A Web-Based Tool for Analysing Normative Documents in English*

John J. Camilleri
Chalmers University of Technology
and University of Gothenburg
Gothenburg, Sweden
john.j.camilleri@cse.gu.se

Mohammad Reza Haghshenas
mrhaghshenas@gmail.com

Gerardo Schneider
Chalmers University of Technology
and University of Gothenburg
Gothenburg, Sweden
gerardo@cse.gu.se

ABSTRACT

Our goal is to use formal methods to analyse normative documents written in English, such as privacy policies and regulations. This requires the combination of a number of different elements, including information extraction from natural language, formal languages for model representation, and an interface for property specification and verification. A number of components for performing these tasks have separately been developed: a natural language extraction tool, a suitable formalism for representing such documents, an interface for building models in this formalism, and methods for answering queries asked of a given model. In this work, each of these concerns is brought together in a web-based tool, providing a single interface for analysing normative texts in English. Through the use of a running example, we describe each component and demonstrate the workflow established by our tool.

CCS CONCEPTS

• **Computing methodologies** → **Information extraction**; **Model development and analysis**; *Knowledge representation and reasoning*; • **Applied computing** → Law;

KEYWORDS

normative texts, contract analysis, information extraction, controlled natural language, model checking

ACM Reference Format:
John J. Camilleri, Mohammad Reza Haghshenas, and Gerardo Schneider. 2018. A Web-Based Tool for Analysing Normative Documents in English. In *Proceedings of ACM SAC Conference (SAC'18).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Normative texts, or *contracts*, are documents which describe the permissions, obligations, and prohibitions of different parties over a set of actions. They also include descriptions of the penalties which must be paid when the main norms of the document are violated. We frequently encounter such texts in the form of privacy policies, software licenses, workflow descriptions, terms of use, regulations, and SLAs (service-level agreements). Despite being written for human consumption and thus expressed in natural language, these kinds of documents are typically long and difficult to follow, making them hard to analyse manually.

> *What commitments am I agreeing to make?*
> *Can my information be shared with third parties?*
> *How late can I make my payment without facing fines?*

These are the kinds of questions about a contract that we may want answered, both as users and as authors. Using text-based search to find such answers can be tedious and unreliable, for example when clauses cross-reference each other, and when the document contains exceptions and timing constraints. Our goal is to bring formal methods to this kind of natural language analysis, packaged as a tool which is usable by non-experts and which requires as little understanding of the underlying technologies as possible. We do this by first modelling these documents using a suitable formalism, which then makes them amenable to verification using standard techniques. This includes answering queries based on a syntactic traversal of the model, as well as using model-checking to verify temporal properties, by converting the model to a timed automata representation.

The main components required for this have been individually described in previous papers by the current authors [8–10]. This paper presents a new web-based tool for the analysis of normative texts in English, bringing each of these components together into a single interface. This integration requires the development of suitable transformations between formats, besides maintaining a *dictionary* of key elements at each level of abstraction in order to guarantee the translation from/to the natural language text and the different formal representations. Our novel contributions also include a set of query templates in natural language and a method for processing counter-example traces, allowing users to interact with the system completely in English.

The rest of the paper continues as follows. Section 2 introduces the structure of the tool and gives an overview of the workflow it establishes. In section 3 we then present the running example which will be used throughout the paper to demonstrate the use of our tool. Section 4 describes extraction and building a model through post-editing of tabular data, including verbalisation of an existing model using controlled natural language. Section 5 describes the analysis which can be performed on the model, by using a set of query templates which can be customised based on the current model. The software architecture is then summarised in section 6. Section 7 takes a look at some related work in this area, and we conclude with a summary of the benefits and limitations of our approach in section 8.

*This work has been supported by the Swedish Research Council under grant number 2012-5746.

SAC'18, April 9-13, 2018, Pau,France
2018. ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/xx.xxx/xxx_x

2 THE *CONTRACT VERIFIER* TOOL

The main contribution of this work is the *Contract Verifier*, a web-based tool for modelling and analysing normative texts in English. Figure 1 shows an overview of the workflow which our tool covers, summarised in the following steps:

(1) Users start with an English text which they wish to build a model out of.
(2) The text is submitted to an **extraction** phase which attempts to automatically extract the clauses from the text, each of which concerning at least an agent, action and modality.
(3) The results of the extraction are shown to the user in a tabular format where each cell is editable. At this point, the user must check the results of the extraction and **post-edit** the clauses which are not completely correct.
(4) Once the user is happy with the model in tabular format, this is **converted** into an actual model, which is internally represented in an XML format.
(5) From here, the model can be verbalised in a controlled natural language and viewed in a compact formal notation.
(6) The user then performs **analysis** by selecting the queries which should be run against the model. Queries are presented as English sentence templates, which may include slots for specifying relevant arguments.
(7) The **queries** are then submitted to the server which computes the **results** and displays them to the user beside each respective query.
(8) If the answer to a query is not as expected, this could point to either:
    (a) a problem with the contract model, in which case the user may go back and edit the model and repeat the steps as above; or
    (b) a problem with the original normative document, where the user may then modify the text and perform the analysis again, or simply leave the original formulation as it is (e.g., the analysis might detect a lack of a deadline associated with a given obligation, but this might be considered desirable by the user).

