Towards Legal Contract Formalization with Controlled Natural Language Templates

Regan Meloche
School of EECS
University of Ottawa
Ottawa, Canada
0009-0004-2418-1990

Daniel Amyot
School of EECS
University of Ottawa
Ottawa, Canada
0000-0003-2414-1791

John Mylopoulos
School of EECS
University of Ottawa
Ottawa, Canada
0000-0002-8698-3292

***Abstract*—Automated formalization of legal texts in order to remove ambiguities, conflicts and incompleteness has been a challenge for Requirements Engineering (RE) research for decades. This work seeks to make an incremental step towards this objective for legal contracts by making use of contract templates. Our proposed approach starts with a natural language contract template, together with a manually formalized specification of that template. A contract writer can make customizations to the template, which trigger the automatic formalization of the corresponding customized contract. Our target specification language is Symboleo, which is created specifically for contract verification and monitoring. Starting with a manually formalized template reduces the complexity associated with a fully automated formalization. Typical contract templates use simple fill-in-the-blank parameters, which serve as customizations to formalize in our framing of the problem. Our approach pushes the boundaries of these templates by allowing the contract writer to enter complex natural language customizations, such as prepositional phrases and conditional statements. This work explores what types of natural language patterns can be used in that context by analyzing relevant linguistics and real legal contracts. It also introduces a tool, SymboleoNLP, that suggests the feasibility of the formalization process.**

***Index Terms*—contract template, formalization, legal requirements, natural language processing, Symboleo**

I. Introduction

A. Motivation

Many daily affairs are mediated by legal contracts and their terms and conditions. For example, a sales contract specifies that a quantity of a product must be delivered to an address before a due date. A lease agreement prohibits tenants from making excessive noise after a certain time of day. Such documents prescribe behaviour of how one party (supplier, tenant) must act towards another party (buyer, property owner). They are often legally binding, so they must be precise, unambiguous, and understandable to the parties involved. A lack of these qualities can lead to different interpretations, at times resulting in costly legal disputes.

An emerging format of contract is the *smart contract*, a digital artifact where certain prescriptions found in traditional contracts are performed and monitored automatically. The growing attention paid to smart contracts is largely due to technologies such as distributed ledgers and the Internet-of-Things [1]. Consider a meat sale contract where the meat being delivered must be maintained below a certain temperature throughout the delivery process. If the temperature climbs above that threshold, then the meat quality may be compromised, and the buyer may be entitled to a discount. A temperature sensor inside a delivery vehicle connected to the Internet can detect the drop in temperature and report this to a smart contract, which can then automatically trigger (or cancel) a specific payment transaction.

There are many challenges involved in the development of smart contracts, one of which is the conversion of natural language (NL) clauses to code that can be executed on a machine. In almost any NL communication, a degree of common sense is assumed, and this assumption is a fundamental problem for NL processing (NLP). For a reliable smart contract, we must translate a contract written in NL into a *formal specification*, which is a structured and unambiguous representation that retains the meaning of the original document. A formal specification is written in a specification language, and the process of converting a NL document to a formal specification is called *formalization*. Automating this process is in the domain of NLP, and there are a wide variety of techniques in this field that may help us achieve the goal of partial automation.

B. Problem Statement

Automation of contract formalization is an active field of study, yet it remains a hard problem [2]. A long-term goal is to automatically formalize a raw NL contract into a target specification language. As this is currently over-ambitious, we focus on a simpler problem, namely on formalizing linguistic sub-structures that regularly appear in contracts by taking advantage of the reality that contracts are often created using templates. For a given type of contract, an organization may start with a standard template that can be customized to suit the specific needs of a party. Rather than drafting a new contract from scratch, they need only to make minor customizations to the template. Accordingly, rather than formalizing raw NL contracts, we can begin with a manually-created formalization of the contract template. The NL template is then customized by a user, and our task is now to automate the formalization

This work was partially funded by an NSERC Discovery Grant #610877 titled *Engineering Requirements for Cyber-Physical Systems* and by the ORF-RE project *CyPreSS: Software Techniques for the Engineering of Cyber-Physical Systems.*

of the customized contract template. To do this, we need only to focus on the *changes* that were made to the NL template. Furthermore, if these customizations are controlled, through a guided user interface and a restricted set of predictable operations, then the problem becomes more manageable.

