International Journal of Cognitive Computing in Engineering 7 (2026) 40–57

Contents lists available at ScienceDirect

International Journal of Cognitive Computing in Engineering

journal homepage: http://www.keaipublishing.com/en/journals/international-journal-of-cognitive-computing-in-engineering/

Transforming legal texts into computational logic: Enhancing next generation public sector automation through explainable AI decision support

Markus Bertl [a,b,*], Simon Price [a,c], Dirk Draheim [d]

[a] NextGen Computing Research Group, Unisys, 801 Lakeview Drive, Blue Bell, PA, USA
[b] Department of Information Systems and Operations Management, Vienna University of Economics and Business, Welthandelsplatz 1, Vienna, 1020, Austria
[c] Department of Computer Science, University of Bristol, Woodland Road, Bristol, BS8 1UB, UK
[d] Information Systems Group, Tallinn University of Technology, Akadeemia Tee 15A, Tallinn, 12618, Estonia

ARTICLE INFO

Keywords:
Artificial intelligence
Decision support systems
Expert systems
Inference mechanisms
Intelligent automation
Law
Logic
Text mining
Text processing

ABSTRACT

This research presents a novel approach for translating legal texts into machine-executable computational logic to support the automation of public sector processes. Recognizing the high-stakes implications of artificial intelligence (AI) in legal domains, the proposed method emphasizes explainability by integrating explainable AI (XAI) techniques with natural language processing (NLP), employing scope-restricted pattern matching and grammatical parsing. The methodology involves several key steps: document structure inference from raw legal text, semantically neutral pre-processing, identification and resolution of internal and external references, contextualization of legal paragraphs, and rule extraction. The extracted rules are formalized as Prolog predicates and visualized as structured textual lists and graphical decision trees to enhance interpretability. To demonstrate the automatic extraction of explainable rules from legal text, we develop a Law-as-Code prototype and validate it through a real-world case study at the Austrian Ministry of Finance. The system successfully extracts executable rules from the Austrian *Study Funding Act*, confirming the feasibility and effectiveness of the proposed approach. This validation not only underscores the practical applicability of our method, but also highlights promising avenues for future research, particularly the integration of Generative AI and Large Language Models (LLMs) into the rule extraction pipeline, while preserving traceability and explainability.

1. Introduction

Legal systems around the world are characterized by an extensive body of written texts that govern human conduct, regulate societal interactions, and administer justice. These texts encompass statutes, regulations, contracts, court opinions, and a myriad of other legal documents, often drafted in dense, complex, and specialized language. Interpreting and applying the law has, historically, been the domain of legal professionals who possess the necessary training and expertise to navigate this intricate terrain. However, as society becomes increasingly data-driven and digitally connected, there is a growing need for tools and methods that can extract, formalize, and reason with legal rules more effectively and efficiently. Turning rules in natural language into machine-processable code is often referred to as Rules as Code, Policy as Code, or, in our case, Law as Code. McKinsey estimates that more than 60 percent of the work done by public administration officials to provide services could, in principle, be automated by these approaches.[1] The Organisation for Economic Co-operation and Development (OECD) stated the following benefits of transforming rules written in natural language into machine-processable code (Mohun & Roberts, 2020):

(i) *Consistency:* Coded rules propose that governments create an authoritative version of rules in software code, which allows rules to be understood and actioned by computer systems in a consistent way.
(ii) *Agility and Responsiveness:* Agility and responsiveness support a truly digital government, with rules created as a digital product and service, rather than having them incorporated into digital processes according to (i). This can make the government more agile, responsive, and innovative in navigating and shaping an unpredictable operating environment.
(iii) *Improved Policy Outcomes:* Coded rules seek to minimize the risk of discord between policy intent and implementation outcomes,

* Corresponding author at: Department of Information Systems and Operations Management, Vienna University of Economics and Business, Welthandelsplatz 1, Vienna, 1020, Austria.
E-mail addresses: markus.bertl@wu.ac.at (M. Bertl), simon.price@bristol.ac.uk (S. Price), dirk.draheim@taltech.ee (D. Draheim).

[1] https://www.mckinsey.com/industries/public-sector/our-insights/unlocking-the-potential-of-public-service-digitization

https://doi.org/10.1016/j.ijcce.2025.07.003
Received 20 May 2025; Received in revised form 8 July 2025; Accepted 26 July 2025
Available online 20 August 2025

achieve better policy outcomes, and create opportunities to improve the speed and consistency of service delivery dramatically.

(iv) *Digital Optimization:* Coded rules address the reality that our world is increasingly digital and that, by extension, governments need to optimize their operations for this.

(v) *Scrutiny of Rules:* The potential for increased efficacy and speed also suggests an increased need for scrutinizing rules and how they are made, to ensure rules are wanted, necessary, and accurately reflect the legal intent.

One promising avenue for addressing this need is the extraction of legal rules in the form of first-order predicate logic. The connection between formal logic and law is deeply rooted in history. Aristotle developed logical frameworks, the so-called syllogistic logic, which influenced legal reasoning and developed further into the medieval scholasticism which was used to interpret laws and resolve legal disputes. The formal system of first-order logic allows for precise and unambiguous representation of knowledge and rules, making it an ideal candidate for capturing the essence of legal provisions. This endeavor involves the transformation of natural language statements found in legal texts into a structured, symbolic representation that can be leveraged for automated reasoning, legal analytics, and decision support systems. The ultimate goal is to hyper-automate public administration and make legal information more accessible, interpretable, and computable, enabling a wide range of applications such as legal research, compliance monitoring, contract analysis, and artificial legal intelligence.

In recent years, significant progress has been made in the development of techniques and methodologies for extracting rules in first-order predicate logic from legal text. These advances are driven by the convergence of natural language processing (NLP), machine learning, and formal logic, offering a multidisciplinary approach to the complex task of translating the nuances of legal language into formalized rules. However, several challenges persist, including the ambiguity inherent in natural language, the context-dependency of legal reasoning, and the need for scalable and accurate algorithms capable of handling vast volumes of legal texts. Addressing these challenges represents a crucial step toward harnessing the power of technology to assist and augment legal professionals. As we delve into the intricacies of this burgeoning field, in this paper, we aim to shed light on the path ahead and the transformative potential it holds for the future of legal practice and scholarship. Our approach to extract rules from text (called "text extraction" in Fig. 1) consists of:

1. Document structure inference from the raw legal text
2. Semantically neutral pre-processing
3. Recognition of internal and external references
4. Target resolution for internal references
5. Legal paragraph contextualization and, finally
6. Rule extraction

Additionally, we convert the extracted rules to Prolog predicates and visualize the rules as textual lists and graphical decision trees. Our developed Law as Code prototype, Law as Code Decision Support System (LCDSS), has been evaluated as a proof-of-concept at the Austrian Ministry of Finance, where it successfully demonstrated the automatic extraction of explainable rules from the Austrian *Study Funding Act*.

Our approach suggests promising future research directions, in particular, the integration of GenAI Large Language Models (LLMs) into the rule extraction process, while retaining provenance and explainability.

We proceed as follows. In Section 2, we start with an overview of the current state of the field, exploring techniques and tools employed in related work, highlighting both their successes and limitations, and comparing these with our own work. In Section 3, we provide a summary of essential background results utilized in our solution. In Section 4, we explain the methodology of our study. In Section 5, we provide a detailed description of our LCDSS, including its architecture, design decisions and implementation. Throughout the section, we use the application of our LCDSS to Austrian legislation as running example. In Section 6, we provide a validation of LCDSS. We provide an in-depth discussion of our results including future directions in Section 7 and finish with a conclusion in Section 8.

2. Related work

Extracting rules from legal text is not a new concept but rather a culmination of decades of research and practice in various fields, such as artificial intelligence, logic, natural language processing, and law. The literature on can be broadly categorized into three main themes: rules mapping, policy/governance as code, and logic-based reasoning.

Wong described seven maturity levels of rules as code, from paper-based legislation to rules and contracts which universally appear as digital, executable code (Wong, 2020). But transforming law into executable code is not a new concept. Already in 1992, Wilson wrote about rules mapping of legal text (Wilson, 1992).

Biagioli et al. proposed a method for annotating normative texts, which are composed of formal partitions or semantic units, to facilitate easier retrieval of norms. Their approach involves an analytical activity to classify text portions into provision types and extract their arguments based on a metadata scheme. In their paper, they introduced two modules designed to automate this process, qualifying text fragments in terms of provision types and extracting their arguments (Biagioli, Francesconi, Passerini, Montemagni, & Soria, 2005).

In 2020, Libal proposed a meta-level annotation language for legal texts which he applied on the GDPR's article 13 (Libal, 2020).

Korger and Baumeister's work focuses on the semantic concepts underlying regulatory documents in various contexts, including daily life, technical, business, and political. They provide examples of these concepts in the domains of nuclear safety and public events, highlighting the complexity of broader semantic concepts in their textual representation. The authors propose a rule-based approach for managing these concepts and extracting their interrelations (Korger & Baumeister, 2021).

Dragoni et al. proposed using a combination of syntax-based patterns extracted using the Stanford Parser, and logic-based ones extracted through the Boxer framework. They evaluated their approach with the Australian "Telecommunications consumer protections code" (Dragoni, Villata, Rizzi, & Governatori, 2016).

Tammet et al. developed an experimental logic-based pipeline capable of performing inference and providing explained answers to natural language queries. The pipeline integrates semantic parsing, large knowledge bases, automated reasoning using extended first order logic, and the translation of proofs back into natural language. The ultimate aim is to develop hybrid neurosymbolic systems that combine machine learning and large language models (LLMs), enhancing trustworthiness and explainability, and interfacing natural language with external tools such as database systems and scientific calculations (Tammet, Järv, Verrev, & Draheim, 2023).

Prolog as a knowledge base for legal reasoning has already been researched extensively in the context of Expert Systems. In 1986, Sergot et al. showed the feasibility of using Prolog as a rule base for legal reasoning by manually formalizing the British Nationality Act as a Prolog program (Sergot et al., 1986). The manual extraction of the rules took a single person about two months (Borrelli, 1989). This approach has been further discussed and evaluated by Gordon (1987). A manual approach to encoding Bulgarian law in Prolog notation has also been undertaken by Yalumov and Gargov (Yalumov & Gargov, 1986).

More recently, Morris surveyed and evaluated declarative logic programming tools designed specifically for legal service automation (Morris, 2020). He argues that currently, no easy-to-use tool for

reasoning over legal text exists. As a result, he published Blawx,[2] a framework for visualizing, managing, and executing rules that are formulated as declarative logic statements and visualized in Blockly, a Google platform for visual coding.

Apart from the legal domain, policy/governance as code principles are also already used in cyber security to e.g., automatically create access control policies (Xiao, Paradkar, Thummalapenta, & Xie, 2012).

In Mowbray, Chung, and Greenleaf (2023), Mowbray et al. propose the programming language *yscript* for expressing legislative rules. The expressive power of *yscript* is, basically, propositional logic. The interpreter of *yscript* generates explanations during its derivations. Also, Mowbray et al. (2023) report on progress in generating *yscript* from legal text.

In Merigoux, Chataing, and Protzenko (2021), Merigoux et al. propose Catala, which is based on OCaml, as a programming language for the law, to express legal rules in an computationally executable way.

What sets our work apart is the automated extraction of rules and integration into a workflow with case management. To our advantage, our extracted rules are understandable by non-specialists, in text and graphical rendering of the extracted computational logic. We further provide a complete security model, an auditable environment where all changes to the rules are documented, and testable using sophisticated impact analysis that allows a detailed assessment of the implication of changes in rules.

