A Meta-level Annotation Language for Legal Texts

Tomer Libal(✉)

University of Luxembourg, Luxembourg City, Luxembourg
shaolintl@gmail.com

Abstract. There are many legal texts which can greatly benefit from the support of automated reasoning. Such support depends on the existence of a logical formalization of the legal text. Among the methods used for the creation of these knowledge bases, annotation tools attempt to abstract over the logical language and support non-logicians in their efforts to formalize documents. Nevertheless, legal documents use a rich language which is not easy to annotate. In this paper, an existing annotation tool is being extended in order to support the formalization of a complex example - the GDPR's article 13. The complexity of the article prevents a direct annotation using logical and deontical operators. This is overcome by the implementation of several macros. We demonstrate the automated reasoning over the formalized article and argue that macros can be used to formalize complex legal texts.

Keywords: Automated reasoning · Knowledge bases · Annotation tools

1 Introduction

Computer systems are playing a substantial role in assisting people in a wide range of tasks, including searching in large data and decision-making; and their employment is progressively becoming vital in an increasing number of fields. One of these fields is legal reasoning: New court cases and legislations are accumulated every day and navigating through the vast amount of complex information is far from trivial. In addition, the understanding of those texts is reserved only for experts in the legal domain despite the fact that they are usually of interest to the general public.

A key component of legal reasoning is the transformation of legal texts into a machine readable format. This transformation must capture the legal understanding of the text in order to allow computers to reason over it.

Supporting automated reasoning over legislation is an old idea, dating back to LEGOL [23] and made popular by the formalisation of the British Nationality Act [21]. These approaches and others (see for example [4,22]) were based on the use of the Prolog programming language for the formalization of the legislation. Since then, many other systems have followed the same path and the formalization of legal texts in Prolog is also done today [13].

Prolog is very suitable for such formalizations but still depends on the works of programmers and logicians. In order to verify that the formalization is correct, methodologies were created which allow legal experts to be able to give back feedback to the programmers and logicians [1]. In addition to Prolog-based, legal knowledge bases were created which are based on other logical formalizations, such as IO logics [19] and modal logics [11].

In order to allow legal experts to create knowledge bases directly, user friendly interfaces can be created which aim at hiding the logical complexity. Annotation editors for legal texts [12,16] allow users to add a legal interpretation to texts by the use of annotations. The editor then produces a logical formalization which can be used for automated reasoning.

Despite the advances described above, "good" logical formalizations are hard to achieve [3]. Among the most important properties of such formalizations, one can list faithfulness to the original text, efficient support for the required reasoning operations and being well engineered [20]. In order to be faithful to the original text, one must not only describe the legal terms and their logical relations in a faithful way but also meta-level concepts such exceptions, counterfactuals and deeming provisions. This normally increases the complexity of the formalization. Being well engineered, on the other hand, normally means being simple and easy to verify, validate, update and maintain.

The tension between the two can be demonstrated by looking at the third paragraph of article 13 of the GDPR[1]:

"Where the controller intends to further process the personal data for a purpose other than that for which the personal data were collected, the controller shall provide the data subject prior to that further processing with information on that other purpose and with any relevant further information as referred to in paragraph 2.".

This paragraph discusses cases not handled by previous paragraphs and ask to apply certain points in a new context. Attempting to faithfully represent this paragraph while keeping to the best engineering principals is not easy.

One approach for tackling such problems is to manually simplify the structure of the sentence such that logical annotations then become easier. In the example above, one can copy the relevant parts of the previous paragraphs and replace parts of their context with the one of paragraph 3.

From the engineering point of view, there are several problems with this approach. First, manual processing and copying is error-prone. By delegating work to the computer, we decrease the chance of error as long as the algorithm is correct. Second, it might be a tedious and time-consuming work which can be automated by a computer. Lastly, the manual work needs to be repeated when a change to the understanding or context of the legislation occurs.

In this paper, an extension to the NAI Suite's annotation editor [12] is discussed. The NAI Suite follows a certain methodology which aims at being faithful and well engineered at the same time. This methodology, closely related to the Isomorphism approach [2], uses two levels of annotations in order to create an

[1] https://eur-lex.europa.eu/eli/reg/2016/679/oj.

intermediary representation of the legal text which is faithful and well engineered, in the sense defined above. This intermediary representation is then translated automatically into logical representations which can then be used for efficient automated reasoning.

This methodology is still insufficient when facing the GDPR's paragraph from above. The two levels of annotations which are supported cover vocabulary and logical relations, but the paragraph requires complex modifications of a cross referenced paragraph as well.

The extension to the NAI Suite discussed in this paper supports an additional level of annotations. These meta-level annotations can handle complex legal structures, such as the one in the above example. In addition to this extension, a domain specific language (DSL) for describing meta-level annotations is presented and applied to these annotations. It is then being shown that with hardly any change to the original text, a formalization is created which can be used for automated reasoning. Some examples of automated query answering are demonstrated.

In the next section, an introduction to the NAI Suite, its theoretical foundations and its graphical user interface is given. The following section describes the extension of the tool, while Sect. 4 demonstrates how the new features can be applied to easily formalize and reason over article 13. A conclusion and future work discussion are given last.

2 The NAI Suite

The NAI suite integrates novel theorem proving technology into a usable graphical user interface (GUI) for the computer-assisted formalization of legal texts and applying automated normative reasoning procedures on these artifacts. In particular, NAI includes

1. a legislation editor that graphically supports the formalization of legal texts,
2. means of assessing the quality of entered formalizations, e.g., by automatically conducting consistency checks and assessing logical independence,
3. ready-to-use theorem prover technology for evaluating user-specified queries wrt. a given formalization, and
4. the possibility to share and collaborate, and to experiment with different formalizations and underlying logics.

