lid 2
Er wordt een lid toegevoegd, luidende: …
In article 30, sub 6, of the Motorised Vehicles Liability Insurance Act, a sentence is inserted after the sentence «…», reading: …

The quoted text is marked using double angle quotation marks.

### Insertion

<table>
<thead>
<tr>
<th>Target</th>
<th>Motorised Vehicles Liability Insurance Act, article 30, sub 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>…</td>
</tr>
<tr>
<td>Position</td>
<td>After</td>
</tr>
<tr>
<td>Text</td>
<td>…</td>
</tr>
</tbody>
</table>

#### 5.1.4 Replacement

There are two main patterns of replacement, which yield a slightly different frame. The first pattern deals with the replacement of text within a given structure part. The text to be replaced is always marked with double angle quotation marks «». The replacing text is always at the end, preceded by a colon. The location of the replacement is at the beginning of the sentence, a reference following the word In. As with the other changes, this should be the only reference in the main sentence (disregarding any references that appear inside the quoted text).

**Act of May 20\textsuperscript{th}, 2010 (Stb. 2010/200), article I, sub B\textsuperscript{125}**

In article 43a, «a conviction» is replaced by: an earlier conviction.

### Replacement

<table>
<thead>
<tr>
<th>Target</th>
<th>article 43a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old Text</td>
<td>a conviction</td>
</tr>
<tr>
<td>New Text</td>
<td>an earlier conviction</td>
</tr>
</tbody>
</table>

The second type of replacement replaces an entire structure element, and does therefore not include a text to replace. The replacing text is again found after the colon, and the location is given as a reference after In at the beginning of the sentence. The location may be missing, which is usually the case if it has been set earlier with a scope declaration.

**Act of May 20\textsuperscript{th}, 2010 (Stb. 2010/205), article IV, sub I\textsuperscript{126}**

Article 31, sub 2, will read:

2. …

### Replacement

<table>
<thead>
<tr>
<th>Target</th>
<th>Article 31, sub 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Text</td>
<td>2. …</td>
</tr>
</tbody>
</table>

\textsuperscript{124} Wet van 20 mei 2010 (Stb. 2010/200), article VII

In artikel 30, zesde lid, van de Wet aansprakelijkheidsverzekering motorrijtuigen wordt na de zin «…» een zin ingevoegd, luidende: …

\textsuperscript{125} Wet van 20 mei 2010 (Stb. 2010/200), artikel I, lid B

In artikel 43a wordt een veroordeling vervangen door: een vroegere veroordeling.

\textsuperscript{126} Wet van 20 mei 2010 (Stb. 2010/205), artikel IV, lid I

Artikel 31, tweede lid, komt te luiden:

2. …
5.1.5 Renumbering

Renumbering changes the index of parts of a text. Though it is very uncommon for articles and document parts above the level of articles to be renumbered, paragraphs and list items are frequently renumbered. The sentence will list the part(s) to renumber and the new numbering. A reference to the part(s) to renumber can be found at the start of the sentence, while the new index can be found after are renumbered to.

<table>
<thead>
<tr>
<th>Modern Migration Policy Act, article VIII, sub B</th>
<th>127</th>
</tr>
</thead>
<tbody>
<tr>
<td>The articles 2a to 2u are renumbered to the articles 2i to 2cc.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Renumbering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target: articles 2a to 2u</td>
</tr>
<tr>
<td>New numbering: articles 2i to 2cc</td>
</tr>
</tbody>
</table>

A second common pattern for renumbering is adding a number to a previously unnumbered paragraph, such as:

<table>
<thead>
<tr>
<th>Care Contract System Revision Act, article II, sub Q, sub 1</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before the text of article 64, the index &quot;-1. &quot; is placed.</td>
<td></td>
</tr>
</tbody>
</table>

This operation is different from the “real” renumbering, as it changes the structure of the article by introducing a subparagraph. Hence, it cannot use the frame shown above, though the fields are very similar. This operation has a target, which may be present in the sentence as a reference following the text of. In the case of our example, this target is article 64. It is more common, though, for this kind of sentence not to name the target, which will have been declared before through a scope provision, and will have to be retrieved from the scope provision when integrating the models. The other field of the frame is the new index. So, for the example above, the complete frame would be:

<table>
<thead>
<tr>
<th>Numbering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target: article 64</td>
</tr>
<tr>
<td>New index: -1.</td>
</tr>
</tbody>
</table>

5.1.6 Enactment Date

The basic sentence that sets an enactment date will include two pieces of information: the document (or parts of document) to be enacted, and the date on which it is to be enacted.

<table>
<thead>
<tr>
<th>Fuel Taxes Environment Tariff Act 1991, article IV, sub 1, first sentence</th>
<th>131</th>
</tr>
</thead>
<tbody>
<tr>
<td>This law is enacted starting on January 1st, 1991.</td>
<td></td>
</tr>
</tbody>
</table>

---

127 *Wet modern migratiebeleid, article VIII, lid B*
De artikelen 2a tot en met 2u worden vernummerd tot de artikelen 2i tot en met 2cc.

128 *Wet herziening overeenkomstenstelsel zorg, artikel II, lid Q, lid 1*
Voor de tekst van artikel 64 wordt de aanduiding "-1." geplaatst.

129 Strictly speaking, the target of the operation is the text of article 64, but as this operation is always aimed at the text, simply identifying it as article 64 will survive.

130 Again, strictly speaking, such sentences do name the target as the text, which is an incomplete reference that can be completed using the scope provision (like other incomplete references that are governed by a scope provision).

131 *Tarievenwet brandstofheffingen milieu 1991, artikel IV, eerste lid, eerste volzin*
Deze wet treedt in werking met ingang van 1 januari 1991.
It is rather uncommon for a law to specify the exact date on which it will be enacted. More commonly, it will defer the decision to a royal decree:

**Exception Situations Coordinating Act, article 11**
This law is enacted on a date to be set by Royal Decree.

**Delegated Enactment Date**

<table>
<thead>
<tr>
<th>Target</th>
<th>this law</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delegation target</td>
<td>Royal Decree</td>
</tr>
</tbody>
</table>

Or it will relate it to the publication date:

**Act of July 7th, 2010 (Stb. 2010/305), article 9**
This law is enacted starting on the day after the date of publication of the Bulletin of Acts and Decrees in which she is included.

<table>
<thead>
<tr>
<th>Target</th>
<th>this law</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>the day after the date of publication of the Bulletin of Acts and Decrees in which she is included.</td>
</tr>
</tbody>
</table>

Both constructions follow fixed text patterns, which can be detected instead of a date. In addition to these straightforward enactment date setting sentences, more complicated constructions exist which include exceptions for which the date is set separately.

### 5.1.7 Short Title

A sentence setting a short title contains two pieces of information: the law for which the short title is set, and the actual short title.

**Nobility Act, article 9**
This law may be cited as Nobility Act.

Or:

**Trade Name Act, article 10**
This law may be cited under the title “Trade Name Act”.

---

132 **Coördinatiewet uitzonderingstoestanden, artikel 11**
Deze wet treedt in werking op een bij koninklijk besluit te bepalen tijdstip.

133 **Wet van 7 juli 2010, artikel 9**
Deze wet treedt in werking met ingang van de dag na de datum van uitgifte van het Staatsblad waarin zij wordt geplaatst.

134 **Wet op de adeldom, artikel 9**
Deze wet kan worden aangehaald als Wet op de adeldom.

135 **Handelsnaamwet, artikel 10**
Deze wet kan worden aangehaald onder de titel "Handelsnaamwet".
The law concerned is indicated with a reference at the start of the sentence. Usually, this is *this law*, but sometimes, the short title of some other law is changed. The new short title follows the text *may be cited as*, *may be cited under the title* or *may be cited under the name*. It may be enclosed in double or single quotes, and is sometimes preceded by a colon.

<table>
<thead>
<tr>
<th>Short title</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>This law</td>
<td></td>
</tr>
<tr>
<td>Trade Name Act</td>
<td></td>
</tr>
</tbody>
</table>

### 5.1.8 Application Provision

An application provision refers to a text that does (or does not) apply in a given context (usually the article in which the application provision). For the most basic application provisions, there is only one piece of information to be extracted: a reference to the text that does or does not apply. This reference can be found at the beginning of the sentence:

**Sickness Benefits Act, article 11a, sub 5**

Article 11, sub 3, applies accordingly.

<table>
<thead>
<tr>
<th>Application</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article 11, sub 3</td>
<td></td>
</tr>
</tbody>
</table>

Many application provisions do come with additional restrictions, which often relate to the subject matter of the law.

**Partnership Taxes Act 1969, article 12b, sub 4**

This article does not apply to brands and logos produced by the taxable person and any similar assets.

If this is the case, then the core of the sentence is still modelled in the same manner, but the additional restrictions need to be processed in the manner presented in section 3.

### 5.1.9 Delegation

A delegation contains two pieces of information: to whom or what the setting of rules is delegated, and what the scope of such rules may be. For example:

**Notary Act, article 56, sub 5**

By order-in-council, rules are set regarding the manner in which the size of the personal capital as meant in sub 4 is determined.

Here, the setting of new rules is delegated to an order-in-council, and their subject is *regarding the manner in which the size of the personal capital as meant in sub 4 is determined*. We can assemble the following frame:

---

136 Ziektewet, artikel 11a, vijfde lid
Artikel 11, derde lid, is van overeenkomstige toepassing.
137 Wet op de venootschapsbelasting 1969, artikel 12b, vierde lid
Dit artikel is niet van toepassing op door belastingplichtige voortgebrachte merken, logo’s, en daarmee vergelijkbare vermogensbestanddelen.
138 Notariswet, artikel 56, vijfde lid
Bij algemene maatregel van bestuur worden regels gesteld met betrekking tot de wijze waarop de hoogte van het in het vierde lid bedoelde eigen vermogen wordt bepaald.
### Delegation (Obligation)\(^{139}\)

<table>
<thead>
<tr>
<th>Target</th>
<th>order-in-council</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>the manner in which the size of the personal capital as meant in sub 4 is determined</td>
</tr>
</tbody>
</table>

As the sentence does not contain an agent, the frame does not contain one either. As a result of integration with other rules (in this case, generic rules which restrict who is allowed to issue an order-in-council), an agent can be added.

The subject usually follows words like regarding or on. In the case of a sentence starting with by order-in-council, the target is always an order in council. In an active sentence, with a pattern like may set rules or sets rules, the subject of the sentence (appearing just before the pattern) is the target:

**Sworn-in Interpreters and Translators Act, article 15, sub 2\(^{140}\)**

> Our Minister sets rules with regards to the certificate.

<table>
<thead>
<tr>
<th>Delegation (Obligation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
</tr>
<tr>
<td>Subject</td>
</tr>
</tbody>
</table>

### 5.1.10 Publication Provision

A publication provision mentions a decision or decree that has to be published, who is responsible for publishing it, and where it should be published. For example:

**Consumer Protection Retainment Act Article 5.1, sub 5\(^{141}\)**

> Our Minister announces the cooperation protocols in the State Gazette.

In this pattern, the person making the announcement is listed before the pattern (announces), the subject is listed after the announcement, and the venue is listed following in.

<table>
<thead>
<tr>
<th>Publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
</tr>
<tr>
<td>Venue</td>
</tr>
<tr>
<td>Subject</td>
</tr>
</tbody>
</table>

An alternate pattern uses a passive sentence, which does not list the agent:

**Announcement Act, article 4, sub 2, sentence 2\(^{142}\)**

> Such an announcement is announced in the State Gazette.

---

\(^{139}\) The header Delegation (Obligation) refers to the fact that the rules must be set, rather than a situation in which rules may be set (which we would denote as a Delegation(Permission).

\(^{140}\) *Wet beëdigde tolken en vertalers, artikel 15, tweede lid*

Onze Minister stelt regels ten aanzien van het legitimatiebewijs.

\(^{141}\) *Wet handhaving consumentenbescherming, artikel 5.1, vijfde lid*

Onze Minister doet mededeling van de samenwerkingsprotocolen in de Staatscourant.

\(^{142}\) *Bekendmakingswet, artikel 4, tweede lid, tweede volzin*

Van zodanige bekendmaking wordt mededeling gedaan in de Staatscourant.
Here, the subject is listed before the pattern (*is announced*), and the resulting frame is:

<table>
<thead>
<tr>
<th>Publication</th>
<th>Venue</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>the State Gazette</td>
<td>such an announcement</td>
</tr>
</tbody>
</table>

### 5.2 Experiment

In order to determine the feasibility of the modelling rules as prescribed in the previous sections, we have conducted a study of the sentences of the corpus that was also used in the experiment described in section 4.5. For each sentence belonging to one of the categories that we wish to model, it was determined by hand whether or not the rules presented in the previous sections could actually be applied in order to decompose the sentence into the different elements needed for the model.

Once again, this is about patterns. However, these patterns are not the ones used to classify the sentences (as presented in chapter 4). Here, we assume that the sentence has been classified correctly (even though that will not always be the case), and apply different patterns; hence the results differ from those in chapter 4. The outcome of this experiment is shown in table 17.

<table>
<thead>
<tr>
<th>Delegation</th>
<th>Sentences</th>
<th>Lists</th>
<th>No problems</th>
<th>Missing references</th>
<th>Additional parsing required</th>
<th>Both missing references and additional parsing required</th>
<th>Patterns do not apply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delegation</td>
<td>19</td>
<td>4</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Publication</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>41</td>
<td>4</td>
<td>19</td>
<td>1</td>
<td>21</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Enactment date</td>
<td>18</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Short title</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change – Scope</td>
<td>55</td>
<td>5</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change – Insertion</td>
<td>44</td>
<td>42</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change – Replacement</td>
<td>111</td>
<td>102</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Change – Repeal</td>
<td>23</td>
<td>18</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Change – Renumbering</td>
<td>9</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Total            | 330       | 13    | 294         | 14                 | 21                          | 1                                                      | 13                    |

Table 17: Expected results for modelling sentences based on patterns

Though most of the sentences could be handled with the patterns as described in section 5.1, this did not automatically guarantee a correct outcome. Within the sentences that did conform to the patterns, we noted two problems that might lead to problems. First of all, several of the application provisions have conditions that need to be parsed in order to be properly modelled. So, despite the main sentence conforming to the pattern, the sentence as a whole might not be modelled correctly. In addition, some sentences contained references to renumbered or newly inserted elements, such as *sub 4 (new)* or *item b that has been renumber to item c*. Some other sentences refer to the header of an element, such as *the header of section 5*. As the reference parser as described in section 3.4 does not yet (completely and fully) detect such references, these sentences would not yet be correctly modelled.
As the table shows, the corpus contains 23 delegations. Eighteen of these follow a standard pattern. Three sentences did not follow the standard pattern of defining a topic on which rules may be created, but rather defined actions that could be taken, thus deviating from the standard pattern. Another one referred back to an earlier provision to describe the topic of the rules allowed, using a non-standard pattern. The last sentence split the topic of those rules in two parts, also deviating from the standard patterns:

| Bill 31 726 nr. 2, article I, sub P, quoted text (to replace text in the Base Databases Act) |
| By order-in-council, to protect the private life of persons to whom data that has been included in the base databases, as meant in article 2, can be traced, restrictions may be set for specific data or categories of data with regards to the stipulations in sub 1. |

The six publication provisions in the corpus all follow the basic pattern. There are 45 application provisions in the corpus. Of these 45, nineteen follow the most basic format, having no conditions. Fourteen sentences include a topic on which the target does or does not apply (following the word on or the words with regards to), four sentences include an explicit condition (an auxiliary if-sentence), and one sentence has both.

For these sentences, the application provision itself conforms to the patterns, and can be decomposed into the different frame elements. However, the topic and conditions will also need to be modelled, and that can be a more complicated task (involving the methods discussed in section 5.3). The next provision shows how complicated these topics may become:

| Bill 31 531 nr. 2, article I, quoted text (to replace a text in the Competition Act) |
| Without prejudice to sub 1, article 6, sub 1, does also not apply to contracts, decisions and collusive behaviour as meant in that article, insofar enterprises or associations of entrepreneurs are involved who are actual or potential competitors on one or more relevant markets, if the combined market shares of the enterprises or associations of entrepreneurs involved in the contract, decision or collusive behaviour does not exceed 10% on any of the relevant markets to which the contract, decision or collusive behaviour pertains. |

---

143 Wetsvoorstel 31 726, artikel I, lid P, geciteerde tekst (ter vervanging van een tekst in de Wet basisregistraties)
Bij algemene maatregel van bestuur kunnen ter bescherming van de persoonlijke levenssfeer van personen tot wie de gegevens die zijn opgenomen in de basisregistraties, bedoeld in artikel 2, herleidbaar zijn voor daarbij aangewezen gegevens of categorieën van gegevens beperkingen worden vastgesteld ten aanzien van het bepaalde in het eerste lid.

144 Wetsvoorstel 31 531 nr. 2, artikel I, geciteerde tekst (ter vervanging van een tekst in de Mededingingswet)
Onverminderd het bepaalde in het eerste lid, geldt artikel 6, eerste lid, voorts niet voor overeenkomsten, besluiten en onderling afgestemde feitelijke gedragingen als bedoeld in dat artikel voor zover daarbij ondernemingen of ondernemersverenigingen betrokken zijn die daadwerkelijke of potentiële concurrenten zijn op een of meer van de relevante markten, indien het gezamenlijke marktaandeel van de bij de overeenkomst, het besluit of de onderling afgestemde feitelijke gedraging betrokken ondernemingen of ondernemersverenigingen op geen van de relevante markten waarop de overeenkomst, het besluit of de onderling afgestemde feitelijke gedraging van invloed is, groter is dan 10%. 

There are two sentences that include restrictions on how the targets should apply, using the phrase *with the understanding that*. They introduce several provisions that are only relevant for the specific situation described in the application provision:

**Bill 31 835 nr. 2, article 5, sub 1**

The articles 9, 11 to 14, 18, 20, 21, 23, 29, 39, sub 1, and 41 of the General Act apply to the board of advisors, with the understanding that:

1. by remuneration or compensation as meant in article 14 of the General Act is understood: holiday allowance; and
2. the board of advisors may issue the annual report as meant in article 18, sub 1, of that law together with the annual report of the management.

Though such sentences do follow a regular format (and the additional provisions follow a regular format as well), the current frame does not accommodate these features.

Two sentences describe the conditions in which their target applies by referring to the application of other sections of the law:

**Bill 30 411 nr. 2, article 3.2, sub 4**

With regards to the application of sub 2, the Financial Services Act as well as the articles 2.7, sub 3, 2.15f and 2.16 of this law do apply.

One sentence referred to a modified target, which is not yet recognised by the reference parser. Three sentences do not use an explicit reference for the target, referring to the law in general, and in that deviate from the standard pattern:

**Bill 30 435 nr. 2, article III, sub 2**

With regards to the untimely decision on a request or appeal which has been lodged before the time of enactment of this law, the law as it applied before that time remains applicable.

Of the eighteen enactment date provisions in the corpus, sixteen follow the generic patterns. One of the deviating sentences includes an (also non-standard) application provision as an auxiliary sentence:

---

145 Wetsvoorstel 31 835 nr. 2, artikel 5, eerste lid
De artikelen 9, 11 tot en met 14, 18, 20, 21, 23, 39, eerste lid, en 41 van de Kaderwet zijn van overeenkomstige toepassing op de raad van advies, met dien verstande dat:
a. onder bezoldiging of schadeloosstelling als bedoeld in artikel 14 van de Kaderwet wordt verstaan: vacatiegeld; en
b. de raad van advies het jaarverslag, bedoeld in artikel 18, eerste lid, van die wet tezamen met het jaarverslag van het bestuur kan uitbrengen.

146 Wetsvoorstel 30 411 nr. 2, artikel 3.2, vierde lid
Met betrekking tot de toepassing van het tweede lid zijn de Wet financiële dienstverlening alsmede de artikelen 2.7, derde lid, 2.15f en 2.16 van deze wet van overeenkomstige toepassing.

147 Wetsvoorstel 30 435 nr. 2, artikel III, tweede lid
Ten aanzien van het niet tijdig beslissen op een aanvraag die of een bezwaarschrift dat is ingediend voor het tijdstip van inwerkingtreding van deze wet, blijft het recht zoals dit gold voor dat tijdstip van toepassing.
Extraordinary Decree Labour Relations 1945, article 33, sub 1

This decree, with regards to which the competence, meant in article 9, sub 2, of the Decree on the Extraordinary State of Emergency cannot be exercised, is enacted starting on October 15th, 1945.

The other also includes an application provision, and is actually a mixed type sentence, consisting of two sentences that are concatenated:

Bill 30 583 nr. A, article II

This law is enacted starting on the day after the date of publication of the Bulletin of Acts and Decrees in which she is included, and applies to the time periods that start after the enactment date of this law.

The four short title provisions in the corpus all follow the generic pattern, as do the 60 scope provisions. Of the 44 insertion provisions, 38 follow the basic pattern, while six others follow a more complicated pattern: four of these provisions also renumber some items, one provision makes multiple insertions (the instruction includes each time to indicate that the insertion has to be repeated), and one has an auxiliary sentence to prevent a clash with another change.

There are 111 replacement provisions, of which 45 follow the basic pattern for the replacement of a structure element, including one sentence that replaces a header. There is one replacement of a structure element that also renumbers, and one that contains an auxiliary sentence to prevent a clash with another change.

There are 49 replacement provisions that follow the basic pattern for the replacement of text. Seven make multiple replacements using each time, and six makes multiple replacements by only specifying multiple targets. One of the multiple replacements also changes a header.

Two replacements deviate from the patterns. One of the text replacements does not specify which text must be replaced by quoting it, but describes it as the definition of ‘Our Minister’:

Bill 22 139 nr. 2, article I, sub A

In article 1, sub 1, the definition of ‘Our Minister’ is replaced by: Our Minister of Agriculture, Nature and Fishing.

Another deviates by using two references rather than one to define the target of the replacement:

---

148 Buitengewoon Besluit Arbeidsverhoudingen 1945
Dit besluit, ten aanzien waarvan de bevoegdheid, bedoeld in artikel 9, tweede lid, van het Besluit op den bijzonderen staat van beleg niet kan worden uitgeoefend, treedt in werking met ingang van 15 oktober 1945.