3. Background – formulating law as logic

Law, in the context of this paper, is defined as a set of rules created by social or governmental institutions to regulate behavior. There are two fundamentally contrasting systems of public law, known generically as legislative law and case-based law.

- *Legislative law, or statutory law*, is created by legislative bodies such as parliaments and congresses. Such legislation is written in a formal, structured manner and provides a set of rules that apply to everyone within the jurisdiction. Legislative laws can be amended, repealed, or new ones can be enacted by the legislative body.
- *Case law* is created by courts through their decisions in individual cases. Case law is written in the form of judicial opinions and operates on the principle of precedent, meaning that the decisions made in previous cases guide the decisions in future similar cases. Case law can only be overturned or modified by a higher court or, in some cases, by legislation.

Both types of law are usually defined using natural language statements written in the official language or languages of the jurisdiction. Natural language can be notoriously imprecise, leading to ambiguities, inconsistencies, contradictions and subjectivity in the interpretation of rules. Translation between languages, interactions between rules in the same law and between different laws further complicate their interpretation. To mitigate these inherent problems of natural language, an entire profession of highly trained legal professionals has emerged, whose job it is to write down the rules more precisely in laws and to more consistently interpret their meaning so that they can be applied to society.

At this point it is relevant to also mention contract law, which shares characteristics with both legislative law and case law. Unlike legislative law and case law, which apply broadly to all individuals within a jurisdiction, contract law applies specifically to the parties that have agreed to the contract. The number of contracts in existence is orders of magnitude greater than the number of jurisdiction-originated laws and so it is important to consider how rule extraction relates to contract law; rule extraction for contract law could have greater applicability, in some sense, than legislative and case laws.

Like legislative law, contract law involves formal written structured documents, that use a specific legal language to establish rules to be followed. However, like case law, contracts can be interpreted by courts, and previous court decisions on similar contracts can influence the interpretation of a contract. In some circumstances, legislation can overrule case law precedents set by courts. In the context of extracting rules from text, contract law can be viewed as a complex blend of legislative and case law. In other words, for the purposes of rule extraction, contract law is covered by our treatment of legislative and case law. Therefore, we do not treat contract law separately in this paper and we leave to future work the open problem of how contract law blends these two contrasting settings for the rule extraction covered in this paper.

Legislation and case law can be conceptualized as a logical model: legislation, composed of laws, forms the base rules of this model, while case law, derived from judicial decisions, adds dynamic, context-dependent rules based on the interpretation and application of these laws in specific cases. For any specific situation, the relevant laws and precedents are interpreted and applied, leading to logical implications. This perspective allows for a structured approach to understanding and applying law, reflecting the dynamic and adaptable nature of legislative systems. Therefore, laws can be viewed as a form of logic, continuously evolving and adapting to societal changes and judicial interpretation.

However, legislative and case law each have their own interpretational approaches. The logical interpretation of the rules in their systems is arrived at by different methods: legislative law is interpreted based on the rules encoded in the legislation by lawmakers, while case law is interpreted based on judicial decisions and precedents. In some sense, they each have their own "algorithms" for interpretation, enacted by legal professionals rather than computationally by software. Below we present logical formulations of both legislative and case law in the context of their respective "algorithms", in preparation for their reformulation as computational logic.

3.1. Law as propositional logic

We begin by formulating legislative law as *propositional logic*, a branch of logic the deals with propositions, which are statements that can be either true or false. Logical connectives are used to combine these propositions into more complex ones. The set of logical connectives is "not", "and", "or", "if-then" and "if-and-only-if", or more formally, $\{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$". From the perspective of Propositional Logic, each rule in natural language legislation aims to precisely define a relation of the form "if p then q" ($p \Rightarrow q$) for propositions p and q. The intended interpretation is that "if p holds true, then q must also be true", and conversely, "if p does not hold true, then q does not hold true". Consider, for instance, the rule "if an individual has an Austrian passport, then the individual is an Austrian citizen", and its converse. Such natural language rules can, in theory, be accurately represented as propositional statements. Assuming that the truth value of each proposition p can be ascertained, an inference algorithm can compute the truth value of q for each rule in a transparent and unambiguous manner. Returning to our example, if it can be verified from a government database that a certain individual has an Austrian passport, it can be inferred from the rule that the individual is an Austrian citizen. Provided that suitable data is accessible to determine the truth value of all propositional representations of legislative rules, their logical interpretation is deterministic and circumvents the aforementioned issues associated with the interpretation of natural language rules by humans.

Although legislative law can be precisely formulated as statements in propositional logic, such a representation lacks the expressive power to do so concisely, as it requires large numbers of statements to represent the complex relations found in legal rules. For this reason, we turn to first-order predicate logic, a more expressive language for representing and reasoning about knowledge.

[2] https://law.mit.edu/pub/blawxrulesascodedemonstration/release/1

3.2. Law as first-order predicate logic

First-order predicate logic (FOPL) extends propositional logic by introducing predicates, quantifiers, and variables. Predicates allow us to make statements about specific entities in the domain of discourse. Quantifiers "for all" (∀) and "there exists" (∃), let us make statements about all or some entities in the domain. Variables represent these entities. Importantly, when we switch to a computational context, FOPL also naturally represents the relational model, including its schema, data and relational queries on that data expressed as either relational algebra or relational calculus. Although beyond the scope of this paper, FOPL's ability to describe the relational model provides a convenient bridge from variables in FOPL representations of rules to the real-world data values held in government and other databases, such as those in the planned EU Single Digital Gateway (SDG).[3]

Returning to our earlier example, the rule "if an individual possesses an Austrian passport, then the individual is an Austrian citizen" can be represented in FOPL as:

$$\forall x(\text{HasAustrianPassport}(x) \Rightarrow \text{AustrianCitizen}(x)).$$

Here, HasAustrianPassport(x) is a unary predicate relation that is true when variable x can be unified (bound) to a ground truth fact of the form, e.g. HasAustrianPassport(Alice), such that x = Alice. Intuitively, a set of such facts is analogous to a table in a relational database, or column/s in a spreadsheet. For example, let our ground truth data be the set:

$$\{\text{HasAustrianPassport(Charlie)}, \text{HasAustrianPassport(Xin)}, \text{HasAustrianPassport(Alice)}, \ldots, \text{HasAustrianPassport(Bob)}\}.$$

By implication, the predicate AustrianCitizen(x) will be true for x if HasAustrianPassport(x) can be unified with a ground fact, e.g. x = Alice, by HasAustrianPassport(Alice), which implies AustrianCitizen(Alice). If we also consider the ∀ quantification, our example FOPL rule states that for all x, if x has an Austrian passport, then x is an Austrian citizen. In our example, this rule quantification implies the truth of AustrianCitizen(Charlie), AustrianCitizen(Xin), AustrianCitizen(Alice), … , AustrianCitizen(Bob).

There are numerous other types of logic that extend FOPL in various ways, including *decision logic* and *higher-order logic*, but given that our objective is to support the legal profession rather than experts in logic, the relative simplicity of FOPL has, in our opinion, readability advantages for our use case. It also has the advantage that there is a widely available computational logic that is based on FOPL.

3.3. Law as computational logic

Although legal rules can be expressed in mathematical FOPL, to make those rules executable requires their translation into a computer program. Although most computer programming languages are capable of encoding rules expressed in FOPL, the Prolog language facilitates the translation from pure FOPL to a practical implementation as machine-executable computational logic — while retaining much of the syntax and semantics of FOPL.

For the remainder of this paper, we adopt Prolog notation as we extract rules from text to formulate laws as programs in a computational logic. Note that in mathematical logic, variables begin with lowercase letters and functors/atoms with uppercase. The reverse is true in Prolog, and so our earlier example in Prolog becomes,

```
austrian_citizen(X) :- has_austrian_passport(X).
```

where `X` is the unbound variable. For further information on Prolog (Flach & Sokol, 2022).

In the following section we discuss rules extraction, formulating extracted rules as Prolog terms with unbound variables, such as `X` in the above example. We do not address the non-trivial problem of binding these variables to data in the real world, for example via queries to government databases. That challenge is currently being addressed by the EU Single Digital Gateway (SDG) initiative (European Comission], 2018), which aims to provide a common interface to citizen data across the EU.

4. Methodology

The Law as Code Decision Support System (LCDSS) has been developed using Action Design Research (ADR) (Sein, Henfridsson, Purao, Rossi, & Lindgren, 2011). The created artifact has been evaluated using scenario-based evaluation (Hevner, March, Park, & Ram, 2004), as well as unstructured interviews with three experts from the Department of Legal Informatics at the University of Vienna, four e-Government experts from the Austrian Ministry of Finance and two IT experts from the Austrian National Computing Center.

5. The Law-as-Code Decision Support System LCDSS

5.1. Architecture

Our LCDSS has been implemented like follows. Initially, a given legal document is passed to our text extraction component, which then returns the corresponding rule candidates. Extracting rules from legal texts alone is not enough to prove a system is usable for public and private organizations. The extracted rules need to be stored, validated, governed and applied. To meet those requirements, we integrated the Text Extraction component described above into a Streamlit user interface, which submits a JSON file with the extracted rules to a REST API. Within the API, this JSON file is stored and versioned in a Couchbase database. Since logic rules can be quite complex to understand, a rules visualizer and editor component serve as a graphical user interface that renders the created FOPL statements in textual, graphical and Prolog formats. This enables a domain expert, like a lawyer, to evaluate whether the extracted rules correctly represent the law. If needed, tracked manual changes to the extracted rules can be made. This human-in-the-loop ensures that only legally sound rules are applied. After the correctness of the rules has been approved, they become accessible to the rules engine. Records of all changes are kept in an audit log. The stored rules are then accessible in the rules engine and their execution can be triggered via REST API.

This process is depicted in Fig. 1. In step one, the text extraction is initiated, and the law about study grants is converted into FOPL rules. Once a lawyer has reviewed the extracted rules in the rules editor and confirmed that they are a valid representation of the legal text, the rules are made accessible in the rules engine. The rules engine can then be queried over REST with to determine whether a citizen, in this case, John Doe, is eligible for a study grant. The rules engine then automatically selects the appropriate rules from the database and is then able to use backward reasoning to look up what type of data is needed to execute the rules. This data can then be loaded from external data sources or registries or solicited from a human. Once all the required data is available, the rules are executed, and the rules engine returns the result.

From a performance perspective, the system operates in two distinct modes. The Text Extraction pipeline (Section 5.2) is an offline, computationally intensive process. Its latency is dependent on the length and complexity of the legal document and the number of extraction patterns, but as it is not a real-time user-facing task, this is acceptable. In contrast, the Rules Engine query is designed for real-time decision support. The backward reasoning can be handled by a standard Prolog

[3] https://ec.europa.eu/isa2/sites/default/files/dlv06.01-_final_report.pdf

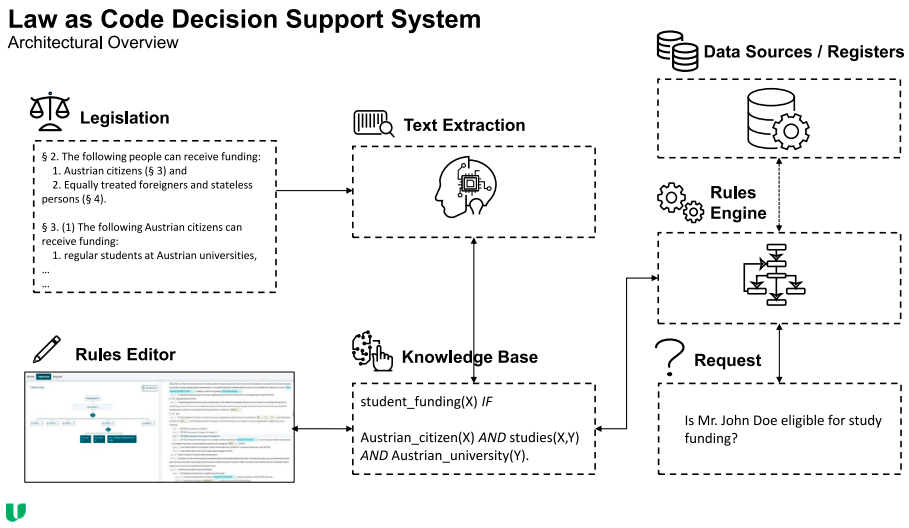

Fig. 1. Architecture of the "Law as Code" Prototype.

engine, whose computational complexity is a function of the number of rules and the depth of the logical search tree. The system's latency for a typical eligibility query is low, making it suitable for interactive decision support applications.

