Representing Normative Regulations in OWL DL for Automated Compliance Checking Supported by Text Annotation

Ildar Baimuratov[1,2,*], Denis Turygin[3]

[1]L3S Research Center, Leibniz University Hannover, Germany
[2]TIB - Leibniz Information Centre for Science and Technology, Hannover, Germany
[3]ITMO University, St. Petersburg, Russia

Abstract

Compliance checking involves determining whether a regulated entity adheres to applicable normative regulations. Currently, this process is largely manual, requiring considerable time and expert knowledge, while remaining prone to human error. To address these challenges, various approaches to automating compliance checking have been explored, including Machine Learning. However, the reliability of ML-based methods remains questionable due to issues such as hallucinations, lack of transparency, and limited reproducibility. In contrast, symbolic reasoning is inherently accurate, reproducible, and explainable. While recent work has predominantly relied on SHACL, this study advocates for modeling regulations using OWL DL. It provides several advantages, such as a more human-readable syntax, semantics grounded in decidable Description Logics, the ability to detect redundant or inconsistent regulations through reasoning and generating explanations that ensure traceability. To support this approach, we introduce an annotation schema and a corresponding algorithm that transforms regulatory text annotations into OWL DL code. We validate our method through a proof-of-concept implementation applied to examples from the building construction domain.

Keywords

Normative Regulations, Text Annotation, OWL DL, Reasoning, Compliance checking

1. Introduction

Normative regulations govern business processes, industry, law, and various other domains. The process of verifying whether a regulated entity meets these regulations is known as compliance checking. Currently, this process is predominantly manual, requiring significant time and expertise while remaining prone to human error. For instance, in building construction, a single compliance review cycle can take several weeks, and multiple review cycles may be necessary due to design modifications. Non-compliance with building regulations can result in fines, penalties, or even criminal prosecution. Moreover, studies have shown significant discrepancies in manual code reviews, with different plan review departments often reaching inconsistent conclusions when evaluating the same set of plans [1]. Additionally, the redundancy of building codes contributes to inefficiencies and increases the likelihood of errors during the compliance checking process [2] [3].

Thus, there is a clear need to automate compliance checking. However, normative regulations are typically presented in human-readable formats, making them incompatible with software processing. Compared to manual compliance checking, automated compliance checking (ACC) is expected to improve efficiency by reducing time, costs, and errors. However, existing compliance checking systems, such as the Solibri[1] building model checker, rely on manually created, hard-coded, proprietary rules to represent normative regulations. While effective for a specific set of regulations within a given timeframe, this rigid approach requires significant effort to adapt to different regulatory codes and

LDAC 2025: 13th Linked Data in Architecture and Construction Workshop, July 09–11, 2025, Porto, Portugal
*Corresponding author.
ildar.baimuratov@l3s.de (I. Baimuratov); turygindenis7@gmail.com (D. Turygin)
0000-0002-6573-131X (I. Baimuratov); 0009-0003-5502-4077 (D. Turygin)

[1]https://www.solibri.com/

maintain over time. Machine learning (ML), particularly large language models (LLMs), offers potential support for ACC. However, the trustworthiness of ML models remains questionable due to issues such as hallucinations, lack of transparency, and limited reproducibility — critical factors in responsible domains like building construction.

In contrast, symbolic reasoning is inherently accurate, reproducible, and explainable. In recent years researchers have investigated the use of Semantic Web technologies, such as RDF, OWL, SPARQL, SWRL and SHACL for compliance checking. However, to the best of our knowledge, no study has applied OWL DL[2] reasoning for ACC. Among the most relevant works, Fitkau and Hartmann [2] modeled part of building regulations using OWL DL but processed them solely with DL querying. Nuyts et al. [4] employed OWL DL to check information availability, while compliance constraints were modeled using SPARQL, SWRL, and SHACL. However, query-based approaches, such as SHACL and DL Query, have several limitations, including the lack of established semantics; the absence of consistency checking; the inability to trace violations back to their sources; non-completely human-readable syntax; and dependence on the RDF graph structure. In contrast, OWL DL reasoning offers several advantages, such as a standardized, human-readable (Manchester) syntax; semantics grounded in decidable description logics (DLs); independence from data complexity; explanations that ensure traceability; and identifying redundant or inconsistent regulations. The primary challenge that hinders researchers from using OWL is the open-world assumption (OWA). However, we argue that OWL DL is expressive enough to produce the same results as closed-world reasoning, provided that the data is modeled correctly.

Converting textual regulations into machine-readable formats such as OWL DL remains a challenging task. Various approaches have been explored to facilitate the formalization of regulations, including NLP techniques to generate Prolog clauses [1] or SHACL shapes [5], deep learning for LegalRuleML [6], and Large Language Models (LLMs) for SPARQL [7]. However, since modeling regulations in OWL DL has been largely unexplored, there are no studies on translating regulation texts into OWL DL. To address these gaps, we propose a text annotation schema and an algorithm for automatically converting annotated regulations into OWL DL code. The annotation schema facilitates the alignment of text with the regulations' semantics, making it accessible to domain experts. It also leverages existing annotation tools, removing the need for custom formalization interfaces, and opens the way for future integration with machine learning models to support the annotation process.

The contributions of this research include: 1) An approach for representing normative regulations in OWL DL that enables ACC through OWL reasoning. 2) A text annotation schema and an algorithm for automatically converting annotated regulations into OWL DL code. The proposed approach is validated through a proof-of-concept implementation applied to examples from the building construction domain.

The paper is organized as follows: section 2 reviews the relevant research, section 3 describes the proposed text annotation schema, section 4 presents the algorithm for converting regulations into OWL DL code, section 5 demonstrates the proof of concept, section 6 discusses the advantages of using OWL DL, and section 7 concludes the work.