149 Wetsvoorstel 30 583 nr. A, artikel II
Deze wet treedt in werking met ingang van de dag na de datum van uitgifte van het Staatsblad waarin zij wordt geplaatst en is van toepassing op tijdvakken die aanvangen na inwerkingtreding van deze wet.

150 Wetsvoorstel 22 139 nr. 2, artikel I, lid A
In artikel 1, eerste lid, wordt de begripsomschrijving van ‘Onze Minister’ vervangen door: Onze Minister van Landbouw, Natuurbheer en Visserij.
In article 15, the subparagraphs 1, 2 and 3 will read: …

There are 23 repeals. Five of these follow the basic pattern for repealing an entire structure element, and eight follow the basic pattern for repealing part of the text of an element. Four sentences modify the header of a text (thus affecting the index as well, and effectively renumbering the element). Two sentences make multiple repeals (using each time, as with the insertion provisions) and three sentences also renumber some items. One repeal does not follow a standard pattern, as it includes a reference to the date of the repeal:

Extraordinary Decree Labour Relations 1945, article 33, sub 2

From that date, the Extraordinary Decree Labour Relations (Decree of July 17th, 1944, no. E 52, last modified by decree of December 29th, 1944, no. E 157) is repealed.

There are nine (pure) renumbering provisions in the corpus; six of those are regular renumberings, and the other three are sentences that add an index to a paragraph that did not have one before. These are both standard patterns.

All in all, a large number (330 out of 343, or 96%) of the sentences can be modelled correctly using the rules as presented. However, this does not yet guarantee perfect results, as two more issues are involved. First of all, the reference parser needs to identify all references, and currently, it does not yet recognise references to renumbered articles, which are used in some of the provisions. Secondly, delegations and application provisions include large portions of domain-specific text, which needs to be modelled further.

For this type of sentences, the pattern-based classifier manages about 92% accuracy (the machine-learning classifier achieves 97%). If we assume that there is errors made in classification are independent of those made in modelling, then this suggests that about 88% (96% of 92%) of these sentences can be correctly classified and modelled. As these sentences comprise about 57% of our corpus, this comes down to about 50% of the entire corpus.

5.3 Sentences Dealing With Subject Matter

Sentences dealing with the subject matter of the law come in many shapes and varieties, and though there are some clear signal words, it is difficult to extract the features of a sentence based upon those signal words. However, a sentence contains more information than just signal words. Sentence structure, grammatical conjugation of verbs, etc., can help us determine the different elements from a sentence and their roles. Thus, when processing sentences that deal with the subject matter of the law, we will first parse the sentence, using the Alpino
parser, created by Bouma, van Noord and Malouf (2001). This is a parser for the Dutch language, which has been evaluated and tested using sentences taken from newspaper articles. It assigns a dependency structure to a sentence. These structures are described by Bouma et al.:

Dependency structures make explicit the dependency relations between constituents in a sentence. Each non-terminal node in a dependency structure consists of a head-daughter and a list of non-head daughters, whose dependency relation to the head is marked.

We will then use the output of the parser (a parse tree, which is stored in XML) as the input for our modelling process.

We will first discuss the general approach for creating a model of a normative sentence or definition from a parse tree. After that, several additional considerations are discussed. We will not discuss calculations. Those have a great variety in different patterns that each require a separate model, and which may be nested as well. Discussing those would require too much effort, as we wish to focus on the (more common) normative sentences.

### 5.3.1 Normative Sentences

We see each normative sentence as describing a situation that is allowed or disallowed. We consider the main verb of a sentence as the action that is allowed or disallowed, with the other elements being modifiers or properties of that action. A number of these other elements are labelled according to their semantic role (or thematic relation) in the sentence. The semantic role describes the role a participant plays in a certain situation, which may differ from the role it has in the sentence (such as subject or object). An example taken from Payne (1997):

If, in some real or imagined situation, someone named John purposely hits someone named Bill, then John is the agent and Bill is the patient of the hitting event. Therefore, the semantic role of Bill is the same (patient) in both of the following sentences:

- John hit Bill.
- Bill was hit by John.

In both of the above sentences, John has the semantic role of agent.

At the moment, we distinguish only the agent, patient and recipient of the action. The other elements are considered as generic modifiers. Other researchers have already been working at classifiers to assign semantic roles (Stevens et al, 2006), and we hope to adopt one of those in the future, but for the moment, we use two simple schemes for labelling them, one for active sentences, and one for passive sentences.

In an active sentence, we assume that:

- the subject is the agent of the action;
- the direct object is the patient of the action;
- the indirect object is the recipient of the action.

For example:

*Our Minister issues a warrant to the negligent person.*
The main verb of this sentence is *to issue*, so that is considered the action. Properties of this action are the agent (*Our Minister*), the patient (*a warrant*) and the recipient (*the negligent person*). All these elements are distinguished by the Alpino parser (as subject, direct object and indirect object), allowing us to extract them for our model.

Within Dutch law, this sentence format expresses an obligation, so the action as a whole is classified as an obligation.

<table>
<thead>
<tr>
<th>Obligation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>issue</td>
</tr>
<tr>
<td>Agent</td>
<td>Our Minister</td>
</tr>
<tr>
<td>Patient</td>
<td>warrant</td>
</tr>
<tr>
<td>Recipient</td>
<td>negligent person</td>
</tr>
</tbody>
</table>

The articles (*the, a*) are left out of the model, though they are stored internally, as they are of importance during later integration of the model; *the negligent person* often is a reference to an earlier sentence, whereas *a negligent person* is not.

The example above is an active sentence, but many sentences in Dutch law are phrased in the passive voice, such as this instruction:

**Protection of Antarctica Act, article 33, sub 3, second sentence**

An English translation is added to this report.

A sentence in the passive voice cannot be modelled in the same way as a regular sentence, as the subject of the sentence is not the agent, but the patient, and should be modelled as such. The parse of the sentence gives us an easy way to do this:

![Alpino parse tree](image)

Figure 14: Alpino parse tree (with reduced information) for *An English translation is added to this report* (in Dutch).

---

153 *Wet bescherming Antarctica, artikel 33, derde lid, tweede volzin*  
Bij dit verslag wordt een Engelse vertaling gevoegd.
The verb clause (vc) of the sentence holds the sentence in active voice, with the subject re-cast in the role of object. This re-cast subject is our cue that we are dealing with a passive sentence. By modelling the verb clause instead of the sentence as a whole, we get the correct model, with the correct object, and without the auxiliary verb. So, we will the frame based on the nodes that appear under the verb clause. As there is no subject listed there, there will be no agent in the frame. There is an object present (which refers to the subject of the sentence as a whole), so we will add a patient to the frame.

If the agent is present in the sentence (for example, if the sentence would read *An English translation is added to this report by the organiser*), then this prepositional object is not re-cast in the role of object in the tree. We will have to detect its presence by scanning for signal words like *by*. As this does not always indicate the agent, this will be one of the cases were human validation is necessary. Further detail can be added by splitting of adjectives and relative clauses from the noun they modify. For example, *negligent person* has two properties: being a person and being negligent. Splitting adjectives from nouns is not always desirable; it is preferable to leave multiword expressions intact. *European Union* is not any union that is also European; *Our Minister of Finance* is not any minister that is also ours, and of finance. Instead, these are references to concepts that have been defined elsewhere: the common sense domain, the legal domain or elsewhere in this law. Common multiword expressions are recognised by the Alpino parser; legal domain or law-dependent expressions need be filtered out separately.

Relative clauses are more complex than adjectives, as they contain a complete new sentence. In this case, we repeat the procedure for the main sentence, identifying the main action and all properties of that action. For example:

*Our Minister issues a warrant to the person that neglected his duties.*

This sentence yields a frame like:

<table>
<thead>
<tr>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
</tr>
<tr>
<td>Agent</td>
</tr>
<tr>
<td>Patient</td>
</tr>
<tr>
<td>Recipient</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AgentOf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
</tr>
<tr>
<td>Direct Object</td>
</tr>
</tbody>
</table>

5.3.1.1 Filtering Out Signal Words

The sentences we showed above are examples of normative sentences that do not use signal words; only the desired situation is described, and it is left implicit that this is an obligation. Other sentences in the law use signal words to make the kind of norm explicit, such as:

---

154 In Dutch laws, *Our Minister of Finance* is a reference to the (Dutch) Minister of Finance. No more detailed model is needed, as no derivations need to be made.
The buyer is obliged to pay the price.

This sentence uses *is obliged* to make it clear that this is an obligation. Other examples of signal words are *must, may* and *is allowed*. These sentences require a different approach than the sentences without signal words. If we were to use the same approach, the result would be something like:

<table>
<thead>
<tr>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
</tr>
<tr>
<td>Agent</td>
</tr>
<tr>
<td>Patient</td>
</tr>
</tbody>
</table>

This is not a desirable outcome, as the action that this norm deals with is *pay* rather than *is obliged to pay*. When modelling these sentences, these signal words should not be included in the model of the situation (their meaning is translated into whether the situation is allowed or disallowed). Ideally, after we have categorised the sentence (based on the signal words), we would like to transform the sentence to a sentence without signal words, like:

*The buyer pays the price.*

We could then model that sentence to come to a correct frame. Simply leaving out the signal words may lead to errors, since the role of the other words might need to shift as well. However, the parse of the sentence actually contains this “transformed sentence” that we want to model. This is shown in figure 15.

---

Figure 15: Alpino parse tree (with reduced information) for *The buyer is obliged to pay the price* (in Dutch).

---

155 *Burgelijk Wetboek Boek 7, artikel 26, eerste lid*
De koper is verplicht de prijs te betalen.
Beneath the body node, we find exactly the “sentence” that we are looking for. Alpino assigns this dependency structure to any sentence that follows this pattern. This makes it easy to filter out the signal words by simply focusing on the part of the parse tree that contains the transformed sentence. Similar patterns, such as is allowed to result in a similar tree, with the actual action described beneath the body node.

For sentences that use a modal auxiliary verb, we can also focus on a specific sub-tree. For example, the sentence:

*The buyer must pay the price.*

The parse tree for this sentence is shown in figure 16.

![Figure 16: Alpino parse tree (with reduced information) for The buyer must pay the price (in Dutch).](image)

Again, there is a sub-tree (this time under the vc node) that corresponds to the action that we want to represent in the model (i.e. the buyer pays the price). For other modal auxiliary verbs, the parse tree is similar.

So, depending on the pattern used, we use a different sub-tree as the basis for our frame:

- no pattern used (normative indicative): use the entire tree;
- modal auxiliary verbs (must, may, can): use the sub-tree under the (top-most) vc node;
- a pattern is ... to (is obliged to, is permitted to, is allowed to): use the sub-tree under the (top-most) ti node.

### 5.3.1.2 Lists

Lists are also recognised by the Alpino parser, and can therefore easily be added to our models as the union or intersection of the different list items, depending on the conjunction used. However, though the conjunction and suggests an intersection, it often expresses a union instead. For example:

*Advances and duties are paid in cash.*

In this sentence, it is the union of advances and duties that is meant. Our current approach is to translate and with a union if it appears in a relative clause, and with an intersection otherwise.
5.3.1.3  Negation
It is important to recognise negations such as no or not. For example:

No bodies are interred on a closed cemetery.

This is an obligation, and the patient of this sentence is no bodies. So, this sentence would lead to the following frame.

<table>
<thead>
<tr>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action:</td>
</tr>
<tr>
<td>Patient:</td>
</tr>
<tr>
<td>Condition:</td>
</tr>
</tbody>
</table>

This frame implies that one is obliged to bury something that is not a body on a closed cemetery, which is not what is intended in the law. The actual meaning is that one should not bury bodies on a closed cemetery, represented as:

<table>
<thead>
<tr>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not</td>
</tr>
<tr>
<td>Action:</td>
</tr>
<tr>
<td>Patient:</td>
</tr>
<tr>
<td>Condition:</td>
</tr>
</tbody>
</table>

If we encounter a negation in a (main) sentence, and it is not part of the pattern used to classify the sentence, then this leads to an negation of the frame (using the not operator), and the word not or no is not included as text in the frame. If the negation was part of the pattern used to classify this sentence, its meaning is already represented in the type of the frame, and the negation does not lead to a negation of the frame, nor is it included in the text.

If we encounter such a negation in an auxiliary sentence, then this leads to the negation of the frame representing that auxiliary sentence, not to negation of the frame for the main sentence.

5.3.1.4  Explicit Exceptions
Sometimes, a normative sentence in a Dutch law includes a prefix to denote that it is an exception to some other rule, like:

In exception to article 12, …

Alternatively, some sentence start with a prefix to denote that it is not an exception, like:

Without prejudice to article 12, …

These prefixes differ from other elements in the sentence in the sense that they do not describe the situation that is allowed or obliged, but instead tell us something about how this rule interacts with some other rule. Hence, this element should not be added to the frame describing the rule. Instead, it becomes a relation between this sentence and the referenced structure element.
5.3.1.5 Conditions

Auxiliary sentences that express a condition to the main sentence need to be modelled separately. The different classes of conditions and their representations have not been studied in detail as part of this research, so the descriptions given here are incomplete.

Just as there are sentences dealing with the law and sentences that deal with the subject matter of the law, there are also conditions dealing with the law and conditions that deal with the subject matter of the law.

Like a normative sentence, a condition that deals with the subject matter of the law specifies a situation, and it can be modelled in a similar manner. The result is a condition frame describing a situation, which is linked to the frame representing the main sentence (which usually represents a norm or an application provision). For example:

<table>
<thead>
<tr>
<th>Coin Act 2002, article 8, sub 3&lt;sup&gt;156&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the coins are counterfeits, as judged by the office meant in sub 2, they are returned cut in half or the raw material value is reimbursed.</td>
</tr>
</tbody>
</table>

Here, the main sentence is an obligation: they are returned cut in half or the raw material value is reimbursed. The auxiliary sentence forms a condition, as is represented as a separate frame linked to the obligation (alternatively, it could also be represented as a nested frame, which has the same meaning). The result is:

![Figure 17: Linked condition and obligation frame](image_url)

An important difference with normative sentences is that conditions may specify situations that lie in the past. This means that when creating the frame for a condition, the tense of the

---

<sup>156</sup> Muntwet 2002, artikel 8, derde lid

Indien de munten naar het oordeel van de in het tweede lid bedoelde instantie vals of vervalst zijn, worden ze doorgesneden teruggegeven of wordt de stoffelijke waarde vergoed.
sentence (i.e. present tense, past tense etc.) needs to be preserved. Also, some situations do not involve an action, but rather describe some property or attribute of involved items, such as the example given above. In this case, rather than modelling this as an action based on the verb to be, it may be more desirable to translate such conditions using subclassOf relations.

The conditions that deal with the law refer to the applying or applicability of other parts of the law, such as:

**Prosecution Code, article 366a, sub 1**
In case article 14a or 77x of the Penal Code has been applied, …

Or:

**Work and Income to Working Capacity Act, article 64, sub 5**
If sub 4 applies, …

Like application provisions, such conditions should be modelled using dedicated frames rather than the frames used for generic situations in the application domain.

**5.3.2 Definitions and Deeming Provisions**
Definitions and deeming provisions attach a meaning to a specific concept. At top level, a definition contains three elements: the definiendum and the definiens, and, optionally, a scope declaration stating for which sources of law this definition applies. For example:

**Medication Act, article 1, introduction and item c**
In this law and stipulations based upon it, it is understood by immunological drug: a vaccine, toxin, serum or allergen.

This definition has the scope this law and stipulations based upon it. The definiendum is immunological drug and the definiens is a vaccine, toxin, serum or allergen. The parse tree for the definiens is shown in figure 18.

![Figure 18: Alpino parse tree (with reduced information) for a vaccine, toxin, serum or allergen (in Dutch).](image)

---

157 Wetboek van strafvordering, artikel 366a, eerste lid
In geval artikel 14a of 77x van het Wetboek van Strafrecht is toegepast, …

158 Wet arbeid en inkomen naar vermogen, artikel 64, vijfde lid
Indien het vierde lid van toepassing is, …

159 Geneesmiddelenwet, artikel 1, aanhef en onderdeel c
In deze wet en de daarop berustende bepalingen wordt verstaan onder immunologisch geneesmiddel: een vaccin, toxine, serum of allergen;
Like the sentences presented in section 2, these top elements can easily be extracted by means of the pattern used to classify the sentence (in this case it is understood by) and some additional features. The scope, if present, will follow the word in (and end at the text it is understood by). The definiens will follow the word by and end at the colon, and the definiendum will follow the colon\(^ {160} \). Thus, we can easily extract a top level frame:

<table>
<thead>
<tr>
<th>Definition</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>This law and stipulations based upon it</td>
</tr>
<tr>
<td>Definiendum</td>
<td>immunological drug</td>
</tr>
<tr>
<td>Definiens</td>
<td>a vaccine, toxin, serum or allergen</td>
</tr>
</tbody>
</table>

However, modeling the definiens in this way is unsatisfactory, as it gives insufficient detail to use this model for practical purposes. To create a more useful model, we need to split up the definiens. To do so, we use the same methods used for concepts in normative sentences. This requires a parse tree; we can either parse the entire sentence or just the definiens.

5.4 Example: Converting Frames to OWL

The frames presented in this chapter do not form an executable specification. However, they contain sufficient information to be converted to a different notation that can be executed. In this section, we will show how the frame we have created for normative sentences can be converted into OWL expressions, which can be evaluated using the HARNESS system (van de Ven et al., 2008).

In HARNESS, a norm is represented in terms of the LKIF Core ontology (Hoekstra et al., 2007) as a deontic qualification of a generic case (Valente, 1995). Such a generic case is a conjunction of conditions that together form a description of the situation expressed by the norm. It is defined as a set of conditions in conjunctive normal form. An individual case is a set of grounded propositions that describe a certain state of affairs.

The norm itself is qualified using the deontic notions Permission, Obligation and Prohibition, which have been defined in the LKIF Core ontology. A norm is a qualification, which normatively qualifies something (as opposed to non-normative qualifications)\(^ {161} \):

Class: Norm
SubClassOf: Qualification and qualifies some Normatively_Qualified

A permission is a norm that allows some situation:

Class: Permission
SubClassOf: Norm
EquivalentTo: allows some Allowed and allows only Allowed

\(^{160}\) There are a number of different patterns, but for each pattern, a similar set of features can be identified to extract these three elements.

\(^{161}\) The OWL statements have been written in the Manchester notation (see Horridge et al., 2006).
An obligation is something that allows something (i.e. that what is obliged) and also disallows something (i.e. the situation in which that what is obliged is not performed).

Class: Obligation
SubClassOf:
   Permission
EquivalentTo:
   allows some Obliged and disallows some Disallowed
   and allows only Allowed and disallows only Disallowed

A prohibition is equivalent to an obligation, as a prohibition to do something is the same as the obligation not to do it:

Class: Prohibition
EquivalentTo:
   Obligation

In an example, van de Ven et al. (2008) describe how a set of library rules can be represented using HARNESS. These rules apply to the checking out of books. The example starts with specifying a default situation. In this case, the default situation is disallowed: students are not allowed to check out books, except when the norms state something else. This default norm and generic case apply to all situations where a book is checked out.

Class: Default_CG
SubClassOf:
   Generic_Case and disallowed_by some {defaultnorm}
EquivalentTo:
   checks_out some Library_Book

Class: Default_Norm
SubClassOf:
   Prohibition and disallows only Default_GC
EquivalentTo:
   (defaultnorm)

These statements say that the default generic case is a generic case, which is disallowed by the default norm. The situations to which it applies are those situations in which a library book is checked out. The default norm is a prohibition which disallows the generic case.

Other norms are represented in a similar manner as the default norm. The example continues with article 1a of the regulations, which states: Students registered at this university are allowed to check out a book from this library. This is represented as the Article 1a Generic Case, which is a registered student checking out a library book. This situation is permitted by article 1a.

Class: Art1a_CG
SubClassOf:
   Generic_Case and allowed_by some {art1a}
EquivalentTo:
   Registered_Student and checks_out some Library_Book
These representations can now be used to assist in the task of legal assessment. This is done by specifying an individual case, such as this:


Using an OWL classifier, this individual case can be classified as belonging to any of the generic cases. The case of Amy, above, matches both the default generic case (as there are books being checked out) as the Article 1a generic case (as it is a registered student checking out books). Based on the norms allowed to these generic cases, it is possible to determine whether this individual case is allowed. This situation is allowed by Article 1a (as it matches the Article 1a generic case), but disallowed by the default norm (as it matches the default generic case). However, any explicit norm is considered an exception to the default norm, and has precedence, leading to the conclusion that this individual case is allowed\(^{162}\).

So, in order to add a normative sentence to this system, we need to specify the generic case, as a set of conditions. We continue with the example from section 5.3.1:

*Our Minister issues a warrant to the negligent person.*

After parsing, this sentence yielded the following frame:

<table>
<thead>
<tr>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
</tr>
<tr>
<td>Agent</td>
</tr>
<tr>
<td>Patient</td>
</tr>
<tr>
<td>Recipient</td>
</tr>
</tbody>
</table>

We can describe the generic case using the elements from this frame. The generic case is an action `issue` with agent *Our Minister*, patient `warrant` and recipient `negligent person`:

Class: Generic_Case

EquivalentTo:

`issue and agent some Our_Minister and patient some warrant and recipient some negligent_person`

This generic case is allowed by the article, and its negation is disallowed by the article, which leads to the following complete statement in HARNESS:

\(^{162}\) This exception relation is derived outside OWL.
The example *The buyer is obliged to pay the price* results in the frame:

<table>
<thead>
<tr>
<th>Obligation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Pay</td>
</tr>
<tr>
<td>Agent</td>
<td>buyer</td>
</tr>
<tr>
<td>Recipient</td>
<td>Price</td>
</tr>
</tbody>
</table>

This translates to the following HARNESS statement:

Class: Generic_Case_P_Example2
SubclassOf:
  | Generic_Case and allowed_by some {example2}
EquivalentTo:
  | pay and agent some buyer
  | and patient some price

Class: Generic_Case_F_Example2
SubclassOf:
  | Generic_Case and disallowed_by some {example2}
EquivalentTo:
  | not Generic_Case_P_Example2

Class: Example2_Obligation
SubclassOf:
  | Obligation and allows only Generic_Case_P_Example2
  | and disallows only Generic_Case_F_Example2
EquivalentTo:
  | {example2}

A sentence denoting a permission, such as *A holding cell may be equipped with an observation camera* is treated in a similar manner. The frame for this sentence is:
The translation for this frame again follows the same approach again, with the difference that this sentence only includes a permission, and not a prohibition:

Class: Generic_Case_P_Example3
SubClassOf:
  Generic_Case and allowed_by some {example3}
EquivalentTo:
  equip and patient some isolation_cell
  and ppobject some observation_camera

Class: Example3_Permission
SubClassOf:
  Permission and allows only Generic_Case_P_Example3
EquivalentTo:
  {example3}

As HARNESS is modelled in OWL, any definitions can be easily modelled as well, by adding an equivalent relation between the definiens and the definiendum. For example, the definition given in section 5.3.2:

In this law and stipulations based upon it, it is understood by immunological drug: a vaccine, toxin, serum or allergen.