5.2. Text extraction

The rules extraction process consists of two steps:

- Document Structure Inference (Section 5.2.1)
  - Semantically Neutral Pre-processing (Section 5.2.1.1)
  - Legal Paragraph Hierarchy Inference (Section 5.2.1.2)
  - Internal Reference Parsing and Resolution (Section 5.2.1.3)
  - External Reference Parsing (Section 5.2.1.4)
- Application of Rule Extraction Patterns (Section 5.2.2)

After the application of the rule extraction logic, rules which can then be rendered as Prolog rules and/or RuleML are produced, and can be sent to an external application to visualize hierarchically or graphically as a decision tree. The rules rendering for the rules editor is described in (Section 5.3).

To demonstrate the rules extraction process, we also implemented an interactive Streamlit application to configure each step in the process and view the intermediate outputs of each processing step. However, the text extraction modules could also be used independently of the user interface. Configuring the process requires technical expertise that is likely to be outside of the skill set of most legal professionals, for instance, knowledge of regular expressions and of natural language grammar patterns. However, our intent is that the extracted rules can be rendered in formats that can be understood and verified by legal professionals with minimal training. As will be discussed later in this paper, the inclusion of a human-in-the-loop is essential when seeking to execute law as code.

5.2.1. Document structure inference

Legal texts are available in a variety of formats across different legislatures, including plain text, HTML, XML, DOCX, PDF. Semi-structured formats, such as HTML and other XML formats, provide readily accessible structural information about the legal texts that they encode. In particular, the delineation of headings, paragraphs, numbering, internal and external cross-references are unambiguously marked with embedded metadata tags. Such structural information provides important semantic context for the understanding of the text and is fundamental to the way that law is written and interpreted by humans. It seems reasonable to assume that document structure is just as important to the machine extraction of rules from natural language text as it is to humans.

Unfortunately, at the time of writing, legislative bodies are in the early stages of digitizing legal texts and their focus to-date has been on making raw human-readable document formats publicly available. While semi-structured representations are sometimes also available, there is no widespread standardization of their format, nor of the metadata schema used. Indeed, we found the availability of semi-structured formats to be incomplete within individual jurisdictions and, worse still, published through bespoke access methods and APIs. Therefore, to assure the widest possible applicability to available formats, we adopted the universally available plain text, purely natural language, format as the input for our solution. We deferred the problem of accessing this information directly from the source web pages or APIs, instead choosing to manually cut and paste plain text into our user interface. Clearly, a production system would need to address the non-trivial problem of automated access to legal texts and, hopefully, as the digitization of laws matures, the emergence of standards would simplify this process.

Starting from plain text, our approach to rule extraction first requires segmentation of the semantically important document structures by identifying the location of structural elements like titles, paragraphs, cross-references, etc., and recording these as metadata. Traditionally, this "mark-up" process is performed manually by inserting HTML/XML tags into the document or through bespoke annotation tools. Such manual processes are time-consuming, error-prone and need to be repeated from scratch when each new version of the legislative text is published. Instead we automatically parse the text to infer the document structure and create the associated metadata. In part this is achieved through trivial use of regular expressions but more sophistication is required to correctly infer the document hierarchy and to resolve the intended meaning of cross-references. The latter merits separate treatment in Sections 5.2.1.3 and 5.2.1.4 below.

5.2.1.1. Semantically neutral pre-processing. Legal texts evolve over time through a series of manual edits that inadvertently introduces inconsistencies in the layout, titling, numbering and cross-referencing that makes the task of parsing more complex than it may seem at first sight. In addition, our workaround of cutting and pasting text from other document formats tends to bring along unwanted artifacts, such as hidden anchor text from HTML pages. In our case, we also worked with a machine translated English version of original German language

Austrian law, which introduced an extra layer of inconsistencies, such as sometimes translating "§" as itself and sometimes as the word "section" or "article". Some pre-processing of the text can normalize these inconsistencies and strip out unwanted artifacts. However, in the legal domain, this pre-processing must be transparent and reproducible so that an audit trail is maintained back to the original source document. It must be demonstrable that any pre-processing changes to the text are semantically neutral; that they do not change the intended meaning of the law. We implement such pre-processing through a domain-specific language that specifies a commented sequence of regular expression substitutions, as shown by the fragment below.

```
# strip out markup anchor text before each section
  ↪ start
'\n§\s\d+[a-z]*\nText\n' --> '\n'

# normalize section titles to always use § as the
  ↪ prefix
'\n(?:[Ss]ection|[Aa]rticle) (\d+)\s' --> '\n§\1 '
```

Lines beginning with # are comments used to document transformations. The result of each 'REGEX' –> 'REPLACEMENT' line is a global search for matches of the specified REGEX pattern and its substitution with a REPLACEMENT string, which may contain back references to captured text as \1, \2, \3, etc.

The 19 regular expression substitutions applied to the Austrian *Study Funding Act*[4] were manually crafted but are likely to work unchanged on future versions of the same document, and over similarly formatted legal texts from the same legislative body. However, it should be noted that, unlike regular expressions, Generative Pre-trained Transformer (GPT) LLMs like (Vaswani et al., 2017) are resilient to minor inconsistencies in text. A future LLM-based enhancement of rule extraction may be able to skip this step entirely or, at least, remove the need for regular expression expertise to create such a transparent and repeatable list of pre-processing transformations.

Ultimately, in the longer-term view of the Law as Code movement, there would be no need to perform this document structure inference step because laws would be written and maintained in a machine-readable structured format in the first place.

5.2.1.2. Legal paragraph hierarchy inference. Each heading or paragraph in the pre-processed text is indexed by a unique legal-format numbered string, e.g. 'III.5.68.1.5', that is extracted or inferred from the text structure. For human-readability, legal headings and paragraph numbers tend to use concise numbering schemes that rely on their location within the document and common sense reasoning to arrive at a global index that is unique within the document. For example, headings and paragraphs in the Austrian *Study Funding Act* are numbered to six levels but at each level they are only indexed by a single number (a local index), which by itself does not uniquely identify the heading or paragraph. However, by keeping track of the location of each heading or paragraph within the document it is straight forward to infer a global index for each local index, as illustrated in Table 1.

Local index numbers for each of the six levels are recognized and extracted by six regular expressions. There is also a special case seventh level for footnotes that have wider scope than their immediate ancestor paragraph. An arbitrary number of levels may be defined to suit the particular numbering scheme of any given legal text format. The parser combines the level (depth) of each heading or paragraph with the local index number and locational context, built up in a dynamically constructed hierarchical structure, to arrive at its global index string. The global index and its associated hierarchical structure proves valuable for de-referencing cross-references, for providing semantic context during rule extraction and, importantly, for transparent linking of extracted rules to their precise origin in the legal text.

[4] https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10009824

Table 1
Inferring global index of paragraph 'III.5.68.1.5' from its location within the text.

Source text	Local index	Global index
III. MAIN PIECE - OTHER …	III	III
SECTION 5 Scholarships	5	III.5
§68. Study support	68	III.5.68
(1) Within the framework of the …	1	III.5.68.1
5. for the promotion of stays abroad,	5	III.5.68.1.5

5.2.1.3. Internal reference parsing and resolution. Legal texts make extensive use of internal cross-references. The Austrian *Study Funding Act* document has 900 internal references that are resolvable to locations within the same document. Table 2 shows a selection of internal references that illustrate the variability of referencing styles and how important the location of the source text is in intelligently resolving local indexes to arrive at global indexes. The resolution of the section range, "§§20 to 24", in the final row of the table is cautionary because one might expect there to be five sections within that range, i.e. 20, 21, 22, 23, 24. In fact, sections 21 and 22 appear to have been repealed and no longer exist. Similarly, the legal practice of inserting additional paragraphs, with non-numeric alphabetical suffixes, in between existing numbered paragraphs has the side-effect of making it impossible to tell how many paragraphs there are in a "§§" range without examining the actual document structure between the start and end point of the range. For example, I.1.4.1a and I.1.4.1b have been inserted between I.1.4.1 and I.1.4.2 at some stage in the document's history.

Resolving internal references is a two step process: first, identifying the reference substring within the text, and second, resolving the substring to a global index. The first step is achieved through regular expressions that match the start of a reference sequence or range, so that another regular expression can be used to create a *partial reference* for each explicit reference in a sequence or each implicit reference in a range. Single references are treated as a sequence of one. The regular expressions use the same syntax as the pre-processing substitution regular expressions, but generate new strings rather than replace existing ones. The example below matches the word "paragraph" followed by a number that, optionally may end with letters. The number is captured and substituted into the string on the right of the arrow at the code\1, e.g. matching against 'paragraph 1b' would output '[_,_,_,1b,_,_,_]'. For more complex reference patterns, a suitable regular expression can capture more than one number and insert multiple numbers into the right hand string, e.g. §26 para. 8 would output '[_,_,26,8,_,_,_]'. In this way, the right hand string becomes a structured representation of a partial reference.

```
# extract paragraph number to depth=4 in partial
  ↪ reference
'(?:paragraph\s+)(\d{1,3}[a-z]?)' -->
  ↪ '[_,_,_,\1,_,_,_]'
```

Partial references are string representations of fixed-length arrays, where each array item is either '_' or a local index string (unquoted). The underscore is treated as a wildcard during the next step in the resolution process.

Resolving a partial reference template to a global reference amounts to finding the best matching global index in the document, given the list of all global indexes represented in the same fixed length array format as the partial reference template, e.g. global index "I.1.4.1b" is represented as '[I,1,4,1b,_,_,_]'. Using the concept of wildcard matching against underscores provides a mechanism for identifying all the candidate global index matches in the document. Here, we introduce **resolve_partial_ref**, a novel algorithm for selecting the match from this potentially large list of candidates that closely accords with the semantic intent of the reference, i.e. that resolves to the global index of the target paragraph that was originally intended by the author of the legal text.

Table 2
Examples paragraphs with internal references from the Austrian *Study Funding Act*. The text of each reference is highlighted in bold but is not delineated in the original document.