NAI is realized using a web-based Software-as-a-service architecture, cf. Fig. 1. It comprises a GUI that is implemented as a Javascript browser application, and a NodeJS application on the back-end side which connects to theorem provers, data storage services and relevant middleware. Using this architectural layout, no further software is required from the user perspective for using NAI and its reasoning procedures, as all necessary software is made available on the back end and the computationally heavy tasks are executed on the remote servers only. The results of the different reasoning procedures are sent back to the GUI and displayed to the user. The major components of NAI are described in more detail in the following.

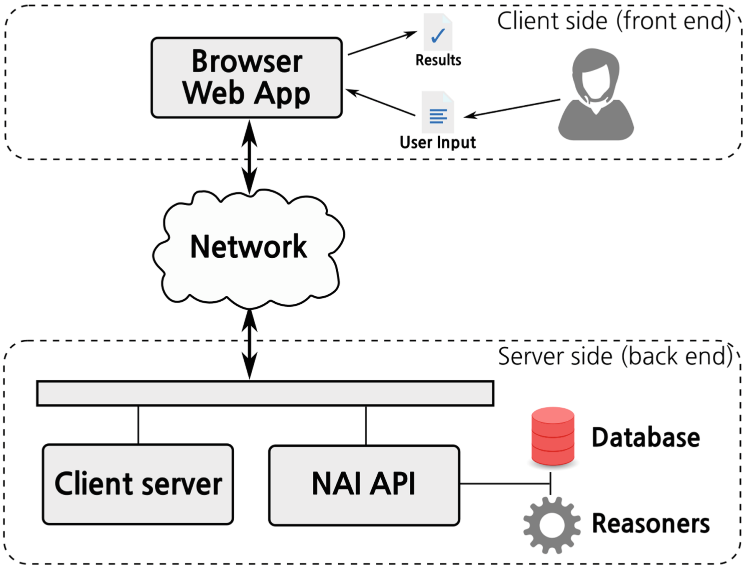


Fig. 1. Software-as-a-service architecture of the NAI reasoning framework. The front end software runs in the user's browser and connects to the remote site, and its different services, via a well-defined API through the network. Data flow is indicated by arrows.

2.1 The Underlining Logic

The logical formalism underlying the NAI framework is based on a universal fragment first-order variant of the deontic logic **DL*** [11], denoted **DL***$_1$. Its syntax is given by

Definition 1 (Syntax of DL*$_1$). *Let V, P and F be disjoint sets of symbols for variables, predicate symbols (of some arity) and function symbols (of some arity), respectively.* ***DL****$_1$ *formulas ϕ, ψ are given by:*

$$\begin{aligned}\phi, \psi ::= {} & p(t_1, \ldots, t_n) \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \Rightarrow \psi \\ & \mid \mathsf{Id}\,\phi \mid \mathsf{Ob}\,\phi \mid \mathsf{Pm}\,\phi \mid \mathsf{Fb}\,\phi \\ & \mid \phi \Rightarrow_{\mathsf{Ob}} \psi \mid \phi \Rightarrow_{\mathsf{Pm}} \psi \mid \phi \Rightarrow_{\mathsf{Fb}} \psi\end{aligned}$$

where $p \in P$ is a predicate symbol of arity $n \geq 0$ and the t_i, $1 \leq i \leq n$, are terms. Terms are freely generated by the function symbols from F and variables from V. ⌟

DL*$_1$ extends Standard Deontic Logic (SDL) with the normative concepts of ideal and contrary-to-duty obligations, and contains predicate symbols, the standard logical connectives, and the normative operators of obligation (Ob), permission (Pm), prohibition (Fb), their conditional counter-parts, and ideality (Id). Free variables are implicitly universally quantified at top-level.

This logic is expressive enough to capture many interesting normative structures. For details on its expressivity and its semantics, we refer to previous work [11].

2.2 The Reasoning Module

The NAI suite supports formalizing legal texts and applying various logical operations on them. These operations include consistency checks (non-derivability of falsum), logical independence analysis as well as the creation of user queries that can automatically be assessed for (non-)validity. After formalization, the formal representation of the legal text is stored in a general and expressive machine-readable format in NAI. This format aims at generalizing from concrete logical formalisms that are used for evaluating the logical properties of the legal document's formal representation.

There exist many different logical formalisms that have been discussed for capturing normative reasoning and extensions of it. Since the discussion of such formalisms is still ongoing, and the choice of the concrete logic underlying the reasoning process strongly influences the results of all procedures, NAI uses a two-step procedure to employ automated reasoning tools. NAI stores only the general format, as mentioned above, as result of the formalization process. Once a user then chooses a certain logic for conducting the logical analysis, NAI will automatically translate the general format into the specific logic resp. the concrete input format of the employed automated reasoning system. Currently, NAI supports only the $\mathbf{DL}^{*}{}_{1}$ logic from Sect. 2.1; however, the architecture of NAI is designed in such a way that further formalisms can easily be supported.

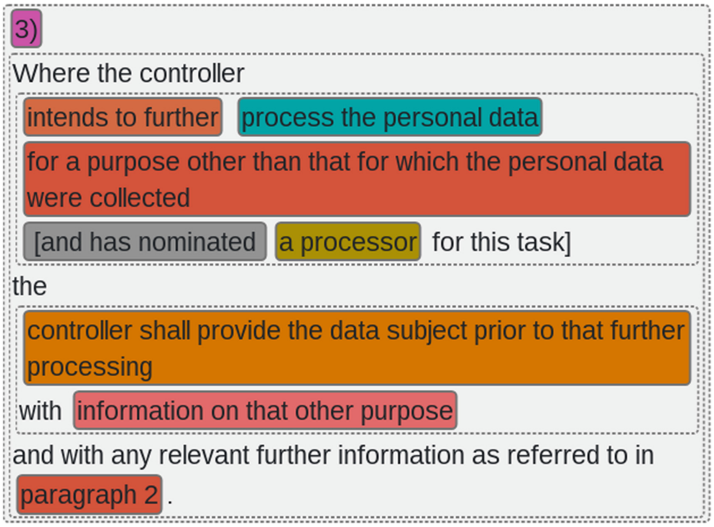
The choice in favor of $\mathbf{DL}^{*}{}_{1}$ is primarily motivated by the fact that it can be effectively automated using a shallow semantical embedding into normal (bi-)modal logic [11]. This enables the use of readily available reasoning systems for such logics; in contrast, there are few to none automated reasoning systems available for normative logics (with the exception of [9]). In NAI, we use the MleanCoP prover [15] for first-order multi-modal logics as it is currently one of the most effective systems and it returns proof certificates which can be independently assessed for correctness [14]. It is also possible to use various different tools for automated reasoning in parallel (where applicable). This is of increasing importance once multiple different logical formalisms are supported.