A demonstration of the *Contract Verifier* is openly available on the web[1]. Upon visiting this URL, users will be asked to log in. New users may create an account so that their work can be saved under their profile. Alternatively, one may log in with a guest account, using username `guest@demo` and password `contract`.

3 RUNNING EXAMPLE

To show how our tool is used in practice, we pick a running example of a normative text describing the rules of a university course (see Figure 2). This example is based on courses held at our own department, covering the requirements for passing the course and the deadlines for the submission and grading of assignments. It has been chosen as an example because it is concise yet contains a variety of temporal constraints and dependencies between clauses. The text itself has been written by ourselves.

Before publishing these rules as an official course description, the authors (teachers/administrators) may wish to ensure that all the requirements for passing the course are enforced and that the rules are consistent. Once published, end users of these rules (students) may wish to query the parts which are relevant to them or work out any flexibilities in their deadlines.

[1] http://remu.grammaticalframework.org/contracts/verifier/

4 BUILDING A CONTRACT MODEL

4.1 Extraction from English

The first step towards building a formal model of our natural language contract is to process the text in order to see what clause information can be extracted automatically. This is done using the ConPar tool [8], which can extract partial contract models from English normative texts. ConPar is a natural language processing (NLP) tool, which uses the Stanford Parser [14] to obtain dependency trees for each sentence in the input text. These trees are then processed by ConPar, which uses the dependency representation to attempt to extract information related to:

(i) subject (agent)
(ii) verb and object (action)
(iii) modality (obligation, permission, prohibition)
(iv) temporal and non-temporal conditions.

In addition, ConPar will also try to identify refinement clauses, where a single input sentence may translate to multiple sub-clauses joined together using a connective such as conjunction, choice or sequence.

The output of the ConPar tool is a tabular representation of the extracted data, which is produced in a tab-separated value (TSV) format. This is a simple format which can be easily loaded in any spreadsheet or table-editing software. In this representation, each row corresponds to a clause while the columns indicate the various components, as listed above. The result of passing our example through this tool is shown in Figure 3.

4.2 Post-editing

While quite a lot of clause information has been correctly extracted by the tool, there are still some errors which need to be corrected manually by the user. The use of a tabular format for displaying the output of the extraction phase facilitates this post-editing. All cells in the table are editable, and rows can be added/deleted as needed.

For example, in the case of clause 4 (Figure 3), the value in the *Modality* field (`{should}`) should be replaced with the obligation specifier `O`. Additionally, the value of *Time* field (`within one week of it being submitted`) can be encoded with the phrase `within 7`, which is the accepted syntax for expressing this kind of time restraint.

4.3 Conversion to contract model

After post-editing the extracted clause information, this can be converted into a formal contract model. The formalism used for representing contracts is based on the deontic modalities of **obligation**, **permission** and **prohibition** of agents over actions. It includes constructs for refining clauses by conjunction, choice and sequence, and includes the possibility to specify reparations for when a clause is violated. Clauses can be constrained by temporal restrictions or guarded based on the status of other clauses. This formalism is based on *C-O Diagrams* [12].