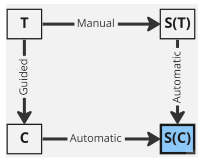

Fig. 1. Problem structure

In summary, we are building a program that allows a user to make controlled customizations to a NL contract template, which will result in predictable changes to a formal specification of the contract. Hence, given a contract template T written in NL, a manually created formal specification of this template S(T), and a customization C of the template, we want to automatically generate the formal specification S(C) for the customization. This problem structure is shown in Fig. 1.

We illustrate this process with a sample obligation that might appear in a sales contract. The specification language that we use is called Symboleo [3], which will be discussed in the next section. For now, we note that it is a formal language that can be reliably interpreted by a machine.

- **Template T:** *The Seller shall deliver the goods to the Buyer [REFINEMENT].*
- **Formalization S(T):** `Obligation(buyer, seller, true, Happens(deliver_goods))`. This formalization is done manually by someone familiar with Symboleo.
- **Customization C:** *The Seller shall deliver the goods to the Buyer within 2 weeks of payment.* This step is done by the user through a guided process.
- **Formalization S(C):** `Obligation(buyer, seller, true, HappensBefore(delivered, Date.add( payment, 2, weeks)))`. This step is automated.

In this example, T is a single obligation, but in practice it will represent a full NL contract template, consisting of multiple obligations and powers. Our task is to use the knowledge from S(T) to determine how to properly formalize the customization *within 2 weeks of payment*, which involves mapping the operator `Happens` to `HappensBefore` in the Symboleo specification. We want to push the boundaries to make user input as flexible as possible, while still having reliable formal mappings. We will therefore need a well-defined characterization of the types of NL customizations that a user can make. This comes in the form of a series of *controlled NL* (CNL) patterns. For each defined pattern, there will be a predictable mapping to a Symboleo operation. The goal of our research questions will be to determine what these patterns are.

II. Background

A. Symboleo

Symboleo [3] is a formal specification language, based on a legal ontology and axiomatic semantics, used for verifying and monitoring legal contracts. It provides constructs for specifying many legal contract concepts such as obligations and powers (norms), which are formatted as follows:

- **Obligation:** `O-id: [trigger →] O(debtor, creditor, antecedent, consequent)`
- **Power:** `P-id: [trigger →] P(debtor, creditor, antecedent, consequent)`

In an obligation, the debtor is the party that must fulfill the obligation, whereas the creditor benefits from the fulfillment of the obligation. In a power, the debtor is the party that has the right to exercise the power (e.g., to terminate or create another obligation or power), whereas the creditor is liable for the power. The antecedent is a legal situation that causes an obligation or power to be in effect. For an obligation, the consequent is a legal situation that the debtor needs to make true, and when this is done, then the obligation is fulfilled. For a power, the consequent is a legal situation which, upon becoming true, the power is exerted. There is also an optional trigger statement, which is used to instantiate a norm. Both antecedent and trigger represent conditions imposed on the norm, but the distinction is important at the formal semantic level. We illustrate these concepts with two examples.

First, consider a *payment* obligation, which specifies that the buyer must pay before the due date: `O(buyer, seller, true, HappensBefore(paid, paid.payDueDate))`. There is no trigger or antecedent for this obligation, so it is in effect as soon as the contract is activated. The buyer plays the debtor role, and therefore has an obligation to the seller, who plays the creditor role. For the obligation to be fulfilled, the *paid* event must take place before the *payDueDate* time point. If this does not happen, then this obligation will be considered *violated*.

Next consider the power: `Happens(Violated( obligations.payment)) → P(seller, buyer, true, Suspended(obligations.delivery))`. This power is created in the case of a violation of the payment obligation (i.e., the buyer has failed to pay by the due date). If that obligation is fulfilled, then we need not concern ourselves with this power. However, if payment does not happen on time, then this power is triggered, and the debtor (the seller) may now suspend the delivery obligation, and the seller no longer needs to deliver the goods.

The formal syntax of Symboleo is specified as an Xtext grammar to which we must adhere in order to produce valid Symboleo contracts. The precise semantics of the language are specified in a series of axioms exploiting event calculus [4]. One of the next main objectives for the development of Symboleo is a tool to support the partially automated translation from NL contracts to Symboleo specifications, with an aim to facilitate the creation of smart contracts.