2.3 The Annotation Editor

The annotation editor of NAI is one of its central components. Using the editor, users can create formalizations of legal documents that can subsequently used for formal legal reasoning. The general functionality of the editor is described in the following.

One of the main ideas of the NAI editor is to hide the underlying logical details and technical reasoning input and outputs from the user. We consider this essential, as the primary target audience of the NAI suite are not necessarily logicians and it could greatly decrease the usability of the tool if a solid knowledge about formal logic was required. This is realized by letting the user annotate legal texts and queries graphically and by allowing the user to access the different reasoning functionalities by simply clicking buttons that are integrated into the GUI. Note that the user can still inspect the logical formulae that result from the annotation process and also input these formulae directly. However, this feature is considered advanced and not the primary approach put forward by NAI.

The formalization proceeds as follows: The user selects some text from the legal document and annotates it, either as a term or as a composite (complex) statement. In the first case, a name for that term is computed automatically, but it can also be chosen freely. Different terms are displayed as different colors in the text. In the latter case, the user needs to choose among the different possibilities (which roughly correspond to logical connectives) and the containing text can be annotated recursively. Composite statements are displayed as a box around the text. An example of an annotation result is displayed in Fig. 2.



3)
Where the controller
intends to further process the personal data
for a purpose other than that for which the personal data were collected
[and has nominated a processor for this task]
the
controller shall provide the data subject prior to that further processing
with information on that other purpose
and with any relevant further information as referred to in paragraph 2 .

Fig. 2. GDPR article 13, paragprah 3: annotation

The editor also features direct access to the consistency check and logical independence check procedures (as buttons). When such a button is clicked, the current state of the formalization will be translated and sent to the back-end provers, which determine whether it is consistent resp. logically independent.

User queries are also created using such an editor. In addition to the steps sketched above, users may declare a text passage as *goal* using a dedicated annotation button, whose contents are again annotated as usual. If the query is executed, the back-end provers will try to prove (or refute) that the goal logically follows from the remaining annotations and the underlying legislation.

2.4 The Abstract Programming Interface (API)

All the reasoning features of NAI can also be accessed by third-party applications. The NAI suite exposes a RESTful (Representational state transfer) API which allows (external) applications to run consistency checks, checks for independence as well as queries and use the result for further processing. The exposure of NAI's REST API is particularly interesting for external legal applications that want to make use of the already formalized legal documents hosted by NAI. A simple example of such an application is a tax counseling web site which advises its visitors using legal reasoning over a formalization of the relevant tax law done in the NAI suite.

3 A Meta-level Annotation Language

An essential element in the compliance checking of privacy policies and data collection procedures, the GDPR's article 13[2] is concerned with their transparency. This article contains 4 paragraphs, where the first two contain each 6 subsections. The third paragraph extends and modify the second one while the last states a situation in which the three previous paragraphs do not hold.

Let us consider the first paragraph.

Paragraph 1 of GDPR's Article 13: Where personal data relating to a data subject are collected from the data subject, the controller shall, at the time when personal data are obtained, provide the data subject with all of the following information:

(a) the identity and the contact details of the controller and, where applicable, of the controller's representative;
(b) the contact details of the data protection officer, where applicable;
(c) the purposes of the processing for which the personal data are intended as well as the legal basis for the processing;
(d) where the processing is based on point (f) of Article 6(1), the legitimate interests pursued by the controller or by a third party;
(e) the recipients or categories of recipients of the personal data, if any;
(f) where applicable, the fact that the controller intends to transfer personal data to a third country or international organisation and the existence or absence of an adequacy decision by the Commission, or in the case of transfers referred to in Article 46 or 47, or the second subparagraph of Article 49(1), reference to the appropriate or suitable safeguards and the means by which to obtain a copy of them or where they have been made available.

The above paragraph is not easy to read but is even harder to annotate. There are several reasons.

[2] https://eur-lex.europa.eu/eli/reg/2016/679/oj.

- The structure of the paragraph is not trivial. It starts by declaring some conditions, then it states the obligation to communicate information. The exact information to communicate and other possible conditions are then specified in each item.
- Most conditions and the obligation are specified once only but the content of the obligation, i.e. the precise information which should be communicated to the data subject, is changing for each point.

In order to understand the paragraph, the reader is expected to reconstruct each of the 6 items with the relevant further conditions and the precise content of the obligation.

Clearly, new types of annotations must be added. For example, one can consider changing the precise information to communicate in each item as a replacement operation. Such operations are normally not a part of any logical language but of their meta-language. We need therefore a new kind of annotations for annotating meta-level concepts.

An even more complex structure appears in the third paragraph.

Paragraph 3 of GDPR's Article 13: Where the controller intends to further process the personal data for a purpose other than that for which the personal data were collected, the controller shall provide the data subject prior to that further processing with information on that other purpose and with any relevant further information as referred to in paragraph 2.

Here, in addition to all the issues which were just discussed, the annotation should also apply it to another, already existing, paragraph. The annotation should not only add further conditions to the referenced paragraph but modify it as well. For example, The sentence "the controller shall provide the data subject prior to that further processing with information on that other purpose and with any relevant further information as referred to in paragraph 2." requires the reader to adapt the obligations of paragraph 2 to the new processing.