2. Related Work

In this section, we review research relevant to our work, focusing on approaches for machine-readable representation of regulations and the streamlining of their formalization.

2.1. Regulation representation

Normative regulations are studied across various disciplines, including law, legal reasoning, deontic logic, and artificial intelligence. Researchers have explored the formalization of these regulations to facilitate ACC. In this subsection, we focus specifically on the manual conversion of the regulations.

Hashmi et al. [8] proposed an abstract regulatory compliance framework for business processes based on deontic logic. Among machine-readable representation, LegalRuleML [9] extended the syntax

[2] https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

of RuleML[3] with concepts and features specific to legal norms. However, LegalRuleML provides only mechanisms to capture and represent different interpretations of legal norms, without relying on any specific logical framework. Gandon et al. [10] explored the application of Semantic Web frameworks to the formalization and processing of normative regulations. They built upon the LegalRuleML model, incorporating notions of regulatory compliance from [8]. Their approach modeled states of affairs as named graphs and utilizes SPARQL for querying. Francesconi [11] also utilized SPARQL to reason over legislation within a DL framework. Lam and Hashmi [12] presented a method for transforming legal norms from LegalRuleML into a variant of Modal Defeasible Logic. This transformation was implemented as an extension to the DL reasoner SPINdle.

In the building construction domain, the use of Semantic Web technologies has also been evaluated. Yurchyshyna and Zarli [13] represented constraints with SPARQL queries. Pauwels et al. [14] utilized N3 logic. The SWRL language was used to encode rules in [15], [16] and [17]. Fitkau and Hartmann [2] developed a Fire Safety Ontology using SPARQL, DL Query and SWRL. The use of SHACL for compliance checking has been evaluated in [18], [19],[20], [21],[5] and [22]. The use of OWL was previously demonstrated in [4], however, the authors applied it solely to model information availability constraints, while compliance constraints were modeled using SPARQL, SWRL, or SHACL.

2.2. Streamlining regulation formalization

Converting textual rules into machine-readable formats is a challenging task. Researchers have explored the potential of NLP and ML techniques to translate natural language regulatory texts into specific representation formats. Hjelseth and Nisbet [23] introduced an annotation schema for normative texts called RASE, although the study does not address the conversion of these annotations into an executable format. Zhang and El-Gohary [1] proposed an ACC system that uses NLP techniques to automatically extract normative information from documents and convert it into logical rules, opting for first-order logic and its implementation in B-Prolog. Donkers and Petrova [5] leveraged the linguistic structure of sentences to automatically generate SHACL representations using predefined templates. Recent studies also explore automating regulation formalization with LegalRuleML via deep learning [6] or with SPARQL using LLMs [7]. To address the lack of resources for ML, Hettiarachchi et al. [24] introduced CODE-ACCORD, a dataset of sentences from the building regulations of England and Finland, annotated with a custom set of entities and relations not grounded in any formal framework.

In contrast to these approaches, we propose an annotation schema that directly aligns with OWL DL syntax, ensuring that the translation of annotated regulations is both transparent and human-controllable. The intermediate annotation step facilitates the alignment of the text with the regulations' semantics, making it more accessible to domain experts, and leverages existing annotation tools, eliminating the need for custom formalization interfaces. Finally, machine learning models can be trained on these annotations to further assist in the annotation process.

3. Annotation Schema

In this section, we introduce a schema for annotating regulatory text to streamline its translation into OWL DL code. The annotation schema includes three layers: Terms, Semantic Types, and Semantic Roles. Each layer includes both span-based tags and arrows connecting them. For clarity, annotation tags are written in *italics* and OWL expressions are written in `teletype`.

Terms. The first annotation layer focuses on domain terms. This layer is essential for aligning the formalized regulations with the domain data for validation and for enabling efficient search and filtering of regulations. This layer utilizes domain vocabularies or ontologies as tags. For instance, in the building

[3] https://www.ruleml.org/

Table 1
Mapping of Semantic Types to the OWL DL syntax

Tag	OWL
Literal	literal
Class	owl:Class
Not	owl:complementOf
Or	owl:unionOf
Relation	owl:ObjectProperty
Property	owl:DataProperty
Some	owl:someValuesFrom
Only	owl:allValuesFrom
Number	owl:maxCardinality, owl:minCardinality, owl:cardinality
Comparison	xsd:minInclusive, xsd:minExclusive, xsd:maxInclusive, xsd:maxExclusive

Table 2
Arrows connecting Semantic Type tags

Arrow	Start	End
Domain	Relation, Property	Class, Relation
Range	Relation	Class, Relation, Property
	Property	Literal
Of	Not, Or	Class, Relation, Property
	Comparison	Literal
	Some, Only, Number	Relation, Property

construction domain, Building Information Models (BIM) [25] are defined using the Industry Foundation Classes (IFC)[4]. Its OWL serialization, ifcOWL [26], can be employed as values within the Term layer.

Semantic Types. The Semantic Types layer includes tags that correspond to syntactic elements of the OWL DL language, enabling the construction of class restrictions. We primarily use Manchester syntax[5] for tag names, though some are slightly modified for better readability by domain experts. For example, `owl:ObjectProperty` is represented by the tag *Relation*, and `owl:DataProperty` corresponds to *Property*. The mapping of these tags to OWL is provided in Table 1. Additionally, the Semantic Types layer includes the specific arrows *Domain*, *Range* and *Of*. Their possible starting and ending tags are provided in Table 2.

If a Term tag appears on the same tokens as a Semantic Type tag, the latter might seem redundant. However, our goal is to provide a comprehensive representation of the regulation's semantics at the Semantic Type layer, independent of the specific vocabulary used in the Term layer. This approach ensures that Semantic Types can be reused across different domain models (or even without any). As a consequence, raw OWL code may contain multiple instances of the same concept, reflecting its various spellings and synonyms. However, since our aim is not to create a single ontology for all regulations but rather a set of individual OWL DL programs, each representing a regulation, we do not need to align these variations.