Becomes the following OWL statement:

Class: immunological_drug
EquivalentTo:
  vaccine or toxin or serum or allergen

This OWL statement does not contain a translation for the scope declaration In this law and stipulations based upon it. Within HARNESS, no method has been defined for dealing with scope declarations, and OWL has no obvious mechanisms to deal with this.

One way to deal with this is to simply define each term with the same label separately – a common_sense_immunological_drug, a medicine_law_immunological_drug, etc. This approach means that each partial model has to be integrated by hand with the correct definition; a human expert will need to determine which definition to use, as the knowledge of the scope of the definition is not modelled. When a new definition is added – suppose the Medicine Law is amended to include a broadened definition of immunological drug that applies to chapter 7 only, thus introducing the concept medicine_law_chapter7_immunological_drug – then all rules that use this definition must be updated by hand.

163 “observation camera” does not fill one of the semantic roles that we currently assign, and hence is described with the more primitive label “Prepositional Object”.

<table>
<thead>
<tr>
<th>Permission</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>equip</td>
</tr>
<tr>
<td>Patient</td>
<td>holding cell</td>
</tr>
<tr>
<td>Prepositional Object</td>
<td>observation camera</td>
</tr>
</tbody>
</table>
A more promising method might be to develop an approach similar to that of Klarman, Hoekstra and Bron (2008) who describe a method for concepts that have multiple versions in time.

5.5 Experiences

At this moment, we do not have a fully automated process to create the models, and have not yet tested this method on a large body of sentences. Instead, random sentences have been selected, parsed using Alpino and then fed into our modeller.

There is a clear difference between the computer generated models and those created by a human expert with regard to the granularity of the model. Our method will create models with model elements that represent one word from the original sentence, whereas a human expert is more likely to include some sentence fragments as a whole. For example, one Dutch law defines an alcoholic drink as the drink that, at a temperature of twenty degrees Celsius, consists of alcohol for fifteen or more volume per cents, with the exception of wine. Our algorithm will split this sentence into many elements, whereas most human modellers will leave the first subordinate sentence intact and add it to the model as a single attribute (most likely abbreviated to alcohol by volume). A more detailed model seems not necessarily wrong, but quite possibly over-the-top and inconvenient for many applications.

The method assigns rather broad categorisations to each object (it is either a direct, indirect or prepositional object), but does not yet assign a legal meaning to such an object. It may be a third party involved or the instrument. Perhaps this is not an obstacle; users dealing with a system based on such models are likely to recognise the roles from the context and language used, whereas a computer does not need this information for the derivations we currently want to make. For future projects, though, the information may be required, and some way to automatically recognise it is desired. Here, the addition of a more advanced semantic role labelling tool can be valuable.

For the modelling of norms, we have been focussing on the sentences that represent an obligation, duty or right. For those sentences, the method seems adequate. However, for other types of sentences, such as delegation, we have not come to an acceptable approach yet. Dealing with these sentences will require first of all that we recognise them. Currently, our classifier distinguishes only between obligation/prohibition and right/permission. Several of the patterns used clearly indicate delegations, but we have not yet established whether these patterns cover all delegations in Dutch laws.

A problem with regard to the parses made by Alpino is that most often, the correct parse is not the one preferred by Alpino, but second, third or fourth. If we make several suggestions (each suggestion based on a parse by Alpino), this means that it will often not be the first suggestion that is correct, which means more effort is needed by a human expert who is verifying the models. This may be caused by the fact that Alpino has been trained on newspaper articles rather than legal texts. We expect that by recalibrating the disambiguation on a written legal corpus, and perhaps by expanding the lexicon used by Alpino, this problems will disappear.
5.6 Integrating models

After a model has been generated for each individual sentence, these individual models need to be integrated. Though some provisions may stand on their own, many rules are somehow based on earlier provisions.

This integration step starts with the models of the individual provisions (sentences) that have been generated in the previous step, as described in sections 5.1 to 5.3.

There are several ways in which provisions can be connected. First of all, a provision can relate to a different provision (or larger element) as a whole. This is the case for modifications, application provisions, etc. as well as for certain conditions.

Provisions can also be linked through the concepts they use. The provisions that deal with the subject matter of the law will contain texts (usually a noun phrase or verb phrase) that refer to certain concepts. When creating a model for the individual sentences, each model will contain a class for that concept. For example, in a law about child subsidy, we may have child appear in article 5, article 6, first sentence, article 6, second sentence and article 7. When modelling these sentences separately, we also get separate classes. So, we end up with classes child according to article 5, child according to article 6 first sentence, child according to article 7 second sentence and child according to article 7. Obviously, these classes are related. There are two possibilities for such connections:

- A provision contains a class that has been defined in another provision. In this case, the class in the provision is a subclass of the class in the definition.
- A provision refers to a class introduced in another provision. In this case, the two classes are equivalent.

In the next sections, the different connections are discussed, and the manner in which they can be detected.
5.6.1 Linking Provisions that are Related to Other Provisions as a Whole

There are a number of operations for which a provision needs to relate to a different provision as a whole. These operations have been described in chapter 4: modification provisions, application provisions, setting the enactment date and setting the short title. As we have identified the references in the text, have classified the sentences and have determined the role of the references in these sentences, they have effectively been integrated with the other text.

Similarly, there are a number of auxiliary sentences that relate to other provisions as a whole: auxiliary sentence setting the scope for definitions and deeming provisions, auxiliary sentences setting explicit exceptions and conditions based on the application of other provisions. Like the provisions mentioned above, these auxiliary sentences have already been linked through their target by identifying the reference and determining the role of that reference.

5.6.2 Linking Concepts to Their Definitions

Linking the concepts that are used with their definitions is a fairly straightforward step. Basically, it is the direct matching of the word, noun phrase or verb phrase representing the concept with the word, noun phrase or verb phrase that forms the definiendum of the definition. There are three minor issues involved in this matching.

First of all, the phrase as it appears in a sentence may differ from the exact phrase that was used in the definition. A noun may be plural instead of singular, a verb may be in a different tense, etc. This is a problem that is very common in natural language processing. It is usually solved using stemming or lemmatising. A stemmer attempts to find the base form (or lemma) of the word. A lemmatiser does the same, but takes into account the context in which the word appears. By applying a stemmer or lemmatiser on the words, we can compare the lemmas rather than the actual words as they appear in the sentences, thus circumventing the problem posed by the many inflections. The Alpino parser, like most parsers, includes such a lemmatiser. This means that after parsing, no additional step is needed to obtain this information.

Secondly, the appropriate definition may be in a different document. This will be most common with lower regulations (i.e. not laws). Many definitions in a law have as scope *this law and the stipulations based upon it*, meaning that the definition does not only apply to the law itself, but to any legislation that is promulgated under the authority of that law as well. Thus, when integrating these norms with their definitions, then definitions from appropriate higher legislation should be included in the search.

Finally, there may be several definitions that define the word, each with a different scope. In this case, the location of the phrase should be compared to the scope of the definitions. Obviously, if the location is outside of the scope of a definition, it should not be linked to that definition. If there are multiple definitions that have the location in scope, then the definition that has the narrowest scope should be chosen. For example, the Labour and Care Act contains the following definition for *employee*. 


Labour and Care Act, article 1:1, introduction and items a and b

Unless it is determined differently, for the application of this law is understood by:

a. employer: he who has another perform labour under an employment contract under civil law or an appointment under public law;
b. employee: the other, meant in item a.

The scope of this definition is the entire law. However, there is a different definition for section 1 of division 2 of chapter 3:

Labour and Care Act, article 3:6, sub 1, introduction and item a

For the application of this section is understood by employee: the employee meant in article 1:1, item b, with the exception of him who, based on Division 1, Section 2, of the Sickness Benefits Act, is not an employee under that act.

This second definition forms an exception to the scope of the first definition; the first definition applies throughout the law, with the exception of chapter 3, division 2, section 1 (which is the section this section in item a refers to), whereas the second definition only applies in that specific section. So, sentences within that section should link to the second definition, whereas sentences elsewhere in the law link to the first definition.

5.6.3 Linking Concepts That Represent the Same Class

Next to linking concepts to their definitions, concepts may also be linked between two (non-defining) sentences, indicating that those two sentences should not be applied independently, but that the second sentence applies to the same situation as the first sentence. The same may occur within a single sentence (usually between an auxiliary sentence and the main sentence).

The fact that a text represents the same class as an earlier text can be indicated in a number of different ways. Firstly, the concept may be explicitly linked by means of a reference, such as a request as meant in sub 7. As references have been classified earlier, these links are easily identified. Secondly, a demonstrative like this or that is used. For example, in the next sentence, that person refers back to a natural person:

Mining Act, article 21, sub 5, introduction and item a

The license expires automatically: a. if the holder is a natural person, from the day after that person dies.

---

164 Wet arbeid en zorg, artikel 1:1, aanhef en onderdeel a
Tenzij anders is bepaald, wordt voor de toepassing van deze wet verstaan onder werkgever:
a. degene die een ander krachtens arbeidsovereenkomst naar burgerlijk recht of publiekrechtelijke aanstelling arbeid laat verrichten;
b. werknemer: de ander, bedoeld in onderdeel a;

165 Wet arbeid en zorg, artikel 3:6, eerste lid, aanhef en onderdeel a
Voor de toepassing van deze paragraaf wordt verstaan onder werknemer: de werknemer, bedoeld in artikel 1:1, onderdeel b, met uitzondering van degene die op grond van de Eerste Afdeling, Paragraaf 2, van de Ziektewet geen werknemer in de zin van die wet is;

166 In this specific case, the first definition is not completely out of effect, as part of it is re-used through a reference in the second definition.

167 Mijnbouwwet, artikel 21, vijfde lid, aanhef en onderdeel a
De vergunning vervalt van rechtswege: a. als de houder een natuurlijke persoon is, met ingang van de dag na die waarop die persoon is overleden;
Often, no demonstrative is used, and the difference is shown by the fact that the article *the* is used instead of *a*, indicating that the text deals with a class that has been introduced earlier. For example:

### Income Tax Act, article 4.50, sub 1 and 2

1. The inspector sets the amount of a loss from substantial interest by means of a decision, which may be objected to.
2. The inspector makes the decision meant at the same time that he sets the assessment for the year in which the loss arose.

In these sentences, *the decision meant* refers back to *a decision* (in this case, the reference is made more explicit by means of the addition of the word *meant*), and *the loss* refers back to *a loss*.

So, if an indefinite article (i.e. *a* or *an*) is used, the text refers to the class referred to by the noun, whereas if a definite article (i.e. *the*) is used, the text may refer to a specific subclass. This specific subclass has been introduced before, and this occurrence of that concept should be linked to the earlier occurrence.

However, it is possible that the specific subclass has not explicitly been introduced before, but has been introduced through some relation, such as *the owner of a steam apparatus*. Here, *the owner* does not refer back to *an owner* that has been introduced before, but to *a steam apparatus*.

In many cases, though, such a relation is left implicit. For example:

### Succession Act 1956, article 2, sub 3

In the case of a gift by a legal person, the place of residence of the giver is considered his home.

In this sentence, *the giver* does not refer back to a giver introduced in an earlier sentence. Instead, *a gift* implies that there is a giver (and a recipient). Thus, by introducing *a gift*, the corresponding giver and recipient are also introduced. This type of integration can only be made if this implicit knowledge is added, and cannot be made automatically based on the legal text alone.

A related situation arises if the relation is made explicit in the definition. For example:

### Successiewet 1956, artikel 2, derde lid

In geval van schenking door een rechtspersoon wordt de plaats, waar de schenker is gevestigd, als zijn woonplaats aangemerkt.

---

168 *Wet Inkomstenbelasting 2001, artikel 4.50, eerste en tweede lid*

1. De inspecteur stelt het bedrag van een verlies uit aanmerkelijk belang vast bij voor bezwaar vatbare beschikking.
2. De inspecteur geeft de bedoelde beschikking gelijktijdig met het vaststellen van de aanslag over het jaar waarin het verlies is ontstaan.

169 *Successiewet 1956, artikel 2, derde lid*

In geval van schenking door een rechtspersoon wordt de plaats, waar de schenker is gevestigd, als zijn woonplaats aangemerkt.
Protection of Antarctica Act, article 1, introduction and item e

In this law and stipulations based upon it, it is understood by organiser: the natural or legal person who organises an activity from the Netherlands.

In this definition, an organiser is linked to an activity. Thus, if a sentence introduces an organiser, it also introduces an activity.

5.6.4 Example

As an example of the different links that may occur, we will consider the following obligation:

Protection of Antarctica Act, article 33, sub 1 (partial)

An organiser draws up a report within six weeks after ending the activity.

<table>
<thead>
<tr>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
</tr>
<tr>
<td>Agent</td>
</tr>
<tr>
<td>Patient</td>
</tr>
<tr>
<td>Condition</td>
</tr>
</tbody>
</table>

This obligation refers to an organiser, as it has been defined in the definition given in the previous section:

Protection of Antarctica Act, article 1, introduction and item e

In this law and stipulations based upon it, it is understood by organiser: the natural or legal person who organises an activity from the Netherlands.

The frame for this definition is:

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definiendum</td>
</tr>
<tr>
<td>Definiens</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>AgentOf</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Scope</td>
</tr>
</tbody>
</table>

As said, the organiser from the obligation needs to be linked to the organiser that is being defined. Moreover, the definition introduces a link between the organiser and the activity. The activity mentioned in the obligation is the activity linked to the organiser through this definition, and should also be linked to the definition, yielding:

170 Wet bescherming Antarctica, artikel 1, aanhef en onderdeel a

In deze wet en de daarop berustende bepalingen wordt verstaan onder organisator: de natuurlijke of rechtspersoon die vanuit Nederland een activiteit organiseert.

171 Wet bescherming Antarctica, artikel 33, eerste lid (ingekort)

Een organisator stelt binnen zes weken na de beëindiging van de activiteit een verslag op.
The next sentence refers back to the obligation given above:

**Protection of Antarctica Act, article 33, sub 2 (partial)**

The report contains a detailed description of the performed activity.

The model of this sentence is:

<table>
<thead>
<tr>
<th>Obligation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
</tr>
<tr>
<td>Agent</td>
</tr>
<tr>
<td>Patient</td>
</tr>
</tbody>
</table>

In this sentence, *the report* refers to the report mentioned in the previous obligation, and the performed activity refers back to the activity mentioned in the previous obligation. Adding this frame with those links gives:

---

172 Wet bescherming Antarctica, artikel 33, tweede lid (ingekort)

Het verslag geeft een nauwkeurige omschrijving van de uitgevoerde activiteit.
There is a lot of work still to be done. A large part of that lies in the actual implementation of the separate sentence models, but within the models for norms, there is also room for improvement. Currently, within the models, all attributes are generated as Boolean attributes. For example, an adjective like “blue” results in a condition testing whether something is blue or not blue. Human experts might create an attribute “colour” and test whether the colour is blue. As our proposed method looks only at the words and their role in the sentence, and not at their meaning, it is not able to distinguish such things. It may be possible to create lists of words for which such an intermediate concept (such as “colour”) is appropriate.

Most existing formal languages do not have support for all the different constructs that occur in a legal text. Most notably, there is no isomorphic translation for application provisions, deeming provisions and scope statements for definitions. Though such statements can be represented in most languages, there is usually some loss of information. For example, a deeming provision will be represented in the same way as a definition. When evaluating a case based upon such a representation, the outcome will be correct, but the knowledge that a deeming provision was involved can be useful in order to explain the reasoning for that outcome to the user. As such, it is desirable to have a formal, executable modelling language capable of supporting such constructs. Within the POWER project, it has been attempted to do so by adding custom constructs to UML, but these models were not executable.

5.7 Conclusion

There is a lot of work still to be done. A large part of that lies in the actual implementation of the separate sentence models, but within the models for norms, there is also room for improvement. Currently, within the models, all attributes are generated as Boolean attributes. For example, an adjective like “blue” results in a condition testing whether something is blue or not blue. Human experts might create an attribute “colour” and test whether the colour is blue. As our proposed method looks only at the words and their role in the sentence, and not at their meaning, it is not able to distinguish such things. It may be possible to create lists of words for which such an intermediate concept (such as “colour”) is appropriate.

Most existing formal languages do not have support for all the different constructs that occur in a legal text. Most notably, there is no isomorphic translation for application provisions, deeming provisions and scope statements for definitions. Though such statements can be represented in most languages, there is usually some loss of information. For example, a deeming provision will be represented in the same way as a definition. When evaluating a case based upon such a representation, the outcome will be correct, but the knowledge that a deeming provision was involved can be useful in order to explain the reasoning for that outcome to the user. As such, it is desirable to have a formal, executable modelling language capable of supporting such constructs. Within the POWER project, it has been attempted to do so by adding custom constructs to UML, but these models were not executable.
6 Applications
In this chapter, we will discuss two applications that have been created that use the research that has been presented in the previous chapters. The Semantic Network for the Dutch Tax and Customs Administration is a content management system intended to deal with large amounts of sources of law which have been provided by various different publishers. It uses parsers to extract useful metadata. The second application is MetaVex, an editor for legal sources. It uses language patterns for templates and to generate natural language texts from models.

6.1 Semantic Network for the Dutch Tax and Customs Administration
The Dutch Tax and Customs Administration (DTCA), like many other organisations, deals with a multitude of (digital) legal data. Part of this data is published by themselves, but other data is purchased from publishers. They run into a number of different problems with these collections. First of all, each publisher uses its own format for its collection, and the DTCA adds to that collection using that format. Should the DTCA wish to switch to a different publisher, they will need to translate their corporate data to the format of that new publisher. Secondly, documents from the same collection have marked-up interrelations, but they are not related to documents from other collections. These relations are added by the DTCA. This is a time and effort consuming task, which needs to be repeated every time one of the collections is updated. Finally, there is a lot of redundancy in the total collection. Most publishers will include the relevant legislation in their collection, which means it turns up several times in the total collection. This can be confusing for the employees of the DTCA, but also means that the DTCA may be able to save on costs if they can avoid buying this duplicate data.

To counter these problems, we have built a prototype content management system (CMS) (see Winkels, Boer, de Maat, van Engers, Breebaart, & Melger, 2005) based on MetaLex XML. In this system, metadata about the documents in the DTCA collection is stored outside these databases using RDF. This metadata is stored at the expression level instead of the item level (see chapter 3).

For each expression in the collection, the CMS contains a RDF representation of the expression and its structure. For example, if the CMS contains the Income Tax Law as it was modified on January 1st, 2011, then the RDF graph contains a node for that version of the Income Tax Law, nodes for each chapter of that version of the Income Tax Law (which are linked to the node representing the entire law), nodes for each article in that version of the law (which are linked to the node representing the chapter they are a part of), etc.

Metadata is linked to these RDF nodes. In particular, references between two documents are stored as a reference between two expressions. The documents in the collection are each linked to the expression they exemplify. Thus, any document is automatically linked to all the metadata that is available for the expression (instead of just the metadata that is part of the collection it is in). If a new document is added to the total collection, it merely needs to be linked to the appropriate node in the RDF graph, which will link it to all relevant metadata.
In order to create and maintain this network a number of parsers were created, which extracted the relevant metadata.

First of all, the identity of each document had to be established. To do so, we need to know the type of document (legislation, part of legislation, case law or legal commentary), the title or name of the document and the version of the document. There are two sources for this information: the actual text of the document and any metadata added by the publisher. Obviously, simply extracting the metadata is the easiest and most reliable method. This information is structured and has already been checked by human experts at the publisher. All the documents used for the prototype included such metadata.

If the metadata is lacking, we can establish the identity using other information. If the publisher has used meaningful filenames or addresses, we may be able to identify the document by means of this filename. If not, we can study the document text itself to see if it contains a title. As some documents in a collection represent only a part of a law (e.g. a single article), this is not always the case. Finally, we may be able to derive the information from the text other documents use to refer to this document (using links provided by the publisher.)

The next step is to find the reference in the texts. Within each collection, documents will often have links to other documents in the collection, but will lack mark-up for references to documents that are not part of the collection. A predecessor of the parser described in chapter 3 was used to detect those references.

In this manner, we can provide the DTCA with a knowledge base of the field they work in that is both format-independent and publisher independent. Other applications used by the DTCA, either for themselves or their clients, can connect to this database with a static link to the correct concept in the network.

Other research, such as the work of Urbani, Kotoulas, Maassen, van Harmelen and Bal (2010), who have worked with a dataset of 100 billion triples (which are the links between RDF nodes) shows that this solution will scale up to the many sources it will have to deal with in the future.

### 6.2 MetaVex

An important part of the work done by ministries, national parliaments, city councils, etc. consists of the drafting of new legislation and of amendments to existing legislation. Much of this work is done by means of general purpose word-processing software. Such generic software, however, does not include support for the legislative process, and is not well-integrated with the systems that are used to store, publish and search through legal documents.

At the Leibniz Center for Law, an editor for legislative drafters has been created with which they can create laws in CEN/MetaLex XML format, which provides better integration with other systems than generic document formats do (see van de Ven, Hoekstra, Winkels, de Maat & Kollar, 2008). This editor has a look and feel similar to normal word processors, and
document editing is done in a WYSIWYG\textsuperscript{173} interface. Legislative drafters do not need any knowledge of the underlying XML in order to create new documents. The editor ensures that the output is always a valid XML document.