Source index	Source text	Target index
I.1.2.1.11	Austrian citizens (§3) and	I.1.3
II.1.6.1.4.c	for disabled students in accordance with § 26 para. 8 by five years,	II.5.26.8
I.1.3.2	The Austrian educational institutions referred to in **paragraph 1** shall be treated in the same way:	I.1.3.1
II.4.19.2	Important reasons within the meaning of **paragraph 1** are:	II.4.19.1
II.5.26.1	The basic amount of the study grant is 335 euros (Note. 1) monthly.	II.5.26.8.1.1.1
II.5.27.3	The annual amount calculated in this way is reduced by 8% (Note 1) to increase, …	II.5.27.4.1.1.1
II.4.19.6.2	in the case of important reasons within the meaning of **no. 1 or paras. 2, 3 and 4**,	II.4.19.6.1 II.4.19.2 II.4.19.3 II.4.19.4
II.1.6.1.4.d	for students who take up a Master's program, …, taking into account **lit. a to c**.	II.1.6.1.4.a II.1.6.1.4.b II.1.6.1.4.c
IV.1.75.16	Instead of the maximum study grants laid down in §§26 to 28, the following maximum …	II.5.26 II.5.27 II.5.28
II.4.16.1.3	submits proof of successful completion of courses and examinations (§§20 to 24).	II.4.20 II.4.23 II.4.24

Informally, **resolve_partial_ref** accepts a partial reference template and the list of all global indexes in the document, along with two pieces of contextual information: the global index of the source paragraph (i.e. the location of the text containing the internal reference) and the previous, if any, partial reference already resolved in the same source paragraph. The function returns the global index of the resolved reference inferred using heuristics and locational context of the source paragraph within the document structure, as inferred by the method described in 5.2.1.2. The heuristics are as follows:

- First try to resolve based on any previously resolved references in a sequence (list or range) of references, e.g. when resolving the "3" in the list "paras. 2, 3 and 4", wildcard match the underscores in the template of the partial reference '[_,_,_,3,_,_,_]' against the resolved previous reference in the list, '[II,4,19,2,_,_,_]', to infer the resolution '[II,4,19,3,_,_,_]', representing the global index "II.4.19.3".
- Otherwise, treat as the first reference in a sequence or a standalone reference (i.e. a sequence of length one). Search each level of the global index hierarchy upwards, starting with the global index of the source paragraph, looking for the best matching resolution in a bottom-up, breadth-first exhaustive tree traversal. The primary measure used for determining the best candidate match is *common ancestry similarity*, which in the case of a tie, is supplemented by a biased version of the difference between the ordinal positions of the candidate and source paragraphs. The common ancestry similarity, $\mathbf{sim}_{CA}(x, y)$, between two global indexes x and y in the same document, is defined as their number of shared ancestors, such that

  $$\mathbf{sim}_{CA}(x, y) = |\mathbf{ancestry}(x) \cap \mathbf{ancestry}(y)|,$$

  where $\mathbf{ancestry}(p)$ is the set of global indexes of all the ancestors of paragraph p in the document hierarchy, excluding the document root, union the global index of p itself.[5] For example, global indexes "II.1.3.6" and "II.1.3" have common ancestry similarity of 3 because,

  $$\mathbf{ancestry}(\text{"II.1.3.6"}) = \{\text{"II.1.3.6"}, \text{"II.1.3"}, \text{"II.1"}, \text{"II"}\},$$
  $$\mathbf{ancestry}(\text{"II.1.3"}) = \{\text{"II.1.3"}, \text{"II.1"}, \text{"II"}\},$$

  and so,

  $$\begin{aligned}\mathbf{sim}_{CA}(&\text{"II.1.3.6"}, \text{"II.1.3"})\\ &= |\{\text{"II.1.3"}, \text{"II.1"}, \text{"II"}\}|\\ &= 3.\end{aligned}$$

  Similarly,

  $$\begin{aligned}\mathbf{sim}_{CA}(&\text{"II.1.3.6"}, \text{"II.1.3.6.1.1.1"})\\ &= |\{\text{"II.1.3.6"}, \text{"II.1.3"}, \text{"II.1"}, \text{"II"}\}|\\ &= 4,\\ \mathbf{sim}_{CA}(&\text{"II.1.3.6"}, \text{"II.1.3.6"}) =\\ &= |\{\text{"II.1.3.6"}, \text{"II.1.3"}, \text{"II.1"}, \text{"II"}\}|\\ &= 4, \text{ and}\\ \mathbf{sim}_{CA}(&\text{"II.1.3.6"}, \text{"I.1.3.6"})\\ &= |\varnothing|\\ &= 0.\end{aligned}$$

  For a given global index x in the set of all global indexes in document I_D, the set of best matching candidate global index $y \in I_D$ is defined as R_x, the resolution candidate set of x,

  $$R_x = \{y \mid \mathbf{sim}_{CA}(x, y) > \mathbf{sim}_{CA}(x, z), \forall y, z \in I_D\}.$$

  Where there are ties, i.e. where R_x is not singleton, a secondary measure, $\delta_d(x, y)$, is used to select the nearest global index from the set of resolution candidates R_x. Heuristically, ambiguous references tend to be backward references to earlier paragraphs rather than forward references. Therefore we define distance metric δ_d with a deliberate bias, so that all backward references are closer than all forward ones, such that

  $$\delta_d(x, y) = \begin{cases} \mathbf{ord}_d(x) - \mathbf{ord}_d(y), & \text{if } \mathbf{ord}_d(x) \geq \mathbf{ord}_d(y) \\ \mathbf{ord}_d(y) & \text{otherwise,} \end{cases}$$

  where $\mathbf{ord}_d(z_i) = i$, the ordinal position, $i \in \mathbb{R}$, of global index z_i in the list of global indexes $d = [z_1, z_2, \ldots, z_{|d|}]$ in the document.

  $$\mathbf{resolve}_d(x) = \begin{cases} \arg_0(R_x), & |R_x| = 1 \\ \operatorname{argmin}_{\forall y \in R_x} \delta_d(x, y), & \text{otherwise.} \end{cases}$$

  where Once the best matching candidate $y = \mathbf{resolve}_d(x)$ is found, the template of y is then wildcard matched against the underscores in the partial reference template of x to complete the inference.

[5] $\mathbf{sim}_{CA}$ is a similarity measure and not a metric, as it violates the reflexive axiom of a metric due to $\mathbf{sim}_{CA}(x, x) \neq 0$, despite satisfying the positivity, symmetry and triangular inequality axioms.

- If no common ancestors are found, i.e. $R_x = \emptyset$, then return a null resolution to indicate that no matches for the partial reference x exist in the document.

```
FUNCTION resolve_partial_ref(all_paragraphs_metadata, partial_ref,
↪  source_ref, previous_ref):
 // INPUTS:
 //  all_paragraphs_metadata: all paragraphs with absolute IDs and
↪  positions.
 //  partial_ref:          incomplete reference to be resolved
↪  (e.g., ["", "", "para", "1"]).
 //  source_ref:           absolute ID of the paragraph containing
↪  the reference.
 //  previous_ref:         resolved ID of the preceding reference in
↪  a list (e.g., for "paras 1, 2", this would be the ID for "1" when
↪  resolving "2").

 // --- HEURISTIC 1: RESOLVE USING SEQUENCE CONTEXT ---

 // First, try to resolve the reference using the context of the
↪  previously resolved item in the same list or range.
 IF previous_ref IS NOT NULL THEN
  // Case 1: The partial reference has the same structure as the
↪  previous one.
  IF length_of(previous_ref) == length_of(partial_ref) THEN
   // Infer by merging the known prefix of previous_ref with the
↪  specified parts of partial_ref.
   // e.g., if previous_ref is "§26.1" and partial_ref is "._.2",
↪  infer "§26.2".
   inferred_ref = merge(prefix_of(previous_ref),
↪  suffix_of(partial_ref))
   IF inferred_ref exists in all_paragraphs_metadata THEN
    RETURN inferred_ref
   END IF
  // Case 2: The partial reference is a single ambiguous element
↪  (e.g., "2").
  ELSE IF length_of(partial_ref) == 1 THEN
   // Try to attach the partial_ref at each level of the
↪  previous_ref's hierarchy, from most specific to least.
   FOR d from deepest_level_of(previous_ref) up to highest_level:
    inferred_ref = attach_at_level(previous_ref, partial_ref, d)
    IF inferred_ref exists in all_paragraphs_metadata THEN
     RETURN inferred_ref
    END IF
   END FOR
  END IF
 END IF

 // --- HEURISTIC 2: RESOLVE USING HIERARCHICAL CONTEXT AND
↪  PROXIMITY ---

 // If sequence context fails or is not applicable, search the entire
↪  document for the best candidate.
 best_candidate_ref = NULL
 highest_ancestry_score = -1
 closest_distance = infinity

 // Iterate through all paragraphs to find potential candidates.
 FOR EACH candidate_paragraph IN all_paragraphs_metadata:
  // A candidate is valid if its ID structure matches the known parts
↪  of the partial_ref.
  IF candidate_paragraph.ID matches_structure_of partial_ref THEN

   // Calculate how similar the candidate's location is to the
↪  source's location.
   ancestry_score =
↪  calculate_common_ancestors(candidate_paragraph.ID, source_ref)

   // Calculate the distance, heavily favoring references that
↪  point backwards in the document.
   distance =
↪  calculate_biased_distance(candidate_paragraph.position,
↪  source_ref.position)

   // A candidate is better if it shares more ancestors with the
↪  source reference.
   IF ancestry_score > highest_ancestry_score THEN
    highest_ancestry_score = ancestry_score
    closest_distance = distance
    best_candidate_ref = candidate_paragraph.ID
   // If ancestry is tied, the one that is closer (with a bias for
↪  backwards references) wins.
   ELSE IF ancestry_score == highest_ancestry_score AND distance <
↪  closest_distance THEN
    closest_distance = distance
    best_candidate_ref = candidate_paragraph.ID
   END IF
  END IF
 END FOR

 IF best_candidate_ref IS NOT NULL THEN
  RETURN best_candidate_ref
 END IF

 // --- FALLBACK ---

 // As a last resort, check if the partial_ref happens to be a valid,
↪  complete reference on its own.
 IF partial_ref exists as an absolute ID in all_paragraphs_metadata
↪  THEN
  RETURN partial_ref
 END IF

 // If no resolution is found, return an empty result.
 RETURN []
END FUNCTION
```

Listing 1: Pseudocode for our resolve_partial_ref function.

There is a final post-processing step for the special case of ranges, such as "§§20 to 24", where up to this point, only the global index of their start and end paragraphs will have been resolved. The resolution of the range can now be completed by including all the global indexes for paragraphs (at the same depth in the hierarchy tree) that lie between the start and end paragraphs. This expansion uses the paragraph ordinal positions within the document and is, therefore, unaffected by gaps in the global index numbering scheme, as discussed earlier.

The **resolve_partial_ref** algorithm successfully resolved all the 900 resolvable references in our test data, the Austrian *Study Funding Act*. We anticipate that it will work well on similarly structured legal-numbered texts and references, based on the observation that the external references to other legal documents cited in this act all share the same referencing notation.

Over and above the 900 resolvable internal references, the reference parsing and resolution also identified a further 83 references that have no obvious resolution within the document. Most of these are intended to be understood as external references by their semantic context; for these examples, regular expression matching on syntax alone is insufficient to distinguish an external reference from an internal one. In a few cases, the reference appears to be a "broken link" to a paragraph that no longer exists or as a result of a typographical error. Perhaps an LLM might reduce the number of misidentified internal references but even so, for comprehensive machine interpretation of references, examples such as the 83 identified in our work need special attention by the legal professionals creating such documents, either to remove ambiguity and to fix broken references.

5.2.1.4. External reference parsing. External references in legal texts follow the same format and conventions used in internal references, but with an extra dimension: they refer to locations in the text of other documents. In other words, external references are citations rather than local cross-references within the same document. Table 3 shows representative examples from a total of 108 external references in the Austrian *Study Funding Act*. As can be seen in the table, some external references cite multiple locations within the referenced (target) document and so this increases the total to approximately 130 — although without resolving each reference it is not possible to know the exact number because of "§§" references to ranges of sections, as discussed in 5.2.1.3. More challenging for rule extraction is that the referenced scope within the target document varies from specifying a precise paragraph or, less precisely, a section or, most imprecisely, the entire target document. Dealing with the implication of such dramatically varying external reference precision is an open research problem outside the scope of this paper.