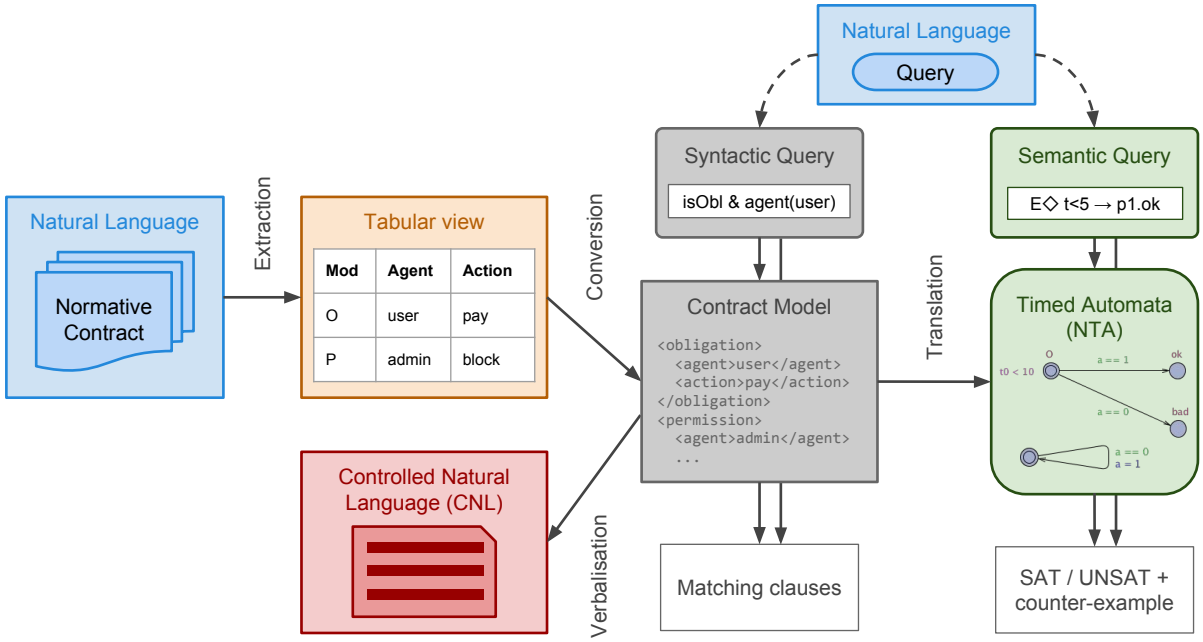

Figure 1: Overview of the contract analysis workflow established by the *Contract Verifier*.

> Students need to register for the course before the registration deadline, one week after the course has started.
> To pass the course, a student must pass both the assignment and the exam.
> The deadline for the first assignment submission is on day 10.
> Graders should correct an assignment within one week of it being submitted.
> If the submission is not accepted, the student will have until the final deadline on day 25 to re-submit it.
> The exam will be held on day 60.
> Registered students must sign up for the exam latest 2 weeks before it is held.

Figure 2: Example of a normative text describing the rules involved in the running of a university course. The course is assumed to start on day 0. This text is used as the running example throughout this paper.

This conversion step is implemented as a straight-forward script which takes the TSV representation as input and produces a contract model file. We refer to the format of this file as COML, which is an XML-based format for storing contract models in our formalism. Once the conversion is complete, the COML file is stored on the server and the user can view it using three different representations simultaneously:

(i) post-edited input text,
(ii) controlled natural language (CNL), and
(iii) a compact formal notation (*C-O Diagram* Shorthand or CODSH).

These are shown in Figure 4 and explained in the sections below.

4.4 Verbalisation using CNL

Given a contract model in our formalism, we have developed a method for linearising it as a phrase in a *controlled natural language* (CNL) [21]. A CNL is a reduced version of a natural language (NL) which has limited syntax and vocabulary, making it in fact a formal language and thus expressible using a grammar. CNLs are often used as interfaces for formal languages which are human-friendly, yet still unambiguous and well-defined.

The CNL designed for this contract formalism is described in [9]. We use the Grammatical Framework (GF) [20] for defining the grammar for our CNL and converting to and from our internal formal representation. This also includes the possibility of building a contract model directly using the CNL (rather than using the extraction step), however we do not cover this input method in the present work. The CNL representation may resemble the original NL text in some ways, however it is characterised by less variation in the expressions used and by certain structural features such as labels before each clause.

The generation of the CNL representation requires that subjects, verbs and objects are present in the lexicon. The lexicon used is a large-scale English dictionary containing over 64,000 entries, but if the contract contains terms which are not present in this lexicon then the generation to CNL will fail. This failure however will not affect the usability of the rest of the tool.

4.5 Compact formal notation

In addition to the post-edited text and the CNL, we also display a view of the model in a formal syntax (Figure 4, bottom-right). We refer to this notation as *C-O Diagram* Shorthand (CODSH). It is designed mostly for developers who understand the formal

Sentence	No	Modality	Subject	Verb	Object	Time	Conditions
Students need to register for the course before the registration deadline , one week after the course has started .	1	D	student	need	to register		
To pass the course , a student must pass both the assignment and the exam .	2	AND[2.1,2.2]					
	2.1	O	student	pass	exam		
	2.2	O	student	pass	assignment		
The deadline for the first assignment submission is on day 10 .	3	D	deadline	be on	day		DAY: 10
Graders should correct an assignment within one week of it being submitted .	4	{should}	grader	correct	assignment	within one week of it being submitted	

Figure 3: Screenshot showing the output from passing our normative text through the extraction tool (image has been cropped to improve readability). Each row indicates a clause in the model. Note how the second English sentence has been refined into multiple sub-clauses. The O in the modalities column stands for *obligation*, while D stands for *declaration*.

structure of the contract model but would like a more condensed representation than the COML format. This can be helpful when debugging. In particular, this notation reveals the names which are automatically assigned to each clause in the conversion phase.

5 ANALYSIS

Once the model has been built, we can perform analysis on the contract by running queries against it. The user is presented with a list of query templates, as shown in Figure 5. Each query may have slots for parameters which the user should provide; these are either names of clauses, agents or actions. The possible completions for these slots are extracted automatically from the contract model.

Internally, queries provided by our tool are computed in one of two ways, either through syntactic filtering or via conversion to timed automata and using the Uppaal verification tool [15]. Both of these techniques are described below.