Semantic Roles. Semantic Roles are essential for constructing axioms in the OWL DL language. OWL axioms are statements that are asserted to be true within the domain of interest. In OWL, two types of assertions are possible: one about the relation between individuals, and the other about the membership of an individual or class in a class. In the context of normative regulations, we focus on asserting relationships between restricted classes (annotated with Semantic Types), so only the relation `owl:SubClassOf`, or General Class Inclusion (GCI) is needed. To annotate this relation, two semantic roles are introduced: 1) *Subject*, which represents the subclass, 2) and *Requirement*, which represents the superclass. In other words, the *Subject* role denotes the OWL class to which the regulation applies, while the *Requirement* role represents the class containing the imposed regulation. The arrow between *Subject* and *Requirement*, corresponding to the `owl:SubClassOf` relation is annotated with the tag *To*, resulting in the triple *<Subject, To, Requirement>*.

[4]https://technical.buildingsmart.org/standards/ifc
[5]https://www.w3.org/TR/owl2-manchester-syntax/

Linguistic arrows. In addition to arrows that represent the semantic relationships between different tags, the proposed annotation schema also includes linguistic arrows, which only connect separate pieces of text related to the same tag. There are three linguistic arrows with distinct properties, which are described as follows:

- *Concatenation*: $a \rightarrow b \Rightarrow ab$,
- *Distribution*: $a \rightarrow b, a \rightarrow c \Rightarrow ab, ac$,
- *Self-Distribution*: $a \rightarrow b \Rightarrow a, ab$,

where a, b, and c are pieces of text and ab or ac represent a and b or a and c respectively concatenated. *Self-distribution* can be considered as *Distribution* in which one of the strings is empty:

$$a \rightarrow 0, a \rightarrow b \Rightarrow a, ab.$$

However, since not all tools allow annotating an empty line in the text, we introduce *Self-distribution* as a separate arrow.

Annotation Interface. To annotate regulations following the proposed schema, we use the INCEpTION tool [27]. One of the key advantages of this tool is its ability to import ontologies, which can then be used as annotation tags. In our use-case, the imported ifcOWL ontology serves as the tagset for the Term layer, and tagsets for other layers are created manually. Additionally, this tool supports the integration of ML-based recommenders, which can streamline the annotation process in future work. The original regulations are imported into INCEpTION in TXT format, and the annotations are exported in WebAnno TSV v3.3. Example 1 and Example 2 illustrate annotated regulations, one qualitative and the other quantitative.

Example 1. As an example, consider a qualitative regulation from the building construction domain with the text "If the degree of fire resistance of a building is III, then the fire resistance limit of the beams in it should be R15". In this regulation, we annotate the terms *ifcBuilding*, *ifcBeam*, *FIRESAFETY*, and *FIREPROTECTION*. The first two are annotated with the Semantic Type *Class*, while the latter two are annotated as *Property*. Additionally, "III" and "R15" are annotated as *Literal*, the preposition "in" is labeled as a *Relation*, and "should be" is annotated with the tag *Only*. Finally, the phrases "If the degree of fire resistance of a building is III" and "the beams in it" are connected with a *Concatenation* arrow and form together a subject of the regulation. Similarly, the phrases "then the fire resistance limit" and "should be R15" are also connected with a *Concatenation* arrow and identified as the regulation's requirement. Figure 1 shows the annotation of this regulation within the INCEpTION interface.

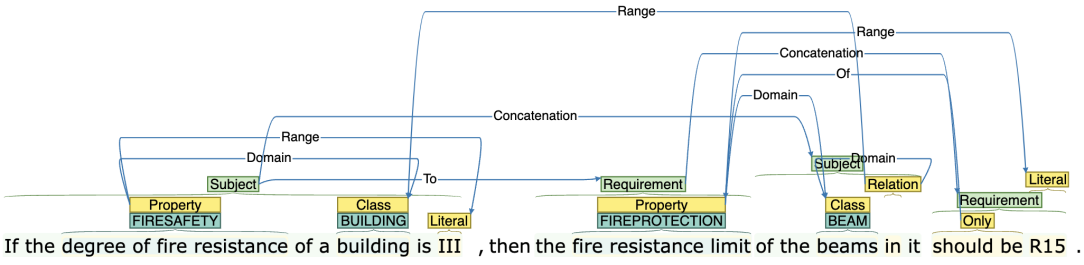


Figure 1: Annotation of Example 1 in INCEpTION

Example 2. As a quantitative requirement, consider the following: "For buildings with a capacity of not more than 300 students the height of classrooms must be at least 3.0 m". Compared to Example 1, the subject of this regulation contains a chain consisting of the relation "for" and the property "capacity", as well as the constrained data type "not more than 300". Its requirement also includes the constrained data type "at least 3.0". Figure 2 illustrates the annotation of this regulation.

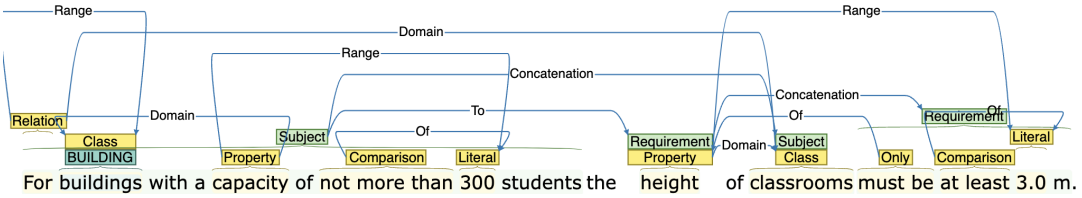


Figure 2: Annotation of Example 2 in INCEpTION

4. Transforming Annotations into OWL DL

In this section, we present our algorithm for transforming annotated regulations into OWL DL code, which enables the classification of domain objects into regulation subjects and the subsequent checking of their compliance with the imposed requirements with OWL reasoning. The algorithm is divided into several steps: 1) data preprocessing, 2) generating elementary OWL entities, 3) aligning domain terms with semantic types, 4) vocabulary-based mapping of numbers and comparisons, 5) constructing class restrictions, and finally, 6) constructing axioms. The algorithm is recursive, allowing it to process annotations of any complexity.