B. Related Work

The use of a CNL in the context of smart contract formalization has precedents in existing research. Finnegan [5] discusses the advantages of using a CNL specifically for specifying contracts. It is motivated largely by standardization and the need for simplicity in legal documents for human use, rather than focusing specifically on the goal of formalization. Ideas for a contract-specific CNL are proposed, such as eliminating synonyms, redundant and archaic words, and ambiguous terms. The paper notes that a contract-writing application is a logical next step. Pace and Rosner [6] propose using a CNL for formally specifying contracts, but not specifically for targeting smart contracts, emphasizing the idea that there are *other* benefits that come with contract formalization, in addition to smart contracts. The authors accurately identify the problems associated with using unrestricted NL, motivating the use of a CNL. Tateishi et al. [7] *do* specifically apply the idea of a CNL to smart contracts by proposing a technique to automatically generate smart contracts from a NL contract using a template and a CNL. While there is indeed overlap, our work focuses specifically on the application of the monitoring of contract events, hence our use of Symboleo.

Since much of the text found in contracts is normative, there is considerable overlap with requirements engineering (RE). Wyner and Peters [8] created a framework for extracting formal specifications from NL regulations using a rules-based approach, which involves parsing and reasoning about the linguistic forms and relations in a text. Breaux and Antón [9] introduce a method that involves first converting NL requirements from privacy policies into a restricted NL and then mapping to a machine-readable format. Sleimi et al. [10] reconcile various approaches to formalization in RE literature and provide a case study targeting traffic regulations. The rule-based approach is sometimes considered in opposition to a statistical approach based on machine learning (ML), which relies on a model trained on a large dataset to extract useful information, such as spans of text that contain an obligation. This has been done in both the legal contract domain [11], [12] and also for RE [13]. Rules-based approaches tend to be more rigorous and precise, yet they are domain-specific and may not scale as well to new data. ML approaches, on the other hand, have achieved strong results, provided they have access to an annotated dataset (a major obstacle in this domain), but the results are often less formal. Buzhinsky [14] highlights this distinction and gives a survey of various techniques for formalizing NL requirements into a temporal logic. Ioannou and Michael [15] explore the idea of formalizing NL for an application where a user is giving feedback to a call assistant with the goal of improving its functionality, and apply many standard NLP techniques such as part-of-speech tagging and named entity recognition. Navas-Loro et al. [16] provided a framework for translating text from a NL contract into a formal specification called PROLEG. This translation is based on frame semantics, an idea realized in FrameNet [17].

III. Research Questions (RQs) and Methodology

A. RQ1: Customization Operations in Symboleo

Ultimately, our task involves mapping one Symboleo specification S(T) to another Symboleo specification S(C), which is achieved by performing a set of concrete operations to S(T). These operations will correspond to NL patterns entered by the user. We begin by asking:

RQ1: which Symboleo customization operations do we want our CNL to support?

We answer RQ1 by considering *all* possible Symboleo operations, and then for each one, deciding if there is any grounds for excluding it. We are working with contract templates, and the key idea of using a template is that the customization of a template serves to *refine or restrict* it to something more specific. This notion of only allowing operations that restrict the template is our main guiding principle. In our sample obligation, we restricted the delivery timeframe from being completely open to a period of within 2 weeks after payment occurs. To clarify the notion of a *restrictive* operation, we consider a sample of the predicates supported by Symboleo, some of which were already introduced:

- `Happens(event)`
- `HappensBefore(event, time)`
- `HappensWithin(event, time1, time2)`

From a purely syntactic perspective, we can see how the `HappensBefore` and `HappensWithin` predicates are refinements of the `Happens` predicate, since they contain the `event` parameter, *in addition to* other parameters. An example of a desirable Symboleo operation is therefore refining the `Happens` predicate to a `HappensBefore` predicate. We apply similar logic to refine our list of Symboleo operations.

B. RQ2: General Semantic Correspondence

From the perspective of the contract authors using a customization tool, they would interact only with NL, and need not even know that Symboleo exists. They will see a NL contract template T to which they make customizations (C) using our CNL and, behind the scenes, a Symboleo contract S(C) is automatically generated. Once we have defined our desirable Symboleo customization operations in RQ1, we need to match their semantics to corresponding NL patterns. In our example, we had the phrase *within 2 weeks of payment* as the NL that corresponded to the Symboleo `Happens -> HappensBefore` refinement operation. RQ1 considered only the syntax of Symboleo, and now we must consider the semantics.

RQ2: What are the NL structures that correspond to our desired Symboleo customization operations?

