# Integrating NLP and Context-Free Grammar for Complex Rule Interpretation towards Automated Compliance Checking

**Yu-Cheng Zhou[a], Zhe Zheng[a], Jia-Rui Lin[a,*], Xin-Zheng Lu[a]**

[a] Department of Civil Engineering, Tsinghua University, Beijing, China
[*]Corresponding author, E-mail: lin611@tsinghua.edu.cn

## Acknowledgment

The authors are grateful for the financial support received from the National Natural Science Foundation of China (No. 51908323, No. 72091512), the National Key R&D Program of China (No. 2018YFD1100900), and the Tsinghua University Initiative Scientific Research Program (No. 2019Z02UOT).

## Abstract

Automated rule checking (ARC) is expected to significantly promote the efficiency and compliance of design in the construction industry. The most vital and complex stage of ARC is interpreting the regulatory text into a computer-processable format. However, existing systems and studies of rule interpretation either require considerable time-consuming manual effort or are based on exhaustive enumerations of matching patterns with a limited scope of application. To address this problem, this research integrates natural language processing (NLP) and context-free grammar (CFG) to propose a novel generalized rule interpretation framework, which can parse regulatory text like a domain-specific language. First, a syntax tree structure with several semantic elements is proposed to represent the roles and relations of concepts in regulatory text. Second, a deep learning model with the transfer learning technique is utilized to label the semantic elements in a sentence. Finally, a set of CFGs is built to parse a labeled sentence into the language-independent tree structure, from which computable checking rules can be generated. Experimental results demonstrate our method outperforms the state-of-the-art methods: it achieves 99.6% and 91.0% accuracies for parsing single- and multi-requirement sentences, respectively, where the multi-requirement sentences are more complex and common but difficult for existing methods to deal with. This research contributes a method and framework for rule interpretation with both a high level of automation and wide scope of application, which can create computable rules from various textual regulatory documents. This research also publishes the first regulation dataset for future exploration, validation, and benchmarking in the ARC area.

**Keywords:** automated rule checking (ARC); automated compliance checking (ACC); rule interpretation; context-free grammar; syntax tree; natural language processing (NLP); building information modeling (BIM); smart design

## 1 Introduction

The entire lifecycle of a built environment is governed by a variety of regulations, requirements, and standards [1,2]. The manual process of regulatory compliance checking is time-consuming, costly, and error-prone [3]. As an alternative to manual checking, automated rule checking (ARC) (also known as automated compliance checking (ACC)) is expected to significantly promote the design process in the architecture, engineering, and construction (AEC) industry. With the increasing complexity of designs and the emergence of building information modeling (BIM), which provides a feasible environment, ARC is vital to the design process [4,5] and it has been extensively studied by many researchers. However, there has been no adoption of ARC as part of official compliance processes to date [4], and the extent of ARC use in real projects is difficult to observe, which is not in line with the high expectations in the AEC industry [6,7].

Currently, regulatory rules are mainly written in textual documents for humans to read. Therefore, these textual rules need to be interpreted into a computer-processable format for ARC, i.e., rule interpretation, and it is regarded as the most vital and complex stage in the ARC process [5]. However, most existing ARC systems are based on proprietary and manual hard-coded rule-based mechanisms. These methods, such as the first large effort in building rule checking, the Singapore

CORENET project [8], and the widely used ARC application Solibri Model Checker (SMC) [9], are costly to maintain and inflexible to modify and are often referred to as black-box approaches [10]. To address this gap, many researchers have proposed automated or semi-automated rule interpretation methods for ARC. For semi-automated methods, Hjelseth [11] soft codes regulatory text by annotating it with four mark-up operators – Requirement, Applies, Select, and Exception – which is called RASE methodology and can help AEC experts develop applicable rules without programmers' help. Nawari [10] proposed a framework for neutral data standard industry foundation classes (IFC) which utilizes language-integrated query (LINQ) programming objects to extract, access, and link BIM and regulatory data via ifcXML. For automated methods, Zhang and El-Gohary [12,13] concluded that domain-specific regulatory text is more regular than general nontechnical text and more suitable for natural language processing (NLP); thus, they proposed an automated rule transformation method consisting of (1) information extraction, which recognizes and labels words and phrases in relevant sentences with predefined tags, and (2) information transformation, which transforms the extracted information instances into logic clauses by regular expression rules.