5.1 Syntactic analysis

Our tool currently provides six different syntactic queries (items 1–6 in Figure 5). These queries are *syntactic* in the sense that each can be solved by traversing the contract model and filtering out the corresponding clauses which match the query. As an example, consider the following query:

What are the obligations of [agent]*?*

This query is internally encoded as the following conjunction of predicates:

$isObl \wedge agentOf([\text{agent}])$

where [agent] is replaced with a concrete agent name chosen by the user. The solution to this query is computed via a Haskell function which takes the contract model as a Haskell term together with the query as a set of predicate functions, and returns a list of matching clauses. The final output is given as a natural language sentence listing the actions corresponding to the matching clauses. For the example, asking for all the obligations for *student* will give the output below:

The following are obligations of student:
- *register for course*
- *submit assignment*
- *sign up for exam*
- *pass exam*

The way the result is phrased will vary based on the query, as well as the number of items in the result: up to two results are inlined, while three or more are given as bullets. This is to make the response more natural for the user.

5.2 Semantic analysis

Our tool also includes four semantic queries (items 7–10 in Figure 5). We use the term *semantic* to refer to those queries which cannot be answered simply by looking at the structure of the model. Consider the following example:

The [agent] *must* [action] *before time* [number].

Determining this must take into consideration the operational behaviour of a contract model, including when actions are performed,

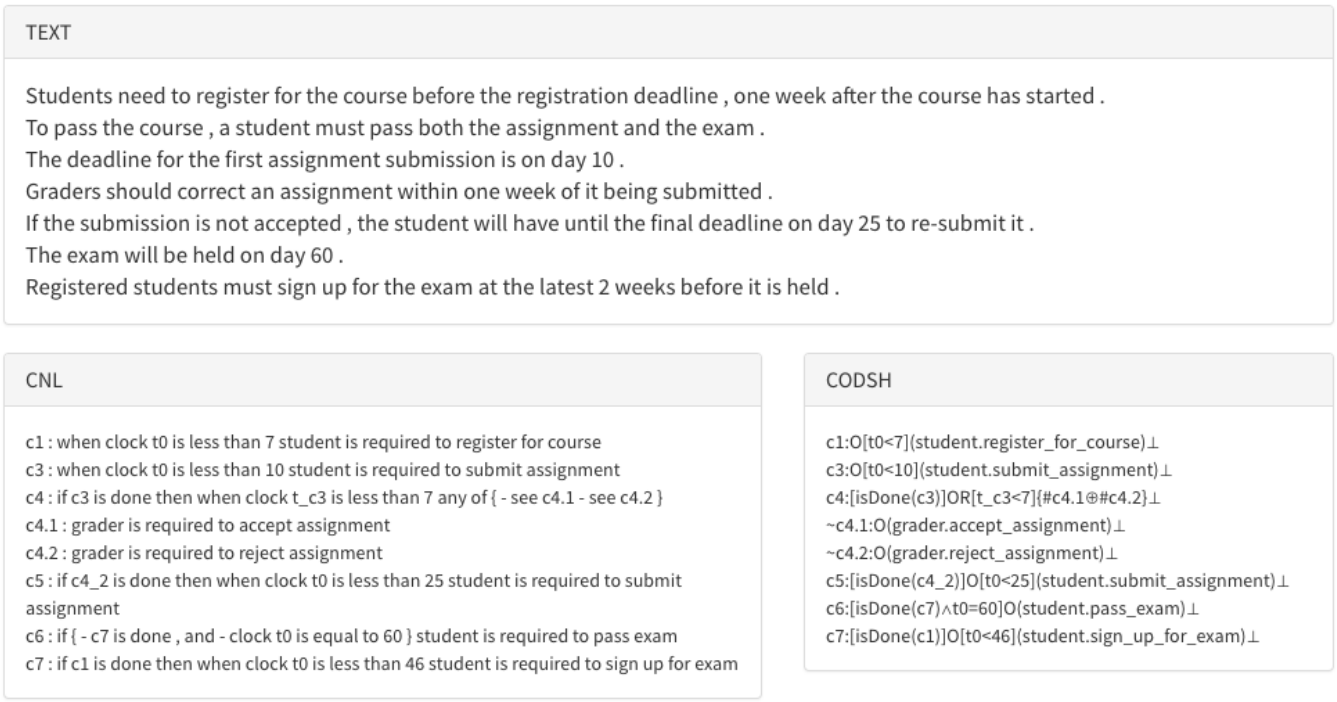

Figure 4: Screenshot showing the different output representations of the contract model (image has been cropped to improve readability). Top: the post-edited input text; bottom-left: controlled natural language; bottom-right: compact formal notation.