We must first point out that, due to the inherent ambiguity of NL, any correspondence between Symboleo operations and NL patterns will be imprecise. Furthermore, the very act of limiting the user input to a controlled set of NL patterns will reduce the expressiveness inherent in NL, but on the other hand will render our mappings to Symboleo operations more reliable. This is an important trade-off, and serves as a reminder that any correspondence we posit will not be precise. This is a trait of many NLP applications, and largely the reason that we aim only for *partial* automation.

In mapping our Symboleo operations to NL patterns, we introduce an important guiding principle. We want our NL patterns entered by the user to be *optional*. If the user leaves the parameter blank, then the resulting Symboleo contract will still be valid, albeit more vague. Consider our example of *within 2 weeks of payment*. If this phrase were omitted, the NL clause would still be grammatically correct, and it would map to valid Symboleo: `O(seller, buyer, true, Happens(delivered))`. We are therefore looking for linguistic constructs that deal with refinements and are grammatically optional.

Our *within 2 weeks of payment* phrase is an example of a more general linguistic concept called the adjunct, which aligns well with our requirements. *Adjuncts* are linguistic constituents that are structurally dispensable, i.e., they may be removed without affecting the remainder of the sentence [18]. They can serve many functions but are typically classified based on semantic meaning. Some examples of adjuncts (italicized) include:

- **Temporal:** The flowers must be delivered *before Friday*.
- **Conditional:** *If there is an emergency,* you must call 911.
- **Locative:** The keycard must be returned *to the front desk.*

In each of these examples, the adjunct gives more information about the event. In other words, they *refine*. Adjuncts can take many grammatical forms, such as adverbs, prepositional phrases, or subordinating conjunctions. For each of our Symboleo operations, we can define a broad set of adjuncts that have a close semantic correspondence. As an example, our operation of refining `Happens` to `HappensBefore` has a semantic correspondence with a subset of temporal adjuncts, particularly those that take the form of prepositional phrases. Each of the following examples would result in such a refinement to the sentence *The Seller shall deliver the goods to the Buyer [PARAMETER]*:

- *within 2 weeks of payment*
- *before March 30, 2023*
- *prior to the contract termination*

We also investigate the Symboleo refinement operation of adding an antecedent to a norm, another result of RQ1. The semantic equivalent would be a conditional adjunct (essentially an *if* statement). Many contracts contain information on how they can be terminated, for example through written notice or violation of certain obligations. Suppose we had the template text *[CONDITION] the buyer may terminate the contract*. The Symboleo code would be: `P(buyer, seller, true, Terminated(self))`. The *CONDITION* parameter can now be filled by the user with a conditional adjunct, such as: *If the delivery is not completed on time*. The refined Symboleo would now be: `Happens(Violated(obligations.delivery)) -> P(buyer, seller, true, Terminated(self))`.

C. RQ3: CNL based on real contracts

Although we have found NL correspondences with our desired Symboleo operations in the form of adjuncts, these are largely heuristics. We want to know the linguistic character of refinements that appear in *real contracts*. To this end, we analyze a set of real contracts, identify norm refinements as well as their corresponding Symboleo operations, and use this data as evidence to construct a CNL, which will be used by our tool. For example, we want to know if refinements similar to *within 2 weeks of payment* appear in real contracts, and if they *do* really map to the operations predicted by our heuristics from RQ2.

RQ3: What refinement patterns exist in real contracts?

We extract a set of refined NL norms from a subset of the contracts in the CUAD dataset [19]. The dataset initially contained 510 contracts, but we filtered this down to include only shorter contracts, in order to facilitate manual analysis. This resulted in 109 contracts, though future work may involve analyzing the full dataset. After manually identifying adjunct refinement patterns and corresponding Symboleo operations, we were able to extract an evidence-based CNL based on the most common pairings of patterns and operations. We offer a sample of our current findings, and illustrate them with selected examples (italicized) from the contract dataset.

- **if EVENT:** This corresponds to the addition of an antecedent. *If the Investment Adviser is required to reimburse the Fund...*
- **before EVENT/DATE:** This corresponds to a refinement of a `Happens` predicate to a `HappensBefore` predicate. *The Franchise Fee shall be paid to Grantor on or before March 31, 2017.*
- **within TIMESPAN of EVENT:** Also corresponds to a refinement of a `Happens` predicate to a `HappensBefore` predicate. *...VNUE shall reimburse Promoter within Thirty (30) Days of receipt of such accounting*.