By means of templates, the editor attempts to speed up the process of generating document structure and content, and supports the user in complying with the official guidelines. These templates are based on the official guidelines and on other patterns described in chapter 4.

MetaVex allows drafters to mark-up references to other sources, but also shows them all the references that point to the document they are editing, so that they can more easily determine the impact of their changes.

More elaborate support is available for the creation of amending documents. This is an important topic, as the majority of laws that are produced are in fact amending laws (i.e. laws that change an existing law rather than introducing completely new legislation). In addition, each law that passes through Parliament will give rise to a number of amendments proposed by members of Parliament.

In many ways, amending documents are more challenging to handle than regular documents. They contain descriptions of the changes to be made rather than the complete new text. Because of this, the actual impact of the new text can sometimes be difficult to perceive. To help the reader with such documents, there are applications that show the new text next to the original text, with any modifications marked using highlights and strikethroughs. The European Parliament makes use of this method, and the Italian Senate has also adopted a tool, called TafWeb, that shows this comparison (see Bacci, Spinosa, Marchetti & Battistoni, 2009).

MetaVex is intended to help the drafters of an amending document in a similar way. It does so by allowing a legislative drafter to make changes to a copy of the original act. The system then compares this modified copy with the original document, detects the changes and represents them in natural language (see de Maat, van de Ven, Winkels & van Engers, 2009).

This approach is similar to that of the Themis system used in Tasmania (ArnoldzMoore, 1997). In that system, the original acts are stored in SGML. When a user wishes to create an amending law, he checks the act out. At this point, the act is converted to a MS Word document. The user then uses MS Word to edit this document. The Themis system includes dedicated MS Word macros, templates and toolbars to assist the user in editing the document. When the user has finished making modifications, he saves the changes in a Rich Text Format file. In this file, the changes are marked using underlines for insertions and strikethroughs for deletions. Based on this file, two new SGML files are generated: one representing the original act, and one representing the modified act. These two SGML files are then compared in order to compile a list of changes. Compared to the Themis system, MetaVex has the advantage that it is a native XML editor, so we do not need to convert the XML file to a different format and then back to XML again.

As mentioned above, the modified act is compared to the original document. This is done in two steps. First of all, the structure of the modified document is compared to the structure of

\textsuperscript{173} What you see is what you get.
the original document. This is made possible through the fact that the structure is marked-up
using MetaLex. This comparison detects three types of modifications:

1. Deletion of an entire element: if an element with a specific ID does exist in the
original document, but not in the modified document, it has been deleted.
2. Insertion of an entire element: if an element with a specific ID does exist in the
modified document, but not in the original document, it has been inserted as a new
element\textsuperscript{174}.
3. Relocation of an entire element: if an element with a specific ID occurs in a different
position in the modified document than in the original document, it has been moved.
Though we can detect such a change, this is not an operation that is applied in Dutch
legislation, as is evident from the fact that there is no sentence type describing a
relocation.

After this structural comparison has been made, all the elements in the modified document
that have not been newly inserted are compared with the original text to find any textual
changes, using a \textit{diff} function\textsuperscript{175}. This process will also detect renumbering, as this is a
modification of the text of the index (which is tagged separately in MetaLex).

Some actions by the user can confuse this system and cause it to come to the wrong
conclusion. For example, a user could copy an article and delete the original. In effect, this is a
move of the article, but because the copy has a different ID than the original article, MetaVex
will detect a deletion and an insertion. However, for most common actions, this procedure
adequately detects the changes.

After all changes have been detected, metadata describing these changes are stored in an XML
file. Now, an amending text can be generated, which describes the changes in natural language.
The user can later add additional texts, such as a motivation or provisions that do not modify
existing legislation. For each insertion, deletion, renumbering, etc. a single sentence is
generated describing that modification. These changes are described using the sentence
formats that are described in section 4.4.3 as templates. Using the patterns described there,
any change can be expressed in natural language by taking an appropriate pattern and fill it out
in the right way. For example, the template for a textual replacement is:

\texttt{In [target], «\textit{old_text}» is replaced by: \textit{new_text}.}

The texts in square brackets indicate fields that have to be filled out. So, if a user has modified
article 7 (target), replacing the words \texttt{director} (old text) with \texttt{board of directors} (new text), then this
information has been detected using the steps described above. Filling out the fields in the
template results in:

\texttt{In article 7, «director» is replaced by: board of directors.}

\textsuperscript{174} MetaVex automatically assigns a new ID to each new element.
\textsuperscript{175} To limit the number of diff operations, MetaVex flags all elements that have been edited by the user in some
way. When searching for changes, MetaVex only checks the elements that have been flagged in this way.
Effectively, this is the reverse of the process we used in section 5.1.4 to extract the information for a frame from a sentence.

The location of the modification is derived from the MetaLex document, in which the category and index of each element is marked. For example, if there is a change in a list item with index “u”, which is part of a paragraph with index “1”, which in turn is part of an article with index “21”, then the entire natural language description would become *artikel 21, eerste lid, onderdeel u* (lit.: article 21, first paragraph, item u). Basically, this is simply the concatenation of the different category labels and indexes, though some modifications are needed. As mentioned in section 2.4, in Dutch laws, the paragraphs and list items do not have a full heading, but are only numbered. This means that the category labels have to be added. Furthermore, in a reference to a paragraph, the index number is replaced by an ordinal number (e.g. *first* instead of *1*), so this has to be converted.

The other necessary texts to fill out the “blanks” in the sentences are simply those texts that have been identified when detecting the changes (i.e. the texts that are deleted or inserted).

This approach works, but does sometimes yield rather ugly constructions, as there is a complete sentence generated for each change. Sometimes, it is more desirable to group these changes, which requires a more sophisticated approach. First of all, generating individual sentence may result in something like:

<table>
<thead>
<tr>
<th>Article I</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Article 12, sub 1, … is replaced by ….</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Article II</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Article 12, sub 2, … is replaced by ….</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Article III</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article 12, sub 3 is repealed.</td>
</tr>
</tbody>
</table>

Though these sentences correctly describe the modifications, it is not how these changes are usually represented in an amending law. If there are several changes in the same part of the document, they are usually grouped, and preceded by a scope declaration as described in section 4.4.3. Any further references are then made relative to that scope. So, in the case of the example above, the preferred way of presenting the modifications would be:

<table>
<thead>
<tr>
<th>Article I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article 12 is modified as follows:</td>
</tr>
<tr>
<td>A. In sub 1, … is replaced by …</td>
</tr>
<tr>
<td>B. In sub 2, … is replaced by ….</td>
</tr>
<tr>
<td>C. Sub 3 is repealed.</td>
</tr>
</tbody>
</table>
To achieve this result, the changes must be grouped according to location before the text is generated\textsuperscript{176}. If a location contains more than one change, a grouping construct as shown above must be used.

Another correct, but ugly representation occurs when the same modification is repeated several times at different locations, such as:

<table>
<thead>
<tr>
<th>Article I</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. In Article 1, «the director» is replaced by: the board of directors.</td>
</tr>
<tr>
<td>B. In Article 3, «the director» is replaced by: the board of directors.</td>
</tr>
<tr>
<td>C. In Article 4, «the director» is replaced by: the board of directors.</td>
</tr>
</tbody>
</table>

This will happen when a reference that is used multiple times needs to be updated, or when a concept needs to be replaced by some other concept. In these cases, the modifications need to be grouped, too, in a sentence like\textsuperscript{177}:

| Chapter 7 is modified as follows: |
| All occurrences of «the director» are replaced by: the board of directors. |

In order to perform this grouping, we need to compare the texts of each (similar) modification, in order to determine whether a modification is repeated multiple times. If we wish to come close to the sentence a human expert would produce, we should also group the indexes\textsuperscript{178}. For example, if articles 112, 113, 114, 115 and 116 are affected, it would usually be described as articles 112-116 or chapter 12, if these chapters combined form chapter 12 (or if the other articles in chapter 12 would not be affected by this modification). When making such a grouping, we need to check if there are no other articles in that range (e.g. an article 114a) that should not be affected.

Next to these problems with grouping, there is another way in which this system produces sentences that describe the modifications correctly, but not in the desired way. When a change is made inside a sentence, we know which words have been removed and which have been added, and we can describe those modifications in natural language. However, the Dutch guidelines prescribe that only complete meaningful units may be replaced. Thus, if Minister of Foreign Affairs is replaced by Minister of Internal Affairs, it should be stated that way, and not by stating that foreign is replaced by internal. At this moment, MetaVex does not have enough knowledge of what constitutes a meaningful unit; incorporating a chunker or parser to split the sentence could help here.

A final situation in which the computer generated texts may differ from a text written by a human expert is in the description of locations in the text. As mentioned above, the computer generates descriptions of locations by concatenating the categories and indexes of each containing element. So, the description for article 34, contained in section 4 of chapter 1

\textsuperscript{176} MetaVex does groups the changes immediately after detecting them, so this step is not actually made when generating the text, but already before.

\textsuperscript{177} Textual modifications will only be grouped if they are exactly the same. If there is a difference in capitalisation, for example, they cannot all be grouped together.

\textsuperscript{178} The question is whether we actually want to replace the enumerations with such ranges; as mentioned in section 2.8, using ranges may result in errors.
would be described as Chapter 1, section 4, article 34. However, it is common for articles to have unique number (i.e. the numbering does not reset at the start of each new chapter, or each index includes the number of the chapter). If this is the case, referring to article 34 is sufficient. In order to generate this shortened description, we need to determine whether or not the articles are continuously numbered, either within the entire law, or within some lower level. We can determine this by first studying all article indexes; if they are different, it is fair to assume that they are continuously numbered. If not, we can repeat the procedure at lower levels to determine if they are continuously numbered at that level. After we have determined if the level at which the articles are continuously numbered, we can then proceed to generate the appropriate description, leaving out any levels that do not need to be mentioned in order to get a precise location.

MetaVex collects a formal description of all the modifications made to an act, so we can use that information to generate a consolidated version of that act, i.e. the original version of the act with all the changes applied to it. A similar system has been developed for Italian acts by Palmirani and Brighi (2002) who created a system that can semi-automatically create consolidations based upon amendment acts that have already been marked-up before. Ogawa, Inagaki and Toyama (2008) have created a system that can consolidate Japanese acts by first parsing the amending clauses and then applying the changes. Spinoza et al. (2009) have a similar full-automatic system under construction for Italian acts.

As MetaVex uses a different way to gather modifications, generating them from a modified version of the act rather than extracting them from the amending text, it may seem as if we already have a consolidated version: the version as modified by the user, which was used to generate the amending document. However, this only works if all changes are made in the right order. However, amending acts may be prepared in parallel rather than sequentially. In such cases, it is not possible to use the modified text resulting from the first modification as input for the second modification, as it possible that the second modification is being prepared at a time the first modification is also still being prepared. Moreover, it is unsure which modification will be applied first; it is even unsure whether they will get enacted. Also, modifications may be prepared in secrecy. Because the modifications are prepared in parallel, the final consolidation will not be one that has originally been created by one of the users, as neither of the two modified texts contains both change sets.

The consolidated version is generated by simply applying the changes to the previous consolidation. This can be done for a single amending document, but it also possible to apply several amending documents at the same time. This can help to detect conflicts between the different amending documents.

Conflicts between modifying acts can occur because they are prepared in parallel. For example, if one amendment introduces a new article dealing with the transport of bulk goods, while another amendment changes the definition of bulk goods, this may lead to unexpected results.

When a bill is pending at parliament, and its members propose amendments, conflicts are not usually a problem. These amendments are all prepared by the legislation bureau of the
parliament\textsuperscript{179}, where they are also checked for conflicts. When parliament votes on which amendments to accept, the voting procedure is arranged in such a manner that two conflicting amendments may not be accepted. However, amending laws can be prepared at different ministries, and at different times. This means there is less coordination, which may lead to clashing modifications being approved. Because of this, it is desirable to have support for the detection of these changes.

With an automated consolidation system, a certain group of clashes is easy to detect: situations in which one modification prevents another modification to be performed, e.g. an amendment repeals article 23 while another amendment wishes to modify article 23. If the automated consolidation system is used to apply these two amendments in this order, an error will occur. By trying out all different orders, such clashes can be detected. With this support for the generation of amending text and consolidations, as well as the basic editing functionalities, MetaVex is a solid, easily extendable and adaptable support for legislative drafters. It is still under construction, and may very well merge with similar tools being developed, such as the Norma Editor (Palmirani & Brighi, 2003) or xmlLeges Editor (Agnoloni, Francesoni & Spinosa, 2007), but we feel that eventually, an editor such as MetaVex will become the standard tool for legislative drafting.

6.3 Conclusion
Automated support for the creation of models of law is not the only product of the research presented in this book. As the examples in this chapter show, both the parsers developed as the knowledge gathered can be applied to other applications that help to make legislation more accessible.
7 Conclusions and Future Work

In this thesis, we have presented methods to translate Dutch law texts to formal representations of that text.

Coming back to the question posed in the introduction:

To what extent can the creation of models of Dutch law be automated?

We have broken the creation of such models into a number of different steps, namely:

1. structuring the law;
2. identifying references in the text;
3. classifying the different sentences;
4. generating models for the individual sentences;
5. integrating the separate models into a complete model.

When it comes to structuring a law and marking any references inside it, an automated approach can be very successful. By means of headings, the start (and the end) of elements in the body of the law is clearly marked. The introduction and the conclusion use fixed formulas which can be used to identify those parts of the text as well. For separating articles and subparagraphs into sentences, existing sentence splitting techniques seem to suffice. Only lists give some trouble, as different lists (and especially sub-lists) use different structures. As a result, lists require more patterns than the other elements, as well as more post-processing.

We have created a prototype parser based on regular expressions, which identifies the headers of chapters, articles, etc., the indexes of subparagraphs and lists and certain fixed formulas that appear in the text of a law. Due to time constraints, this prototype did not include full support for dealing with lists and quoted text. In an experiment (see section 2.8), this prototype parser achieved 100% accuracy on detecting the overall structure of ten different laws. For detecting the structure within each element, this prototype achieved an accuracy of 96%. All errors were related to lists.

Just as legislative texts are clearly structured, the references referring to those texts follow a clear structure as well. Elements in the text are generally referred to by the category of the element, such as chapter or article, and its index. Since the number of different categories used in the laws is limited, it is possible to find these combinations of category and index. More complicated, derived references have more than one target, combining a single category label with multiple indexes. Others use several steps to identify their target, using a category label and index combination for each step. However, these complicated structures are also limited and can be detected using regular expressions. Only the titles of laws cannot be detected using simple patterns, as they are too varied and too similar to regular text. A list of names is required to find those in the text. A parser based on these ideas was tested on six laws detected 97% of all the references in those texts correctly, with very few false positives (less than 0.1%).

All in all, it seems that the first steps of structuring a law and finding the references in it can be effectively done by a computer. The next step, the actual modelling, is significantly harder.
Recognising the type of a sentence is still a doable task. We have identified a number of different sentence types that occur in the law:

- obligations
- rights
- application provisions
- penalisations
- calculations
- delegation
- publication provisions
- definitions
- deeming provisions
- enactment date
- short title
- change provision (scope, insertion, replacement, repeal, renumbering)

Within each category, there are language patterns (mostly verb phrases) that are commonly used. There is only one exception. The official guidelines advise that obligations should preferably be written as a description of the desired situation, without using words like must or should. As a result, many of these sentences do not have a common language pattern.

We have created two classifiers to classify the sentences. One of them is pattern-based, and classifies sentences based upon the patterns found. If no pattern can be found, it defaults to classification. A second classifier is based on machine learning, using Support Vector Machines. Both classifiers achieve an accuracy of 90-95%.

For the actual modelling, we divide the sentences into two broad groups: sentences that deal with the law, and sentences that deal with the application domain of the law. Some sentences straddle these groups, for example consisting of a main sentence that deals with the law combined with an auxiliary sentence, such as a condition, that deals with the application domain.

The sentences that deal with the law cover a limited set of operations, and follow the same structure most of the time. Because of this structure, it is relatively easy to create a model of the law by filling the slots of a frame. A manual count has predicted that this will work for 96% of such sentences, provided that all references have been correctly identified, and disregarding any auxiliary sentences that deal with the application domain of the law, and that have to be handled separately.

The real hard task is the creation of models for those sentences that deal with the domain of the law. Using a parser, it is possible to identify the different elements of the sentence, and by doing so, create a model of the situation or activity described in the sentence. However, such norms need to be integrated, and this integration requires a lot of implicit knowledge that the computer does not (yet) have. Thus, we can conclude that automation is a realistic option for those sentences that deal with the law itself, but that for those sentences that deal with the actual domain of the law, an automated process remains limited to suggesting model
fragments for the individual sentences. Even with that restriction, though, an automated process can make useful contributions to the creation of portals for legislation and expert systems.

By recognising the structure of documents, and by adding a clear identification to each structure element, we make it possible to refer to the elements of a document. This makes navigating such documents easier, as an application can now redirect a user to the appropriate element, rather than leaving the user to search through the document by himself. Moreover, identifying the elements makes it possible to add metadata to the individual elements, which is useful for other applications. For example, in a law, different structure elements, such as articles may be enacted, changed and repeals at different dates. In such case, this information needs to be tracked for each structure element separate rather than for the law as a whole.

References refer to the documents and structure elements of the documents. By automatically recognising and resolving the references, we make the “web” of the law explicit, making it easier for the user to navigate that web. When a reference has been marked-up, a computer application can automatically present the correct document, which means the user does not have to search for the document by hand (and, as mentioned above, if the target’s document structure has been identified, then the application can even send the user to the appropriate structure element as well). In addition, it is possible to follow the references the other way: an application can show what other regulations refer to the document that is currently viewed by the user.

A reference parser is not only valuable in detecting references in documents in which references have not yet been annotated. In existing collections, documents may have references annotated with links to other documents within that collection. Applying a generic reference parser can help to link such documents to legislation outside that collection, effectively linking multiple collections together (see Winkels et al., 2005).

Making the web of the law explicit also opens the door for other research. For example, the number of references in a law can be considered as an indication of the complexity of that law (Bourcier & Mazzega, 2007). When the references are made explicit, this measure of complexity can more easily be determined. Also, using notions from graph theory, it is possible to draw other conclusions with regard to this web of the law, such as identifying key articles (see Smith, 2005, Mazzega, Bourcier & Boulet, 2009, Bommarito, Katz & Zelner, 2009 and Liiv, Vedeshin, & Täks, 2007).

The categorisation of sentences is intended as a preparatory step for the creation of models. However, the categorisation itself can further support a user that is searching through the documents. It allows a user to search through a document in a simpler manner. For example, if a user is searching for specific definition, an application could show only the definitions. If a user is reading the law in order to find about the norms contained in it, the application could hide all modifying provisions\(^\text{180}\).

\(^{180}\) This corresponds with the manner in which wetten.nl presents Dutch regulations; on this portal, all modifying provisions have been removed from the regulations and are not displayed.
The search functions are further improved by the frames that are generated, as those frames make it possible to search for a definition of a specific term or for modifications to a specific location. Other frames, such as setting the citation title, can be used to generate appropriate metadata for the document. The frames that are generated for modifying sentences contain sufficient information to generate consolidated versions of the legislation that has been changed by them (see Palmirani & Brighi, 2002, Ogawa, Inagaki & Toyama, 2008, de Maat et al., 2009).

Finally, as was the original intention, the frames can be used as a basis for the model of an expert system. The automated generation of models for individual sentences, and possibly, the automated partial integration of such sentences can reduce the effort needed to create complete models, and will improve the uniformity of such models.

So, in various ways, the automated processing of legislation can improve access to the law and increase the quality of the law.

7.1 Regularity of Legislative Sources
The success of this method hinges on the regularity of the legislative texts. As a result of tradition and guidance through the official guidelines, the Dutch legislative texts are very structured. Still, there are a number of issues where a stricter adherence to the guidelines, or small additions to the guidelines, could improve the (automated) understanding of the text.

For the first step, the recognition of the structure of documents, the parsers can be simplified if new legislation coheres more closely to the official guidelines. Specifically, adhering to the following guidelines would help:

- Use the correct order of the different levels in a law (section, division, title, chapter, part and book).
- End all list items in a semi-colon, except for the last one, which ends in a full-stop if the list has no conclusion.
- Use the correct format for the indexes.

On other issues, the guidelines could be expanded in order to facilitate the comprehension of the texts. First of all, it would help to add punctuation to denote the end of quoted structure elements (i.e. entire list elements, paragraphs, articles, chapters, etc.) Likewise, marking the end of sub-lists using some dedicated punctuation also eliminates problems with the detection of the end of a sub-list. Sub-headers are also difficult to detect, and not using such sub-headers would reduce the chance of errors in the recognition of the document structure. Finally, the use of a colon at the end of an introduction of a list makes it easier to recognise the list, and avoids the chance of errors with lists. With regard to references, the current practices do not interfere with automated recognition. There are some occurrences of references that deviate from the common practice, but nothing that warrants new guidelines. However, when it comes to resolving the references, there is one type of reference that is difficult to resolve correctly: the range. The problem with a reference to a range is that it cannot simply be resolved by knowing the reference; you also need to know the structure of the document being referred to. For example, a reference to article 41 is simply a reference to article 41. A reference to articles 41 to 44 may be a reference to articles 41, 42, 43 and 44 or articles 41, 42, 43,
43a and 44, etc. This also means that a range cannot be permanently resolved, as its set of targets may change over time.

The fact that the set of targets may change over time does not only make it harder to create an easily maintained computer model of the text, but also increases the chance of introducing errors. A newly created article may be included in a reference unintended. The same is true for references that refer to the previous article, the previous subparagraph, etc. After the text has been changed, these references may no longer refer to their intended targets. Thus, to avoid errors, it may be preferable to avoid references to ranges and relative references.

When it comes to categorisation of the sentences, the Dutch legislation has one significant advantage to some other jurisdictions, and that is that the rules in a law are separated from the motivation for such rules. As a result, no steps need to be taken to separate rules and motivations. On the other hand, in many jurisdictions, clear signal words like ought, must or should are used in obligations, whereas the Dutch guidelines advise against using such words. Because of this, no clear signal words are present in obligations, making them harder to detect.

Though such changes in the legislative texts themselves would increase the performance of the processing by a computer (and, perhaps, processing by humans as well, though additional research will be needed to determine if this is actually the case), the largest improvement is still to be gained by further implementing the parsers and other tools presented in this book.