Two possible solutions for annotating such complex legal structure come to mind. First, one can ask the user to simplify the structure manually. In the above case, the user will transform the complex sentence into many simple ones and will take care of replacing different values in the right places. This is the approach taken by current formalizations of the GDPR [19].

The solution suggested in this paper is to automatize this process by providing an additional layer of annotations for describing the complex structures of legal texts. These annotations will be called **macros**. Macros are tailored to specific situations and are capable of arbitrary modifications of the result. For example, in the case of paragraph 1, a macro could take the original sentence and break it down automatically into the several required sentences.

From the engineering point of view, the second approach is better. First, it saves the user from the need to copy and duplicate parts of the sentence. In addition, it generates automatically the relevant conditions of each item and thus is more rigorous. Lastly, by automatizing parts of the formalization process, future editions of this paragraph become easier - one just needs to change the relevant parts of the paper and the macro can be called again.

Clearly, a disadvantage of such an approach is the possible high number of required macros. For the purpose of having a precise and simple annotation of article 13, there is a need of 4 such macros. Nevertheless, as we will see next, these macros are general in nature and will possibly fit a wide range of legal sentences, which are usually of a restricted form. As a possible extension of the work discussed in this paper, there is also a plan to add to the NAI suite the ability to design new macros using a Domain Specific Language (DSL).

A possible such DSL is defined next. This DSL will be used in this paper to describe the macros which are required for the formalization of article 13.

Definition 2 (Annotation's syntax). *Annotations will be denoted using English capital letters $A, B, \ldots$ with possibly subscripts and superscripts. There are two types of annotations. Simple annotations, denoted by A^s are term annotations (also called vocabulary annotations), i.e. annotations applied to term and therefore do not contain any nested annotation. Complex annotations are of the form $\texttt{ANNOT}(args)$ where $\texttt{ANNOT}$ is the name of the logical connective which is used for the annotation and args is an ordered list of the top level annotations which are included in it. The connectives were described in Sect. 2.1.*

For example, a conjunction annotation over the term annotations $t(a, b)$ and $s(X, b)$ is denoted as $\texttt{AND}(t(a, b), s(X, b))$.

Definition 3 (Parsing state). *Labels are defined as simple annotations which denote a name and are normally purely propositional. The parsing state is a pair $(annots, map)$ where annots is an array of annotations which were extracted from the annotated texts and map is a mapping between labels and annotations. We denote the map which is obtained by setting the value y for the label x in map by $map(x, y)$. We denote the value which is associated with the label x in map by $map[x]$.*

Definition 4 (Macros). *A macro is a transformation from one parsing state into another and is denoted by $(annots_1, map_1) \Longrightarrow (annots_2, map_2)$. Macros normally apply to only one annotation J_1 in the annots array. In these cases, we will simplify the notation and write $(J_1, map_1) \Longrightarrow (J_2, map_2)$. In addition, when the map does not change, we will sometimes further simplify the notation and write $J_1 \Longrightarrow J_2$.*

Lastly, we need to define occurrences of subterm annotations within annotations in order to be able to formally desscribe replacements.

Definition 5 (Subterm occurrences). *Given an arbitrary annotation A, we denote by $A[x]$ all occurrences of a subterm annotation x appearing in it. In the definition of a macro $(A[x], map_1) \Longrightarrow (A[y], map_2)$, $A[y]$ on the right hand side is obtained from $A[x]$ by replacing all occurrences of x in A with y.*

For example, assuming that the conjunction in the previous example is denoted by A, the occurrences of the subterm b in it are denoted by $A[b]$.

We can denote the macro which replaces those occurrences of b with c using $A[b] \Longrightarrow A[c]$.

We can now give a formal definition of the specific macros.

The Multi-obligation Macro. This macro takes two annotations, where the second annotation has the following restrictions. First, it must be a conjunction. Second, each conjunct except the first must be either a term, an "If/Then" or an "Always/If" annotation. Third, its first conjunct must be a term which contains the placeholder `VAR`. This placeholder can also appear anywhere in the first annotation.

Definition 6 (The multi-obligation macro). *The multi-obligation macro is defined by* $(\mathtt{M\text{-}OBS}(C[\mathtt{VAR}], \mathtt{AND}(A[\mathtt{VAR}], B_1, .., B_n)), map)$
$\Longrightarrow$
$(\mathtt{AND}(\mathtt{IF\text{-}THEN\text{-}OB}(\mathtt{AND}(C[B_1^1], B_1^0), A[B_1^1]), \ldots,$
$\mathtt{IF\text{-}THEN\text{-}OB}(\mathtt{AND}(C[B_n^1], B_n^0), A[B_n^1])), map)$.
Where

- $n \geq 1$
- *For each* $0 < i \leq n$, B_i *is one of the following*
  - $\mathtt{IF\text{-}THEN}(B_i^0, B_i^1)$
  - $\mathtt{ALWAYS\text{-}IF}(B_i^1, B_i^0)$
  - *A simple annotation* B_i^1. *In this case* B_i^0 *is empty.*

Informally, when applied to two annotations, the macro does the following. For each conjunct beyond the first in the second annotation, the macro creates a new conditional obligation. The type of obligation and the form of conditions is defined according to the type of annotation:

- In case the annotation is simple, the set of conditions is the one defined in the first annotation and the obligation is the first conjunct, where the `VAR` placeholder is replaced by the simple annotation.
- In case the annotation is an "If/Then" or an "Always/If", we define the condition part of the annotation to be the first formula of the "If/Then" annotation and the second in "Always/If", while the conclusion part is defined to be the second formula and first, respectively. The set of conditions is a conjunction of the one defined in the first annotation and conditions of the complex annotation. The obligation is the first conjunct, where the `VAR` placeholder is replaced by the conclusion of the complex annotation.

Similarly to the above, any occurrence of the placeholder `VAR` in the conditions is replaced with the same term as in the obligation.