Table 3
Examples paragraphs containing external references from the Austrian *Study Funding Act*. The text of each reference is highlighted in bold but is not delineated in the original document. Source Index is the global index of the Source Text paragraph within the source document. Target Scope is the precision of the reference to the external document.

Source index	Source text	Target scope
I.1.3.1.2	Students at a theological college located in Austria Art. V §1 para. 1 of the Concordat, Federal Law Gazette II No. 2/1934) after passing a matriculation examination,	paragraph
II.2.9.1.22	the amounts referred to in §10, §11, §18 (6), §24 (4) and §41 (3) EStG 1988, in so far as they were deducted from the determination of income;	section or paragraph
I.1.4.1a.1	are migrant workers within the meaning of (45) of the Treaty on the Functioning of the EU (TFEU) or self-employed workers within the meaning of (49) TFEU or members of their families; or	paragraph
I.1.3.2.11	educational institutions located in Austria that are accredited as private universities or private universities in accordance with the provisions of the Private Higher Education Act (PrivHG), Federal Law Gazette I No. 77/2020,	document

Any solution to the external reference problem will first require the resolution of external references to a location, a global index, within the target document. We suggest that this task is amenable to the same approach we adopted for internal references. In which case, the resolution of external references would first require structural inference parsing to be performed on each of the cited documents, so that the location of the cited text can be determined. The heterogeneity of legal document formats would require document- or format-specific parsing rules but the underlying structure of legal texts is hierarchically numbered paragraphs. In principle, it would be possible to parse every published legal document in the legislature, past and going forwards, to infer the structure and resolve both internal and external references. Doing so would create a complete web of links between legal texts, thereby enabling resolution of external references.

Currently, legislation is not automatically "link checked" to ensure that every external reference can be resolved and that there are no broken "links" between documents. Document authors are responsible for ensuring the referential integrity of their texts — even as the documents they cite are themselves evolving over time. The situation has much in common with the Web, where links (external references) to other pages (documents) are not guaranteed to exist nor to contain the same information as when the link was first created. Web pages change on their own cadence and can be deleted (repealed); the same is true for the implicit web of legal documents. Building and maintaining a database of links between entire bodies of legal texts would undoubtedly be valuable but it is outside the scope of our present work. Consequently, we restrict our treatment of external references to identifying their location within the text, using regular expressions, so that they can be excluded from resolution. Manual markup would have been a more accessible alternative for less technical users. A machine learning or LLM-based approach might prove more cost-effective in future work but was not investigated in this project.

5.2.2. Application of rule extraction patterns

Rule extraction is performed by the application of matcher patterns to text. When matched, each pattern extracts information from the text in order to emit a triple of the form, $(field, condition, value)$. These logical triples are emitted in one of two modes, depending on the presence or absence of a reserved keyword `THEN()` in the *field*.

- If `THEN()` is absent, then the logical triple is interpreted as a term of the form `condition(field, value)` and emitted as either the first or subsequent n-ary argument of a logical conjunction (AND) or disjunction (OR) of terms that collectively define the body of a single rule. Whether AND or OR is used is defined as the *combinator* property of the pattern emitting the triple. If a pattern defines a different logical combinator to the previously emitted term then a new n-ary logical operator is started. In this usage mode, every term emitted by any pattern matcher becomes part of the same overall rule: for conjunctions, the term is appended to the same body; for disjunctions, a new body is started for each term. The head of the rule is defined external to the pattern matching mechanism — in our proof-of-concept implementation the head is defined as the name of a process to execute if the body evaluates to `true`. The head is also accompanied with a top-level logical operator that is used to combine the bodies below it. All bodies emitted share the same head signature, effectively forming part of the same rule. Rules in this mode all share a similar structure, as illustrated below for a top-level disjunction followed by N conjunction operators, where `process/1` is the externally defined head, or consequence.

```
process :-
  condition11(field11, value11),
  condition12(field12, value12),
  ...

process :-
  condition21(field21, value21),
  condition22(field22, value22),
  ...

process :-
  conditionN1(fieldN1, valueN1),
  conditionN2(fieldN2, valueN2),
  ...
```

The `process/1` rule is `true` if any of the emitted bodies is `true`. Note that in practice, lazy evaluation of logical operators is used when evaluating rules and so there is an implicit cut at the end of each clause. Turning to another example, when the top-level logical operator is a conjunction, there is an extra implicit insertion of an implied head for any disjunctive operator terms emitted. For example, for a top-level conjunction with N subsequent disjunction operators emitted beneath it will have a rule structure of the following form.

```
process :-
  implied_head1,
  implied_head2,
  ...,
  implied_headN.

implied_head1 :-
  condition11(field11, value11),
  condition12(field12, value12),
  ...
```

```
implied_head2 :-
  condition21(field21, value21),
  condition22(field22, value22),
  ...
```

The `process/1` rule is `true` if all of the emitted bodies are `true`.

- If `THEN()` is present in the *field* item of the logical triple, then the triple is interpreted differently. The string between the parentheses is used as the head of the current body, thereby enabling multiple rules, with different heads, to be emitted. However, that is a useful side-effect of `THEN()`; in the context of the logical triple, `THEN()` always evaluates to `true` and so may be incorporated into the body in two different ways:
  - For logical triples emitted into a disjunction, each triple may be structured as `(THEN(`*head*`),`*condition*, *value*`)`, where *condition* is usually `is` and *value* can be either an atom or a functor. The emitted rule for each pattern in a disjunction will have the form,
    ```
    head :- value.
    ```
  - For logical triples emitted into a conjunction, the nth triple emitted into an n-ary AND will have the form, `(THEN(`*head*`),`*condition*`,`*value*`)`, where *condition* is usually `is` and *value* will always be `TRUE`. The emitted rule for the current body of a conjunction will have the form,
    ```
    head1 :-
      condition11(field11, value11),
      condition12(field12, value12),
      ...,
      condition12(field1N-1, value1N-1).
    ```
    The triple with the `THEN()` does not emit anything into the body; it emits into the head, which is why there are $N-1$ conditions rather than N in the body. In this way, an arbitrary number of rules can be emitted from a single list of pattern matchers. This mechanism relies on the side-effect of `THEN()` and on lazy evaluation but facilitates the generation of core diverse rules than using a single externally defined head to define the consequences. Using `THEN()` enables multiple consequences from a single set of patterns.

We now turn to the specifics of defining patterns, such as the example in Listing 2, which is a JSON dictionary that follows the syntax and semantics set out in Table 4. A JSON list of such dictionaries is the core of the pattern matching and rule extraction capability. Here we list just one pattern for brevity:

Each value in match defines a spaCy[6] matcher that is matched against each paragraph of text within the defined scope of the pattern. Each matcher is initialized with the vocabulary from a pre-loaded spaCy language model (either *en_core_web_sm* for English or *de_core_news_lg* for German), which are used to identify words and phrases based on their linguistic attributes (e.g. Part of Speech tagging). The configuration is set with `validate=True` to check for pattern correctness and explicitly disables fuzzy matching (`fuzzy_compare=None`). Patterns are added to the matcher using the `greedy="LONGEST"` parameter, which instructs the matcher to return the longest possible token sequence if multiple matches overlap. When the matcher is called on a text, it is configured to return the start and end token indices of the match, not Span objects (`as_spans=False`). All matchers in match must match in their order of definition for the overall pattern to match; the sequence of matcher is treated as if it were one large matcher. The reason for specifying the overall matcher as multiple smaller

[6] https://spacy.io/usage/rule-based-matching#matcher

```
{
 "id": "deductions1",
 "scope": "II.5.30.1",
 "level": 4,
 "match": {
  "relation": [{"POS": "NOUN"},{"TEXT": "from"},
  ↪  {"POS": "DET"},{"TEXT": "age"}],
  "number": [{"POS": "NUM"}]
 },
 "emit": {
  "field": "age",
  "condition": ">=",
  "value": "{number}"
 },
 "combinator": "AND",
 "ruleset": "{paragraph_id}"
}
```

Listing 2: Example pattern for matching against text.

matchers is so that the matching text of each matcher can be made available inside the strings defined in "emit", as string interpolation variables. The keys in "match" are used for the names of variables, e.g. "larg_relation_rarg" would be interpolated to the string,

"student_at(austrian_universities)" if the matchers for "l_arg", "relation", "r_arg" had matched against the text "student", "at" and "Austrian universities" respectively. Matched non-numeric text is automatically converted to valid atom/functor names in Prolog, e.g. converting spaces to underscores and text to lower case. By contrast, matched numbers are already valid Prolog atoms and are not altered. An exception to string conversion is where the text is contained in single quotes in the interpolation string, e.g. "'currency'" will be converted to 'EUR' if the matched text was "EUR" and will not be converted to eur as it would be without the quotes.

This string interpolation of matched text enables the construction of logical triples, as specified in "emit" dictionary keys:value pairs listed in Table 5.

If the interpolated value of "field" results in a string of the form, `THEN(x)`, where `x` is a Prolog atom or functor, then the resultant logical triple will be treated as the head of the emitted rule.

5.3. Rules rendering for the rules editor

One of the challenges of legal text analysis is to render first-order predicate rules that have been extracted from legal text in the graphical form of a decision tree, in Prolog notation, and in RuleML. These formats can help to represent the logical structure and the conditions of the rules, as well as to facilitate their verification and validation. However, a graphical visualization is also important to be able to compare the extracted rules to the original source text, and to identify any errors or inconsistencies caused by the automated extraction. For that, we provide an explainability component, which highlights which part of the rule was created from which part of the text. For that, the tree is displayed on the left side of the screen and the text on the right side. Lawyers can then compare the extraction results and provide feedback or corrections. Prolog and RuleML are used as data exchange formats, which can enable interoperability and integration with other systems or applications. RuleML is a specific XML-based vocabulary for expressing rules, while XML is a general purpose markup language for structuring data (Boley, Tabet, & Wagner, 2001).

Prolog rendering is achieved by transforming and simplifying the logical triples emitted during pattern matching and represents a procedural interpretation of the FOPL rules extracted (see Fig. 2). The corresponding Prolog rendering for each of these texts is shown in Listings 3, 4 respectively.

Table 4
Pattern dictionary keywords and values. "*" denotes optional.

Keyword	Value
"id"	Unique identifier string for this pattern, e.g. "citizenship".
"scope" *	Global index of a paragraph to start pattern matching at, e.g. "I.1.3". Only this paragraph and its descendent paragraphs will be considered for pattern matching. May be used in conjunction with level to constrain the extent of matching.
"level" *	Depth number in the document hierarchy to match at, e.g. 4. Only paragraphs at this depth in the hierarchy will be considered for pattern matching. Depths start at 1. May be used in conjunction with scope to constrain the extent of matching.
"match"	A dictionary with at least one key:value pair. Values are JSON format spaCy Token Matcher dictionaries.
"emit"	A dictionary with three mandatory keys, "field", "condition", "value", and their associated values as defined in Table 5. These collectively define the logical triple to output if this pattern matches the text of a paragraph. Key order is unimportant.
"combinator" *	"OR" (default) or "AND" specifying the logical operator for combining logical triples emitted by this pattern.
"ruleset" *	A string identifier that defines the destination ruleset to which logical triples are to be added. By default this is the pattern "id", so that each pattern will generate its own ruleset. However, the ruleset id can be any string and make use of embedded variables, paragraph_id or pattern_id, which will be interpolated to their respective values, e.g. "paragraph_id" expands to the global index of the paragraph being matched against, e.g. "I.1.4.1b". This is a powerful mechanism for grouping logical triples emitted by different patterns into the same rulesets; it makes pattern definition simpler because each pattern needs to only match against one relation in the text; it also enables pattern re-use across multiple rulesets.