First of all, not all of the ideas presented in this book have been implemented. The structure parser needs to be extended so that it can better deal with lists, including the separation of list items and sub-list items and the recognition of the conclusion of a list. It also needs to be extended in order to recognise quoted text elements (which, in Dutch legislation, is not marked using quotation marks) and to recognise appendices.

The sentence classifier would perform significantly better on lists if each list item was first converted into a sentence (by combining it with the list introduction and the list conclusion, if it exists) and then classified. We tested approaches that limited themselves to either the introduction or the list item, but this lead to errors that should be prevented by the conversion proposed. Furthermore, since the sentence classifier is supposed to classify the main sentence, it would be improved by basing it on main sentences only, leaving auxiliary sentences out. The current classifiers, which do not leave the auxiliary sentences out will sometimes misclassify the sentence based on patterns or keywords appearing in an auxiliary sentence.

### 7.2 Architecture for an Integrated Application

Next to completing the current tools, the entire process can be improved by creating a more integrated application. Currently, the different tasks that are needed to go from a natural language text to different models are all achieved by different applications, which have to be run by hand in order to achieve a full conversion. The prototypes that have been developed during the course of this research are:

- a structure parser, consisting of various GATE modules and a post-processor in JAVA;
- a reference parser, built using the JAVA Compiler Compiler (JAVACC);
• a sentence classifier, built in JAVA, using the JAVA regular expressions library;
• a model generator for norms and definitions, also built in JAVA, using XML models generated by the Alpino parser as input.

At the moment, no model generator for sentences other than definitions and norms exists. The Alpino parser is based on a Prolog grammar, and can be run on a Linux machine.

Obviously, running all these programs by hand is not a good setup for an automated system. It would be desirable to have a pipeline of integrated modules rather than a set of separate applications. A good way to achieve such a pipeline would be by converting all modules that have been developed to GATE modules. GATE is specifically developed for natural language processing tasks, and is supported by an active community.

As mentioned, the structure parser has already been implemented using various modules from GATE combined with new JAPE rules. The reference parser can also easily be converted to JAPE rules, as it is mostly based on different patterns. This has a small disadvantage, since JAPE is based on regular expressions instead of a grammar. List items can be nested, and since regular expressions are not recursive, it is not possible to support any level of nesting using JAPE. This is not a big problem, however, since a fixed maximum depth of three or four should be sufficient to cover most, if not all, references.

Likewise, the patterns used for the sentence classifier can be converted to JAPE rules, making it possible to include the sentence classifier as a JAPE transducer inside the GATE framework.

A parser like Alpino cannot be converted to JAPE regular expressions. However, GATE makes it possible to include functions as plug-ins using wrapper classes. In this way, GATE includes several parsers for the English language. Assuming a parser for Dutch becomes available for GATE, we can then include the model generator as a set of JAPE rules on top of the annotations made by the parser. Models for sentences that do not require the parser but can be generated using language patterns can also be generated using JAPE, with rules again based on the actual words instead of the parser's annotations. The frames as presented in chapter 5 will be included as annotations in the document.

So, if all these conversion are made, a GATE pipeline can be constructed using the following processing modules in order, mimicking the process as it has been with the individual applications:

1. tokeniser;
2. JAPE Transducer which corrects tokens which are part of an index that mixes numbers, letters and punctuation;
3. sentence splitter;
4. JAPE Transducer to identify structural elements;
5. JAPE Transducer to identify references;
6. JAPE Transducer to classify the sentences;
7. Dutch parsing module(s);
8. JAPE Transducer to generate the frame annotations.
After the GATE modules have been run, a postprocessor can take care of those tasks that remain: add containers to the output (see chapter 2) and resolve the references to proper URIs (see chapter 3). The result of this process will be a XML file in which the structure is marked, references are marked, and elements of each sentence are marked according to their role in the sentence (in terms of the legal meaning, not their semantic role or syntactical role)(see chapter 4). This XML document can later be used to generate models in a specific formal language.

Running the postprocessors last, rather than in-between steps, also means that they can profit from the additional information found. The postprocessor for adding containers benefits from the knowledge of which sentences are modifying sentences, as those may introduce quoted text elements. The postprocessor for resolving references needs to know which sentences are scope provisions, as this will affect the references made following that scope provision.

As said, this pipeline follows the process as it has been executed using the individual applications. It can be streamlined a bit further by creating a dedicated tokeniser that already recognises the indexes, so that no corrections are needed. Furthermore, since it is desirable to recognise subordinate sentences before classifying sentences in order to avoid confusing due to patterns appearing inside these subordinate sentences (see chapter 4), the parsing modules can be moved to before the classifying module, as parsing the sentence will also identify subordinate sentences, resulting in the following GATE pipeline:

1. “legal” tokeniser;
2. sentence splitter;
3. JAPE Transducer to identify structural elements;
4. JAPE Transducer to identify references;
5. Dutch parsing module(s);
6. JAPE Transducer to classify the sentences;
7. JAPE Transducer to generate the frame annotations.

Again, the GATE pipeline is followed by a postprocessor.

There are two issues with the Dutch parsing module that are relevant for this automatic processing pipeline. First of all, the Alpino parser tends to have problems with references that contain a lot of punctuation. It is expected that this holds true for any parser. To avoid this, any references (which have already been recognised) should be annotated in the input of the parser so that it does not attempt to analyse it (but simply regards the entire reference as a name).

The second issue is that a parser usually generates multiple parses for each input sentence. When including it in a pipeline, only one parse will be considered (i.e. the one that is considered “best”). This may not be the correct parse. In order for the pipeline (and the automated processing) to function properly, the preferred parse should usually be the correct parse.

As it is unlikely that the preferred parse will always be the correct parse, a second tool should be available next to this main pipeline, which allows users to correct any errors introduced because the preferred parse was not the correct parse. The tool uses the XML file that is the output of the main pipeline as input. Users can select sentences that are not modelled correctly. These sentences are then put through a smaller version of the pipeline, consisting of
variants the Dutch parsing module(s) and the transducer to generate frame annotations. The variant of the parsing modules should return multiple parses\(^{181}\) (i.e. the top ten, or some other fixed number, rather than only the preferred parse). The transducer is run once for each parse, thus generating multiple models. The user should be able to select the best model (or even annotate the sentence manually), which is then saved in the XML file.

There is an overlap in the patterns searched for by both the reference parser and the structure parser, as the label and the number as they appear in a header are the same as they appear in a reference parser. With regards to performance, it may be beneficial to combine this search. This can be achieved by switching the order of the reference parser and the structure parser. The reference parser will find both the references as the label/number combinations that appear in the header. The structure parser can then use the references that have been identified as input to find the headers, changing the annotation from “reference” to “header”.

### 7.3 Future Expansions

As discussed in the previous paragraphs, this method still requires a lot of fine-tuning. More importantly, it also requires more testing. This is especially true of the actual modelling steps, which have not yet been tested systematically. Without doubt, this will result in more issues to be addressed.

In addition to the fine-tuning, though, the current method can be extended beyond the ideas presented here. There are two important directions for such expansions:

1. The current method is aimed at laws, and should be expanded to include other regulations and case law.
2. The current method has not explored all possibilities for modelling sentences that deal with the domain of the law, and it may be possible to generate more specific models.

As for the extension of the range of legislative sources covered, this is something that mostly comes down to expanding the different tools with new patterns for the additional documents. All (national) legislation in the Netherlands falls under the guidelines, and thus should be using similar language and structure. The structure parser has already been tested on a ministerial decree, and with some additions to detect the introduction of such a document, which differs from that for a law, the parser was able to handle this decree.

Extending the reference parser to handle references to other legislation mostly comes down to adding the names of such legislation to the list of names of laws. As lower regulations often have a similar structure to a law, the references to them also follow the same structure as references to the law. Still, it is likely that some new patterns emerge and have to be added.

Here, it may be interesting to include other documents that refer to legislation, such as case law and doctrine. These documents will refer to legislation in much the same way as legislation refers to legislation, though there are a few differences. First of all, these documents are much more likely to use informal names for legislation, which, like the formal names, will have to be collected in a list before they can be properly detected. Secondly, any incomplete references in

---

\(^{181}\) Note that GATE annotations differ from XML annotations in that they can overlap; this makes it possible to have multiple parses annotated within one document.
such a document will have a different meaning. A reference to article 9 in a regulation refers to “article 9 of this regulation”, while in case law or doctrine, it refers to “article 9 of the regulation that is the discussion of this (section of the) text”. Thus, the manner in which such references are resolved needs to be changed, and will be more complicated.

Finally, the sentence classifier and the method for generating model fragments should perform the same on the sentences that appear inside legislation that appear in regulations other than laws with regard to the types of sentences already included. However, it may be that other types of sentences exist in those documents, and the methods need to be expanded to cover those.

As for expanding the modelling, the current method treats all prepositional phrases (that are not linked to the direct object or indirect object) in the same way: as a general modifier of the action described in the sentence. In order to make the models more informative, it would be interesting to somehow classify these phrases in categories, such as temporal or spatial constraints. Research into this area is already being conducted (see for example O’Hara & Wiebe, 2003). Such a classification will allow for a more meaningful model of these phrases.

Looking at the core of the sentences, there is also room for some more specific modelling. In the current method, only two types of norms are distinguished: rights and obligations. Most theories on norms distinguish several other categories. Possibly the largest generalisation made in our simple division is that it treats power-conferring norms the same as behaviour regulating norms. In doing so, it ignores the fact that a power-conferring norm does not only confer a certain ability, but a certain responsibility as well.

In order to treat power-conferring norms different from other norms, we need to be able to distinguish them. Our current methods for classification, using language patterns or machine learning based on the words appearing in the text cannot make this classification. However, it may be possible to recognise the power-conferring norms after the sentence has been parsed, and additional information can be added to the analysis, such as the agent of the action. Since a power-conferring norm will (generally) target a civil servant or government institute and not a civilian, such information may help make a more advanced distinction between the different types of norms. However, even when classification is made more precise, power-conferring norms will likely remain a troublesome category, as they often are broad statements, which are difficult to integrate with more precise statements.

There seem to be few (practical) expert systems that focus on these norms. This seems understandable. The broad terms used in these norms do not lend themselves well to a computer application. Furthermore, there will be few routine questions that involve these norms (and an expert system is most useful for routine questions). Hence, it remains to be seen whether it is useful to focus on these power-conferring norms.

Next to the power conferring norms, the procedural norms also deserve more attention. For the norms that form a procedure together, the order in which they appear in the text is important, as the steps in the procedure have to be followed in that order. Currently, the method does not include any patterns or other handholds to discover whether a norm is part of a procedure, nor any ways to model the outcome. The same is true for procedural
calculations, though those are easier to recognise, as any calculations that try to establish the same value and apply at the same time are usually part of a procedure and should be applied in order.

These two extensions are aimed at improving the method presented in this thesis. However, there also needs to be more attention to the use of the metadata added by the different tools discussed here. Currently, such data is not offered to the users in a manner that allows them to utilise it to the fullest. For example, Hoekstra (2011) argues that the current Dutch government portal for legislation, wetten.nl, presents its information in a too restricted way:

> Wetten.nl presents regulations as books with hyperlinks; the position of an article within the running text of a regulation is the only context provided. Given the highly networked structure of legislation, this traditional restricted presentation is suboptimal:

potential alternative ways of serialising one or more regulation texts (e.g. by topic) are discarded. This is not only a potential problem for businesses and citizens trying to understand the norms applying to their case, it is problematic for the civil servants and government organisations that have to apply these norms as well.

Hoekstra shows how the current contents of the portal can be published as “5-star open data”, which should be more flexible. Such an approach will need to be elaborated upon in order to accommodate the additional data (such as sentence types) we can extract.

### 7.4 Text First or Model First?

In the introduction, it was mentioned that this method is one of three methods that are being explored with regards to the creation of models of legislation. The other two methods are:

1. Creating the model at the same time that the legislative text is drafted, using a specialised editor;
2. Creating the model first, and then generating the legislative text from the model.

Just like parsing, the editor allows legal drafters to create a law using natural language, whereas starting with the model does not. So, the main question becomes: is it better to specify the law in a natural language or in a formal language?

An important difference between these two approaches is, of course, the output. The processes each have two outputs, two representations of the law, one in natural language and one in formal language. When starting with natural language, we will end up with a good and correct text, but it costs some effort to create a correct computer model. Starting out with the model will give us a correct computer model, but it will create some effort to create a human readable text.

In itself, generating a natural language description of a formal model is doable; each statement in a formal language can be described in a sentence in natural language. However, a collection of such statements does not yet form a complete and readable text. Ruiter (1987) mentions that a law could be written in “legal normative sentences”, with each sentence including a normative operator (such as “may” or “must”), the norm addressee, the behaviour and conditions. Such legal normative sentences are quite close to a description of a formal model – but Ruiter states that if a law were completely composed of such sentences, they would be tedious, complicated and full of unnecessary repetitions. So, in order to generate an actual text
we would need to integrate those sentences, which would involve removing unnecessary repetitions. Also, the common sense knowledge that needs to be added to a legal text when creating a formal model needs to be removed when creating a legal text from such a formal model.

In both approaches, additional effort will be required to generate the “translation”. We have already seen that the integration of generated model fragments requires human intervention; it is likely that the same is true for the integration of generated sentences.

The main difference between the two approaches will be that the quality of the original specification, either in natural language or formal language, will be higher than that of its translation. So, the question is: which one do we want to see as “leading”? We should ensure that the leading specification is the one that has the highest quality.

Opting for natural language seems to be the better choice. Even though formal languages retain snippets of natural language, they are not as rich as natural language, which means that there will always be things we cannot express in a formal language that we can express in natural language. So, in general, starting with natural language seems is preferable.

Still, there are situations in which starting with more formal models may be preferable, such as writing amending provisions, where an approach that starts out with making the actual modifications wins out over describing them in text. Likewise, Voermans (1998) concludes that in some cases, a more formal model, like a decision tree, is better. In such cases, using a formal model is more desirable than using natural language – in fact, it would be better not to translate the formal model to natural language, as the formal model is better at communicating the information than the natural language translation.

So, in the end, a mix of both approaches is best. Situations that are difficult to explain clearly in natural language should be drafted and presented in a more formal language. Other elements of the law should be drafted in natural language and then translated to a formal model – preferably, as much as possible, by the computer itself.
Appendix A  Detecting Structure Using JAPE

In chapter 2, a method was outlined for the detection of the structure of Dutch legislative documents. This method has been used to construct a prototype using GATE (General Architecture for Text Engineering)\(^\text{182}\). In this appendix, we'll describe the different elements of the JAPE regular expressions that make up this prototype.

A.1  A Short Introduction to JAPE

In GATE, JAPE regular expressions are used to describe patterns that we want to recognise. Before applying such patterns, the document is first split into tokens by the tokeniser. Words, numbers and punctuation are marked using tokens, and whitespace is marked using a spacetoken. We can specify patterns using these tokens.

The most basic pattern is: \{\textit{Token}\}, which will match any word, number or punctuation. More specific patterns can be created by checking for certain properties of the token. The most commonly used property is the string property, which is equal to the actual word, number etc. For example, if we want to search for the word Netherlands, the appropriate pattern is \{\textit{Token.string} == “Netherlands”\}.

Tokens may be combined to form more complicated patterns. For example, we can search for the phrase of the Netherlands using the pattern \{\textit{token.string} == “of”\} \{\textit{SpaceToken}\} \{\textit{Token.string} == “the”\} \{\textit{SpaceToken}\} \{\textit{Token.string} == “Netherlands”\}. If we are not interested in the whitespace, we can instruct the pattern recogniser to ignore the whitespace, in which case the pattern \{\textit{token.string} == “of”\} \{\textit{Token.string} == “the”\} \{\textit{Token.string} == “Netherlands”\} would suffice.

Using specific operators, more complicated patterns may be created:

- A question mark after a pattern enclosed by round brackets indicates that that pattern is optional; it may appear once or not at all.
- A plus-sign after a pattern enclosed by round brackets indicates that that pattern may appear multiple times.
- An asterisk after a pattern enclosed by round brackets indicates that that pattern may appear multiple times and is optional.

To generate output (i.e. to add our own annotations) we can add instructions to each pattern. These instructions are given in the JAVA programming language. To make it easier to add the output, we can add temporary labels to the pattern. For example (\{\textit{token.string} == “of”\} \{\textit{Token.string} == “the”\} \{\textit{Token.string} == “Netherlands”\}):\textit{DutchLabel} would search for the string \textit{of the Netherlands} and then add the temporary label \textit{DutchLabel} to it, which we could later replace by an actual annotation.

A complete JAPE grammar may consist of multiple regular expressions. Furthermore, these expressions may be organised in different phases, which are applied after each other. This means that annotations that are made in the earlier phases can be used as input for later phases.

\(^{182}\) http://gate.ac.uk/
We’ll discuss two common patterns used for the determining the structure of laws: the patterns to detect a header, and patterns to detect a fixed element.

### A.2 Detecting Headings

As was discussed in chapter 2, a header consists of three elements: a category label, an index and optionally, a title.

The pattern for the category label is simply that label, e.g. for an article header, the pattern for the category label is:

```
{Token.string == "Artikel"}
```

If we assumed that the index can be any number or a combination of numbers, dots and letters that has been identified by the appropriate JAPE grammar, then the pattern for the index is\(^{183}\):

```
{Token.kind == "number"}|{Token.kind == "index"}
```

Finally, the title will start with a word which may be followed by a combination of words and punctuation, making the pattern for the title:

```
{Token.kind == "word"}
{((Token.kind == "punctuation")|{Token.kind == "word"})}*
```

The beginning and the end of the header have been determined by the sentence splitter, meaning that we can expect a split at the beginning and the end of the header. Also, between the index and the title, there may be a dot or a dash. If we add those, combine the individual patterns, and add labels, we get the following complete pattern:

**Rule: TitledArticleRule**

```
{Split.kind == "external"}
{Token.string == "Artikel"}:CategoryLabel
{Token.kind == "number"}|{Token.kind == "index"}:IndexLabel
{Token.string == "."}|{Token.string == "-"})?
{Token.kind == "word"}
{((Token.kind == "punctuation")|{Token.kind == "word"})}*
:TitleLabel
{Split.kind == "external"}

:CategoryLabel.Category = {},
[IndexLabel.Index = {}],
>TitleLabel.Title = {}
```

The lines after the arrow (\(\rightarrow\)) assign annotations to the sections marked by label. The section marked with CategoryLabel is annotated with the annotation Category, etc. Similar rules can be made for other formats (headers without titles, headers using ordinals, etc.)

\(^{183}\) Tokens of kind “index” are not introduced by the GATE tokeniser, but are the result of applying the JAPE grammar that combines numbers/dots/letters.
A.3 Detecting Fixed Elements

As was mentioned in chapter 2, the introduction of a Dutch law starts with *We* and ends with *understands as follows*. This corresponds with the following pattern in JAPE (in Dutch):

```plaintext
(Token.string == "Wij")
((Token))*
(Token.string == "verstaan")
(Token.string == "bij")
(Token.string == "deze")
(Token.string == ":")
```

A disadvantage of using this pattern is that it may find multiple matches. As the word *We* will probably occur multiple times in the introduction, JAPE may match this pattern to each region that starts with an occurrence of *Wij* and ends at *verstaan bij deze*. We can instruct JAPE to make only one match. In that case, he will take the match that is the longest, starting at the first *Wij* and ending at the last *verstaan bij deze*. So, as long as *verstaan bij deze* occurs nowhere else in the document, this is a safe method. This seems like a reasonable assumption.

However, if we do not want to make this assumption, we can get our result by using temporary annotations, by first marking up the first *Wij* and the first *verstaan bij deze* and then creating a single annotation spanning them.

This requires three phases, two to make the temporary annotations and one to create the final one. The first phase looks as follows:

Phase: IntroductionTemporaryAnnotations
Input: Token
Options: control = once

Rule: Wij
{
    (Token.string == "Wij")
} :Wij --> :Wij.Wij = {}

These lines are somewhat more complicated as the pattern shown earlier, as they do not only include the actual pattern, but also all the options and the lines generating the output. We start with

Phase: IntroductionStartTemporaryAnnotation
Input: Token
Options: control = once

The first line indicates that the patterns that follow belong to a separate phase in the annotation process. The next line indicates that we will only look at Tokens (which means that we will ignore Spacetokens, which actually means that we are not interested in whitespace). The final line indicates that this phase exit after one pattern has been matched – which should be the first occurrence of *Wij*.

Then, we get the actual rule, which reads:
Rule: IntroductionStartRule
{
    (Token.string == "Wij")
}:StartLabel -->
:StartLabel.IntroductionStart = {}

The first line indicates that a new rules starts, named IntroductionStartRule. Then we get the actual pattern, which consists of a single token matching the word Wij. This area is assigned the label StartLabel. The final line generates the output: it indicates that an annotation IntroductionStart should be applied to the area marked with the label StartLabel.

After detecting the first We in this manner, we can move on to detecting understands as follows, which results in a similar phase:

Phase: IntroductionEndTemporaryAnnotation
Input: Token
Options: control = once

Rule: IntroductionEndRule
{
    (Token.string == "verstaan")
    (Token.string == "bij")
    (Token.string == "deze")
    (Token.string == ":")
}:EndLabel -->
:EndLabel.IntroductionEnd = {}

Now, we can add a pattern that combines these two temporary annotations to the actual annotation marking the entire introduction.