These studies have shown that it is feasible to improve the efficiency of extracting and interpreting rules from regulatory text based on NLP, and they have greatly facilitated rule interpretation for ARC. However, limitations still exist because they cannot achieve both a high level of automation and wide scope of application for rule interpretation. The semi-automated methods still require considerable manual effort toward writing query languages, pseudocodes, marking up regulatory documents, and so forth. The automated methods, however, are based on regular expression pattern matching, which is also a hard-coded manner and has limited application scope. Actually, regular expression has low expressiveness such as the lack of recursion: a regular expression cannot be placed inside another one unless hard code it by hand for each level; this drawback will easily lead to their increasing number of usages which quickly becomes unmaintainable [14]. This may explain why existing automated rule interpretation methods mainly considered simple regulatory sentences and use a small size of dataset for validation. For example, Zhang and El-Gohary's experiments [12,13] limited the extracted regulatory sentences to having at most one instance of compliance checking attribute, comparative relation, requirement quantity value, etc. (e.g., "habitable rooms shall have a net floor area of not less than 70 square feet" presented in [13]), which can be called single-requirement sentences; and their validation for pattern matching-based information transformation used 62 extracted sentences. However, having multiple requirements (multi-requirement) in a regulatory sentence is very common such as in compound and complex sentences (e.g., "habitable rooms shall have a net floor area of not less than 70 square feet and a glazing area of not less than 5 square feet"), but it can hardly be seen in the papers of related studies. It is also worth noting that, the datasets of regulatory text used in all related studies are not open, and this data isolation causing non-reproducibility of research is also a problem to be solved [15].

To address these problems, this research aims at (1) proposing an automated rule interpretation method with wide application scope, which can interpret more complex multi-requirement sentences, and (2) developing and opening a dataset of regulatory sentences with annotation for validation, which has a larger size than existing studies. This method mainly consists of semantic labeling and syntactic parsing by utilizing deep learning, NLP, and context-free grammar (CFG) techniques. The remainder of this paper is organized as follows. Section 2 reviews and classifies the related work and highlights the research gap. Section 3 describes the proposed automated rule interpretation method. Section 4 illustrates and analyzes the results of the experiment and its application. Section 5 discusses the advantages and contributions of this research and notes the limitations. Finally, Section 6 concludes this research.

## 2 Related work

ARC has been extensively studied in recent decades since Fenves investigated the application of decision tables for rule checking in the 1960s [16]. These efforts took various approaches and focused on various ARC purposes or subdomains. For example, Garrett and Fenves proposed a knowledge-based strategy to represent design standards using data items, decision tables, information networks, and organizational systems [17]. Delis and Delis proposed an approach to encode engineering code requirements in a knowledge-based expert system, in which the rules are stored in "if-then" form [18]. Han et al. proposed a hybrid prescriptive/performance-based approach for ARC, which models building code prescriptive statements where there is no indeterminacy and conflict and adopts a performance-based approach if such problems surface [19]. Ding et al. proposed an approach to represent building codes using object-based rules and represent designs using an IFC-based internal model for ARC of accessibility regulations [20].

Currently, the increasing complexity of designs and the widely adopted BIM highlight the significance of ARC in the AEC industry. The rule checking process can be structured into four stages [8]: (1) rule interpretation, (2) building model

preparation, (3) rule execution, and (4) reporting checking results. Among these stages, rule interpretation, which translates written rules into a computer-processable format, is the most vital and complex stage [5].

Since regulatory text is domain-specific and usually written more regularly than general text [13], many studies have explored different automated or semi-automated rule interpretation methods to replace manually hard-coded-based ARC systems. However, limitations still exist in these studies because they cannot achieve both a high level of automation and wide scope of application. This research classifies current rule interpretation methods into three categories: rule annotation, rule formalization, and rule transformation. The first two are semi-automated, and the last is a (fully) automated method.