The full CNL as well as research artifacts, code, and an EBNF-like grammar specifying valid input can be found online [1]. The resulting CNL is evaluated by splitting off a separate test set, and verifying that the patterns and operations from this test set are covered by our constructed CNL.

IV. SymboleoNLP

A. Manual Preprocessing

We have established a hypothetical CNL consisting of a set of NL patterns that will correspond to Symboleo operations. We now discuss the design of SymboleoNLP, a tool that partially automates this refinement. The tool requires as input a properly formatted NL contract template and its corresponding Symboleo specification. The creation of these inputs is a manual process, which we will now illustrate with a toy example. We start with a raw contract T0:

- *The buyer must pay the seller on or before March 20, 2024.*
- *The seller must deliver the goods within 3 weeks of payment*
- *The buyer may terminate the contract upon providing 30 days written notice*

We manually convert this to Symboleo to get S(T0):

- `O(buyer, seller, true, HappensBefore(payment, '2024/03/20'))`

[1] https://github.com/reganmeloche/symboleo-cnl

- `O(seller, buyer, true, HappensBefore(delivery, Date.add(payment, 3, weeks)))`
- `Happens(provides_notice_days(30)) -> P(buyer, seller, true, Terminated(self))`

This Symboleo specification S(T0) will now contain norms that could be *broadened*. We broaden these overly refined norms in Symboleo to get the simpler S(T):

- `O(buyer, seller, true, Happens(payment))`
- `O(seller, buyer, true, Happens(delivery))`
- `P(buyer, seller, true, Terminated(self))`

The final step is to broaden the corresponding NL on T0 to give us T by replacing the adjuncts with parameters:

- *The buyer must pay the seller REFINEMENT1*
- *The seller must deliver the goods REFINEMENT2*
- *The buyer may terminate the contract CONDITION*

We now have the required inputs, T and S(T), for SymboleoNLP. These are toy examples, but we have completed this process on 5 real contracts in varying domains. Given T and S(T), the contract author can enter refinements to T that are based on the CNL. SymboleoNLP will then automatically convert these refinements to their corresponding Symboleo operations. The result is a more specific Symboleo specification that can be used to generate a smart contract [3]. Our main requirement for the tool is therefore the mapping from a CNL refinement into the required Symboleo operation.

B. Design of SymboleoNLP

SymboleoNLP is written in Python, since it has support for many popular and powerful NLP libraries, such as nltk and SpaCy. It is also an easy-to-learn language, which lowers the bar for other developers who work on this project. Among the early decisions we make for our tool is how important entities are represented in Python, including the Symboleo specification language, the representation of complex NL concepts such as events, and also the user input.

Since the user input must adhere to the CNL, we take a structured approach where we represent the input as a series of input tokens, where each token corresponds to one or more words from the CNL. For example, consider the **within TIMESPAN of EVENT** pattern. We need a token representing ‘within’, a token representing a TIMESPAN, a token representing ‘of’, and a token representing an EVENT. Some of these tokens are *dynamic* in the text that they represent. For example, the ‘within’ token simply corresponds to the NL word “within”, but the TIMESPAN token can correspond to phrases like “2 days”, “3 weeks”, etc. The specific text for the TIMESPAN is entered by the user. Each pattern will have a set of *slots* that must be filled with values from the user. Our example pattern requires a TIMESPAN and an EVENT. In this way, our pattern-based approach is reminiscent of frame semantics [17]. The EVENT concept is especially complex since there are many ways to specify events in NL. Our tool can only represent simplified forms of events, which we acknowledge as a limitation.

With all of our patterns defined as lists of these tokens, the next goal is to provide a means for the user to enter *only* these valid sets of tokens. We want to specify a data structure that facilitates predictable and controlled selection of these tokens. Since the tokens are sequentially linked to each other in our defined patterns, we use a directed graph. We represent our tokens as nodes, which are connected based on the pattern sequences; a node will have as its children any tokens that follow it in any of our patterns. The user will traverse the graph, selecting from nodes that are the children of the previously selected node. Once they reach a leaf, they will have generated a valid CNL pattern, which will correspond to a formal operation on our Symboleo contract. Since the user will always begin with empty input, this is a rooted graph, a subset of which is shown in Fig. 2.