Phase: Introduction
Input: IntroductionStart IntroductionEnd
Options: control = once

Rule: IntroductionRule
{
    (IntroductionStart)(IntroductionEnd)
}:IntroductionLabel -->
{
    gate.AnnotationSet as =
    (gate.AnnotationSet/bindings.get("IntroductionLabel"));
    outputAS.add(as.firstNode(), as.lastNode(), "Introduction",
        Factory.newFeatureMap());
    outputAS.removeAll(as);
}

The pattern links the start and end annotations for the introduction. As the parser has been instructed to ignore everything except the introduction start and the introduction end, we do not need to specify what other tokens may occur in between the start and end of the introduction. The lines that generate the output are a bit more complicated than in the previous rules. This is because we used a shorthand notation in the previous rules. However, we do want to remove the temporary annotations, which cannot be done is shorthand.
The first line of the output generation is:

gate.AnnotationSet as =
    (gate.AnnotationSet)bindings.get("IntroductionLabel");

This line asks for a set of all annotations that appear in the area that has been labelled
IntroductionLabel. This set will include the IntroductionStart annotation and the IntroductionEnd
annotation.

outputAS.add(as.firstNode(), as.lastNode(), "Introduction",
        Factory.newFeatureMap());

This line adds an Introduction annotation to the area that corresponds to this set (and thus to
the IntroductionLabel).

outputAS.removeAll(as);

This last line removes all the annotations that are part of the set (i.e. the temporary
annotations IntroductionStart and IntroductionEnd).
Appendix B  Detecting References using JAPE

In chapter 3 of this thesis, the different language formats that are used for references have been discussed. These language formats can be detected using patterns. In this appendix, it is shown how such patterns look like in JAPE, with the goal of applying annotations in the style of MetaLex 1.3.1\(^{184}\). Note that these patterns are not the ones used in the experiment described in section 3.4, though they are very similar. The patterns used in the experiment were written down in a different notation, and we have opted to include the JAPE patterns instead as we have also used JAPE grammars in appendix A, as we feel these are easier to read.

Please refer to appendix A for a short introduction of JAPE. For the detection of references, we assume that a GATE tokeniser has been applied to the input. Furthermore, since the names of Dutch legal sources do not conform to a specific pattern, they must be found using a list of names. This is done using a separate GATE resource, a GATE Gazetteer, and is not part of this JAPE grammar (though the grammar uses the annotations left by the Gazetteer as input).

Since the patterns for references overlap with the patterns for headers, these patterns will also find annotate parts of headers, unless these have already been marked. It is therefore assumed that the headers have already been marked in such a way that they no longer appear as input for the reference, or that references found in headers will be filtered out afterwards.

### B.1 Basic References

The simplest references consist of a category label and an index – similar to a heading. So, we can use a similar same pattern. To simplify the patterns, we first apply a temporary annotation Index to each token (or group of tokens) that may be an index, i.e. each number, single letter, a number followed by °, etc.

Now, for a reference to a list item, several labels are used: sub, onder and onderdeel. All of them may appear with or without a capital. So, the pattern for a list item is\(^{185}\):

\[
\text{(\{Token.string == "Sub"\}|\{Token.string == "sub"\}|
\text{\{Token.string == "Onder"\}|\{Token.string == "onder"\}|
\text{\{Token.string == "Onderdeel"\}|\{Token.string == "onderdeel"\})}\
\text{(Index)}:ListItem
\]

Now, we would like to mark these using a Cite annotation, with the information that is needed to resolve the reference (in this case, the information that it is a reference to a list item with the specific index found). The following block of code does that:

\[^{184}\text{http://legacy.metalex.eu/}\]

\[^{185}\text{It is possible to simplify this pattern somewhat: Token.string =~ \"[Ss]ub\"\) matches both Sub and sub.}\]
gate.AnnotationSet set = (gate.AnnotationSet)bindings.get("ListItem");
gate.Annotation index = (gate.Annotation)set.get("Index");
gate.Annotation indexAnn = (gate.Annotation)aanduiding.iterator().next();
gate.FeatureMap features = Factory.newFeatureMap();
features.put("onderdeel", indexAnn.getFeatures().get("value"));
features.put("kind", "onderdeel");
outputAS.add(set.firstNode(), set.lastNode(), "Cite", features);
inputAS.removeAll(Index);

The first line selects all the annotations within the set marked as ListItem, i.e. the entire reference found. Then, the second line selects all the Indexes that appear within that set. The third line selects the first element of that set, i.e. the first Index (which should also be the only index).

Now, we start creating a set of attributes (called features in GATE). We first create a new set, add the attribute onderdeel and give it a value equal to indexAnn.getFeatures().get("value"), which is the actual index found. Also, we add an attribute kind which we give the value onderdeel, to indicate that this specific Cite annotation refers to a list item.

Finally, we add the actual Cite annotation and remove the temporary Index annotation.

B.2 References to Ranges

In MetaLex, a reference to a range, such as items a to d, is marked up differently from simple references. The reference as a whole is marked as a CiteRange, with the first index of the range marked as CiteFrom and the last index as CiteTo, e.g.:

<CiteRange><CiteFrom>items </CiteFrom>a</CiteFrom> to </CiteTo>d</CiteTo></CiteRange>

To simplify the patterns needed for ranges, we can use temporary annotations for the indices, such as:

Rule: Range
{
   {{Index}}:Start
   {
      {Token.string == "-"}
   |
      {Token.string == "tot"}
      {Token.string == "en"}
      {Token.string == "met"}
   }
   {{Index}}:End
} -->

{ gate.AnnotationSet start =
  (gate.AnnotationSet)bindings.get("Start");
gate.Annotation index = (gate.AnnotationSet)start.get("Index");
gate.Annotation indexAnn =
  (gate.Annotation)index.iterator().next();
gate.FeatureMap features = Factory.newFeatureMap();
}
This rule recognises patterns like 2 – 5 and 3 up to 5, and marks such patterns with a Range annotation, with the indices marked as CiteFrom and CiteTo, which will make it easy to convert the Range to a CiteRange. No kind attribute is added to the CiteFrom and CiteTo attributes, as it is not yet known what kind of element (list item, subparagraph, article, etc.) they refer to. An attribute value holds the actual index found. After all references have been found, any remaining Range annotations can be removed, together with the corresponding CiteFrom and CiteTo annotations.

Rule: EnkelOnderdeelRange

```java
{{Token.string == "Sub"}||{Token.string == "sub"}||{Token.string == "Onder"}||{Token.string == "onder"}||{Token.string == "Onderdeel"}||{Token.string == "onderdeel"}||{Token.string == "Onderdelen"}||{Token.string == "onderdelen"}) |
(Range) :Onderdeel -->
{
    gate.AnnotationSet set = (gate.AnnotationSet)bindings.get("Onderdeel");
gate.AnnotationSet start = (gate.AnnotationSet)inputAS.get("CiteFrom", set.firstNode().getOffset(), set.lastNode().getOffset());
gate.AnnotationSet end = (gate.AnnotationSet)inputAS.get("CiteTo", set.firstNode().getOffset(), set.lastNode().getOffset());
gate.AnnotationSet range = (gate.AnnotationSet)set.get("Range");
gate.Annotation startAnn = (gate.Annotation)start.iterator().next();
startAnn.getFeatures().put("onderdeel", startAnn.getFeatures().get("value"));
startAnn.getFeatures().remove("value");
gate.Annotation endAnn = (gate.Annotation)end.iterator().next();
endAnn.getFeatures().put("onderdeel", endAnn.getFeatures().get("value"));
endAnn.getFeatures().remove("value");}
```
With this range annotation, the pattern for the range becomes similar to the pattern for a simple annotation: a category label\(^{186}\) followed by a range token:

\[
\{ \text{Token.string} == "Sub" \} | \{ \text{Token.string} == "sub" \} | \{ \text{Token.string} == "Onder" \} | \{ \text{Token.string} == "onder" \} | \{ \text{Token.string} == "Onderdeel" \} | \{ \text{Token.string} == "onderdeel" \} | \{ \text{Token.string} == "Onderdelen" \} | \{ \text{Token.string} == "onderdelen" \} \]
\]

In order to apply the correct annotation, we need to perform the following steps:

- update the CiteFrom and CiteTo annotations to indicate they refer to a list item;
- remove the temporary Range annotation;
- add a CiteRange annotation around the entire reference.

Adding the CiteRange and removing the Range is done in the same way as new annotations were added and temporary annotations were removed in the examples above.

Adding the additional feature to the CiteRange annotation is done using the following code:

```java
gate.AnnotationSet set = (gate.AnnotationSet)bindings.get("ListItemRange");
gate.AnnotationSet start = (gate.AnnotationSet)inputAS.get("CiteFrom", set.firstNode().getOffset(), set.lastNode().getOffset());
gate.Annotation startAnn = (gate.Annotation)start.iterator().next();
startAnn.getFeatures().put("onderdeel", startAnn.getFeatures().get("value"));
startAnn.getFeatures().remove("value");
```

The first line selects all the annotations within the text labeled as ListItemRange, i.e. the entire reference that has been found. The second line selects all CiteFrom annotations, and the third line selects the first annotation from that set, which should also be the only CiteFrom annotation in the range. The fourth line adds an attribute onderdeel to the annotation, giving it the same value as the attribute value. This attribute value was added when we detected the range, and holds the value of the index, e.g. “1”, “a” or “1°”. After copying the attribute value, it is deleted. (So, effectively we have renamed the value attribute to onderdeel.)

\(^{186}\) There are two additional labels, as for a range, the plural of one of the labels is also used.
### B.3 Multiple References

A reference may also have several different targets that are not grouped together in a range (though some of them may be). In MetaLex, these are marked using a `CiteGroup` tag, with any individual indices marked with `Cite` and each range marked with a `CiteRange` (with the indices of the range marked using `CiteFrom` and `CiteTo`, as in the previous section). For example, we want to recognise a reference like items \textit{a, b to f and g}, and mark it up as follows:

\begin{verbatim}
<CiteGroup>items <Cite>a</Cite>, <CiteRange><CiteFrom>b</CiteFrom> to <CiteTo>f</CiteTo></CiteRange> and <Cite>g</Cite></CiteGroup>
\end{verbatim}

Using the `Index` and `Range` temporary annotations, the pattern for such a reference can be described as:

\begin{verbatim}
{
    (Token.string == "Sub")|{(Token.string == "sub")}
    |{(Token.string == "Onder")|(Token.string == "onder")}
    |{(Token.string == "Onderdeel")|(Token.string == "onderdeel")}
    |{(Token.string == "Onderdelen")|(Token.string == "onderdelen")}

    ({{Index}}|{{Range}})
    ({{Token.string == ","}}|{{Index}}|{{Range}})*

    ({{Index}}|{{Range}})
    ({{Token.string == "en"}}|{{Token.string == "of"}})
    ({{Index}}|{{Range}})
}
):ListItemGroup
\end{verbatim}

The pattern starts with a category label, followed by an index or a range, then optionally several times a comma followed by another index or range, and finally the word \textit{and}, \textit{or} and one more index or range.

In order to achieve the desired mark-up, the following steps need to be taken:

- add an `CiteGroup` annotation to the entire reference;
- replace all `Index` temporary tags by an `Cite` tag, and change the `value` attribute to an `onderdeel` attribute;
- replace all `Range` temporary tags by a `CiteRange` tag;
- change the `value` attribute on the `CiteFrom` and `CiteTo` tags to an `onderdeel` attribute.

We have seen these operations in the example above. The main difference is that there may now be multiple annotations of the same type. This means that we will have to do certain operations for all the elements in a given set. The code below shows how this is done for the set of all `Indices` (which should be replaced by a `Cite` tag).

\begin{verbatim}
gate.AnnotationSet set = (gate.AnnotationSet)bindings.get("ListItemGroup");
gate.AnnotationSet index = (gate.AnnotationSet)set.get("Index");
gate.Annotation current;
\end{verbatim}
java.util.Iterator i = index.iterator();
while(i.hasNext()){
    current = (gate.Annotation)i.next();
    gate.FeatureMap features = Factory.newFeatureMap();
    features.put("onderdeel", current.getFeatures().get("value");
    outputAS.add(current.getStartNode(), current.getEndNode(), "Cite",
    features);}
}
inputAS.removeAll(index);

As before, the first two lines select the set of all the indices. Then, we declare an iterator, which is a JAVA object that allows us to select all the items of the set by one. For each element of the set, a new Cite annotation is added. Finally, the entire set of temporary Index annotations is removed.

### B.4 Zooming-In Layered References

The patterns above can be used to detect references that refer to a single “layer” of a document, i.e. only items, or only articles. But in chapter 3, we also mentioned the layered references, which refer to a specific part of a document and then zoom in to a specific subpart, such as article 12, item b.

Such patterns can be detected by applying multiple phases for each level. For example, if we first detect all references at item level, followed by all references at subparagraph level, a part of the example above will already have been tagged, as follows:

article 12, <Cite>item b</Cite>

The pattern for detecting such a reference is the same as the pattern for detecting a single article reference followed by a Cite tag. Since the Cite tag may also mark a subparagraph, or a layered reference to an item within a subparagraph, such a pattern deals with several possibilities in one go. Written out in JAPE, the pattern is:

```japecode
( {Token.string == "Artikel"} | {Token.string == "artikel"} |
 {Token.string == "Art"} {Token.string == "."} ) |
{Token.string == "art"} {Token.string == "."} ):{Index}
{Token.string == ","} {Cite}
):ArticleLayered
```

As with the previous sections, we now need to modify the existing annotations to get the correct result. In this case, we need to add a new Cite annotation. This should have a kind attribute (with value artikel) and an artikel attribute (with the value of the index). Also, we need to copy any attributes of the lower level Cite (i.e. the attributes for the subparagraph and/or item) to this new Cite annotation. Then, we can delete the old Cite annotation.
Similar patterns can be used articles that include a CiteGroup or CiteRange sub layer. In those cases, the existing Cite, CiteFrom and CiteTo will need to be updated with the information that they are part of a reference to an article. Likewise, a reference to multiple articles, each of which may have a sub layer, can be dealt with in such a manner.

B.5 Zooming-Out Layered References
In addition to zooming-in layered references, there are also zooming-out layered references, as well as references that first zoom in, then zoom-out (see section 3.2), such as article 3, sub 3 of the Coin Act. If we first run the patterns as described above for all structure levels, then the different elements of such a reference will already be marked using Cite, CiteGroup and CiteRange annotations. In the case of the example given above, the annotations would be:

\[
\text{<Cite>article 3, sub 3</Cite> of <Cite>the Coin Act</Cite>}
\]

So, the pattern that is needed here (including the possibility of a comma to separate the two parts) amounts to:

\[
\{\text{Cite}\ (\{\text{Token.string == ","}\})? \{\text{Token.string == "van"}\} \{\text{Cite}\}
\]

Similar patterns should be added for a CiteGroup and CiteRange that are part of some higher level Cite. We can add a check to see whether the levels being merged are appropriate, which results in something like:

\[
\{
\{\text{Cite.kind == "artikel"}\}|\{\text{Cite.kind == "paragraaf"}\}\{\text{Cite.kind == "afdeling"}\}|\{\text{Cite.kind == "titel"}\}\{\text{Cite.kind == "hoofdstuk"}\}|\{\text{Cite.kind == "deel"}\}\{\text{Cite.kind == "boek"}\}
\}
\{\{\text{Token.string == ","}\})? \{\text{Token.string == "van"}\}
\{\text{Cite.kind == "wet"}\}
\]

So, we use the kind attribute that we added to each of the Cites, CiteGroups and CiteRanges to determine whether the first reference can actually be a subpart of the second reference187.

After two references have been found that are related, they need to be replaced by a single Cite (or CiteGroup or CiteRange) that combines the attributes of the different parts. This can be done with code similar to that presented in the previous sections.

B.6 Putting It All Together
As was mentioned in the previous sections, the different patterns rely on other patterns being applied before them. The result is a multi-phase JAPE grammar, with a number of phases.

The first phase adds the temporary Index annotations. The second phase adds temporary Range annotations. After that, the actual detecting of references starts with a phase that detects references to list items: single references, references to ranges and group references. The next two phases combines two references that form a zooming-in reference from a list item to a sub list item.

187 This check is most likely superfluous, as a text like article 12 of article 28 is unlikely to appear in a legislative source.
Then follow the phases that detect references to the higher structure levels, starting with a phase that detects references to subparagraphs, including zooming-in references to sub-items (as discussed in section B.4). This phase is followed by similar phases at the level of articles, sections, etc. After that, phases follow to combine references that form a single zooming-out reference, starting at the level of list item and going up. Finally, there is a clean-up phase that removes any temporary annotations that still remain, and that apparently were not part of a reference.

In order to detect zooming-in and zooming-out references, this method orders the different structure layers in the way that is prescribed in the official guidelines (i.e. section is the lowest level, followed by division, title, chapter, part and book). This means that a multi-layered reference to such a layer in a document that does not follow this structure may not be recognised correctly. For example, when referring to a document which has chapters within a division, a reference to division 3, chapter 2 is a layered reference, while this approach will annotate it as two separate references. For this specific example, it would be better if the order was not prescribed in the grammar and instead we simply assumed that the first level mentioned was the higher level. After all, if the intention was to have two separate references, it would have been written as division 3 and chapter 2. However, this assumption does not always hold if the reference is part of a list, like division 3, chapters 2 and 3.

In general, though, such references seldom occur. Most references are aimed at the article level or below, and do not need to address the higher-level layers because the articles have been numbered uniquely throughout the document. Hence, the number of errors introduced by choosing one method over the other is very low. In a complete system, which includes a resolver which attempts to attach a URL to each reference found, it will be better to include the assumption mentioned above, as the resolver is more likely to detect errors introduced because of that assumption.
Appendix C  Sentence Classification Patterns

In section 4.5, a pattern-based approach for the classification of sentences in Dutch legislative documents was introduced. In order to provide some insight into these patterns, we’ll give an overview of the patterns included in the parser for the categories Obligation/Duty and Short Title. The patterns are presented as JAVA regular expressions, and include a number of special markers:

<table>
<thead>
<tr>
<th>Marker</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\s+</td>
<td>one or more spaces</td>
</tr>
<tr>
<td>+</td>
<td>one or more (random) characters</td>
</tr>
<tr>
<td>\b</td>
<td>end of a word</td>
</tr>
<tr>
<td>?</td>
<td>the preceding word or pattern enclosed by brackets is optional</td>
</tr>
</tbody>
</table>

Table 18: JAVA regular expression elements

C.1  Patterns for Obligations and Duties

As mentioned in chapter 4, many obligations do not follow a pattern, but are instead presented using the normative indicative. The table below shows the patterns that are used to identify the sentences that do follow a pattern.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>English translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\s+(is</td>
<td>zijn)\s+verplicht\b</td>
</tr>
<tr>
<td>\s+(is</td>
<td>zijn)\s+.\s+verboden\b</td>
</tr>
<tr>
<td>\s+(is</td>
<td>zijn)\s+.\s+gehouden\b</td>
</tr>
<tr>
<td>\s+(is</td>
<td>zijn)\s+verantwoordelijk\s+voor\b</td>
</tr>
<tr>
<td>\s+{moeten</td>
<td>moet}\b</td>
</tr>
<tr>
<td>\s+{dienen</td>
<td>dient}\b</td>
</tr>
<tr>
<td>\s+{draagt</td>
<td>dragen}\s+{zorg}\b</td>
</tr>
<tr>
<td>\s+{is</td>
<td>geen}\s+.\s+{gelaten}\b</td>
</tr>
<tr>
<td>\s+(is</td>
<td>zijn)\s+niet\s+{bevoegd}\b</td>
</tr>
<tr>
<td>\s+{kunnen</td>
<td>kan</td>
</tr>
<tr>
<td>\s+{is</td>
<td>zijn}\b{bij</td>
</tr>
<tr>
<td>\s+{is</td>
<td>zijn}\s+niet\s+{verenigbaar}\s+{met}\b</td>
</tr>
<tr>
<td>\s+{is</td>
<td>zijn}\b{omlast}\s+{met}\b</td>
</tr>
<tr>
<td>\s+{heeft</td>
<td>hebben}\s+{mede</td>
</tr>
</tbody>
</table>

Table 19: Patterns for obligations and duties

The table includes many different synonyms for *must* and *cannot*. Also, there are a few of (rare) patterns that are relevant for specific situations, such as *is not combinable with*, which implies a specific condition.

C.2  Patterns for Short Titles

The setting of a short title is more standardized than the obligations; as a result, there are fewer different patterns used.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>English translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\s+{kan\s+worden}\s+aangehaald\s+als{:}\s+</td>
<td>may be cited as</td>
</tr>
<tr>
<td>\s+{wordt}\s+aangehaald\s+als{:}\s+</td>
<td>is cited as</td>
</tr>
</tbody>
</table>

Table 20: Patterns for sentences setting a short title

---

188 The patterns contain both the singular and plural of the verb; in the English translation, only the singular is included.
References


Summary

The rules of a modern society are codified in statutes, regulations and case law. These rules provide a specification of how a society should be. They tell us how civilians in a society should behave, and how the society’s government and civil servants should behave. Those specifications differ from technical specifications as they are written in natural language rather than in a formal language or schematics. However, regulations are based on established best practices and design patterns. As such, legislation is far more regular than most other natural language text, and not that different from technical specifications.

Within the field of Computer Science & Law (the branch of computer science that deals with the law), this regularity of the legislation is used as the basis for applications that increase the accessibility of the legislation. One common type of application is a portal that allows users to search for legislation using keywords, titles, etc. In such portals, regulations are not stored as plain text, but in a more structured format. Such a format makes it possible to point users to a specific section of a document, and it also makes it possible to store different kinds of metadata, which can also help a user to find the information he needs.

For users without a legal background, however, such applications are not helpful enough. Such users are often looking for an answer to a specific legal problem, rather than mere legal information. For those users, getting a link to the relevant regulations is an insufficient answer. Thus, a second type of application is developed: systems that provide these answers by questioning the user about his situation and then applying the rules. Such systems are not only useful for citizens that need answers to their legal problem, but can also help civil servants process routine cases (for example, when processing permit applications).

This second type of application requires complete, formal and executable representations of the law. Creating such representations takes a lot of effort. This is a common problem with any knowledge-based system, and has become known as the knowledge acquisition bottleneck. The research presented here aims to (partially) overcome this bottleneck by attempting the automated translation of Dutch laws to computer models.

This automated translation is not done in a single step. Instead, it is broken up into several phases:

1. structure recognition: the structure of the law is recognised and marked-up;
2. detecting references: any references in the law to other legislative sources are detected and marked-up;
3. sentence classification: the sentences in the law are assigned a specific type;
4. modelling sentences: a model of each sentence is made;
5. integrating models: the models of individual sentences are integrated into a single model of the law.

The (Dutch) official guidelines for legislative drafting form the basis of the recognition of structure elements in Dutch legislative documents (discussed in chapter 2). These guidelines divide a law into six parts: the title, preamble, body, conclusion, signatures and appendices. The body is subsequently split into articles, which may be split into subparagraphs and may...
contain a list with list items. The articles may be grouped into sections, divisions, titles, chapters and books.