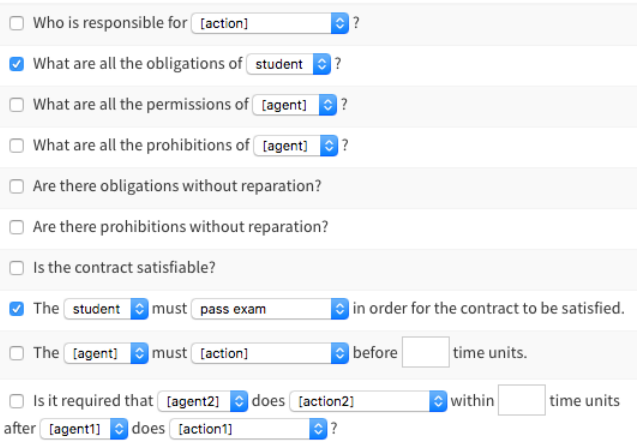

Figure 5: Available query templates which users may execute against the contract model, with drop-downs for specifying agents and actions extracted from the model. Queries 1–6 are *syntactic*, while queries 7–10 are *semantic*.

how new clauses are enabled and others expire. Processing semantic queries is achieved through using model checking techniques, by first converting a contract model into a network of timed automata [1] and then using the Uppaal tool to verify temporal properties against the translated model. This idea was introduced for *C-O Diagrams* in [12], but we have provided our own improved translation. By using verification, we are able to quantify over all possible sequences of events with respect to the contract.

This approach requires that the query itself is encoded as a property in a temporal logic which the model checker can process. In the case of Uppaal, the property specification language is a subset of TCTL [6]. The example query above is encoded as the following Uppaal property:

$$\forall \Box\, allComplete() \implies \Big(isDone([\text{agent}]\_[\text{action}]) \wedge t_0 - Clocks\big[[\text{agent}]\_[\text{action}]\big] < [\text{number}]\Big)$$

Here, *allComplete* and *isDone* are helper functions included in the translated Uppaal system which allow the status of clauses and actions to be queried from within the timed properties. t_0 is a never-reset clock representing global time, while the *Clocks* array contains a clock for each action in the system, which is reset when that action is performed. The expression $t_0 - Clocks[a]$ thus gives the absolute time at which action a was completed.

The property is then verified using Uppaal, which will return a result of **satisfied** or **not satisfied**. In cases where a symbolic trace is produced as part of the verification, this is parsed by our tool in order to provide a meaningful abstraction of it. In this processing step, we pick out the actions performed in the trace along with their time stamps and present these as part of the result to the user. For example, when running the query below on our contract example:

The student *must* register for course *before time* 5.

we get the following result in the tool:

> *NOT Satisfied*
> *The property is violated by the following action sequence:*
> *- student register for course at time 6*
> *- student submit assignment at time 6*
> *- …*

where the remainder of the trace contains the other obliged actions at time 6 or later. This is in fact as we expect; the contract states that students have up to 7 days to register for the course, and thus it is not true that they must have registered before day 5 in order to satisfy the entire contract. If we change the time value in the query from 5 to 7, then the result returned is *Satisfied*.

6 SOFTWARE ARCHITECTURE

The *Contract Verifier* tool is implemented as a PHP web application, using a MySQL database for storing user accounts and the query templates available in the system. Contract models in COML and Uppaal-XML format are saved as files on the server. No server-side framework is used. The client-side interface is based on the AdminLTE Control Panel Template[2], and the tabular editing interface makes use of the editableTable jQuery library[3].

The ConPar extraction tool is written in Java, primarily because it uses the Stanford parser which is also implemented in Java. The core of our system is written in Haskell, using algebraic data types to define the structure of a contract model. The conversion from TSV and translation to NTA are thus also written as Haskell functions to and from this data type. The linearisation of a contract model to CNL uses the GF runtime, which is a standalone application. Similarly, executing semantic queries requires running Uppaal as an external process.

Because of the variety of languages and programs used in our tool chain, we provide a convenient layer over these components in the form of a small server application which provides these separate functionalities as individual web services. This modular approach allows the web application providing the user interface to consume each component via a web API, removing limitations on implementation language and hosting requirements, and allowing a clean separation of concerns between front-end and back-end.

Table 1 shows a summary of the API covering all the web services provided by the server. This API is fully documented and publicly accessible online[4]. The server itself is also implemented in Haskell.

7 RELATED WORK

AnaCon [2] is a similar framework for the analysis of contracts, based on the contract logic $\mathcal{CL}$ [18, 19], which allows for the detection of contradictory clauses in normative texts using the CLAN tool [13]. By comparison, the underlying logical formalism we use, based on *C-O Diagrams*, is more expressive than $\mathcal{CL}$ as it includes temporal constraints, cross-referencing of clauses and more. Besides this, our translation into Uppaal allows for checking more general properties, not only normative conflicts. In addition, their interface for specifying contracts is purely CNL-based and there is no extraction tool from English as in our case.