Paragraph 3 is relatively short syntactically but complex semantically. It expands on paragraph 2 and places further conditions and obligations. While paragraph 2 describes the obligations in case of the first data processing, paragraph 3 describes those in all subsequent ones. Clearly, additional macros are required. The first macro is used for cross-reference and allows the users of the editor to label annotated sentences with a certain name. Using this name, a second macro then takes the referenced annotated sentence, copies it and replaces relevant parts.

The Labeling Macro. This macro, which helps other macros to function by changing the state, expects two annotations. The first is a simple annotation denoting the label while the second can be any possible annotation. It enables the use of labels in other macros which are defined later. In our example, the simple annotation is just the term `a13_p2` which is used to name the second paragraph, while the second annotation is the whole content of the second paragraph.

Definition 7 (The labeling macro). *The labeling macro is defined as follows* $(\texttt{LABEL}(A^s, B), map) \Longrightarrow (B, map(A^s, B))$

The Copying Macro. This macro takes three arguments. An optional annotation containing further conditions, a conjunction of further obligations of the form stated in the Multi-obligation Macro and a label which is used in order to copy an annotated sentence. It uses the first conjunct of the second annotation to replace the obligation of the copied annotation and adds further obligations according to the other conjuncts. In our example, this macro copies the second paragraph, while adding further conditions referring to subsequent processing and the order between them. It then replaces the obligation to refer to the subsequent processing and adds a further obligation to communicate information about the purpose of the processing. It also states that the information should be communicated to the data subject before the processing takes place and not at the time of the collection, as is the case in paragraph 2.

Definition 8 (The copying macro). *The copying macro is defined as follows*
$(\texttt{COPY}(D_?[\texttt{VAR}], \texttt{AND}(E[\texttt{VAR}], F_1, ..F_m), G^s), map)$
$\Longrightarrow$
$(\texttt{M-OBS}(\texttt{AND}(D_?[\texttt{VAR}], C[\texttt{VAR}]), \texttt{AND}(E[\texttt{VAR}],$
$B_1, \dots, B_n, F_1, \dots, F_m)), map)$ *Where*

- $m \geq 0$
- $map[G^s] =$ $\texttt{M-OBS}(C[\texttt{VAR}], \texttt{AND}(A[\texttt{VAR}], B_1, .., B_n))$ *with all the conditions as in Definition 6.*
- *For each* $0 \leq j \leq m$, F_i *is one of the following*
  - $\texttt{IF-THEN}(F_i^0, F_i^1)$
  - $\texttt{ALWAYS-IF}(F_i^1, F_i^0)$
  - *A simple annotation* F_i^1. *In this case* F_i^0 *is empty.*
- $D_?$ *refers to an optional arbitrary annotation.*

The last macro is based on the forth paragraph.

Paragraph 4 of GDPR's Article 13: Paragraphs 1, 2 and 3 shall not apply where and insofar as the data subject already has the information.

The last paragraph in the article has a standard legal form. Its purpose is to set exceptional circumstances in which other paragraphs do not hold. In our example, none of the obligations in the previous paragraphs should hold in case the data subject already has the required information.

In the previous subsection, we have seen a utility macro for labeling annotations. This macro is handy here as well as we will need to be able to refer to other annotations, in order to apply a macro for exceptional circumstances.

The Exception Macro. This macro gets a list of simple annotations, which denote labels of other annotations. It additionally gets an annotation which serves as the exceptional circumstances. When applied, this macro will add the negation of the exceptional circumstances to the conditions of all referred obligations in the state. In our example, it will make sure that all the obligations described in this article hold only in case the relevant information in the specific obligation is not already known to the data subject. Since many of these obligations are generated by one of the other macros, the exceptional circumstances can contain the `VAR` placeholder, similarly to the Multi-obligation and Copy macros. In our example, this placeholder is indeed used and is replaced, for every obligation, with the exact information which should be communicated and should not be already known.

Definition 9 (The exception macro). *The exception macro is defined as follows*
$(\mathtt{EXCEP}(A_1^s, \ldots, A_n^s, B), map) \Longrightarrow (\emptyset, map(A_1^s, C_1)(\ldots)(A_n^s, C_n))$ *where*

- $n \geq 1$
- *For each* $0 < i \leq n$
  - $map[A_i^s] = C_i'$
  - *In case* $C_i' = \mathtt{M\text{-}OBS}(A, D)$*,* $C_i = \mathtt{M\text{-}OBS}(\mathtt{AND}(\mathtt{NOT}(B), A), D)$
  - *Otherwise* $C_i = \mathtt{IF\text{-}THEN}(\mathtt{NOT}(B), C_i')$

A note about the above interpretation of legal exceptions. The macro defined in Definition 9 applies further conditions to previously defined sentences. The purpose of these further conditions is to specify situations in which these sentences are defeated. Most approaches to defeasible reasoning are based on non-monotonic logics [18]. Nevertheless, monotonic logics have been proposed as well [6]. Clearly, more discussion is required in order to justify the choice in Definition 9. While this discussion is beyond the scope of this paper, I would like to point out the main difference between the two approaches and suggest, in a very informal and imprecise way, a remedy, which is made possible by using the NAI Suite. Improving and implementing this remedy is planned as a future work.

In contrary to classical logics, non-monotonic logics allow a decrease in the amount of possible deductions given an increase in factual information. This advantage is no longer relevant if all possible knowledge is known in advance (please refer to Section 6 in [20] for more information). While knowing everything in advance is not feasible, knowing what is known in advance is. After all, the known information must be input into the NAI Suite and the total amount of relevant information is finite, since it must appear in the legislation, which is finite (or more precisely, can be finitely denoted). The NAI Suite can decide for each piece of information if it is known or not and adapt the reasoning process in order to take it into account.

It should be further noted that the macros abstract over the exact interpretation of exceptions. The precise treatment is handled by the underlining logic and theorem prover. It is true that in Definition 9 an explicit negation was used. The reason for that is that right now there is no other underlining logic. A better definition would use a new operator for denoting evidence which defeats obligations. The current underlining logic will interpret this operator as classical negation while a Prolog based solution would interpret it as a "negation as failure", etc.