Preprocessing. The algorithm takes as input a TSV file exported from INCEpTION, which contains the annotated regulation in a tabular format. In this table, each row corresponds to a word-based token, indexed by its order of appearance in the text, and each column represents a distinct annotation layer. Before generating OWL code, the data undergoes preprocessing in two stages. First, linguistic arrows are processed, so that each table row corresponds to exactly one tag. New tags derived from these operations are added to the table, while the old ones are removed. Second, tokens are grouped and extracted into three separate tables, each corresponding to an annotation layer: the table of Terms Tr, the table of Semantic Types ST, and the table of Semantic Roles SR. Additionally, we use lowercase letters to denote single elements, while uppercase denotes sets. For example, tr refers to a single row from the table of Terms Tr. Table 3a, Table 3c, and Table 3b present the layer tables for Example 1 and Table 4a, Table 4c, and Table 4b correspond to Example 2.

Table 3
Layer tables from Example 1

(a) Terms

Index	Token	Tag
1	beams	BEAM
2	building	BUILDING
9	fire resistance limit	FIREPROTECTION
8	degree of fire resistance	FIRESAFETY

(b) Semantic Roles

Index	Token	Tag	To
0	If the degree of fire resistance of a building is III of the beams in it	Subject	1
1	the fire resistance limit should be R15	Requirement	-

(c) Semantic types

Index	Token	Tag	Domain	Range	Of
1-20	beams	Class	-	-	-
1-9	building	Class	-	-	-
1-11	III	Literal	-	-	-
1-25	R15	Literal	-	-	-
5	degree of fire resistance	Property	1-9	1-11	-
6	fire resistance limit	Property	1-20	1-25	-
1-21	in	Relation	1-20	1-9	-
7	should be	Only	-	-	6

Table 4
Layer tables from Example 2

(a) Terms

Index	Token	Tag
5	buildings	BUILDING

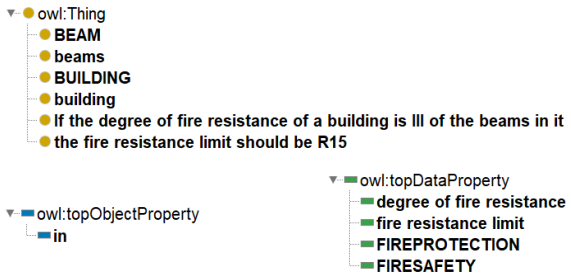
(b) Semantic Roles

Index	Token	Tag	To
0	For buildings with a capacity of not more than 300 students classrooms	Subject	1
1	height must be at least 3.0 m	Requirement	-

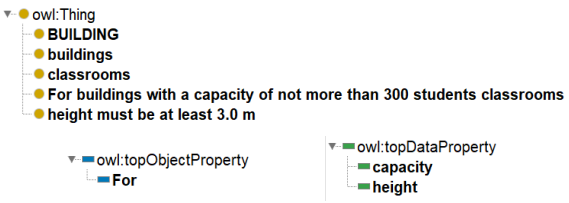
(c) Semantic types

Index	Token	Tag	Domain	Range	Of
1-15	classrooms	Class	-	-	-
1-2	buildings	Class	-	-	-
3	not more than	Comparison	-		1-10
5	at least	Comparison	-		1-20
1-10	300	Literal	-	-	-
1-20	3.0	Literal	-	-	-
1-13	height	Property	1-15	1-20	-
1-5	capacity	Property	1-2	1-10	-
1-1	For	Relation	1-15	1-2	-
4	must be	Only	-	-	1-13

Generating entities. To transform an annotated regulation into OWL Dl code, we start with an ontology O that includes entities imported from a domain ontology corresponding to Tr (in our use-case, ifcOWL). The first step involves generating elementary entities in O based on the tags *Class*, *Relation*, and *Property* from ST, according to the mappings in Table 1. Additionally, OWL classes are created based on the tags *Subject* and *Restriction* from SR. The original tokens from the regulation text are assigned as labels for the created entities. Figure 3a demonstrates OWL entities generated based on Example 1 and Figure 3b for Example 2.



(a) Based on Example 1



(b) Based on Example 2

Figure 3: Generated OWL entities

Entity alignment. Once the elementary entities are created, we establish connections between them across different annotation layers based on token inclusion. Specifically, for each Term tag, we check whether its tokens also belong to any *Class* tag. If so, we add the `owl:equivalentTo` relation. These connections ensure that domain objects align with regulation classes and are further classified through class restrictions into regulation subjects. In Example 1, the equivalence between the `ifc:IfcBeam` class and the `beams` OWL class is generated. This guarantees that any object identified as belonging to the *ifc:IfcBeam* class in a BIM model also belongs to the `beams` class in the ontology. The same for the `ifc:IfcBuilding` and `buildings` classes.

Vocabulary mapping. To balance annotation complexity with ontology generation accuracy, we employ vocabulary-based mappings for certain tokens from ST to OWL operators. Specifically, we

define two mappings: $Card : T \to N$, which matches strings with integers N for constructing cardinal restrictions, and $Constr : T \to \{\leq, =, \geq\}$, which is used to define constrained data types. For instance, in Example 2, the token "not more than", labeled with the tag *Comparison*, is mapped through $Constr$ to `xsd:maxInclusive`. For the $Card$ mapping, no relevant cases appear in our examples, however, it could be used to map, for instance, the word "two" to the integer 2.