## 2.1 Rule annotation method

This method aims to capture the metadata of regulatory text and parse them at a shallow level. For example, the RASE methodology [11] soft codes regulatory text by annotating it with four mark-up operators – Requirement, Applicability, Selection, and Exception – which can help AEC experts to develop applicable rules without programmers' help. It also concluded that regulatory text contains a number of checks, where each check contains several requirements, applicabilities, selections, and exceptions, and the method identified a 1:1 relation among the four RASE operators and logical operators (and, or, not) in the software code. Furthermore, Beach et al. [21] expanded the RASE methodology to represent regulatory documents at the block or paragraph level and inline and finally mapped the marked document to semantic web rule language (SWRL) for rule execution. Lau and Law [22] developed a shallow parser to consolidate different formats of regulations, including their hierarchical and referential structures, into extensible mark-up language (XML) to semi-automate the rule translational process. They also developed parse trees capable of tagging regulation provisions with the list of references they contain.

The rule annotation method is interpretable and can semi-automate the rule interpretation process. However, it can process text only at a coarse level of granularity, and it requires considerable human effort. The RASE methodology provides an analysis only at the sentence level but not the word/term level, as does the shallow parser parses text into a marked-up format such as XML. Expanding the RASE tags can extend the analysis but simultaneously increase human annotation effort.

## 2.2 Rule formalization method

This method aims to develop a formalized and generalized representation of regulatory text for rule execution. A common approach with this method is formalizing rule representations into a specific query language, and it is often used with building data formalization. For example, Dimyadi et al. [23] developed a computable regulatory knowledge model (RKM) and then used a high-level domain-specific query language to extract regulatory information for ARC. Nawari [10] proposed a framework for neutral data standard IFC that utilizes LINQ programming objects to extract, access, and link BIM and regulatory data via ifcXML. Yurchyshyna and Zarli [24] presented an ontology-based method for the formalization and application of rules for ARC. They further developed a base of semi-formal accessibility queries by formalizing them as SPARQL queries based on the IFC-based conformance checking ontology.

Another approach of this method is developing a domain-specific language (DSL) to be an intermediate rule representation that is more intuitive or easier than programming languages. Lee et al. [25] implemented a DSL, Building Environment Rule and Analysis (BERA) language, based on the use of IFC as given BIM models and SMC as the IFC engine, and it attempts to deal with BIM models intuitively by providing efficiency in defining, analyzing and checking rules. Park et al. [26] used intermediate pseudocodes to represent regulatory sentences; these pseudocodes have an intuitive naming convention and can be further translated into a computer-executable form. Sydora and Stroulia [27] described a simple and extendable DSL for computationally representing building interior design rules, and they demonstrated the use of the DSL in the automated generation of multiple valid alternative model interior designs. Preidel and Borrmann [28] introduced a new method for ARC using a flow-based, visual programming language (VPL), such as Dynamo for Autodesk Revit. Häußler et al. [29] examined the feasibility of implementing railway design rules using VPLs, including business process model and notation (BPMN) and decision model and notation (DMN), for code compliance checking. They also stated that the better interpretability of VPL improves the sufficiency of its transparency and visibility.

Among the three categories of rule interpretation methods, the rule formalization method has been studied the most. The formalized representation of rules can be independently developed or managed by AEC experts without programmers' help, and it has wide application scope. However, human effort is also needed in this method because it only reduces the burden of writing codes but does not eliminate it. Formalized building data (e.g., ifcXML, ifcOWL) are sometimes required

when using this method, which also affects automation.

## 2.3 Rule transformation method

This method aims to achieve fully automated rule interpretation, i.e., translating regulatory text into computer-processable rules automatically without manual intervention. An analysis shows that domain-specific regulatory text is more suitable for automated NLP than general nontechnical text (e.g., news articles, general websites) because regulatory text has fewer homonym conflicts, exhibits fewer coreference resolution problems, and can facilitate ontology development to capture domain knowledge [13]. Therefore, Zhang and El-Gohary [12,13] proposed an automated rule transformation method consisting of (1) information extraction, which recognizes the words and phrases in the relevant sentences and labels them with predefined information tags via part-of-speech (POS) tagging, gazetteer compiling, etc., and (2) information transformation, which transforms the extracted information instances into logic clauses by regular expression-based mapping rules and conflict resolution rules. Their framework is summarized in Figure 1. Furthermore, by integrating information extraction and transformation into a unified system, they achieved fully automated rule checking [3]. Zhou and El-Gohary [30] proposed an ontology-based information extraction method to support fully automated building energy compliance checking. In their studies, the pattern-matching method was enhanced by ontology, preprocessing, and many other domain-specific techniques, which focused on the building energy domain and improved the accuracy. Li et al. [31] presented a framework to automate utility compliance checking by integrating NLP and spatial reasoning, where the pattern-matching-based NLP algorithm is used to translate the textual descriptions of spatial configurations into computer-processable spatial rules. Xu and Cai [32] proposed a semantic frame-based information extraction method focusing on domain and lexical semantics to support ARC, and the method is characterized by the enrichment of lexical semantic frames and mapping with the domain semantic framework. Later, Xu and Cai [33] adopted an ontology and rule-based NLP framework to automate the interpretation of textual regulations on underground utility infrastructure, where pattern-matching rules were encoded for information extraction, and the extracted information elements were mapped to their semantic correspondences via ontologies and finally transformed into deontic logic (DL) clauses.