4 Example: Automated Reasoning over GDPR Article 13

This section describes the formalization of article 13 of the GDPR using the NAI Suite and the extensions to the suite which were implemented in the previous section. The macros currently appear in the suite in the same drop list as the logical and normative connectives, just below a separator. In later versions of the tool, they will probably be allocated their own drop list.

The reader is invited to follow this section while simultaneously looking at the formalization in the tool itself. The formalization and queries which resulted from this work are integral artefacts of this paper and can be accessed via the NAI Suite web application[3]. This section will constantly make references to the tool.

4.1 Annotating Paragraph 1

The NAI Suite, as described in Sect. 2.3, requires us first to create a new legislation and then copy the original text into the editor pane. The first paragraph describes a situation in which a controller is obliged to communicate different information to the data subject, according to different conditions. Although not explicitly written, this paragraph also talks about processors and the processing of the data itself, as well as of its collection. In order to formalize the article, we must make these elements explicit in the text. We therefore add this information in brackets ('[','[') as can be seen in the already annotated text in the editor pane.

Given the explicit text of the paragraph, we are ready to annotate it. The first step of every formalization using the NAI Suite is to annotate all terms which are part of the vocabulary of the text. These terms correspond to the colorful annotations in the editor. There are many relations between the different legal terms. For example, the personal data of the data subject is being collected and

[3] Please login to https://nai.uni.lu using the email address: gdpr@nai.lu and password: nai. Please note that this account is write protected and cannot be changed. Note also that no registration is required in order to use the above account!.

processed and is subject to the supervision of a Data Privacy Officer (DPO). Such complex relations require an expressive language such as fist-order logic, which is used in the annotations of the article.

As an example of term annotations, one can consider the phrases "data subject" and "personal data", for which the following first-order terms were assigned (respectively): `data_subject(Subject)` and`personal_data(Data, Subject)`. When annotating terms, words starting with a lower-case letter are considered as constants while those starting with a capital letter are considered as variables which are quantified over the whole logical expression. The full list of the annotated legal terms can be found on the "Vocabulary" tab in the legislation editor.

Once all legal terms are annotated, we can proceed with annotating the relationsips between them. The structure of the paragraph is the following. A set of conditions for the whole paragraph is followed by an obligation to communicate information. The precise information to communicate then follows in each of the items, possibly with some further conditions.

The NAI suite supports such a sentence structure via the "Multi-obligation Macro". This macro accepts an annotation denoting the general conditions and a second annotation denoting the additional obligations and their specific conditions. When applied, the macro generates a conjunction of obligations, one for each of the annotated obligations and with the general conditions as well as the specific ones. We therefore annotate the whole paragraph 1 with this annotation.

We now proceed with annotating the conditions and obligations. First, we use the "And" connective to annotate all the general conditions, such as the existence of a processor and a controller, etc. We then proceed by using the "And" connective to annotate all the obligations. Each obligation can have one of three forms. A simple obligation contains just a term. The macro will convert this obligation into a conditional obligation where the conditions are all the general ones from the first conjunction. The more complex obligations are either "If/Then" and "Always/If".

The difference between these two connectives is syntactical only. While in "If/Then", the conditions are specified by the first annotation and the conclusion with the second, the order is reversed in "Always/If". By using one of these two annotations for the different obligations, the macro will know to add the additional, specific conditions to the conditions of each resulted obligation.

The application of the macro to the annotation of paragraph 1 generates multiple annotations. The DSL denotation of the generated annotation of the first item of paragraph 1 can be seen in Fig. 3

```
IF-THEN-OB(AND(processor(Processor),nominate(Controller,
Processor),personal_data_processed(Processor,Data,Time,
Justification,Purpose),personal_data(Data,Subject),
collected_at(Data,Time0),data_subject(Subject),
controller(Controller, Data)),communicate_at_time(Controller,
Subject,Time0,contact_details(Controller)))
```

Fig. 3. Generated annotation of item 1 in paragraph 1

The specific annotations can be seen by hovering with the mouse over the relevant parts of the text. The full formalization of this article can be seen on the "Formalization" tab. This formalization is a conjunction of the obligations specified in the article. Note that since some items contain more than just one obligation, the conjunction contains more than 6 conjuncts.

The annotation of paragraph 2 is similar.

4.2 Annotating Paragraph 3

Using the "Copy" macro, annotating this paragraph becomes relatively simple. The annotation takes 3 elements. The first contains optional additional conditions. Indeed, since we consider now further processing of the data, there are additional conditions such as the processor of the new processing, its purpose, etc. We also need, in the conditions, to stress that the new processing is different from the previous one. Finally, we group these further conditions in an "And" annotation.

The second element contains the further obligations. The first of which is the obligation template which contains the `VAR` placeholder. This element will replace the obligation template in the copied annotation which is referenced by the third element. The remaining items in this element are additional values which should be substituted for `VAR` in each of the generated obligations. These items add to those from the referenced annotation.

The annotated text can be seen in Fig. 2.

The resulting formalization, which can be seen at the bottom (the third element) of the "Formalization" tab, contains therefore more obligations that the referenced paragraph 2. In addition, all the obligations refer to the new purpose and state a new communication time, as stated in the new conditions.

Facilitating Correct Formalization. There is a further interesting issue which relates to this paragraph. Clearly, its annotation is far from trivial and is error prone. Still, by using macros and annotations, the chance of error occurring is reduced since there is a clear connection between the text itself and its annotation, as demanded by the Isomorphism approach to legal formalization [2]. This is not the case when the paragraph is translated into a logical formula manually. As an example, consider the DAPRECO formalization of the GDPR [19]. As mentioned in the introduction to this paper, this ambitious knowledge base contains a manual translation of almost all articles of the GDPR.