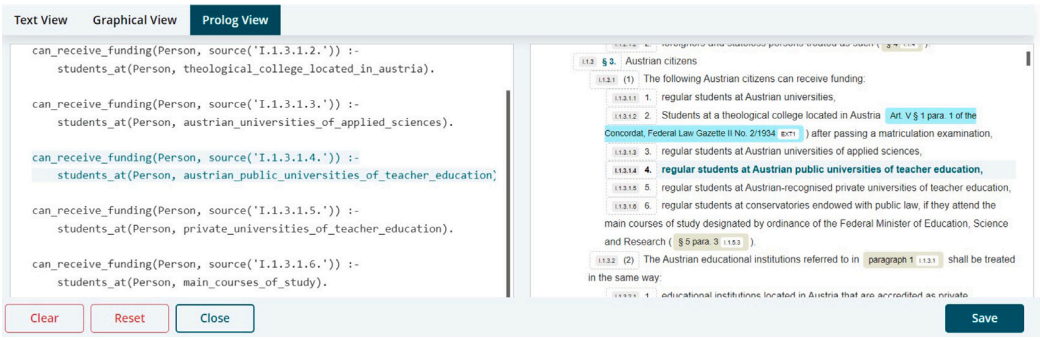

Fig. 2. Prolog view on the left with corresponding source text highlighted on the right.

```
can_receive_funding(Person, source('I.1.3.1.1.')) :-
  students_at(Person, austrian_universities).

can_receive_funding(Person, source('I.1.3.1.2.')) :-
  students_at(Person, theological_college_located_in_austria).

can_receive_funding(Person, source('I.1.3.1.3.')) :-
  students_at(Person, austrian_universities_of_applied_sciences).

can_receive_funding(Person, source('I.1.3.1.4.')) :-
  students_at(Person, austrian_public_universities_of_teacher_education).

can_receive_funding(Person, source('I.1.3.1.5.')) :-
  students_at(Person, private_universities_of_teacher_education).

can_receive_funding(Person, source('I.1.3.1.6.')) :-
  students_at(Person, conservatories_endowed_with_public_law).
```

Listing 3: Prolog rules extracted with disjunctive conditions.

```
each_contributor(Person, source('II.5.30.1.1.'), 6000, eur) :-
  age(Person, Age), =<(Age, 3).

each_contributor(Person, source('II.5.30.1.2.'), 6400, eur) :-
  age(Person, Age), >=(Age, 4), =<(Age, 14).

each_contributor(Person, source('II.5.30.1.3.'), 14200, eur) :-
  age(Person, Age), >=(Age, 5), =<(Age, 18).

each_contributor(Person, source('II.5.30.1.4.'), 18123, eur) :-
  age(Person, Age), >=(Age, 4).
```

Listing 4: Prolog predicates extracted as a disjunction of conjunctive conditions.

I.1.3 **§ 3.** Austrian citizens

I.1.3.1 (1) The following Austrian citizens can receive funding:

I.1.3.1.1 1. regular students at Austrian universities,

I.1.3.1.2 2. Students at a theological college located in Austria Art. V § 1 para. 1 of the Concordat, Federal Law Gazette II No. 2/1934 EXT1) after passing a matriculation examination,

I.1.3.1.3 3. regular students at Austrian universities of applied sciences,

I.1.3.1.4 4. regular students at Austrian public universities of teacher education,

I.1.3.1.5 5. regular students at Austrian-recognised private universities of teacher education,

I.1.3.1.6 6. regular students at conservatories endowed with public law, if they attend the main courses of study designated by ordinance of the Federal Minister of Education, Science and Research (§ 5 para. 3 I.1.5.3).

Fig. 3. Hyperlinked text view of inferred structure.

II.5.30 **§ 30.** Base

II.5.30.1 (1) The basis of assessment of the student, his parents and his spouse or registered partner comprises the income in accordance with §§ 8 II.2.8 , 9 II.2.9 , 10 II.2.10 less the allowances in accordance with para. 4 II.5.30.4 and the following tax deductions for the persons for whom either the student, one of his parents or his spouse or registered partner is obliged by law to provide maintenance:

II.5.30.1.1 1. EUR 6000 for each person up to the age of 3;

II.5.30.1.2 2. EUR 6400 for each person from the age of 4 to the age of 14;

II.5.30.1.3 3. EUR 14200 for each person from the age of 5 to the age of 18;

II.5.30.1.4 4. EUR 18123 for each person after the age of 4 who is considered a relative in accordance with Paragraph 6 (720) of the ASVG EXT36 or who is insured under the health insurance scheme or who receives the study grant; in the case of students who are abroad within the meaning of § 26 (3) II.5.26.3 , EUR 9610;

Fig. 4. Hyperlinked text view of inferred structure.

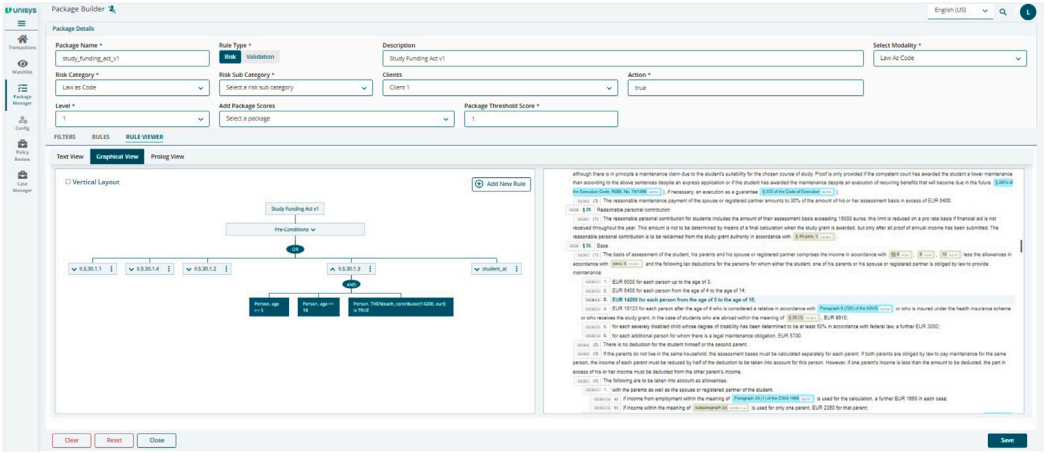

Fig. 5. Visualization of Rules.

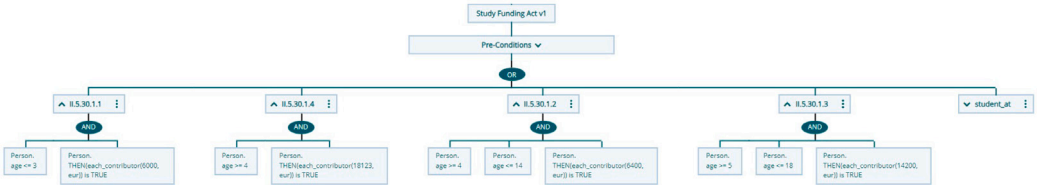

Fig. 6. Rules Tree.

Table 5
`emit`Dictionary keywords and values.

Keyword	Value
"field"	A string, optionally including interpolation variables.
"condition"	A string denoting an operator: "is", "<", "<=", ">", ">=". These have their traditional meanings apart from "is" which we use in place of "==".
"value"	A string, optionally including interpolation variables; or, a number.

Important for the transparency and explainability of the extracted rules, we also render the parsed document as HTML, with internal cross-references hyperlinked, and embedded associations between the extracted rules and their source paragraphs in the legal text (indicated by the green background color of the rules and text in Fig. 2). Example HTML renderings are shown in Figs. 3 and 4.

The visualizations of the rules have been developed and validated with domain experts and lawyers to make sure that they allow an easy comparison of rules and legal text. The UI screen is pictured in Fig. 5. A larger image of the tree with the extracted rules from Fig. 5 is shown in Figs. 6 and 7.

6. Validation

The LCDSS prototype (see Fig. 1 and Figs. 2, 3, 4,5, 6, and 7) was rigorously evaluated following best practices of action design research (ADR) (Sein et al., 2011), ensuring a comprehensive and iterative assessment of its functionality and effectiveness. This evaluation process involved collaboration with three experts from the Department of Legal Informatics at the University of Vienna, four e-Government experts from the Austrian Ministry of Finance and two IT experts from the Austrian National Computing Center, see Table 6.

6.1. Expert opinion from the department of legal informatics

The expertise in legal informatics provided critical insights into the prototype's compliance with legal standards, its potential impact on legal processes, as well as on the usability of the visualization of the extracted rules.

The legal analysis part has been elaborated in an exhaustive, 53-pages internal expert opinion (Schmautzer, Pfister, Braun, & Schweighofer, 2024). As main outcome of this validation, it can be concluded that the LCDSS system, together with its suggested work routines, can be applied according to the EU GDPR (General Data Protection Regulation) (European Commission, 2016a) and the EU AI Act (European Commission, 2016b).

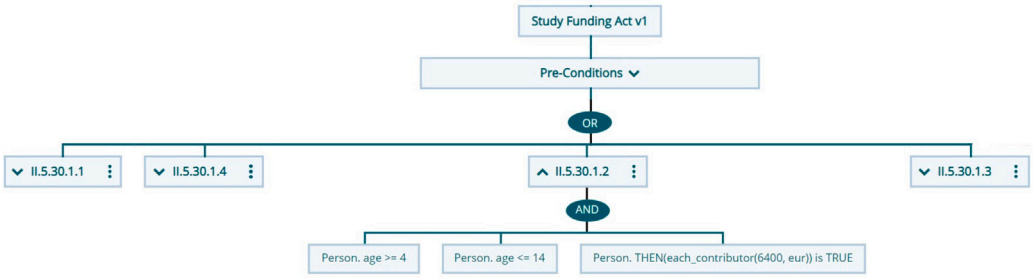

Fig. 7. Expanded Rules Tree.

Table 6
Interviewees — Organizations and Roles, BRZ = Bundesrechenzentrum (Austrian National Computing Center).

Stakeholder	Organization	Role
#1	University of Vienna	Department Head
#2	University of Vienna	Vice Managing Scientist
#3	University of Vienna	Researcher
#4	Ministry of Finance	Department Head
#5	Ministry of Finance	e-Government Expert
#6	Ministry of Finance	e-Government Expert
#7	Ministry of Finance	IT Consultant
#8	BRZ	Deputy Head of AI
#9	BRZ	Infrastructure Consultant

Furthermore, the experts confirmed that the graphical representation of the rules in the LCDSS system is useful (Davis, 1989) as well as easy to use (Davis, 1989) for the working legal expert.

6.2. Validation by the ministry of finance

The practical experience and knowledge in e-Government applications from the Ministry of Finance in Austria was instrumental in assessing the prototype's applicability and integration within existing governmental frameworks. The combined expertise of these professionals ensured a thorough validation of our LCDSS prototype, highlighting its strengths, and identifying areas for further improvement.

According to the experts, the strength of the LCDSS system is in increasing efficiency and effectiveness of e-Government work routines through automatization; effectively, to cope with staff short shortages and retirement waves.

According to areas of potential improvement, the expert pointed out that the customization effort for extracting rules from new legal documents can be quite challenging. Additionally, a binding process must be developed, in order to allow the LCDSS system to map variables to datapoints in e-Government registers. All this leads to the further direction to utilize GenAI in the process as will be discussed in Section 7.

6.3. Validation by the national computing center

The knowledge about national IT operations from experts from the Austrian National Computing Center continuously helped to evaluate the architecture of our prototype, especially with regards to ensuring compliance with on-premise hosting options and federal policies on AI use for public sector clients.