Currently, all existing rule transformation methods use a regular expression-based pattern-matching approach. However, these methods have limited application scope and are difficult to be expanded because (1) regular expression can hardly match long and complex patterns because of its low expressiveness such as lack of recursion (e.g., a regular expression cannot be placed inside another one unless hard code it by hand for each level, which is hard to maintain), and (2) regular expressions cannot be used to match a wide range of varied patterns because their increasing number of usages will quickly become unmaintainable [14]. These limitations suggest why existing rule transformation methods focus on simple single-requirement sentences (e.g., limiting the extracted sentences to having only one attribute and one requirement element in [13]), and use a small number of regulatory sentences as the dataset for validation (e.g., 62 sentences in [12]).

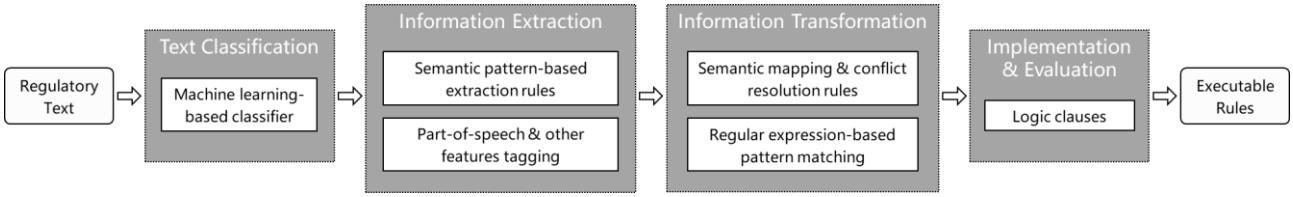



Figure 1. Framework of an automated rule transformation method (adapted from [12]).

## 2.4 Summary

According to the above review, it can be concluded that existing rule interpretation methods cannot achieve both a high level of automation and wide scope of application: the first two semi-automated methods require manual effort, while the last automated method mainly focuses on simple single-requirement sentences and is difficult to interpret complex multi-requirement sentences. Therefore, the aim of this research is addressing this research gap by proposing an automated rule interpretation method with wide application scope, which can interpret multi-requirement sentences. Table 1 summarizes and compares existing methods and our proposed method.

Table 1. Comparison of existing rule interpretation methods and our proposed method.

| Method | Level of automation | Simple single-requirement sentences | Complex multi-requirement sentences |
|---|---|---|---|
| Rule annotation | Semi-automated | Support | Support |

| | | | |
|---|---|---|---|
| [11,21,22] | | | |
| Rule formalization [10,23-29] | Semi-automated | Support | Support |
| Rule transformation [3,12,13,30-33] | Automated | Support | Not or hardly support |
| **Our method** | **Automated** | **Support** | **Support** |

# 3 Methodology and framework

To propose an automated rule interpretation method with wide application scope, we redesign the framework of automated rule transformation methods by introducing syntactic parsing based on CFGs, to replace the widely used regular expression-based pattern matching. CFG has much more powerful expressiveness (e.g., represents recursively-defined concepts) than regular expressions and is commonly used to describe and parse the structure of programming languages, and can significantly expand the method's application scope. Based on this, we also propose a syntax tree and semantic elements to represent the structure and concepts in regulatory text and BIM models. Figure 2 shows a framework of the proposed automated rule interpretation method, which has the following four steps.