Nevertheless, if we focus on the translation of this paragraph[4], we can see several errors. First, the translation does not mention at all the fact that all of the required information mentioned in paragraph 2 is required here as well. Even more important though, there is no distinction between the two processing events of the data, except in the name of the variable used to denote them. There is a

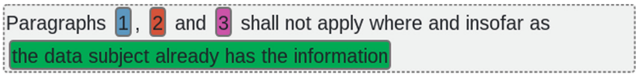
[4] Please search for the text "statements51Formula" in https://raw.githubusercontent.com/dapreco/daprecokb/master/gdpr/rioKB_GDPR.xml, of a version no later than 11/2019.

relation between the times of the two occurrences but it is defined as "greater or equal". Clearly, this formalization will always apply to any processing, whether first or not, since one can substitute for the universally quantified variables the same processing and the same time.

Such an error is not easy to spot, when one translates regulations manually. On the other hand, when using annotations and macros, we could more easily spot this and use a correct annotation.

4.3 Annotating Paragraph 4

Annotating exceptions is now also relatively straightforward. We use the "Exception" macro to state all the annotations which should take an additional condition and the condition itself. The result can be seen in Fig. 4.



Paragraphs 1, 2 and 3 shall not apply where and insofar as the data subject already has the information

Fig. 4. Annotating the exception in paragraph 4

While in the intermediary annotations-based level we add a new annotation to faithfully capture the exception, in the underlining logical representation no new formula is added but existing formulae are modified to accommodate the exception. This is an example of an hierarchical formalization [20] which aims at being both faithful, well engineered and efficient.

4.4 Automated Reasoning over the GDPR

Section 2 has described several automated deduction based tools, such as consistency or independence checking. These tools can be used for checking the correctness of the formalization. Nevertheless, the main usage of automated deduction within the NAI Suite is for allowing the computer to answer questions and make legal deductions. Given a correctly formalized legislation, NAI can currently answer Yes/No questions. This is done by employing the state-of-the-art theorem prover MleanCoP [15], which in turn tries to build a formal proof that the question logically follows from the formalization and assumptions. Since first-order modal resolution is only semi-decidable [8], some negative answers cannot be given. The NAI Suite displays a warning in this case.

The main expected usage of this feature is by third-party tools, which will use the deduction engine of the NAI Suite over an already formalized legislation in order to answer arbitrary questions. For example, privacy policies can be checked automatically for compliance by constructing the relevant queries [17] and executing them in the NAI Suite.

Nevertheless, the NAI Suite also supports the possibility of writing queries directly in the tool. This feature is mainly used for testing and as a support tool for lawyers and jurists. Similarly, this feature can be used in order to demonstrate automatic reasoning over the article. The example consists of five questions relating to the precise time the controller is obliged to communicate different information to the data subject. These questions can be found and executed on the "Queries" menu of the tool. In the remaining of this section, they are described in detail.

Precise Time of Communicating Information in Case There is at Most One Processing of the Data. In order to be able to answer this question, three different scenarios are given. In each, we check if the controller is obliged to communicate specific information.

In all the example questions, the variable denoting the required information is instantiated with a specific one. In general, queries can also be executed over free variables and can represent a more general question.

The basic scenario which is described in all queries is the following. The birth date of the data subject Albert was collected on the 1/8/2017. The Controller Brian has nominated the Processor John to process the data for the purpose of improving the business. The processing took place on the 1/9/2017.

In the first question we assume that Albert does not yet know the Contact Details of the Data Protection Officer, who is named Charles. We ask if, in this case, Brian is obliged to communicate this information to Albert at the time of processing. When we execute the query, NAI tells us that this is not the case.

The way a theorem prover answers Yes/No questions is by trying to prove them. In case a proof is found, we have a mathematical argument that the answer is Yes. When a proof cannot be found, the theorem prover might answer that either the query is counter-satisfiable, or that the search for proof has timed out.

In the case the query is counter-satisfiable, we know that based on the information we have given, the prover has found cases where the opposite of what we have asked is true and that therefore, proving the validity of the argument is impossible. If we have given all relevant information, then this answer is effectively a No. On the other hand, if we have forgotten some important information, such as the fact that Brian is the Controller, then being counter-satisfiable means that in some cases, the answer is No, but in others, it might be Yes. For example, when we omit the information about the controller, it might be that it is not Brian who is obliged to communicate the data and therefore, we have found a counter example. This means that the quality of the answer is based on the quality of the question.

In the case the prover times out, we can only know that we have terminated its execution before an answer was found. Usually, the theorem prover answers in less than a second. While we can normally assume that a futile search for a proof usually means that there is none, we cannot count on this result.

For the first question, we have obtained the answer that a counter-example exists. By considering the "Vocabulary" tab, we can confirm that we have considered all relevant information and that therefore, this answer means no - Brian has no such obligation.

For the second question, the obligation was changed from communicating the information at the time of processing to communicating it at the time of the collection of data. The prover answers in this case Yes - Indeed, Brian is under such a legal obligation.

The last question checks if this obligation also holds when the information is not Charles contact details, but the purpose of processing. The prover still answers Yes.

Precise Time of Communicating Information in Case of a Second Processing of the Data. In the following two questions, we expand the case as follows. Besides nominating John to process the data, Brian has also nominated Chris to process it for the purpose of a collaboration with Facebook. Chris has processed the data on the 1/10/2017.

The first question in this group tries to determine, again in case Albert is not aware of this second purpose of processing his Data, if Brian is obliged to communicate this information at the time of the second processing. The prover answers No.

The last query states the same question, but places the time of communicating the information to before the time of the second processing. The prover affirms that this is indeed an obligation of Brian.

The examples above have shown how arbitrary Yes/No questions can be answered by the tool. In a similar fashion, questions can be asked by third-party tools via the exposed API (see Sect. 2.4), such as whether a certain privacy policy complies with the GDPR.