C. Example Walkthrough

We now provide an illustrative example. The user is presented with the sentence: *The seller shall deliver the goods to the buyer [P1]*. The user refines this obligation by filling out the *P1* parameter. SymboleoNLP guides the user through the process of selecting a valid NL pattern by traversing the graph. Beginning from the ROOT, the user can choose any of the children of the ROOT node as the first token (IF, WITHIN, BEFORE, BETWEEN). The user chooses the WITHIN token, which corresponds to the text ‘within’. As the only child of this node is the TIMESPAN node, it is automatically selected for the user. This node requires input from the user, who is prompted for a valid timespan. The user enters ‘2 weeks’, which is verified as valid. The TIMESPAN token corresponds to the text entered by the user plus the ‘of’ keyword. Our text at this point is ‘within 2 weeks of’. The children of the TIMESPAN node include EVENT and DATE. The user selects the EVENT node. The user is prompted to refine the specific event by further traversing the graph. The result is the following set of nodes, with their corresponding text values in parentheses: ROOT, WITHIN (within), TIMESPAN (2 weeks of), EVENT, OBLIGATION_EVENT, OBLIGATION_EVENT_NAME (payment), OBLIGATION_EVENT_ACTION (being completed). The processing takes care of rendering the text in a grammatically correct format: ‘within 2 weeks of payment being completed’.

The structured CNL must then be mapped to the proper Symboleo operation. This is done by first determining the pattern expressed by the CNL (in this case, the within TIMESPAN of EVENT pattern). This pattern is then hooked up to a special handler function, which will process the TIMESPAN and EVENT arguments on the pattern into the refined Symboleo: `O(seller, buyer, t, HappensBefore(delivery, Date.add(2, weeks, payment)))`. This mapping process is straightforward, but there is potential to incorporate various NLP tools as it grows to handle more complex event specifications. The tool is evaluated using a full suite of unit tests as well as integration tests which validate its operation on a set of real contracts.

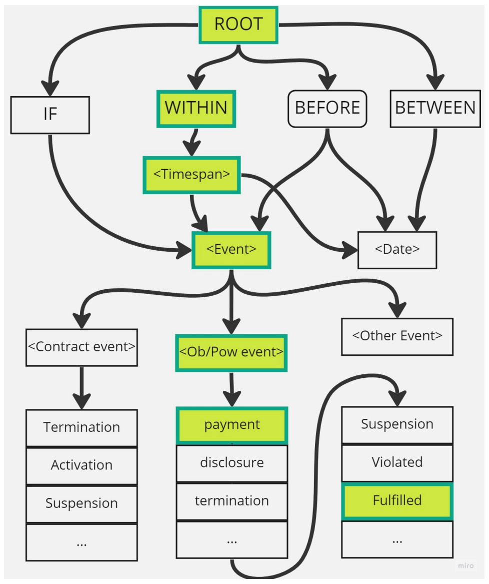

Fig. 2. Diagram showing the rooted graph structure

We have shown a straightforward example of SymboleoNLP to illustrate the customization process. In the bigger picture, contracts are complex and varied, presenting many challenges to this approach. Much of the legalese contained in contracts takes the form of long sentences, which are generally harder to formalize. Ambiguities in NL are also challenging and can be present in the form of attachment ambiguities or coreferences. The CNL patterns defined here for the application are based on patterns found in the CUAD contract dataset [19]. While this dataset covers a wide variety of contracts in many domains, any bias in that dataset could be carried over to our work. For example, the application may have trouble scaling to a specific domain that was not represented in the dataset. The code for the tool can be found online [2].

V. Conclusion

In this paper, we have presented preliminary answers to our research questions, which aim to find valid CNL patterns to use as parameters in a template-based contract customization tool. For RQ1, we defined broad and refined concepts with respect to Symboleo to decide which operations serve to *refine* a contract. RQ2 focused on finding semantic NL alignment with these Symboleo operations, and we found that the linguistic adjunct aligns well with our needs, although it is an open-ended category. RQ3 builds on RQ2 by analyzing real contracts to determine the most relevant patterns that will make up our CNL. We validate the CNL patterns by manually constructing Symboleo specifications from real contracts and applying the various refinement operations. SymboleoNLP is a tool that allows a contract author to refine a broad NL contract template using the CNL, and automatically generates a corresponding refined Symboleo specification, which can be used to generate a smart contract for different applications. The tool is currently a proof of concept, and future work will involve validating the utility of the tool in real world contract authoring scenarios and exploring use cases beyond Symboleo. Currently our CNL only covers contract refinements, but there is potential to expand it to incorporate broader contract elements (for example, entire obligations) and work towards fully automated formalization. This may involve integration with other NLP techniques and libraries.