1. Preprocessing. This step selects relevant sentences from regulatory text and preprocesses these sentences by such as sentence splitting to facilitate later work.
2. Semantic labeling. This step extracts semantic elements by assigning words or phrases in a sentence with semantic labels, indicating different semantic roles and relations. A deep neural network (DNN) model with the transfer learning technique is utilized for the labeling via the begin-inside-outside (BIO) tagging format. The result is validated by the F1-score of assigning each label in sentences.
3. Syntactic parsing. This step mainly uses CFGs to parse a labeled sentence into a syntax tree structure that represents hierarchies and relations of the semantic elements. The syntax tree is language-independent, from which computable checking rules can be generated. The result is validated by the accuracy of parsing labeled sentences into the syntax trees.
4. Rule generation. This step converts the tree representation into language-specific rules for execution, such as if-then statements, horn clauses, and programming language.

Note that the preprocessing method is not presented in detail in this research because it is relatively simple and well developed, as has been detailly discussed by many other studies [34]. The rule generation, which is language-specific, is illustrated in Section 4.4, where an actual rule checking case in a Revit model is presented.

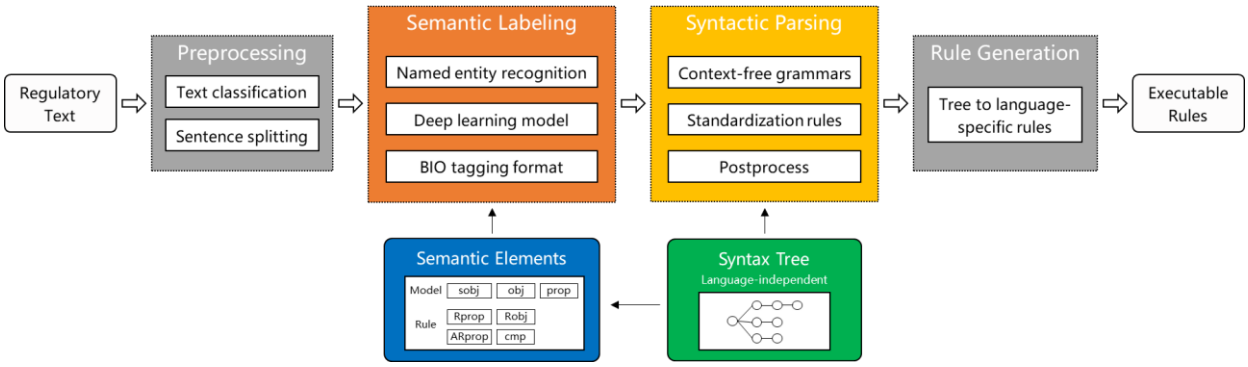



Figure 2. Framework of the proposed automated rule interpretation method integrating NLP and CFG.

## 3.1 Definition of the semantic elements and syntax tree

Regulatory text mainly contains two types of requirements or rules [13]: (1) quantitative requirement, which defines the relationship between an attribute/property of a certain building object and a specific quantity value/range, such as "the thickness of the protection layer shall not be less than 10 mm"; and (2) existential requirement, which requires the existence

of a certain building object, such as “the roof shall have a protection layer”. It can be concluded that each rule expressed by regulatory text consists of three parts: (1) the building or BIM objects to be checked, (2) the required constraints or conditions for checking, and (3) the comparative or existential relationship between them. A BIM model can be described by objects that possess many properties [35,36], where a property can also be an object. Therefore, rule checking can be regarded as a process that first locates and identifies objects in the hierarchical structure of a BIM model and then checks the objects against corresponding requirements. This process can be represented by a tree structure where some nodes representing checked objects are linked to requirements, where a requirement contains a comparative or existential relationship and a required constraint or condition.