Information extraction from natural language is of course a field within itself, and even in the domain of contractual documents in English there are many other works with similar goals. The extraction task in our work can be seen as similar to that of Wyner & Peters [22], who present a system for identifying and extracting rules from legal texts using the Stanford parser and other NLP tools within the GATE system. Their approach is somewhat more general, producing as output an annotated version of the original text, whereas we are targeting a specific, well-defined output format. Other similar works include that of Cheng et al. [11], who combine surface-level methods like tagging and named entity recognition (NER) with hand-crafted semantic analysis rules, and Mercatali et al. [16] who extract the hierarchical structure of the documents into a UML-based format using shallow syntactic chunks.

One crucial aspect in any work targeting formal analysis of natural language documents is the confidence in the extraction from a source document to the target formal language. In our tool, the result of the extraction is usually incomplete and some amount of manual post-editing is always generally required. Azzopardi et al. [5] handle this incompleteness using a deontic-based logic including *unknowns*, representing the fact that some parts have not been fully parsed. Furthermore, the same authors present in [4] a tool to support legal document drafting in the form of a plugin for Microsoft Word. Though the final objective of their work diverges from ours, we are both concerned with the translation of natural language documents into a formal language by using an intermediate CNL. The main difference is that they target a more abstract formal language (very much like $\mathcal{CL}$), and as a consequence their CNL is also different. Our formalism allows for richer representations not present in their language (e.g. real time constraints and cross-references). Additionally, they do not target complex analysis of contracts (they only provide a normative conflict resolution algorithm), and we do not provide assistance in the contract drafting process.

There is considerable work in modelling normative documents using representations other than logic-like formalisms such as our own. LegalRuleML [3] is a rule interchange format for the legal domain, allowing the contents of legal texts to be structured in a machine-readable format. The format aims to enable modelling and reasoning, allowing users to evaluate and compare legal arguments using tools customised for this format. A similar project with a broader scope is MetaLex [7], an open XML interchange format for legal and legislative resources. Its goal is to enable public administrations to link legal information between various levels of authority and different countries and languages, improving transparency and accessibility of legal content. Semantics of Business Vocabulary and Business Rules (SBVR) [17] uses a CNL to provide a fixed vocabulary and syntactic rules for expressing the terminology, facts, and rules of business documents of various kinds. This allows the structure and operational controls of a business to have natural and accessible descriptions, while still being representable in predicate logic and convertible to machine-executable form.

We note that none of the works mentioned above present a single tool for end-to-end document analysis, starting from a natural

[2] https://almsaeedstudio.com/
[3] http://mindmup.github.io/editable-table/
[4] http://remu.grammaticalframework.org:5446/

Table 1: API of services provided in the web server, showing the relevant URL path and input/output formats for each service. The various formats are: TSV (tab separated values), COML (Contract-Oriented XML), CODSH (*C-O Diagram* Shorthand), CNL (Controlled Natural Language), and UPPAAL-compatible XML.

Path	Description	Request	Response
`/nl/tsv`	Clause extraction (ConPar)	English text	TSV
`/tsv/coml`	Convert TSV to COML	TSV	COML
`/coml/codsh`	Show contract in shorthand	COML	CODSH
`/coml/cnl`	Verbalise contract using CNL	COML	CNL
`/coml/syntactic`	Execute syntactic query	Query + COML	Clause names
`/coml/uppaal`	Translate contract to NTA	COML	UPPAAL XML

language text and finally allowing for rich syntactic and semantic queries, as in the case of our tool.

8 CONCLUSION

In this paper we have presented *Contract Verifier*, a web-based tool for analysing normative documents written in English. The tool brings together a number of different components packaged together as a user-friendly application. We demonstrate a typical workflow through the system, starting with an English text, extracting a contract model from it, and executing different kinds of queries against it. Each of the components used by our tool is implemented as a standalone module, with a web-based API exposing each module as a web service. Individual modules can be easily replaced and new ones can be added, such as introducing a new back-end for runtime verification. Similarly, new interfaces can be built around the existing modules without having to make changes to the underlying modules.

An important feature of the *Contract Verifier* tool is the level of automation it provides: everything except the post-editing of the extracted model (and of course choosing the queries to be performed) is automatic. For example, the names of clauses, agents and actions are automatically extracted from the contract so the user can select them using drop-down menus when making queries. Also, each clause is given a unique identifier as well as a *clock* that is reset when that clause is activated. Though these are mainly intended for internal use when performing semantic queries, users may even use them explicitly in the post-editing phase to encode relative timing constraints.

We see this tool as a successful implementation of a user interface for bringing together various separate components and providing a clear workflow for analysing normative texts in English. That being said, it is as such a proof-of-concept tool and has not undergone any extensive usability testing or application in real-world scenarios. We conclude here with a critical look at the shortcomings and limitations of the current work.