Constructing class restrictions. Next, we use the remaining tags from ST to construct restricted classes based on the previously generated elementary OWL entities. For shortness, we refer to the combined set of ObjectProperties and DataProperties as predicates P. First, we identify the set of range classes and literals, denoted as R, that appear at the end of predicate chains, i.e. those that are not domains for other predicates. We then apply backward induction to iteratively construct intermediate restricted classes utilizing the ST table. In each iteration, we check if any *Not* or *Or* tags are associated with elements in R and apply `owl:complementOf` or `owl:unionOf` respectively. For each $r \in R$ (a class if it corresponds to a *Relation* or a literal if it corresonpds to a *Property*), we identify its incoming predicates P through ST and determine for every predicate its associated restriction. Predicate restrictions are generated using the tags *Some*, *Only* or *Number*. If a predicate has no annotated restriction, we assign *Some* by default if it belongs to the subject of the regulation and *Only* if it belongs to the requirement. The $Card$ mapping is used to define cardinal restrictions. If a *Literal* is accompanied by a *Comparison* tag, the mapping $Constr$ is used to generate a constrained data type. Each intermediate restricted class c_{restr} is added to the resulting set C_{restr}, while the original r is removed from R. These intermediate restricted classes are then passed to the next iteration of the algorithm. Once all predicate chains are processed, the terminal restricted classes along with any remaining classes in R, i.e. those that are not ranges of any predicates but possibly with complements or unions, form the complete set of the building blocks for defining the regulation's subject and requirement. Algorithm 1 outlines the recursive construction of a set of intermediate restricted classes.

Constructing axioms. Finally, using all intermediate classes C_{restr} obtained from Algorithm 1, we construct class restrictions that accurately capture the semantics of regulation subjects and requirements. This enables domain object classification and compliance checking with OWL reasoning. To achieve this, for each $sr \in SR$, we identify the corresponding class restrictions $C_{sr} \subset C_{restr}$ based on textual token inclusion. The selected classes from C_{sr} are then combined using the `owl:intersectionOf` relation to form a single complex class c_{sr}, which is declared equivalent (`owl:equivalenTo`) to the given sr. Finally, we utilize the *To* arrow between semantic roles in SR to establish an `owl:subClassOf` relation between the class s, which represents the subject of the regulation, with the class r, which represents the requirement of the regulation. Algorithm 2 details this final step. Figure 4a and Figure 4b demonstrate the resulting axioms, which represent the semantics of the regulations from Example 1 and Example 2, respectively.

Algorithm 1 Constructing class restrictions

Require: ST, R
  $R \leftarrow checkOr(R)$
  $R \leftarrow checkNot(R)$
  $C_{restr} \leftarrow \emptyset$
  for $r \in R$ **do**
    $P \leftarrow ST(r)$
    for $p \in P$ **do**
      $restr \leftarrow ST(P)$
      $c_{restr} \leftarrow restrictedClass(p, r, restr)$
      $C_{restr} \leftarrow C_{restr} \cup \{c_{restr}\}$
    end for
    $R \leftarrow R \setminus \{r\}$
  end for
  Apply Algorithm 1 to ST, C_{restr}
  $C_{restr} \leftarrow C_{restr} \cup R$

Algorithm 2 Constructing axioms

Require: SR, C_{restr}
  for $sr \in SR$ **do**
    $C_{sr} \leftarrow select(C_{restr}, sr)$
    $c_{sr} \leftarrow \bigcap C_{sr}$
    $sr \equiv c_{sr}$
  end for
  for $To \in SR$ **do**
    $(s, r) \leftarrow SR(To)$
    $s \subseteq r$
  end for

As a result, given an annotated regulation, the proposed method, with certain limitations, generates an ontology in machine-interpretable OWL DL code. In this ontology, domain terms are connected to restricted classes representing the subject and requirement of the regulation. Finally, subjects and requirements are connected using the `owl:subClassOf` relation. Consequently, the ontology enables the classification of objects described by domain terms into regulation subjects and facilitates trustworthy compliance checking through OWL reasoning.

5. Proof of Concept

To validate the proposed approach, we developed a proof of concept that converts annotated regulations, exported from INCEpTION, into OWL DL code using the described algorithm. We applied this prototype to the regulations from Example 1 and Example 2 and validated the resulting OWL DL representations through a compliance-checking scenario with modeled data.

Prototype. The prototype receives annotated regulations exported from INCEpTION in WebAnno TSV v3.3 format and generates OWL DL code with machine-actionable regulations following the proposed algorithm. The prototype is implemented in Python using the Owlready2 library [28].

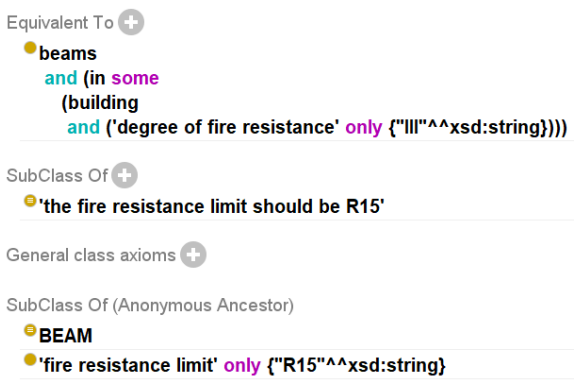
Data. To validate the OWL code generated based on Example 1, we create six "closed" individuals: building (Listing 1), building with the degree of fire resistance III (Listing 3), beam (Listing 2), beam in the building (Listing 4), beam in the building of the degree of fire resistance III (Listing 5), and beam in the building of the degree of fire resistance III with fire resistance limit R15 (Listing 6). According to the complexity levels defined in [29], the last one is classified as level L3, as evaluating this individual involves three triples: *(beam, FIREPROTECTION, R15)*, *(beam, in, building_III)* and *(building, FIRESAFETY, III)*.

```
Individual: building

    Types:
        BUILDING,
        in only owl:Nothing
```