[2]https://github.com/reganmeloche/symboleo-nlp

References

[1] V. Dwivedi, V. Pattanaik, V. Deval, A. Dixit, A. Norta, and D. Draheim, "Legally enforceable smart-contract languages: A systematic literature review," *ACM Comput. Surv.*, vol. 54, no. 5, jun 2021.
[2] M. Soavi, N. Zeni, J. Mylopoulos, and L. Mich, "From legal contracts to formal specifications: A systematic literature review," *SN Comput. Sci.*, vol. 3, no. 5, p. 345, 2022.
[3] A. Parvizimosaed, S. Sharifi, D. Amyot, L. Logrippo, M. Roveri, A. Rasti, A. Roudak, and J. Mylopoulos, "Specification and analysis of legal contracts with Symboleo," *Softw. Syst. Model.*, vol. 21, no. 6, pp. 2395–2427, 2022.
[4] M. Shanahan, *The Event Calculus Explained*. Springer, 1999, pp. 409–430.
[5] M. Finnegan, "From a natural language to a controlled contract language," *Jusletter IT May*, vol. 24, 2018.
[6] G. J. Pace and M. Rosner, "A controlled language for the specification of contracts," in *Controlled Natural Language*, N. E. Fuchs, Ed. Springer, 2010, pp. 226–245.
[7] T. Tateishi, S. Yoshihama, N. Sato, and S. Saito, "Automatic smart contract generation using controlled natural language and template," *IBM J. Res. Dev*, vol. 63, no. 2/3, pp. 6:1–6:12, 2019.
[8] A. Wyner and W. Peters, "On rule extraction from regulations," in *Legal knowledge and information systems*. IOS Press, 2011, pp. 113–122.
[9] T. Breaux and A. Anton, "Analyzing goal semantics for rights, permissions, and obligations," in *13th IEEE International Conference on Requirements Engineering (RE'05)*, 2005, pp. 177–186.
[10] A. Sleimi, N. Sannier, M. Sabetzadeh, L. Briand, and J. Dann, "Automated extraction of semantic legal metadata using natural language processing," in *2018 IEEE 26th International Requirements Engineering Conference (RE)*, 2018, pp. 124–135.
[11] R. Funaki, Y. Nagata, K. Suenaga, and S. Mori, "A contract corpus for recognizing rights and obligations," in *Proceedings of the Twelfth Language Resources and Evaluation Conference*, May 2020, pp. 2045–2053. [Online]. Available: https://aclanthology.org/2020.lrec-1.251
[12] I. Chalkidis, I. Androutsopoulos, and A. Michos, "Obligation and prohibition extraction using hierarchical RNNs," in *56th Annual Meeting of the Association for Computational Linguistics*, vol. 2, Jul. 2018, pp. 254–259. [Online]. Available: https://aclanthology.org/P18-2041
[13] A. Sainani, P. R. Anish, V. Joshi, and S. Ghaisas, "Extracting and classifying requirements from software engineering contracts," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, 2020, pp. 147–157.
[14] I. Buzhinsky, "Formalization of natural language requirements into temporal logics: a survey," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, pp. 400–406.
[15] C. Ioannou and L. Michael, "Knowledge-based translation of natural language into symbolic form," in *7th Linguistic and Cognitive Approaches To Dialog Agents Workshop-LaCATODA*, 2021, pp. 24–32.
[16] M. Navas-Loro, K. Satoh, and V. Rodríguez-Doncel, "Contractframes: Bridging the gap between natural language and logics in contract law," in *New Frontiers in Artificial Intelligence*. Springer, 2019, pp. 101–114.
[17] C. J. Fillmore and C. Baker, "313 A Frames Approach to Semantic Analysis," in *The Oxford Handbook of Linguistic Analysis*. Oxford University Press, 12 2009.
[18] J. Lyons, *Introduction to theoretical linguistics*. Cambridge University Press, 1968, vol. 510.
[19] D. Hendrycks, C. Burns, A. Chen, and S. Ball, "CUAD: an expert-annotated NLP dataset for legal contract review," *CoRR*, vol. abs/2103.06268, 2021.