These different parts are what we want to recognise and identify, i.e. we want to know that certain sections of the document belong together, but we also want to know that they form article 12, or chapter II.

The preamble, conclusion and signatures use specific formulas, which are prescribed in the guidelines. As a result, we can easily find and identify them by searching for the fixed elements of those formulas. Most other elements are marked with a header or an index. Such a header consists of a category label (such as *article* or *chapter*), an index (such as 4, II or 5:128) and optionally a title. By finding these headers, we can find the start of a new structure element (and the end of the previous element). Finally, articles and subparagraphs need to be split into sentences. This can be achieved using existing tools.

A prototype parser built on these principles using the General Architecture for Text Engineering (GATE) was tested on ten different laws. It recognised the structure on article level and above without errors. In 2% of the articles themselves, there was an error at the level of subparagraphs, lists or sentences. The structure of the remaining 98% of the articles was correctly detected by the prototype parser. The main issue that remains in the detecting of structure of laws is the inclusion of quoted text inside articles that modify existing laws. Such quoted text may contain structure elements that are not part of the law itself. Dutch laws do not include quotation marks to recognise the beginning and end of such parts, and as a result, this needs to be determined by comparing the indices of different elements (as the indices of the quoted text will usually not be continuous with the indices of the quoting texts).

Like headers, references conform to specific patterns, which we can use to detect them (see chapter 3). References to a legislative document are made using its (short) title or its signing date. Specific structure elements of a document are described using a category label (*chapter*, *article*, *list item*) and the index of the element. A complete reference includes the document that is being referred to, but incomplete references, which only refer to a structure element, are very common. Their exact target depends on their context. Within a law, an incomplete reference will often refer to a location within that law itself, unless it is part of a list of changes to be made in some other law, in which case it refers to the law being modified. References may also have multiple targets, either by listing multiple indexes (such as *articles 12, 13 and 14*) or by combining them in a range (such as *articles 12 – 14*).

The references that consist of a category label and an index (or multiple indices or a range) can easily be detected using patterns. The same holds true for documents being referred to by their signing date. Short titles, on the other hand, are very varied, and cannot be described using a pattern. In order to detect these, a list of all available titles is needed.

A prototype parser has been constructed that uses such a list of available titles as well as patterns for documents by signing date and category label plus index. This parser has been tested on six different Dutch laws containing over 1,000 references. It detected 97% of those references successfully, with only a few false positives.
In chapter 4, the classification of sentences that appear in Dutch laws is discussed, splitting the sentences into fifteen broad categories. First of all, there are the actual normative sentences: rights/permissions and obligations/duties. These sentences are supported by definitions, which define the terms being used by the normative sentences. Similar to definitions are the deeming provisions, which deem certain items to be equivalent for the application of certain norms. Application provisions specify additional situations in which certain norms do (not) apply, effectively creating additional conditions for those norms. Penalisation provisions set the punishment for violating a norm. A publication provision is a specific obligation ordering the publication of certain information. A value assignment calculates some value, which is used in some norm.

Next to these rules that form the actual law, there are some other sentence types that are about the law itself. There are sentences that set the enactment date or the short title of a law, and delegations delegate the right (or duty) to set additional rules. Other sentences modify existing regulations, by inserting new text, replacing text, repealing text or renumbering existing elements. Scope definitions set the scope for these changes, thereby simplifying the references that are needed in the modifying sentences.

Each class of sentence uses specific keywords or language patterns which can be used to classify the sentences, with one exception. Though many obligations do use keywords like must or obligated, it also common to describe an obligation as a fact, i.e. by stating that the obligation action happens rather than that it must happen. Such a description lacks a keyword like must.

Since the only sentences that do not (always) conform to a pattern are obligations, we can still use the patterns to classify the sentences. If no pattern applies to a sentence, then we assume it is an obligation. A classifier was built using such patterns. It was tested on eighteen different laws with 591 sentences (not counting lists). Of these 591 sentences, 157 were “statements of fact”, i.e. obligations that were written down as a fact. The classifier was capable of correctly classifying 91% of the sentences.

When it comes to text classification, a machine learning approach is often more successful than knowledge-based approached. Therefore, a machine-learning approach using Support Vector Machines (SVMs) was tested, in addition to the pattern-based approach. This was done on the same data set that was used to test the pattern-based approach, using a leave-one-out (LOO) approach. This means that a classifier is trained on all sentences but one. This classifier is then used to classify the one sentence that was left out. This is then repeated for all the sentences. From the number of correct classifications, a LOO accuracy can be derived, which is an indicator for the actual accuracy of such a classifier. For the classification of legislative sentences, the machine-learning approach achieved a LOO accuracy of 95%. In a test on two new laws, the machine-learning classifier scored an accuracy of 95.77% and 88.78%, compared to 94.37% and 95.61% for the pattern-based approach, suggesting that neither approach is strictly better than the other. The pattern-based approach does have the advantage of transparency: the knowledge of what pattern was used to classify the sentence helps with the modelling of the sentences.

For the modelling, discussed in chapter 5, there is a difference between two groups of sentences: sentences that deal with the law itself and sentences that deal with the subject
matter of the law. Sentences that deal with the law itself are those sentences that modify existing laws, set the enactment date or citation title of a law, etc. These sentences describe a specific situation, and follow a strict, more or less standardised format. On the other hand, sentences that deal with the subject matter of the law are norms and definitions. These can describe a great variety of situations, and do not follow standard formats.

For sentences that deal with the law itself, we can rely on the few standardised phrases that are used in order to extract the information that is needed to model the sentence. For example, when dealing with a sentence that replaces some text, we need to know where we should replace the text, what text to replace and by what to replace it. In these sentences, the location is always mentioned in the form of a reference, the text to replace is marked using quotation marks, and the text to replace it with follows the (first) colon in the sentence. For other sentence types that deal with the law, such as setting the enactment date and repealing a law, similar patterns can be applied to extract the needed information. A manual count of a corpus consisting of 343 sentences that deal with the law, suggests that about 96% of these sentences can be modelled using this approach.

Sentences that deal with the subject matter of the law are more diverse, and do not conform to a few standardised phrases. For those sentences, a more generic method is proposed. For normative sentences, a frame is created describing the action or situation that is being allowed or disallowed. Such a frame includes information such as action, agent and patient. This information can be extracted from the sentence using existing natural language processing tools.

These frames form a basic tool for applying the rules. By describing a case in similar terms (i.e. action, agent, patient, etc.) we can compare the situation in the case with the situations described in the rules. If the situation matches, the rule applies, and it becomes clear if the situation in the case is allowed or not (under that rule).

A similar approach can be followed for conditions and definitions. When dealing with a condition, if the case matches the description, this indicates that the rule to which the condition applies should be followed. When dealing with a definition, a match means that any rules that apply to the defined concept also apply to this case.

After the sentences have been modelled, the individual models must be integrated. Provisions can be linked because they explicitly refer to each other, using a reference or because they use the same concepts. Provisions that are linked through references are easily detected, as the references have already been detected. The link between provisions that use the same concepts, and that use the same word(s) to identify these concepts is also easily detected. However, many sentences are linked through common sense relations between the concepts they use. For example, the concepts gift and giver are related, and thus sentences that use these two different concepts are related. However, since this relationship is not defined within the law, the models of these two sentences cannot be integrated automatically unless we add this knowledge to the process. Since the body of common sense knowledge is huge, the automated translation process is unlikely to have all knowledge that is needed available. This means that only specific parts of the integration step can be completed automatically.
As this integration step cannot be performed automatically, it is not possible to automatically create models of the law. However, automated tools can help reduce the knowledge acquisition bottleneck significantly, as they are successful in detecting the structure of a law, finding the references contained in that law, classifying the sentences and suggesting model fragments for those sentences. Furthermore, the intermediate results of the process also have their uses. For example, the recognition of references has been used in the construction of a semantic network of legislation for the Dutch Tax and Customs Administration, and the patterns identified for the classification of sentences are included in MetaVex, an editor for legislation. These applications are discussed in chapter 6.

In order to actually start using this method, it will need to be refined. The accuracy can be improved by gathering more patterns from a wider training set and by implementing various changes suggested in this thesis. Furthermore, legislative drafters can also improve the automated processing of laws. Several pieces of information in the law can be expressed in a more precise manner, which may also make them easier to read for humans.

Next to improving the accuracy, the method also needs to be expanded. With regards to the classification, it may be desirable to introduce several subclasses. The method also needs to be expanded for other documents than laws, such as royal decrees and case law. For this, patterns that are specific for those documents will have to be added.
Samenvatting
De regels van een moderne samenleving zijn vastgelegd in wetten, maatregelen en jurisprudentie. Deze regels vormen een specificatie van hoe de samenleving dient te zijn. Ze vertellen ons hoe de burgers in die samenleving zich horen te gedragen, en ook hoe de regering en ambtenaren van die samenleving zich horen te gedragen. Qua tekst verschillen deze specificaties van technische specificaties, aangezien ze zijn geschreven in natuurlijke taal in plaats van een formele taal of diagrammen. Echter, wetgeving is gebaseerd op best practices en ontwerppatronen. Daardoor is wetgeving veel regelmatiger dan andere teksten in natuurlijke taal, en verschilt het niet zoveel van technische specificaties.

Binnen de rechtsinformatica, de tak van de informatica die zich bezighoudt met juridische onderwerpen, wordt deze regelmatigheid van de wetgeving gebruikt als basis voor toepassingen die de toegankelijkheid van de wetgeving vergroten. Een veel voorkomende toepassing is een portal dat het voor gebruikers mogelijk maakt om wetgeving te doorzoeken door middel van sleutelwoorden, titels, etc. In dergelijke portals is de wetgeving niet opgeslagen als platte tekst, maar in een meer gestructureerd opslagformaat. Deze opslagformaten maken het mogelijk om te verwijzen naar een specifiek deel van een document. Ook bieden ze de mogelijkheid om metadata op te slaan. Deze metadata kan ook worden gebruikt om de gebruiker te helpen de informatie te vinden die hij nodig heeft.

Voor gebruikers zonder een juridische achtergrond zijn deze toepassingen echter onvoldoende behulpzaam. Deze gebruikers zijn vaak op zoek naar het antwoord op een specifiek juridisch probleem, in plaats van alleen juridische informatie. Voor deze gebruikers is een verwijzing naar de relevante regelingen dan ook onvoldoende. Daarom wordt een tweede soort toepassingen ontwikkeld: systemen die de gebruiker vragen naar zijn situatie, die regels toepassen en een antwoord geven. Deze systemen zijn niet alleen nuttig voor burgers die een antwoord zoeken op hun juridische vraag, maar kunnen ook ambtenaren helpen bij het verwerken van routine zaken (bijvoorbeeld bij het verstrekken van vergunningen).

Voor toepassingen van deze tweede soort zijn complete, formele en uitvoerbare modellen van de wet nodig. Het maken van deze modellen kost veel inspanning. Dit is een probleem dat zich bij veel kennisgebaseerde systemen voordoet, en wordt ook wel de knowledge acquisition bottleneck genoemd. Het onderzoek dat hier wordt gepresenteerd is er op gericht om het effect van deze bottleneck te verminderen door Nederlandse wetten automatisch te vertalen naar computermodellen.

Deze automatische vertaling wordt niet in één stap gedaan. Het proces is opgesplitst in een aantal stappen:
1. structuur herkennen: de structuur van de wet wordt herkend en gemarkeerd;
2. verwijzingen zoeken: alle verwijzingen in de wet naar andere juridische bronnen worden opgezocht en gemarkeerd;
3. zinnen classificeren: de zinnen in de wet worden geclasseerd;
4. zinnen modelleren: van elke zin wordt een model gemaakt;
5. modellen integreren: de losse modellen per zin worden geïntegreerd tot één compleet model van de gehele wet.
De Aanwijzingen voor de Regelgeving vormen de basis voor het herkennen van structurelementen in Nederlandse regelgeving (beschreven in hoofdstuk 2). In de aanwijzingen wordt een wet verdeeld in zes onderdelen: opschrift, aanhef, lichaam, slotformulier, ondertekening en bijlagen. Het lichaam is vervolgens verdeeld in artikelen, die op hun beurt kunnen bestaan uit leden, en die lijsten met lijstonderdelen kunnen bevatten. De artikelen kunnen worden gegroepeerd in paragrafen, afdelingen, titels, hoofdstukken en boeken.

Deze verschillende onderdelen willen we kunnen herkennen en identificeren. Dat wil zeggen: we willen weten welke delen van een document bij elkaar horen, en we willen ook weten dat ze samen artikel 12 of hoofdstuk II vormen.

Voor de aanhef, het slotformulier en ondertekening van een regeling worden sjablonen gebruikt. Deze sjablonen worden voorgeschreven in de richtlijnen. Dit betekent dat ze eenvoudig opgespoord en geïdentificeerd kunnen worden door te zoeken naar de vaste onderdelen van deze sjablonen. De meeste andere onderdelen van een regeling zijn gemankeerd met een kop of een index. De kop bestaat uit de aanduiding van het niveau (zoals artikel of hoofdstuk), een index (zoals 4, II of 5:128) en eventueel een titel. Door deze koppen op te sporen kan het begin van een nieuw structurelement (en het einde van het vorige element) worden gevonden. Daarna moeten de artikelen en leden nog worden opgedeeld in zinnen. Dit kan worden gedaan met reeds bestaande toepassingen.

Met behulp van de General Architecture for Text Engineering (GATE) is een prototype parser gebouwd, gebaseerd op bovenstaande ideeën. Deze parser is getest op tien verschillende wetten. Op artikelniveau en hoger werd de structuur van de wetten foutloos bepaald. Binnen 2% van de artikelen was er een fout binnen de zinnen, lijsten of leden. Bij de overige 98% werd ook de structuur van de artikelen zonder fouten bepaald. De grootste uitdaging bij het bepalen van de structuur van regelgeving is het herkennen van wijzigingsteksten. Deze teksten kunnen structurelementen bevatten die geen onderdeel zijn van de (wijzigende) wet. In Nederlandse regelgeving zijn deze teksten niet gemankeerd met aanhalingstekens. Dit betekent dat de structurelementen die deel uitmaken van een wijzigingstekst herkend moeten worden door de indices te vergelijken met die van de naastliggende elementen (aangezien de indices van de elementen in de wijzigingstekst meestal geen nette reeks zullen vormen de indices van de omliggende tekst).

Net zoals koppelen volgen verwijzingen specifieke patronen, die gebruikt kunnen worden om ze op te sporen (zie hoofdstuk 3). Verwijzingen naar regelgeving worden gemaakt door middel van de (citeertit)el of de datum van ondertekening. Onderdelen van een document worden aangeduid door middel van het niveau (hoofdstuk, artikel, onderdeel) en de index. Een complete verwijzing bevat een aanduiding van het document waarnaar verwezen wordt, maar incomplete verwijzingen, zonder een dergelijke aanduiding, komen veel voor. Het doeldocument van dergelijke verwijzingen hangt af van de context waarin ze voorkomen. Binnen een wet verwijst een incomplete verwijzing vaak naar een ander deel van die wet, tenzij de verwijzing deel uitmaakt van een groep wijzigingsinstructies. In dat geval is het vaak een verwijzing naar een onderdeel van de wet die wordt gewijzigd. Verwijzingen kunnen ook naar
meerdere locaties verwijzen, door meerdere indices op te nemen (zoals artikelen 12, 13 en 14) of door ze te combineren in een reeks (artikelen 12-14).

De verwijzingen die bestaan uit een niveau-aanduiding en een index (of meerdere indices, of een reeks) kunnen eenvoudig worden opgespoord met patronen. Hetzelfde geldt voor verwijzingen die de datum van ondertekening gebruiken om te verwijzen naar een document. Daarentegen variëren citeertitels enorm. Deze kunnen daarom niet worden beschreven met een patroon. Om ze op te sporen is daarom een lijst met alle bestaande titels nodig.

Er is een prototype parser ontwikkeld gebaseerd op een dergelijke lijst van titels en patronen voor verwijzingen op basis van ondertekendatum en niveau-aanduiding en index. Deze parser is getest op zes Nederlandse wetten met meer dan 1.000 verwijzingen. De parser spoorde 97% van deze verwijzingen succesvol op, met nauwelijks fout-positieven.

In hoofdstuk 4 wordt de classificatie van zinnen in Nederlandse wetten behandeld. Er worden vijftien verschillende soorten zinnen onderscheiden. Een deel hiervan zijn de eigenlijke normzinnen: rechten/permissies en plichten. Deze zinnen worden ondersteund door definities, die de termen definiëren die door de normzinnen worden gebruikt. Gerelateerd aan de definities zijn de fictiebepaling, die bepaalde concepten gelijk verklaren voor de toepassing van bepaalde normen. Van toepassing verklaringen geven aan dat bepaalde normen wel (of niet) van toepassing zijn in bepaalde omstandigheden. Strafbepalingen stellen vast wat de straf is voor het overtreden van een norm. Een publicatiebepaling is een specifieke verplichting die opdracht geeft bepaalde informatie te publiceren. Een waardebepaling berekent een waarde, die binnen een norm wordt gebruikt.

Naast deze regels, die de eigenlijke wet vormen, zijn er ook een aantal zinnen die over de wet zelf gaan. Dit zijn zinnen die de datum van inwerkingtreding of de citeertitel vastleggen, en zinnen die het recht (of de plicht) tot het stellen van aanvullende regels delegeren. Andere zinnen wijzigen bestaande regelgeving door nieuwe tekst in te voeren, tekst te vervangen of te verwijderen, of door tekstelementen te hernummeren. Door middel van scopebepalingen wordt de scope voor zulke wijzigingen gezet, waarmee de verwijzingen in de tekst versimpeld worden.

Elke soort zinnen maakt gebruik van specifieke signaalwoorden en taalpatronen. Deze signaalwoorden en patronen kunnen worden gebruikt om zinnen te classificeren. Er is slechts één uitzondering. Hoewel veel verplichtingen signaalwoorden gebruiken, zoals moeten en verplicht, is het ook gebruikelijk om een verplichting te beschrijven als een feit. Dit wil zeggen dat de verplichting wordt beschreven alsof de betreffende actie altijd gebeurt (in plaats van dat hij moet gebeuren). In een dergelijke zin mist een signaalwoord zoals moeten. Aangezien alleen (bepaalde) verplichtingen geen gebruik maken van een patroon, kunnen de patronen nog steeds gebruikt worden voor de classificatie van zinnen. Als een zin voldoet aan geen enkel patroon, dan kunnen we aannemen dat het een verplichting is.

Er is een classifier gebouwd die gebruik maakt van dergelijke patronen. Deze classifier is getest op achtzien verschillende wetten met daarin 591 zinnen (lijsten buiten beschouwing latend). Van deze 591 zinnen waren er 157 verplichtingen die waren beschreven als een feit. De classifier was in staat om 91% van deze zinnen correct te classificeren.
Met betrekking tot het classificeren van tekst blijkt automatisch leren vaak succesvoller te zijn dan een kennisgebaseerde aanpak. Daarom is er, naast de classifier gebaseerd op patronen, ook een test uitgevoerd gebaseerd op automatisch leren, met Support Vector Machines (SVMs). Hierbij is gebruik gemaakt van een leave-one-out (LOO) aanpak met dezelfde dataset die bij het testen van de op patronen gebaseerde classifier is gebruikt. De LOO aanpak houdt in dat er een classifier wordt getraind op alle zinnen op één na. Deze laatste zin wordt vervolgens geclassificeerd met de gemaakte classifier. Dit wordt herhaald voor alle zinnen. Op basis van het aantal correcte classificaties kan vervolgens een LOO nauwkeurigheid worden bepaald. Dit is een voorspeller voor de nauwkeurigheid van de uiteindelijke classifier. Voor de classificatie van de zinnen in wetgeving haalde de aanpak gebaseerd op automatisch leren een nauwkeurigheid van 95%. In een test op twee nieuwe wetten classificeerde de automatisch leren classifier 95,77% respectievelijk 88,78% van de zinnen correct, vergeleken met 94,37% en 95,61% door de patroongebaseerde classifier. Dit suggereert dat geen van beide methodes strikt beter is dan de ander. De patroongebaseerde methode heeft wel tot voordeel dat het duidelijk is op basis van welke kenmerken de classificatie van een zin heeft plaatsgevonden.

De volgende stap is het eigenlijke modelleren, besproken in hoofdstuk 5. Bij het modelleren geldt dat er een belangrijk verschil is tussen de zinnen die gaan over de wet zelf en de zinnen die gaan over het onderwerp van de wet. De zinnen die over de wet gaan zijn de zinnen die bestaande wetten wijzigen, de datum van inwerkingtreding of de citeertitel van een wet vastleggen, etc. Deze zinnen beschrijven een specifieke situatie, en volgen een min of meer vast patroon. Dit is niet het geval voor de zinnen die over het onderwerp van de wet gaan: normen en definities. Deze zinnen beschrijven allerlei verschillende situaties en onderwerpen, en volgen geen vast patroon.

Voor de zinnen die over de wet zelf gaan kan er voor elke standaardzin worden bepaald welke onderdelen daarvan nodig zijn om een model van die zin te maken. Voor een model van een zin die tekst vervangt is bijvoorbeeld de locatie waar de tekst vervangen wordt, de te vervangen tekst en de vervangende tekst nodig. In deze zinnen is de locatie altijd aanwezig als een verwijzing, de tekst die wordt vervangen is gemaarkeerd met aanhalingstekens en de vervangende tekst volgt na de (eerste) dubbele punt in de zin. Voor andere zinnen die over de wet gaan, zoals het vastleggen van de citeertitel en het intrekken van een wet, kunnen soortgelijke regels worden gebruikt om de noodzakelijke informatie te verkrijgen. Een handmatige telling binnen een corpus bestaande uit 343 zinnen van deze soort suggereert dat ongeveer 96% van deze zinnen op deze manier kan worden gemaakte.

Voor de zinnen die over het onderwerp van de wet gaan bestaan dergelijke standaardzinnen niet. Het is dus ook niet voldoende om slechts voor een aantal standaardzinnen een vast te leggen welke informatie nodig is voor een model. Voor deze zinnen wordt daarom een generieker aanpak voorgesteld. Voor een normatieve zin wordt de actie of situatie die is toegestaan of verboden beschreven in een frame. Een dergelijk frame bevat informatie zoals de actie, de agens en patiëns. Deze informatie kan uit de zin worden gehaald middels bestaande natuurlijke taalverwerking toepassingen.