Evaluation

Our goal is to make the task of analysing a normative document easier and more reliable than if one were to do it completely manually. Measuring whether we achieve this, and to what extent, requires proper assessment. Some evaluation has already been carried out for the natural language extraction part of the workflow (the ConPar tool), measuring the accuracy of tool for extracting a correct model from a normative document. By calculating precision and recall, F_1 scores of 0.49 to 0.86 were obtained for the test set of four documents [8].

However, we currently do not have a thorough evaluation of the complete *Contract Verifier* workflow as presented here. This would take the form of an empirical study comparing document analysis using our tool with a purely manual approach, measuring the amount of post-editing required to build a correct model, the time required to formulate a query and obtain a result, and the overall ease of use of the tool in a qualitative sense. We consider a study of this kind important future work.

Limitations

Extraction. The extraction phase relies on dependency trees and thus takes a syntactic-level approach to parsing. While a fair deal of information can be extracted in this way from simpler sentences, a deeper understanding of a phrase often involves using related or opposite concepts which cannot be determined without more elaborate processing on the semantic level. In addition, we assume that each input sentence translates into one or more clauses, and have no support for detecting when a phrase should actually modify an existing clause instead.

Modelling. Our tool uses a tabular interface to help make the task of modelling user-friendly, but some understanding of the underlying formal language and its semantics is necessary in order to work efficiently with it. For example, our formalism is essentially *action-based*, where clauses prescribe what an agent should or should not *do*. However, empirically we have found that normative documents often describe what should or should not *be*, i.e. referring to state-of-affairs. While these can often be paraphrased to fit into our formalism, this is a non-trivial task which currently must be done completely by the user. The formalism itself has its own limitations, for example we are unable to encode percentages over quantities or any kind of arithmetic in actions, which are common features in some types of contracts (e.g., in SLAs).

Verification. When it comes to running queries, those which are syntactic can quickly be answered by an algorithm which is linear in the size of the model. For semantic queries however, the conversion to timed automata means that the state-space explosion problem typical of model checking is a potential problem. Certain optimisations made during the translation process could improve this somewhat. For instance, our generated NTA contain many

parallel synchronising automata as a result of the modularity of the translation, and some *ad hoc* heuristics could likely be used to reduce the number of automata or the need for certain synchronisations, thus improving the performance. That said, this is ultimately a theoretical problem which we cannot avoid altogether.

Scalability

Extraction. We are limited here by the speed of the ConPar tool, which itself uses the Stanford dependency parser. While parse time is related to the length of the input sentence, in our tests we have found that parsing and extracting a single sentence takes on average roughly half a second.[5]

Post-editing. The tabular interface for editing the extracted clauses has no concrete limits in terms of the number of clauses it can handle. What it does lack is support for managing a document's internal hierarchy (sections and sub-sections), which is a common feature of normative texts.

Analysis. As discussed above, the running of semantic queries is the biggest barrier to the scalability of the system due to the state-space explosion problem. For our small example here, a query requiring a search of the entire search space requires a few milliseconds to complete, but this performance may degrade drastically as the model size increases.

Queries. Our current implementation only offers a limited number of syntactic and semantic queries. New query templates can easily be added without any theoretical constraints. However, the user interface may need to be updated to help users navigate and possibly search through a long list of queries.

In summary, the *Contract Verifier* tool in its current state can handle documents of essentially any size in what concerns the extraction of a formal representation from a natural language text, its post-editing, and the execution of syntactic queries. This however could be improved by adding an extra layer of document hierarchy management to the tool, allowing the task to be segmented into smaller sub-parts. In what concerns semantic queries, the tool can realistically only handle smaller individual contracts containing tens of clauses. This is essentially due to our choice to translate contract models into Uppaal timed automata. An alternative here could be to use SAT solvers, or some other verification technology, to process the kind of semantic queries we perform. In any case, we believe *Contract Verifier* is an important step towards a rich analysis of normative texts, even if the analysis were to be restricted to just syntactic queries.

[5] Tests carried out on a dual-core MacBook Air from 2013.

REFERENCES

[1] Rajeev Alur and David L. Dill. 1994. A Theory of Timed Automata. *Theoretical Computer Science* 126, 2 (1994), 183–235. https://doi.org/10.1016/0304-3975(94)90010-8

[2] Krasimir Angelov, John J. Camilleri, and Gerardo Schneider. 2013. A Framework for Conflict Analysis of Normative Texts Written in Controlled Natural Language. *Journal of Logic and Algebraic Programming* 82, 5-7 (2013), 216–240. https://doi.org/10.1016/j.jlap.2013.03.002

[3] Tara Athan, Harold Boley, Guido Governatori, Monica Palmirani, Adrian Paschke, and Adam Wyner. 2013. OASIS LegalRuleML. In *ICAIL'13)*. ACM, 3–12. https://doi.org/10.1145/2514601.2514603

[4] Shaun Azzopardi, Albert Gatt, and Gordon Pace. 2016. Integrating Natural Language and Formal Analysis for Legal Documents. In *Conference on Language Technologies and Digital Humanities*. Academic Publishing Division of the Faculty of Arts, 32–35.