5 Conclusion and Future Work

The formalization of legal texts is a non-trivial and error-prone task. Annotations can help generating correct formalizations. Nevertheless, legal texts contain sentence structures which go beyond logical and deontic connectives. The solution which was described and demonstrated in this paper uses macros to describe meta-level properties and to annotate such structures.

The four macros which were introduced are relatively general and appear in other legal texts, as well as in other articles of the GDPR. The immediate followup of the current work is the formalization of articles 5 and 6, which have a similar structure. Other macros will be added when needed.

In addition, the macro DSL which was introduced in Sect. 3 can be extended into a macro editing functionality within the NAI Suite. Such a feature will allow users to create arbitrary macros and handle any kind of legal text efficiently.

Currently, the NAI suite supports an expressive deontic first-order language. This language is rich enough to describe many scenarios which appear in legal texts. Nevertheless, more work is required in order to capture all such scenarios. Among those features with the highest priority, we list support for exceptions, temporal sentences and arithmetic. In Sect. 4.3, one possible direction for addressing exceptions was given. Other possible solutions for these issues already exist in the form of tools such as non-monotonic reasoners [10], temporal provers [24] and SMT solvers [7].

On the level of usability, the tool currently does not give any information as to why a query is counter-satisfiable. The user needs to look on the vocabulary in order to determine possible reasons. Integrating a model finder, such as Nitpick [5], will help "debugging" formalizations and enriching the query language.

NAI's graphical user interface (GUI) aims at being intuitive and easy to use and tries to hide the underline complexities of the logics involved. A continuously updated list of new features can be found on the GUI's development website[5].

References

1. Bartolini, C., Lenzini, G., Santos, C.: An agile approach to validate a formal representation of the GDPR. In: Kojima, K., Sakamoto, M., Mineshima, K., Satoh, K. (eds.) JSAI-isAI 2018. LNCS (LNAI), vol. 11717, pp. 160–176. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31605-1_13
2. Bench-Capon, T.J., Coenen, F.P.: Isomorphism and legal knowledge based systems. Artif. Intell. Law **1**(1), 65–86 (1992)
3. Bench-Capon, T.J., Robinson, G.O., Routen, T.W., Sergot, M.J.: Logic programming for large scale applications in law: a formalisation of supplementary benefit legislation. In: Proceedings of ICAIL, pp. 190–198 (1987)
4. Biagioli, C., Mariani, P., Tiscornia, D.: Esplex: a rule and conceptual model for representing statutes. In: Proceedings of ICAIL, pp. 240–251. ACM (1987)
5. Blanchette, J.C., Nipkow, T.: Nitpick: a counterexample generator for higher-order logic based on a relational model finder. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 131–146. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14052-5_11
6. Boutilier, C.: Conditional logics of normality as modal systems. Proc. AAAI. **90**, 594–599 (1990)
7. Bouton, T., Caminha B. de Oliveira, D., Déharbe, D., Fontaine, P.: veriT: an open, trustable and efficient SMT-solver. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 151–156. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02959-2_12
8. Fitting, M., Mendelsohn, R.L.: First-Order Modal Logic, vol. 277. Springer, Dordrecht (2012). https://doi.org/10.1007/978-94-011-5292-1
9. Governatori, G., Shek, S.: Regorous: a business process compliance checker. In: Proceedings of ICAIL, pp. 245–246. ACM (2013)
10. Kifer, M.: Nonmonotonic reasoning in FLORA-2. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 1–12. Springer, Heidelberg (2005). https://doi.org/10.1007/11546207_1
11. Libal, T., Pascucci, M.: Automated reasoning in normative detachment structures with ideal conditions. In: Proceedings of ICAIL, pp. 63–72. ACM (2019)
12. Libal, T., Steen, A.: NAI: the normative reasoner. In: Proceedings of ICAIL, pp. 262–263. ACM (2019)
13. de Montety, C., Antignac, T., Slim, C.: GDPR modelling for log-based compliance checking. In: Meng, W., Cofta, P., Jensen, C.D., Grandison, T. (eds.) IFIPTM 2019. IAICT, vol. 563, pp. 1–18. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33716-2_1

[5] https://github.com/normativeai/frontend/issues.

14. Otten, J.: Implementing connection calculi for first-order modal logics. In: Proceedings of IWIL, pp. 18–32 (2012)
15. Otten, J.: MleanCoP: a connection prover for first-order modal logic. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 269–276. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_20
16. Palmirani, M., Cervone, L., Bujor, O., Chiappetta, M.: RAWE: an editor for rule markup of legal texts. In: Proceedings of RuleML (2013)
17. Palmirani, M., Governatori, G.: Modelling legal knowledge for GDPR compliance checking. In: Proceedings of JURIX, pp. 101–110 (2018)
18. Prakken, H., Sartor, G.: The three faces of defeasibility in the law. Ratio Juris **17**(1), 118–139 (2004)
19. Robaldo, L., Bartolini, C., Palmirani, M., Rossi, A., Martoni, M., Lenzini, G.: Formalizing gdpr provisions in reified I/O logic: the DAPRECO knowledge base. J. Logic Lang. Inf. 1–49 (2019). https://doi.org/10.1007/s10849-019-09309-z
20. Routen, T., Bench-Capon, T.: Hierarchical formalizations. Man-Mach. Stud. **35**(1), 69–93 (1991)
21. Sergot, M.J., Sadri, F., Kowalski, R.A., Kriwaczek, F., Hammond, P., Cory, H.T.: The British Nationality Act as a logic program. Commun. ACM **29**(5), 370–386 (1986)
22. Sherman, D.M.: Expert systems and ICAI in tax law: killing two birds with one AI stone. In: Proceedings of ICAIL, pp. 74–80. ACM (1989)
23. Stamper, R.: LEGOL: modelling legal rules by computer. In: Computer Science and Law, pp. 45–71 (1980)
24. Suda, M., Weidenbach, C.: A PLTL-prover based on labelled superposition with partial model guidance. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 537–543. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_42