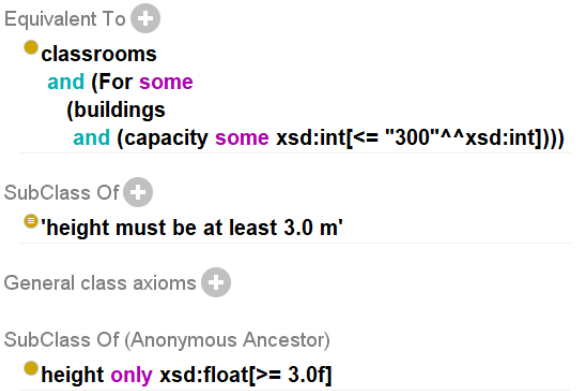
Listing 1: Building

```
Individual: beam

    Types:
        BEAM,
        in only owl:Nothing
```

Listing 2: Beam



(a) From Example 1



(b) From Example 2

Figure 4: Constructed axioms

```
Individual: building_III

    Types:
        BUILDING,
        in only owl:Nothing,
        FIRESAFETY only {"III"^^xsd:
            string}

    Facts:
     FIRESAFETY  "III"^^xsd:string
```

Listing 3: Building of the degree of fire resistance III

```
Individual: beam_in

    Types:
        BEAM,
        in only ({building})

    Facts:
     in  building
```

Listing 4: Beam in the building

```
Individual: beam_in_III

    Types:
        BEAM,
        in only ({building_III})

    Facts:
     in  building_III
```

Listing 5: Beam in the building of the degree of fire resistance III

```
Individual: beam_in_III_R15

    Types:
        BEAM,
        in only ({building_III})

    Facts:
     in  building_III,
     FIREPROTECTION  "R 15"^^xsd:string
```

Listing 6: Beam in the building of the degree of fire resistance III with fire resistance limit R15

Validation script. To validate the generated OWL DL code, a user scenario was developed. It consists of the following steps:

1. Choose a regulation r.
2. Generate onto O from r using the prototype.
3. Import into O individuals representing domain objects that comply with r.
4. Run reasoner.
5. Modify the individuals so that they do not comply with r.
6. Run reasoner.

If the generated OWL DL code is correct, the reasoning in Step 4 should successfully classify the individuals as subjects of r. However, after Step 6, the reasoner is expected to detect noncompliance as O becomes inconsistent. This scenario was implemented in Python using the Owlready2 library and Pellet reasoner [30].

Results. We applied the described scenario to the knowledge graph obtained by importing the individuals listed above into the ontology generated from Example 1. The logs from the initial reasoner run are provided in Listing 7. As expected, the process completed successfully in 3.28 sec. Among other results, it correctly classified the instances `beam`, `beam_in_III`, and `beam_in_III_R15` as subjects of the regulation. Finally, we modified the fire resistance limit of `beam_in_III_R15` from "R15" to '"R14", and reran the reasoner. As expected, it raised an error due to ontology inconsistency. Listing 8 Provides Pellet's explanation for this inconsistency. These results confirm the validity of the generated OWL DL code.

6. Discussion

Researchers have raised concerns that OWL, based on the Open World Assumption (OWA), may not effectively handle constraints for compliance checking. As a result, recent studies have predominantly

```
* Owlready2 * Running Pellet...
* Owlready2 * Pellet took 3.277837038040161 seconds
* Owlready2 * Pellet output:
...
* Owlready * Reparenting reg.beam: {reg.BEAM} => {reg.Subject}
* Owlready * Reparenting reg.beam_in: {reg.BEAM} => {reg.Subject, reg.BEAM}
* Owlready * Reparenting reg.beam_in_III: {reg.BEAM} => {reg.Subject}
* Owlready * Reparenting reg.beam_in_III_R15: {reg.BEAM} => {reg.Subject}
...
```

Listing 7: Successful reasoning

```
Explanation for: owl:Thing SubClassOf owl:Nothing
1) EquivalentProperties: FIREPROTECTION, 'fire resistance limit'
2) 'beam_in_III_R15' Type 'If the degree of fire resistance of a building is III the
    beams in it'
3) 'the fire resistance limit should be R15' EquivalentTo 'fire resistance limit' only
    {"R15"^^xsd:string}
4) 'If the degree of fire resistance of a building is III the beams in it' SubClassOf '
    the fire resistance limit should be R15'
5) beam_in_III FIREPROTECTION "R14"^^xsd:string
```

Listing 8: Explanation of inconsistency

used SHACL, which relies on the Closed World Assumption (CWA). In this section, we advocate for using OWL DL to model regulations.

First, we compare OWL DL and SHACL in terms of their expressive power and computational decidability by relating them to the common framework of description logics (DLs). Leinberger et al. [31] map SHACL shapes to DLs, enabling shape containment to be addressed through DL reasoning. They conclude that the corresponding description logic for SHACL shapes is ALCOIQ($\circ$), which is undecidable for infinite models. A restricted fragment of SHACL, limiting path concatenation, corresponds to the description logic SROIQ, which underpins OWL DL. Bogaerts et al. [32] explore the relationship between SHACL and OWL, arguing that SHACL is, in fact, a description logic. However, they point out that other tasks typically studied in DLs, such as consistency checking, are undecidable in SHACL, except for finite model checking.