7. Discussion

In this paper, we presented a method for extracting rules in first-order predicate logic (FOPL) from legal text, which is the first step of bridging the gap between natural language and formal legal reasoning. Additionally, we show how the extraction can be integrated into our LCSS system to allow hyperautomation of public sector processes. Our method is based on the human-in-the-loop principle, which means that the rules are validated and refined by human experts before being used automatically by a business rules engine.

The pattern-matching approach has practical limitations, particularly in terms of scalability, maintainability, and adaptability. The high customization effort for each legal text is a significant bottleneck. This challenge is magnified when considering scaling the system across different jurisdictions and languages. Cross-jurisdictional scaling would require a complete redesign of the document structure inference and rule extraction patterns to accommodate variations in legal formatting, terminology, and structure (e.g., statutory vs. case-law systems). Similarly, multilingual adaptation beyond the demonstrated German-English context would necessitate creating entirely new, language-specific grammatical and regular expression patterns, a task requiring deep linguistic expertise for each target language.

In terms of maintainability and adaptability, our current pattern-based methods offer high transparency, as the extraction logic is explicit and debuggable. However, maintaining a large and complex set of patterns for evolving legislation is a considerable engineering challenge. Emerging transformer-based legal text parsing approaches, while potentially more opaque, offer superior adaptability. Pre-trained on vast legal and multilingual corpora, they can often generalize across minor variations in text and structure without explicit reprogramming, reducing the manual effort of adaptation. A naive application of LLMs alone currently performs too poorly to produce useful rules with adequate coverage (Blair-Stanek, Holzenberger, & Van Durme, 2023). However, a promising way in which LLMs may be safely used without loss of explainability is in the replacement of pattern matching, transforming pre-processed paragraphs and their contextual ancestors into logical triples. This technique retains the provenance link between the original text and the extracted rules and is an area we are actively investigating.

Furthermore, our method does not bind the variables in the rules to specific data sources, such as e-Government registers or databases.

This means that the rules cannot be directly executed by a rules engine without additional input from the user. To solve this problem, we plan to develop a rules binding module that can automatically link the variables in the rules to relevant data sources based on their semantic meaning and context.

The system's architecture is an intentional implementation of Explainable AI (XAI), a critical requirement for high-stakes legal domains. Unlike opaque "black box" models, our approach is founded on symbolic, rule-based logic (i.e., FOPL), which is inherently transparent. This explainability is operationalized through several key features. The visualization of extracted rules as both textual lists and graphical decision trees translates the formal logic into an interpretable format for legal experts. The mandatory human-in-the-loop validation step ensures that every rule is understandable and verifiable by a domain expert before it becomes active. Most importantly, the system provides strong provenance by maintaining a direct, hyperlink-enabled connection between each computational rule and the specific sentence in the source legal text from which it was derived. This traceability ensures that every decision made by the rules engine can be audited and justified by referencing the authoritative legal source, fulfilling a core tenet of XAI.

Having a human compare and refine the extracted rules is critical because the Law as Code approach touches upon significant ethical issues, particularly in a public administration context. On the one hand, our method can facilitate the automation of legal reasoning, potentially improving the efficiency, consistency, and transparency of public services. On the other hand, it poses risks such as the loss of human oversight, the codification of bias, and the need for robust accountability. Bias can be introduced if ambiguities in the source text are interpreted in a specific way by the expert creating the extraction patterns, leading to that bias being systematically applied at scale. Accountability is addressed through the human-in-the-loop design and the system's audit trail, which logs all automated extractions and manual modifications. However, this raises questions about who is ultimately responsible for a flawed rule approved by an expert. Therefore, our Law as Code approach should be used with caution as a decision support tool, never as a fully autonomous solution, ensuring it is always complemented by human expertise and ethical oversight. This maintains transparency and public trust, ensuring that the codified version of the law remains a faithful and just representation of the original legislative intent. This way, we can ensure that the rules are accurate, consistent and aligned with the legal text. Moreover, by using rules in FOPL, we can make the AI system transparent and explainable, as the rules can be easily inspected and understood by humans. Our method is computationally cost efficient, compared to GenAI LLM approaches, as it does not require extensive fine-tuning/training or GPU compute resources for the inference. This also makes our method more environmentally friendly, as it contributes to green IT.

Our work opens up new avenues for future research and applications in the field of natural language processing and artificial intelligence for law. Some possible directions include extending our method to handle more complex and diverse legal texts, such as contracts, judgments, or regulations; developing methods for verifying, validating, comparing and debugging the extracted rules; and integrating our method with other components of a legal reasoning system, such as knowledge bases, inference engines, or dialog systems.

Derived from the limitations of the current approach, there are also additional directions for further research that we would like to pursue. One direction is to analyze the legal implications of automatically extracting legal text in a formal and executable language and applying it for automation. This analysis is currently being done by our project partners at the Faculty of Law at University of Vienna. Another direction is to test our prototype with multiple users, such as lawyers, judges or citizens, and evaluate its usability and effectiveness in different scenarios and domains. We are also working on generalizing our prototype so that it can be more efficiently applied to more laws and regulations from different jurisdictions and languages. This will allow us to extend our prototype to other types of legal documents, such as contracts or policy documents, and see if our method can handle them as well as statutory laws. The last open research direction is to implement the rules binding module that we mentioned above. By addressing those points for further research, we plan to increase the technology readiness level four of LACDSS (Mankins et al., 1995).

8. Conclusion

We have presented a method for extracting rules in first-order predicate logic from legal text, which holds the potential to bridge the gap between natural language and formal legal reasoning. Our work contributes to the emerging field of Law/Rules/Policy as Code, which aims to transform natural language into executable code that can be automatically processed by computers.

While established initiatives like OpenFisca provide powerful engines for executing manually coded legislation, and standards like LegalRuleML offer interoperable formats for representing legal rules, the primary novel contribution of this research lies in the semi-automated extraction of these rules from unstructured natural language text. Our Law as Code Decision Support System (LCDSS) prototype demonstrates an end-to-end pipeline that begins with raw legal documents and produces a validated, executable knowledge base. This approach fundamentally shifts the role of the human expert from a manual coder of rules to a validator of machine-generated candidates, facilitated by an integrated rules editor and visualization tools.

Furthermore, our methodology places a strong emphasis on explainability through provenance. We have shown how the system maintains a transparent and auditable link between each extracted logical predicate and its precise origin in the source text, a critical feature for ensuring regulatory compliance and building trust with legal professionals. Our work thus complements existing Law-as-Code execution frameworks by addressing the critical upstream challenge of rule acquisition and validation directly from the source.

By formalizing the extraction process and integrating it into a validated decision support workflow, we offer a practical pathway to enhancing the accessibility, clarity, and consistency of legal services in both the public and private sectors. We have discussed the limitations of our pattern-based approach and proposed how GenAI and LLMs could enhance the prototype's adaptability. We hope that our work will inspire further research and development in creating robust, explainable, and scalable systems that translate law into computationally executable logic.

CRediT authorship contribution statement

Markus Bertl: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Simon Price:** Writing – original draft, Visualization, Validation, Supervision, Software, Data curation, Conceptualization. **Dirk Draheim:** Writing – review & editing, Supervision, Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported by the Digi-Fund through the Austrian Ministry of Finance, Austria, under the guidance of Björn Lellmann, as well as co-funded by the European Union and Estonian Research Council via the project TEM-TA141.