[5] Shaun Azzopardi, Albert Gatt, and Gordon J. Pace. 2016. Reasoning About Partial Contracts. In *JURIX'16*. IOS Press, 23–32. https://doi.org/10.3233/978-1-61499-726-9-23

[6] Gerd Behrmann, Alexandre David, and Kim G. Larsen. 2006. *A Tutorial on UPPAAL 4.0.* Technical Report. Department of Computer Science, Aalborg University. http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf

[7] Alexander Boer, Radboud Winkels, and Fabio Vitali. 2008. MetaLex XML and the Legal Knowledge Interchange Format. In *Computable Models of the Law*. LNCS, Vol. 4884. Springer, 21–41. https://doi.org/10.1007/978-3-540-85569-9_2

[8] John J. Camilleri, Normunds Grūzītis, and Gerardo Schneider. 2016. Extracting Formal Models from Normative Texts. In *International Conference on Applications of Natural Language to Information Systems (NLDB 2016) (LNCS)*, Vol. 9612. Springer, 403–408. https://doi.org/10.1007/978-3-319-41754-7_40

[9] John J. Camilleri, Gabriele Paganelli, and Gerardo Schneider. 2014. A CNL for Contract-Oriented Diagrams. In *CNL'14 (LNCS)*, Vol. 8625. Springer, 135–146. https://doi.org/10.1007/978-3-319-10223-8_13

[10] John J. Camilleri and Gerardo Schneider. 2017. Modelling and Analysis of Normative Documents. *Logical and Algebraic Methods in Programming* 91 (2017), 33–59. https://doi.org/10.1016/j.jlamp.2017.05.002

[11] Tin Tin Cheng, Jeffrey Leonard Cua, Mark Davies Tan, Kenneth Gerard Yao, and Rachel Edita Roxas. 2009. Information extraction from legal documents. In *SNLP'09*. IEEE, 157–162. https://doi.org/10.1109/snlp.2009.5340925

[12] Gregorio Díaz, María-Emilia Cambronero, Enrique Martínez, and Gerardo Schneider. 2014. Specification and Verification of Normative Texts using C-O Diagrams. *Transactions on Software Engineering* 40, 8 (2014), 795–817. https://doi.org/10.1109/TSE.2013.54

[13] Stephen Fenech, Gordon J. Pace, and Gerardo Schneider. 2009. CLAN: A Tool for Contract Analysis and Conflict Discovery. In *ATVA'09 (LNCS)*, Vol. 5799. Springer, 90–96. https://doi.org/10.1007/978-3-642-04761-9_8

[14] Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. In *ACL'03*. ACL, 423–430. https://doi.org/10.3115/1075096.1075150

[15] Kim G. Larsen, Paul Pettersson, and Wang Yi. 1997. UPPAAL in a Nutshell. *Software Tools for Technology Transfer* 1, 1 (1997), 134–152. https://doi.org/10.1007/s100090050010

[16] Pietro Mercatali, Francesco Romano, Luciano Boschi, and Emilio Spinicci. 2005. Automatic Translation from Textual Representations of Laws to Formal Models through UML. In *JURIX'05 (Frontiers in Artificial Intelligence and Applications)*, Vol. 134. IOS Press, 71–80.

[17] Object Management Group (OMG). 2015. Semantics of Business Vocabulary and Business Rules (SBVR). Document number: formal/2015-05-07. (2015). http://www.omg.org/spec/SBVR/1.3/PDF

[18] Cristian Prisacariu and Gerardo Schneider. 2007. A Formal Language for Electronic Contracts. In *FMOODS'07 (LNCS)*, Vol. 4468. Springer, 174–189. https://doi.org/10.1007/978-3-540-72952-5_11

[19] Cristian Prisacariu and Gerardo Schneider. 2012. A Dynamic Deontic Logic for Complex Contracts. *Journal of Logic and Algebraic Programming* 81, 4 (2012), 458–490. https://doi.org/10.1016/j.jlap.2012.03.003

[20] Aarne Ranta. 2011. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI.

[21] Adam Wyner, Krasimir Angelov, Guntis Barzdins, Danica Damljanovic, Brian Davis, Norbert Fuchs, Stefan Höfler, Ken Jones, Kaarel Kaljurand, Tobias Kuhn, Martin Luts, Jonathan Pool, Mike Rosner, Rolf Schwitter, and John Sowa. 2010. On Controlled Natural Languages: Properties and Prospects. In *CNL'09 (LNCS)*, Vol. 5972. Springer, 281–289. https://doi.org/10.1007/978-3-642-14418-9_17

[22] Adam Wyner and Wim Peters. 2011. On rule extraction from regulations. In *JURIX'11*. Frontiers in Artificial Intelligence and Applications, Vol. 235. IOS Press, 113–122. https://doi.org/10.3233/978-1-60750-981-3-113