Second, although OWL is based on OWA, it is still expressive enough to support closed-world reasoning. To achieve this, missing information and disjointness must be explicitly defined. For instance, if an individual a has only one relation $R(a, b)$ (in DL notation), this fact must be explicitly declared in the ontology at the time of compliance checking as $a \in \forall R.\{b\}$. Alternatively, if a has no relations at all, and there exists a relation R in the ontology, it must be formulated as $a \in \forall R.\bot$. Software tools like Owlready2 [28] provide methods to algorithmically apply local closure to specific individuals, classes, or even entire ontologies. Once closed, these ontologies can be processed by OWL reasoners to perform constraint checking similarly to closed-world reasoning. Therefore, the OWA is more relevant to data modeling than to regulation modeling, and the former does not require additional effort from users.

Finally, OWL offers better human-computer interaction. Ahmetaj et al. [33] emphasize that in SHACL, validation reports provide limited information, mainly identifying the node and indicating constraint violations. In contrast, OWL reasoners offer detailed explanations of classifications and inconsistencies, allowing users to trace them back to their sources. Additionally, OWL DL is supported by the Manchester syntax, which is more human-readable compared to the serialization format available for SHACL.

7. Conclusion

In this study, we addressed the challenge of converting normative regulations into a machine-interpretable OWL DL code to enable compliance checking using automated reasoning. To facilitate the formalization, we proposed an annotation schema for regulation texts that includes three tag layers: domain Terms, Semantic Types and Semantic Roles, and a number of arrows between the tags. Additionally, we developed an algorithm to convert annotated regulations into OWL DL code. In the resulting OWL DL representations, domain terms are connected to restricted OWL classes that represent regulation subjects, enabling the automated identification of applicable regulations. Regulatory requirements are similarly modeled as restricted OWL classes and are connected to their respective subjects using general class inclusion axioms. As a result, if an object is classified as a regulation subject but fails to satisfy the corresponding requirement, an inconsistency is detected during reasoning. To validate the proposed method, we implemented a proof of concept and a validation script. The prototype was successfully applied to examples from the building construction domain.

Limitations. The proposed approach has several limitations. First, it relies on vocabulary mappings to generate cardinality restrictions and restricted data types. While the set of words representing natural numbers or comparisons is countable, any new ones or typos in regulatory texts require manual processing. Second, not all relational restrictions or logical connectives are explicitly stated in natural language text. Our approach assigns default values in such cases, but this can potentially lead to inaccuracies in regulation modeling. Specifically, in Example 1, it can be stated that the fire resistance limit of the beams should be *at least* R15, rather than strictly R15. If this is the case, the automated generation of the corresponding OWL code from the annotation becomes infeasible, as the comparison is not explicitly mentioned in the text. However, with human intervention, this interpretation can still be formalized in OWL DL as `'fire resistance limit' only xsd:float[>= 15.0f]`. Finally, no ontology is entirely comprehensive for any application domain. For instance, the regulation in Example 2 applies to classrooms, yet the ifcOWL ontology lacks a dedicated class for classrooms. This gap between domain data and regulations must be addressed to enable automated compliance checking.

Future work. In the future, we will focus on automating regulation annotation using machine learning models. We believe that approaches relying on automatically generating code for ACC or performing direct compliance checking with ML will never be trustworthy enough for full automation. In contrast, our approach will leverage machine learning solely to suggest annotation tags, while the semantic modeling of regulations remains under human control. This ensures that human autonomy and decision-making are preserved, aligning with ethical principles for AI development and deployment, such as the Artificial Intelligence Act[6] and the EU Ethics Guidelines for Trustworthy AI[7].

Acknowledgments

We would like to acknowledge the funding by the German Ministry of Education and Research (BmBF) for the project KISSKI AI Service Center (01IS22093C) and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany´s Excellence Strategy – EXC 2163/1 - Sustainable and Energy Efficient Aviation – Project-ID 390881007.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

[6] https://artificialintelligenceact.eu/

[7] https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

References

[1] J. Zhang, N. M. El-Gohary, Integrating semantic NLP and logic reasoning into a unified system for fully-automated code checking, Autom. Constr. 73 (2017) 45–57. doi:10.1016/j.autcon.2016.08.027.

[2] I. Fitkau, T. Hartmann, An ontology-based approach of automatic compliance checking for structural fire safety requirements, Advanced Engineering Informatics 59 (2024) 102314.

[3] L. van Berlo, G. Costa, R. Klooster, K. Breitenfelder, R. Lavikka, K. Schneider, P. Paasiala, Bim information reliability consequences for digital permit checking, 2024.

[4] E. Nuyts, M. Bonduel, R. Verstraeten, Comparative analysis of approaches for automated compliance checking of construction data, Advanced Engineering Informatics 60 (2024) 102443.

[5] A. J. Donkers, E. Petrova, Converting fire safety regulations to shacl shapes using natural language processing, in: Proceedings of the 3rd NLP4KGC: Natural Language Processing for Knowledge Graph Construction co-located with the 20th International Conference on Semantic Systems (SEMANTiCS 2024), CEUR-WS. org, 2024.

[6] S. Fuchsa, J. Dimyadia, M. Witbrocka, R. Amora, Transformer-based semantic parsing of building reg-ulations: Towards supporting regulators in drafting machine-readable rules, in: Digital Building Permit Conference 2024, 2024.

[7] N. Chen, X. Lin, H. Jiang, Y. An, Automated building information modeling compliance check through a large language model combined with deep learning and ontology, Buildings 14 (2024) 1983.

[8] M. Hashmi, G. Governatori, M. Wynn, Normative requirements for regulatory compliance: An abstract formal framework, Information Systems Frontiers 18 (2015). doi:10.1007/s10796-015-9558-1.