Appendix

The following regular expression templates were used for processing the textual data in Sections 5.2 and 5.3:

```
body_regex = (.*)

preprocessing_substitutions = # strip whitespace at start of lines
        '\n\s+' --> '\n'

        # strip out markup anchor text, that is non-displayable, ahead of each section start
        '\n§\s\d+[a-z]*\nText\n' --> '\n'

        # join number-only lines onto next line
        '\n(\d+[a-z]*\.)\n' --> '\n\1 '

        # join "x)"-only lines onto next line
        '\n([0-9a-z]+\))\n' --> '\n\1 '

        # differentiate main part sections from other sections
        '\nSection (\d+)\n' --> '\n_SEC_\1\n'

        # normalise top-level section numbering so that "Section x (y)" or "Article x" becomes "§x (y)"
        # Note: this repairs an artefact of translation from German, whereby § gets translated differently sometimes.

        # normalise titles
        '\n(?:[Ss]ection|[Aa]rticle) (\d+)\s' --> '\n§\1 '
        # normalise remaining "Section x" to "(x)" references
        '(?:[Ss]ection|[Aa]rticle) (\d+)[ .]' --> '(\1) '

        # merge section ids with section names and optional un-numbered line into 1/2 lines (from 2/3 lines)
        '\n(.+?)\n§\s+(\d+[a-z]*\.)\s+([A-Z].*?)\n' --> '\n§\2 \1\n(1) \3\n'
        '\n([^\(].+?)\n§\s+(\d+[a-z]*\.)' --> '\n§\2 \1\n'

        # strip out spaces between section paragraph symbol and number
        '\n§\s+(\d)' --> '\n§\1'

        # merge _SEC_ lines with optional title on next line
        '\n(_SEC_\d+[a-z]*)\n([^§\n]+)\n' --> '\n\1 - \2\n'

        # expand _SEC_ back to "SECTION"
        '_SEC_' --> 'SECTION '

        # normalize main headings
        '\n(I|II|III|IV|V|VI|VII|VIII|IX|X)\.\s+([A-Z ]+)\n([A-Z ]+)?\n' --> '\n\1. \2 - \3\n'

        # strip whitespace at start of lines
        '\n\s+' --> '\n'

        # insert cosmetic space after "(x)" at start of lines
        '(\n\(\d+[a-z]*\))' --> '\1 '

        # fixup euro amounts that use space as thousands separators in the number
        'EUR (\d+) (\d\d\d)' --> 'EUR \1,\2'
        '(\d+) (\d\d\d) [Ee]uro[s]?' --> '\1,\2 Euros'
        # strip commas from numbers to simplify monetary patterns
        '(\d+),(\d+)' --> '\1\2'

        # fixup spacing before superscript footnote references
        '(\S)\(' --> '\1 ('

paragraph_split_regex = \n

paragraph_levels_regex = #level 0
        ^(I|II|III|IV|V|VI|VII|VIII|IX|X)\.\s(.*)$

        #level 1
        ^SECTION\s(\d+[a-z]*)(?:\s-\s(.*))?$

        #level 2
        ^§(\d+[a-z]*)\.\s*(.*)$

        #level 3
        ^\(((\d+[a-z]*)\)\s+(.*)$

        #level 4
        ^(\d+[a-z]*)\.\s+(.*)$

        #level 5
        ^([0-9a-z]+)\)\s+(.*)$

        #level 6 - numbered notes
        ^Note\s(\d+):(.*)$

levels_output_format = %s.
        SECTION %s
```

```
        § %s.
        (%s)
        %s.
        %s)
        Note %s:
        %s

[External References]
external_references_regex = \(([^\)]+ Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= with the provisions of the )([^_]*? Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= listed in )([^_]*? Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= to the contrary, )([^_]*? Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= in accordance with )([^_]*? Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= within the meaning of the )([^_]*? Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= within the meaning of )([^_]*? Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= under the )([^_]*? Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= within the period of )([^_]*? Federal Law Gazette [IVX]* ?[Nn]o. \d+[/\d]*)
        (?<= measure pursuant to )([^_]*? Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= pursuant to )([^_]*? Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= on the basis of the )([^_]*? Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= as amended by the )([^_]*? ?Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= as amended by )([^_]*? ?Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= ordinances of the )([^_]*? ?Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= of the )(Federal Minister [^_]*? ?Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= according to )([^_]*? ?Federal Law Gazette [IVX]* ?[Nn]o\. \d+[/\d]*)
        (?<= within the meaning of )(\(\d+\) of the Treaty on the Functioning of the EU \(TFEU\))
        (?<= within the meaning of )(\(\d+\) TFEU)
        (?<= within the meaning of )(\(\d+\) of Directive .*? of the Union)
        (?<= referred to in )([^_]*? EStG \d+)
        (?<= pursuant to )([^_]*? EStG \d+)
        (?<=\()(§[^_]+? EStG \d+)
        (?<= within the meaning of )([^_]*? EStG \d+)
        (?<= under )([^_]*? Income Tax Act \d+)
        (?<= pursuant to )([^_]*? Income Tax Act \d+)
        (?<=ursuant to )([^_]*? B-VG)
        (?<=ursuant to )([^_]*? VwGVG)
        \(([^\)]+ Execution Code, RGBl\. No\. \d+/\d+)
        \(([^\)]+ Code of Execution)
        (?<= in accordance with )([^_]*? ASVG)
        (?<= factor of )([^_]*? ASVG)
        (?<= in accordance with )([^_]*? Regulation \(EU\) No\.? \d+/\d+)
        (Directive \d+/\d+/EC, O\. J\. OJ No\.? L \d+, [\d.]+)
        (OJ No\.? L \d+ of [\d.]+ p\.? 19)
        (?<= pursuant to )([^_]*? Volunteer Act, BGBl [IVX]+ Nr\. \d+/\d+)
        (?<= pursuant to )([^_]*? Universities Act \d+)
        (?<= in accordance with )([^_]*? \d+ GDPR)
        (?<= requirements of )([^_]*? \d+ GDPR)
        (?<= referred to in )([^_]*? \(\d{4}\))
        (Student Support Act \d+)
        (StudFG \d+)
        (?<= \()([^_]*?UG)
        (?<= according to )([^_]*?UG)

[Internal References]
internal_references_sequence_regex = (§§?\s+.+)
        ([Ss]ections[s]? .+)
        (?<=[^a-z])(paragraph[s]? .+)
        (?<=[^a-z])(para[s]?\.?\s.+)
        (?<=[^a-z])(subsection[s]? .+)
        #(?<=[^a-z])(nos\. .+)
        (?<=[^a-z])(no[s]?\.[s]? .+)
        (?<=[^a-z])(lit\. .+)
        (?<=[^a-z])([Nn]ote[s]?\.? .+)
        (?<=[^a-z])(subparagraph[s]? .+)

internal_reference_regex = '\((\d{1,3}[a-z]?)\)' --> '[_,_,_,\1,_,_,_]'
        '(?:[Ss]ection[s]?\s+(\d{1,3}[a-z]?))' --> '[_,_,\1,_,_,_,_]'
        '(?:paragraph\s+)(\d{1,3}[a-z]?)' --> '[_,_,_,\1,_,_,_]'
        '(?:para[s]?\.?\s+(\d{1,3})?)(?:\s+no[s]?\.?\s+(\d{1,3})?)' --> '[_,_,_,\1,\2,_,_]'
        '(?:para[s]?\.?\s+(\d{1,3})?)(?:\s+sub-para[s]?\.?\s+(\d{1,3})?)' --> '[_,_,_,\1,\2,_,_]'
        '(?:para[s]?\.?\s+(\d{1,3})?)' --> '[_,_,_,\1,_,_,_]'
        '(?:subsection\s+)\((\d{1,3}[a-z]?)\)' --> '[_,_,_,\1,_,_,_]'
        '(?:no\.\s+)(\d{1,3}[a-z]?)' --> '[_,_,_,_,\1,_,_]'
        '(?:sub-?para(?:graph)?\.?\s+\(([a-z]+)?\))' --> '[_,_,_,_,_,\1,_]'
        '(?:lit\.?\s+\(?([a-z]+)?\)?)' --> '[_,_,_,_,_,\1,_]'
        '(?:([a-z]))(?![a-z])' --> '[_,_,_,_,_,\1,_]'
        '(?:[Nn]ote[s]?\.?\s*(\d{1,3}))' --> '[_,_,_,_,_,_,\1]'
        '§§?\s+(\d{1,3}[a-z]?)(?:\s+sub-para\.?\s+(\d{1,3})?)' --> '[_,_,\1,1,\2,_,_]'
        '§§?\s+(\d{1,3}[a-z]?)(?:\s+para\.?\s+(\d{1,3})?)(?:\s+sub-para\.?\s+(\d{1,3})?)' --> '[_,_,\1,\2,\3,_,_]'
        '§§?\s+(\d{1,3}[a-z]?)(?:\s+para\.?\s+(\d{1,3})?)(?:\s+no[s]?\.?\s+(\d{1,3})?)' --> '[_,_,\1,\2,\3,_,_]'
        '§§?\s+(\d{1,3}[a-z]?)(?:\s+para\.?\s+(\d{1,3})?)' --> '[_,_,\1,\2,_,_,_]'
        '§§?\s+(\d{1,3}[a-z]?)\s+\((\d{1,3}[a-z]?)\)' --> '[_,_,\1,\2,_,_,_]'
        '§§?\s+(\d{1,3}[a-z]?)\.(\d{1,3}[a-z]?)' --> '[_,_,\1,\2,_,_,_]'
        '§§?\s+(\d{1,3}[a-z]?) no[s]?\. (\d{1,3}[a-z]?)' --> '[_,_,\1,1,\2,_,_]'
        '§§?\s+(\d{1,3}[a-z]?)' --> '[_,_,\1,_,_,_,_]'
        '(\d{1,3}[a-z]?)\s+\((\d{1,3}[a-z]?)\)' --> '[_,_,\1,\2,_,_,_]'
```

```
        # following pattern is ambiguous and resolution depends on context, so can't say now which level the match refers to
        '(\d{1,3}[a-z]?)' --> '[\1]'

internal_reference_separator_regex = '\s*,\s*' --> 'DELIMITER'
        '\s+(and|or)\s+' --> 'DELIMITER'
        '\s+(to)\s+' --> 'RANGE'
```

References

Biagioli, C., Francesconi, E., Passerini, A., Montemagni, S., & Soria, C. (2005). Automatic semantics extraction in law documents. In *Proceedings of iCAIL'05 – the 10th international conference on artificial intelligence and law* (pp. 133–140). Association for Computing Machinery, http://dx.doi.org/10.1145/1165485.1165506.

Blair-Stanek, A., Holzenberger, N., & Van Durme, B. (2023). BLT: Can large language models handle basic legal text? arXiv preprint arXiv:2311.09693.

Boley, H., Tabet, S., & Wagner, G. (2001). Design rationale for RuleML: A markup language for semantic web rules. In *Proceedings of sWWS'01, the first semantic web working symposium*: *Vol. 1*, (pp. 381–401).

Borrelli, M. A. (1989). PROLOG and the law: Using expert systems to perform legal analysis in the UK. *Software Law Journal*, *3*(4), 687–715.

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, *13*(3), 319–340.

Dragoni, M., Villata, S., Rizzi, W., & Governatori, G. (2016). Combining NLP approaches for rule extraction from legal documents. In *Lecture notes in artificial intelligence*: *Vol. 10791*, *AI approaches to the complexity of legal systems* (pp. 287–300). Springer.

European Comission] (2018). *DLV06.01 – Final report: Study on functional, technical and semantic interoperability requirements for the single digital gateway (SDG) implementation*. European Comission, https://ec.europa.eu/isa2/sites/default/files/dlv06.01-_final_report.pdf.

European Commission (2016a). *Regulation 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. European Commission.

European Commission (2016b). *Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence and amending Regulations (EC) No 300/2008, (EU) No 167/2013, (EU) No 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2014/90/EU, (EU) 2016/797 and (EU) 2020/1828 (Artificial Intelligence Act)*. European Commission.

Flach, P., & Sokol, K. (2022). Simply logical–Intelligent reasoning by example (fully interactive online edition). arXiv preprint arXiv:2208.06823.

Gordon, T. F. (1987). Some problems with prolog as a knowledge representation language for legal expert systems. *International Review of Law, Computers & Technology*, *3*(1), 52–67. http://dx.doi.org/10.1080/13600869.1987.9966253.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 75–105.

Korger, A., & Baumeister, J. (2021). Rule-based semantic relation extraction in regulatory documents. In *Proceedings of the LWDA 2021 workshops*: *Vol. 2993*, (pp. 26–37). CEUR Workshop Proceedings, https://ceur-ws.org/Vol-2993/paper-03.pdf.

Libal, T. (2020). A meta-level annotation language for legal texts. In *Lecture notes in artificial intelligence*: *Vol. 12061*, *Proceedings of cLAR'2020 – the 3rd international conference on logic and argumentation* (pp. 131–150). Springer.

Mankins, J. C., et al. (1995). Technology readiness levels. *White Paper*.

Merigoux, D., Chataing, N., & Protzenko, J. (2021). Catala: a programming language for the law. *Proceedings of ACM Programming Languages*, *5*(77), http://dx.doi.org/10.1145/3473582.

Mohun, J., & Roberts, A. (2020). *Cracking the code – rulemaking for humans and machines (Draft working paper)*. Organisation for Economic Co-operation and Development, Open and Innovative Government Division, https://oecd-opsi.org/wp-content/uploads/2020/05/OECD-OPSI-draft-Innovation-Primer-Rules-as-Code.docx.

Morris, J. (2020). Spreadsheets for legal reasoning: The continued promise of declarative logic programming in law. http://dx.doi.org/10.2139/ssrn.3577239, SSRN.

Mowbray, A., Chung, P., & Greenleaf, G. (2023). Computer law and security ReviewVolume 48April 2023 article number 105772 document type. *Computer Law and Security Review*, *48*(105772), 1–12.

Schmautzer, F., Pfister, J., Braun, J., & Schweighofer, E. (2024). *Law as code – Rechtliche Aspekte (einschl. rechtsinformatische Überlegungen)*. Vienna: Arbeitsgruppe Rechtsinformatik, Juridicum, Universität Wien.

Sein, M. K., Henfridsson, O., Purao, S., Rossi, M., & Lindgren, R. (2011). Action design research. *MIS Quarterly*, 37–56.

Sergot, M. J., Sadri, F., Kowalski, R. A., Kriwaczek, F., Hammond, P., & Cory, H. T. (1986). The british nationality act as a logic program. *Communications of the ACM*, *29*(5), 370–386.

Tammet, T., Järv, P., Verrev, M., & Draheim, D. (2023). An experimental pipeline for automated reasoning in natural language. In *Lecture notes in computer science*: *Vol. 14132*, *Proceedings of cADE'23 – the 29th international conference on automated deduction* (pp. 509–521). Springer, http://dx.doi.org/10.1007/978-3-031-38499-8_29.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Wilson, E. (1992). Guiding lawyers: mapping law into hypertext. *Artificial Intelligence Review*, *6*, 161–189.

Wong, M. W. (2020). *Rules as code: Seven levels of digitisation*: *Tech. rep.*, Yong Pung How School of Law, Singapore Management University, https://ink.library.smu.edu.sg/sol_research/3093.

Xiao, X., Paradkar, A., Thummalapenta, S., & Xie, T. (2012). Automated extraction of security policies from natural-language software documents. In *Proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering*. New York, NY, USA: Association for Computing Machinery, ISBN: 9781450316149, http://dx.doi.org/10.1145/2393596.2393608.

Yalumov, K., & Gargov, G. (1986). Representing legal texts by logic programs. In *Cybernetics and systems' 86: proceedings of the eighth European meeting on cybernetics and systems research, organized by the Austrian society for cybernetic studies, held at the university of vienna, Austria, 1–4 April 1986* (pp. 751–758). Springer.