Dergelijke frames vormen een (rudimentair) instrument voor het toepassen van de regels. Door een casus in overeenkomstige termen te beschrijven (dus actie, agens, patiëns, etc.) kan
de situatie in de casus vergeleken worden met de situatie in de regels. Indien de situatie overeenkomt, is de regel van toepassing, en is duidelijk of volgens die regel de situatie is toegestaan of niet.

Een soortgelijke methode kan worden toegepast voor conditionele bijzinnen en definities. Als het gaat om een conditionele bijzin en de situatie casus komt overeen met die in de bijzin, dan geeft dit aan dat de regel (hoofdzin) waar de bijzin bij hoort moet worden toegepast. Als het gaat om een definitie, dan geeft een overeenkomst aan dat aan de definitie voldaan doen, en dat regels die van toepassing zijn op het gedefinieerde concept ook van toepassing zijn op deze casus.

Nadat de zinnen zijn gemodelleerd, moeten de losse modellen worden geïntegreerd. Modellen kunnen aan elkaar verbonden worden omdat de zinnen waar ze bij horen naar elkaar verwijzen middels een verwijzing, of doordat ze dezelfde concepten gebruiken. Deze verbanden zijn makkelijk op te sporen. Verwijzingen zijn al opgespoord, en als twee zinnen dezelfde concepten gebruiken, dan worden vaak ook dezelfde woorden gebruikt om dat concept aan te duiden. Echter, zinnen zijn ook vaak verbonden door *common sense* relaties tussen twee concepten die ze gebruiken. Zo zijn bijvoorbeeld de concepten *gift* en *gever* aan elkaar gerelateerd, en als gevolg daarvan zijn ook de zinnen die deze twee verschillende concepten gebruiken aan elkaar gerelateerd. Omdat deze relatie nergens expliciet gedefinieerd wordt in de wet kan dit verband niet automatisch worden gelegd, tenzij we deze kennis toevoegen aan het proces. De hoeveelheid van dergelijke *common sense* informatie is echter enorm, en het is dus onwaarschijnlijk dat alle benodigde informatie beschikbaar is in het automatisch modellerenproces. Dit betekent dat slechts een deel van de integratie automatisch kan worden uitgevoerd.

Aangezien de integratiestap niet automatisch kan worden uitgevoerd, is het niet mogelijk om volledig automatisch modellen van een wettekst te maken. Echter, computerprogramma’s kunnen wel helpen om de kennisacquisitie bottleneck te verminderen, aangezien ze succesvol zijn in het bepalen van de structuur van een wettekst, het opsporen van verwijzingen in die tekst, het classificeren van zinnen en het helpen modelleren van die zinnen. Daarnaast hebben ook de tussenresultaten van dit proces hun nut. Zo wordt het automatisch opsporen van verwijzingen gebruikt bij het bouwen van een semantisch netwerk voor de Belastingdienst, en zijn de patronen die zijn geïdentificeerd voor de zinsclassificatie gebruikt als sjablonen in MetaVex, een tekstverwerkingsprogramma voor wetgeving. Deze toepassingen worden beschreven in hoofdstuk 6.

Om de voorgestelde methode daadwerkelijk in gebruik te nemen, zal hij moeten worden verfijnd. De nauwkeurigheid kan worden verbeterd door meer patronen te verzamelen uit een grotere testset. Ook kunnen een aantal verbeteringen worden doorgevoerd die in dit proefschrift beschreven worden. Daarnaast kunnen wetgevingsjuristen ook helpen bij het verbeteren van de automatische verwerking van wetteksten. Sommige stukken informatie kunnen op een meer precieze manier onder woorden worden gebracht, die ze mogelijk ook beter te begrijpen maakt voor de menselijke lezer.

Niet alleen kan de nauwkeurigheid worden verbeterd; de methode kan ook worden uitgebreid. Met betrekking tot de classificatie zou het wenselijk kunnen zijn om meer subcategorieën te
herkennen. Ook is uitbreiding nodig naast wetten ook andere bronnen aan te kunnen, zoals Koninklijke Besluiten en jurisprudentie. Hiervoor zijn extra patronen nodig, die specifieker voor dergelijke teksten zijn.
Acknowledgements
Tom van Engers, my promoter, has been the instigator of this thesis. Without his encouragement, I would not have embarked upon this project. Also, he has worked hard to ensure that I kept looking beyond the borders of this project. Thanks, Tom.

Radboud Winkels, my co-promoter, has been my guide and sparring partner for this project. Without his assistance, many problems, big and small, would have taken a lot longer to get solved. Thanks, Radboud.

I would also like to thank four students who have contributed to my research as part of their own courses:

Marieke Franssen, for making an inventory of the different language patterns that are used for obligations and permissions;

Gijs Kruitbosch, for his work on programming the knowledge-based classifier for sentences in legal texts;

David de Bos, for running the first experiments with the knowledge-based classifier;

Kai Krabben, for running the experiments with the machine-learning classifier.
<table>
<thead>
<tr>
<th>SIKS Dissertation Series</th>
<th>2009-01</th>
<th>Rasa Jurgelenaite (RUN)</th>
<th>Symmetric Causal Independence Models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2009-02</td>
<td>Willem Robert van Hage (VU)</td>
<td>Evaluating Ontology-Aligment Techniques</td>
</tr>
<tr>
<td></td>
<td>2009-03</td>
<td>Hans Stol (UvT)</td>
<td>A Framework for Evidence-based Policy Making Using IT</td>
</tr>
<tr>
<td></td>
<td>2009-04</td>
<td>Josephine Nabukwanyi (RUN)</td>
<td>Improving the Quality of Organisational Policy Making using Collaboration Engineering</td>
</tr>
<tr>
<td></td>
<td>2009-05</td>
<td>Sietse Overbeek (RUN)</td>
<td>Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality</td>
</tr>
<tr>
<td></td>
<td>2009-06</td>
<td>Muhammad Subianto (UU)</td>
<td>Understanding Classification</td>
</tr>
<tr>
<td></td>
<td>2009-07</td>
<td>Ronald Poppe (UT)</td>
<td>Discriminative Vision-Based Recovery and Recognition of Human Motion</td>
</tr>
<tr>
<td></td>
<td>2009-08</td>
<td>Volker Nannen (VU)</td>
<td>Evolutionary Agent-Based Policy Analysis in Dynamic Environments</td>
</tr>
<tr>
<td></td>
<td>2009-09</td>
<td>Benjamin Kanagwa (RUN)</td>
<td>Design, Discovery and Construction of Service-oriented Systems</td>
</tr>
<tr>
<td></td>
<td>2009-10</td>
<td>Jan Wielemaker (UVA)</td>
<td>Logic programming for knowledge-intensive interactive applications</td>
</tr>
<tr>
<td></td>
<td>2009-11</td>
<td>Alexander Boer (UVA)</td>
<td>Legal Theory, Sources of Law &amp; the Semantic Web</td>
</tr>
<tr>
<td></td>
<td>2009-12</td>
<td>Peter Massarthe (TUE, Humboldt-Universitaet zu Berlin)</td>
<td>Operating Guidelines for Services</td>
</tr>
<tr>
<td></td>
<td>2009-13</td>
<td>Steven de Jong (UM)</td>
<td>Fairness in Multi-Agent Systems</td>
</tr>
<tr>
<td></td>
<td>2009-14</td>
<td>Maksym Korotkiy (VU)</td>
<td>From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)</td>
</tr>
<tr>
<td></td>
<td>2009-15</td>
<td>Rinke Hoekstra (UVA)</td>
<td>Ontology Representation - Design Patterns and Ontologies that Make Sense</td>
</tr>
<tr>
<td></td>
<td>2009-16</td>
<td>Fritz Reul (UvT)</td>
<td>New Architectures in Computer Chess</td>
</tr>
<tr>
<td></td>
<td>2009-17</td>
<td>Laurens van der Maaten (UvT)</td>
<td>Feature Extraction from Visual Data</td>
</tr>
<tr>
<td></td>
<td>2009-18</td>
<td>Fabian Groffen (CWI)</td>
<td>Armada, An Evolving Database System</td>
</tr>
<tr>
<td></td>
<td>2009-19</td>
<td>Valentin Robu (CWI)</td>
<td>Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets</td>
</tr>
</tbody>
</table>

2009-20 Bob van der Vecht (UU) | Adjustable Autonomy: Controlling Influences on Decision Making |
2009-21 Stijn Vanderlooy (UM) | Ranking and Reliable Classification |
2009-22 Pavel Serdyukov (UT) | Search For Expertise: Going beyond direct evidence |
2009-23 Peter Hofgesang (VU) | Modelling Web Usage in a Changing Environment |
2009-24 Annerieke Heuvelink (VUA) | Cognitive Models for Training Simulations |
2009-25 Alex van Ballegooij (CWI) | "RAM: Array Database Management through Relational Mapping" |
2009-26 Fernando Koch (UU) | An Agent-Based Model for the Development of Intelligent Mobile Services |
2009-27 Christian Glahn (OU) | Contextual Support of social Engagement and Reflection on the Web |
2009-28 Sander Evers (UT) | Sensor Data Management with Probabilistic Models |
2009-29 Stanislav Pokravec (UVA) | Model-Driven Semantic Integration of Service-Oriented Applications |
2009-30 Marcin Zukowski (CWI) | Balancing vectorized query execution with bandwidth-optimized storage |
2009-31 Sofiya Kattenko (UVA) | A Closer Look at Learning Relations from Text |
2009-32 Rik Farenhorst (VU) and Remco Hoer (VU) | Architectural Knowledge Management: Supporting Architects and Auditors |
2009-33 Khiet Truong (UT) | How Does Real Affect Affect Affect Recognition In Speech? |
2009-34 Inge van de Weerd (UU) | Advancing in Software Product Management: An Incremental Method Engineering Approach |
2009-35 Wouter Koelewijn (UL) | Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling |
2009-36 Marco Kalz (OUN) | Placement Support for Learners in Learning Networks |
2009-37 Hendrik Drachler (OUN) | Navigation Support for Learners in Informal Learning Networks |
2009-38 Riina Vuorikari (OU) | Tags and self-organisation: a metadata ecology for learning resources in a multilingual context |
2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution -- A Behavioral Approach Based on Petri Nets
2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language
2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings
2009-42 Toine Bogers (UvT)
Recommender Systems for Social Bookmarking
2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients
2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations
2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful
2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion
2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter
2010-02 Ingo Wassink (UT)
Work flows in Life Science
2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents
2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multi-display Environments
2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems
2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI
2010-07 Wim Fikkers (UT)
Gesture interaction at a Distance
2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software, Protecting user freedoms in a world of software communities and eGovernments
2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging
2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children
2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning
2010-12 Susan van den Braak (UvT)
Sensemaking software for crime analysis
2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques
2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration
2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models
2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice
2010-17 Spyros Kotsoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications
2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation
2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems
2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative
2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation
2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to Heterogeneous Linked Data
2010-23 Bas Steenbeek (UU)
The Logical Structure of Emotions
2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies
2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective
2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines
2010-27 Marten Voulon (UL)
Automatisch contracteren
2010-28 Arne Koopman (UU)
Characteristic Relational Patterns
2010-29 Stratos Ildiros(CWI)
Database Cracking: Towards Auto-tuning Database Kernels
2010-30 Marnieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval
2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web
2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems
2010-33 Robin Aly (UT) Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dingahaya (UT) Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT) Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU) Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE) Correctness of services and their composition

2010-38 Dirk Fahland (TUE) From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU) Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU) Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM) Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU) Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU) A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE) An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT) A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU) e3Alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT) Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Witthaut

2010-49 Iohn Takeshi Saito (UM) Solving difficult game positions

2010-50 Bouke Huurnink (UVA) Search in Audiovisual Broadcast Archives

2010-51 Alla Khatri Amin (CWI) Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU) Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA) Combining Concepts and Language Models for Information Access

2010-54 Botond Cseke (RUN) Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2010-55 Nick Timmermeier (UU) Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2010-56 Jan Martijn van der Werf (TUE) Compositional Design and Verification of Component-Based Information Systems

2010-57 Hado van Hasselt (UU) Insights in Reinforcement Learning: Formal analysis and empirical evaluation of temporal-difference learning algorithms

2010-58 Base van der Raadt (UU) Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2010-59 Yiwen Wang (TUE) Semantically-Enhanced Recommendations in Cultural Heritage

2010-60 Tim de Jong (OU) Contextualised Mobile Media for Learning

2011-01 Bart Bogaert (UvT) Cloud Content Contention

2011-02 Dhaval Vyas (UT) Designing for Awareness: An Experience-focused HCI Perspective

2011-03 Carmen Bratosin (TUE) Grid Architecture for Distributed Process Mining

2011-04 Xiaoyu Mao (UvT) Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-05 Nieske Vergunst (UU) BDI-based Generation of Robust Task-Oriented Dialogues

2011-06 Milan Lovric (EUR) Behavioral Finance and Agent-Based Artificial Markets

2011-07 Marijn Koolen (UvA) The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-08 Maarten Schadd (UM) Selective Search in Games of Different Complexity

2011-09 Jiyin He (UVA) Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-10 Mark Ponsen (UM) Strategic Decision-Making in complex games
<table>
<thead>
<tr>
<th>Year</th>
<th>Name</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>Ellen Rusman (OU)</td>
<td>The Mind ‘s Eye on Personal Profiles</td>
</tr>
<tr>
<td>2011</td>
<td>Qing Gu (VU)</td>
<td>Guiding service-oriented software engineering - A view-based approach</td>
</tr>
<tr>
<td>2011</td>
<td>Linda Terlouw (TUD)</td>
<td>Modularization and Specification of Service-Oriented Systems</td>
</tr>
<tr>
<td>2011</td>
<td>Junte Zhang (UVA)</td>
<td>System Evaluation of Archival Description and Access</td>
</tr>
<tr>
<td>2011</td>
<td>Wouter Weerkamp (UVA)</td>
<td>Finding People and their Utterances in Social Media</td>
</tr>
<tr>
<td>2011</td>
<td>Herwin van Welbergen (UT)</td>
<td>Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior</td>
</tr>
<tr>
<td>2011</td>
<td>Syed Waqar ul Qounain Jaffry (VU)</td>
<td>Analysis and Validation of Models for Trust Dynamics</td>
</tr>
<tr>
<td>2011</td>
<td>Matthijs Aart Pontier (VU)</td>
<td>Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots</td>
</tr>
<tr>
<td>2011</td>
<td>Aniel Bhulai (VU)</td>
<td>Dynamic website optimization through autonomous management of design patterns</td>
</tr>
<tr>
<td>2011</td>
<td>Rianne Kaptein(UVA)</td>
<td>Effective Focused Retrieval by Exploiting Query Context and Document Structure</td>
</tr>
<tr>
<td>2011</td>
<td>Faisal Kamiran (TUE)</td>
<td>Discrimination-aware Classification</td>
</tr>
<tr>
<td>2011</td>
<td>Egon van den Broek (UT)</td>
<td>Affective Signal Processing (ASP): Unraveling the mystery of emotions</td>
</tr>
<tr>
<td>2011</td>
<td>Ludo Waltman (EUR)</td>
<td>Computational and Game-Theoretic Approaches for Modeling Bounded Rationality</td>
</tr>
<tr>
<td>2011</td>
<td>Nees-Jan van Eck (EUR)</td>
<td>Methodological Advances in Bibliometric Mapping of Science</td>
</tr>
<tr>
<td>2011</td>
<td>Tom van der Weide (UU)</td>
<td>Arguing to Motivate Decisions</td>
</tr>
<tr>
<td>2011</td>
<td>Paolo Tuurini (UU)</td>
<td>Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations</td>
</tr>
<tr>
<td>2011</td>
<td>Maaike Harbers (UU)</td>
<td>Explaining Agent Behavior in Virtual Training</td>
</tr>
<tr>
<td>2011</td>
<td>Erik van der Spek (UU)</td>
<td>Experiments in serious game design: a cognitive approach</td>
</tr>
<tr>
<td>2011</td>
<td>Adriana Burlatia (RUN)</td>
<td>Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference</td>
</tr>
<tr>
<td>2011</td>
<td>Nyree Lemmens (UM)</td>
<td>Bee-inspired Distributed Optimization</td>
</tr>
<tr>
<td>2011</td>
<td>Joost Westra (UU)</td>
<td>Organizing Adaptation using Agents in Serious Games</td>
</tr>
<tr>
<td>2011</td>
<td>Viktor Clerc (VU)</td>
<td>Architectural Knowledge Management in Global Software Development</td>
</tr>
<tr>
<td>2011</td>
<td>Luu Alam (UT)</td>
<td>Cryptographically Enforced Distributed Data Access Control</td>
</tr>
<tr>
<td>2011</td>
<td>Michal Sindlar (UI)</td>
<td>Explaining Behavior through Mental State Attribution</td>
</tr>
<tr>
<td>2011</td>
<td>Henk van der Schuur (UU)</td>
<td>Process Improvement through Software Operation Knowledge</td>
</tr>
<tr>
<td>2011</td>
<td>Boris Reuderink (UT)</td>
<td>Robust Brain-Computer Interfaces</td>
</tr>
<tr>
<td>2011</td>
<td>Herman Stehouwer (UvT)</td>
<td>Statistical Language Models for Alternative Sequence Selection</td>
</tr>
<tr>
<td>2011</td>
<td>Beibei Hu (TUD)</td>
<td>Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work</td>
</tr>
<tr>
<td>2011</td>
<td>Azizi Bin Ab Aziz(VU)</td>
<td>Exploting Computational Models for Intelligent Support of Persons with Depression</td>
</tr>
<tr>
<td>2011</td>
<td>Mark Ter Maat (UT)</td>
<td>Response Selection and Turn-taking for a Sensitive Artificial Listening Agent</td>
</tr>
<tr>
<td>2011</td>
<td>Andrea Niculescu (UT)</td>
<td>Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality</td>
</tr>
<tr>
<td>2011</td>
<td>Terry Kakeeto (UvT)</td>
<td>Relationship Marketing for SMEs in Uganda</td>
</tr>
<tr>
<td>2011</td>
<td>Muhammad Umair(VU)</td>
<td>Adaptivity, emotion, and Rationality in Human and Ambient Agent Models</td>
</tr>
<tr>
<td>2011</td>
<td>Adam Vany (VU)</td>
<td>Supporting Architecture Evolution by Mining Software Repositories</td>
</tr>
<tr>
<td>2011</td>
<td>Jurtaan Souer (UU)</td>
<td>Development of Content Management System-based Web Applications</td>
</tr>
<tr>
<td>2011</td>
<td>Martijn Plomp (UU)</td>
<td>Maturing Interorganisational Information Systems</td>
</tr>
<tr>
<td>Year</td>
<td>Title</td>
<td>Author(s)</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>2012-06</td>
<td>Awareness Support for Knowledge Workers in Research Networks</td>
<td>Wolfgang Reinhardt (OU)</td>
</tr>
<tr>
<td>2012-07</td>
<td>When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions</td>
<td>Rianne van Lambalgen (VU)</td>
</tr>
<tr>
<td>2012-08</td>
<td>Kernel Methods for Vessel Trajectories</td>
<td>Gerben de Vries (UVA)</td>
</tr>
<tr>
<td>2012-09</td>
<td>Trust and Privacy Management Support for Context-Aware Service Platforms</td>
<td>Ricardo Neisse (UT)</td>
</tr>
<tr>
<td>2012-10</td>
<td>Towards a Generic Distributed Adaptive Hypermedia Environment</td>
<td>David Smis (TUE)</td>
</tr>
<tr>
<td>2012-11</td>
<td>Process Mining in the Large: Preprocessing, Discovery, and Diagnostics</td>
<td>J.C.B. Rantham Prabhakara (TUE)</td>
</tr>
<tr>
<td>2012-12</td>
<td>Model Driven Design and Data Integration in Semantic Web Information Systems</td>
<td>Kees van der Sluijs (TUE)</td>
</tr>
<tr>
<td>2012-13</td>
<td>Fun and Face: Exploring non-verbal expressions of emotion during playful interactions</td>
<td>Suleman Shahid (UvT)</td>
</tr>
<tr>
<td>2012-14</td>
<td>Generic Adaptation Framework for Unifying Adaptive Web-based Systems</td>
<td>Evgeny Knutov (TUE)</td>
</tr>
<tr>
<td>2012-15</td>
<td>Social Agents, Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.</td>
<td>Natalie van der Wal (VU)</td>
</tr>
<tr>
<td>2012-16</td>
<td>Helping people by understanding them - Ambient Agents supporting task execution and depression treatment</td>
<td>Fiemke Both (VU)</td>
</tr>
<tr>
<td>2012-17</td>
<td>Towards a Comprehensive Framework for Business Process Compliance</td>
<td>Amal Elgammal (UvT)</td>
</tr>
<tr>
<td>2012-18</td>
<td>Improving Solution Architecting Practices</td>
<td>Eltjo Poort (VU)</td>
</tr>
<tr>
<td>2012-19</td>
<td>What's Next? Operational Support for Business Process Execution</td>
<td>Helen Schonenberg (TUE)</td>
</tr>
<tr>
<td>2012-20</td>
<td>Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing</td>
<td>Ali Bahramisharif (RUN)</td>
</tr>
<tr>
<td>2012-21</td>
<td>Querying Sparse Matrices for Information Retrieval</td>
<td>Roberto Cornacchia (TUD)</td>
</tr>
<tr>
<td>2012-22</td>
<td>Intelligence, politie en veiligheidsdienst: verenigebare grootheden?</td>
<td>Thijs Vis (UvT)</td>
</tr>
<tr>
<td>2012-23</td>
<td>Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction</td>
<td>Christian Muehl (UT)</td>
</tr>
<tr>
<td>2012-24</td>
<td>Evaluation of Noisy Transcripts for Spoken Document Retrieval</td>
<td>Laurens van der Werff (UT)</td>
</tr>
<tr>
<td>2012-25</td>
<td>Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application</td>
<td>Silja Eckarra (UT)</td>
</tr>
</tbody>
</table>