[9] T. Athan, G. Governatori, M. Palmirani, A. Paschke, A. Wyner, LegalRuleML: Design Principles and Foundations, Springer International Publishing, Cham, 2015, pp. 151–188. doi:10.1007/978-3-319-21768-0_6.

[10] F. Gandon, G. Governatori, S. Villata, Normative requirements as linked data, in: JURIX 2017-The 30th international conference on Legal Knowledge and Information Systems, 2017, pp. 1–10. doi:10.3233/978-1-61499-838-9-1.

[11] E. Francesconi, Semantic model for legal resources: Annotation and reasoning over normative provisions, Semantic Web 7 (2016) 255–265. doi:10.3233/SW-140150.

[12] H.-P. Lam, M. Hashmi, Enabling reasoning with legalruleml, Theory and Practice of Logic Programming 19 (2018) 1–26. doi:10.1017/S1471068418000339.

[13] A. Yurchyshyna, A. Zarli, An ontology-based approach for formalisation and semantic organisation of conformance requirements in construction, Automation in Construction 18 (2009) 1084–1098.

[14] P. Pauwels, D. Van Deursen, R. Verstraeten, J. De Roo, R. De Meyer, R. Van de Walle, J. Van Campenhout, A semantic rule checking environment for building performance checking, Automation in construction 20 (2011) 506–518.

[15] T. H. Beach, Y. Rezgui, H. Li, T. Kasim, A rule-based semantic approach for automated regulatory compliance in the construction sector, Expert Systems with Applications 42 (2015) 5219–5231.

[16] M. Fahad, N. Bus, F. Andrieux, Towards mapping certification rules over bim, in: CIB W78 Conference, volume 3, 2016.

[17] L. Shi, D. Roman, From standards and regulations to executable rules: A case study in the building accessibility domain, in: N. Bassiliades, A. Bikakis, S. Costantini, E. Franconi, A. Giurca, R. Kontchakov, T. Patkos, F. Sadri, W. V. Woensel (Eds.), Proceedings of the Doctoral Consortium, Challenge, Industry Track, Tutorials and Posters @ RuleML+RR 2017 hosted by International Joint Conference on Rules and Reasoning 2017 (RuleML+RR 2017), London, UK, July 11-15, 2017, volume 1875 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2017.

[18] S. Stolk, K. McGlinn, Validation of ifcowl datasets using shacl, in: Proceedings of the 8th Linked Data in Architecture and Construction Workshop, 2020, pp. 91–104.

[19] A. T. Kovacs, A. Micsik, Bim quality control based on requirement linked data, International Journal of Architectural Computing 19 (2021) 431–448.
[20] S. Zentgraf, P. Hagedorn, M. König, Multi-requirements ontology engineering for automated processing of document-based building codes to linked building data properties, in: IOP Conference Series: Earth and Environmental Science, volume 1101, IOP Publishing, 2022, p. 092007.
[21] E. Nuyts, J. Werbrouck, R. Verstraeten, L. Deprez, Validation of building models against legislation using shacl, in: LDAC2023: Linked Data in Architecture and Construction Week, volume 3633, CEUR, 2023, pp. 164–175.
[22] P. Patlakas, I. Christovasilis, L. Riparbelli, F. K. Cheung, E. Vakaj, Semantic web-based automated compliance checking with integration of finite element analysis, Advanced Engineering Informatics 61 (2024) 102448.
[23] E. Hjelseth, N. N. Nisbet, Capturing normative constraints by use of the semantic mark-up rase methodology, in: Proceedings of CIB W78-W102 Conference, 2011, pp. 1–10.
[24] H. Hettiarachchi, A. Dridi, M. M. Gaber, P. Parsafard, N. Bocaneala, K. Breitenfelder, G. Costa, M. Hedblom, M. Juganaru-Mathieu, T. Mecharnia, et al., Code-accord: A corpus of building regulatory data for rule generation towards automatic compliance checking, Scientific Data 12 (2025) 170.
[25] C. M. Eastman, C. Eastman, P. Teicholz, R. Sacks, K. Liston, BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors, John Wiley & Sons, 2011.
[26] P. Pauwels, W. Terkaj, Express to owl for construction industry: Towards a recommendable and usable ifcowl ontology, Automation in Construction 63 (2016) 100–133. doi:https://doi.org/10.1016/j.autcon.2015.12.003.
[27] J.-C. Klie, M. Bugert, B. Boullosa, R. E. de Castilho, I. Gurevych, The inception platform: Machine-assisted and knowledge-oriented interactive annotation, in: Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations, Association for Computational Linguistics, 2018, pp. 5–9. Event Title: The 27th International Conference on Computational Linguistics (COLING 2018).
[28] J.-B. Lamy, Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies, Artificial Intelligence in Medicine 80 (2017) 11–28. doi:https://doi.org/10.1016/j.artmed.2017.07.002.
[29] M. Bonduel, J. Oraskari, P. Pauwels, M. Vergauwen, R. Klein, The ifc to linked building data converter: current status, in: 6th International Workshop on Linked Data in Architecture and Construction, CEUR-WS. org, 2018, pp. 34–43.
[30] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical owl-dl reasoner, Web Semant. 5 (2007) 51–53. doi:10.1016/j.websem.2007.03.004.
[31] M. Leinberger, P. Seifer, T. Rienstra, R. Lämmel, S. Staab, Deciding shacl shape containment through description logics reasoning, in: The Semantic Web–ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part I 19, Springer, 2020, pp. 366–383.
[32] B. Bogaerts, M. Jakubowski, J. Van den Bussche, Shacl: A description logic in disguise, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2022, pp. 75–88.
[33] S. Ahmetaj, R. David, M. Ortiz, A. Polleres, B. Shehu, M. Simkus, Reasoning about explanations for non-validation in shacl, in: Description Logics, 2021.