This tree structure for rule representation is similar to the syntax tree in a programming language. Thus, the tree structure of a regulatory sentence can be obtained by first assigning several types of labels to words or phrases in the sentence to indicate their semantic meanings and then using grammar to parse the labeled sentence into the tree structure like a syntax tree. To achieve this, we define several semantic elements and their syntax in the following steps:

1. A simple and basic rule (e.g., “thickness of the protection layer shall not be less than 10 mm” as shown in Figure 3a) contains four elements: checked object (protection layer), checked object’s property (thickness), comparative relation (not be less than), and requirement of the property (10 mm). These are defined as four semantic elements correspondingly: “obj”, “prop”, “cmp”, and “Rprop”.
2. A rule can contain more hierarchy levels, as shown in Figure 3b. Thus, a semantic element “sobj” is defined to represent the parent object (roof) of the “obj” (protection layer). The “sobj” can be used multiple times to recursively represent object hierarchies (e.g., one “sobj” is the parent object of another “sobj”), so as the “prop” recursively represent a property of another “prop”. Besides, the “Rprop” can also have a parent object and a “Robj” element is defined for representing it.
3. A requirement of a rule can be a condition like the “If” clause in an If-Then rule, as shown in Figure 3c. Thus, a semantic element “ARprop” is defined to represent the applicability condition (B1 level) of a corresponding “prop” (insulation material) to distinguish it from the meaning of “Rprop”. The “Robj” can also be used as the parent object of “ARprop”.

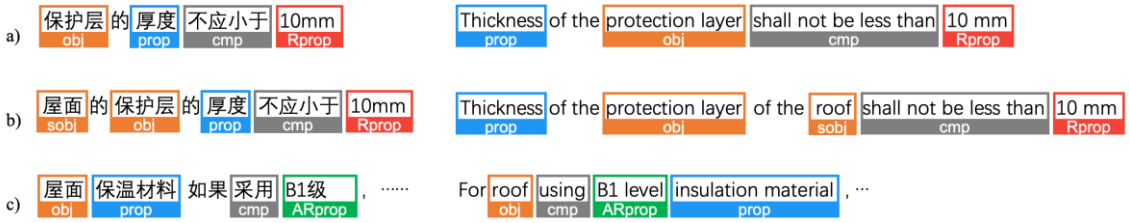



Figure 3. Example of semantic elements.

Therefore, this research proposes seven semantic elements to represent different semantic roles and relations of words or phrases in a regulatory sentence, as defined in Table 2. Meanwhile, a syntax tree to represent the hierarchical structure and relations of semantic elements is proposed, namely, rule check tree (RCTree): a tree structure representing object hierarchies and relations where at most one tree node has multiple child nodes, and at least one node is linked to a requirement. A Node in the RCTree is a semantic element, which is words or phrases with a semantic label. Figure 4 illustrates the RCTree in different representation formats. Figure 5 shows an illustrative example of semantic labeling and syntactic parsing, in which the regulatory sentence is selected from a Chinese building code for fire protection.

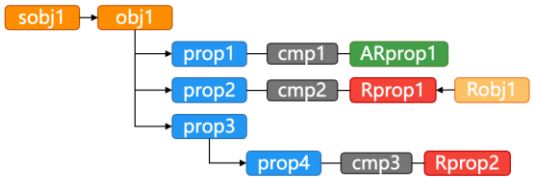
As shown in Table 2, the “sobj”, “obj”, and “prop” are used to identify checked elements in a BIM model, where both “sobj” and “prop” can locate at multiple hierarchies but “obj” is in a fixed hierarchy and can have multiple child nodes. The “cmp” is the comparative/existential relation between the checked element “prop” and the requirement “Rprop” or “ARprop”. The “Rprop” and “ARprop” are used to represent two types of requirements according to the If-Then format: “Rprop” is a constraint applied to a “prop” that shall be satisfied, and “ARprop” is an applicability condition applied to a “prop” that if satisfied then other constraints will be checked. A “Rprop” or “ARprop” could be a reference to another element, and the “Robj” is defined as the refereed element (e.g., the “B” in “A shall be greater than 10 times of B”).

In Figure 4, an example RCTree is presented in graph-based representation, text-based representation, and the equivalent if-then format for illustration. The graph-based format (Figure 4a) is designed for human-readable, where the hierarchy of RCTree nodes is represented by arrows from left to right according to high to low levels. The text-based format (Figure 4b), like the tree structure of a file system, is used for algorithm output and result storage. In Figure 4b, the first

line represents “sobj-sobj” or “sobj-obj” relations from left to right. In the following lines, each line represents an “obj-prop” or “prop-prop” relation and the requirement applied to the “prop” if any: it starts from a “|”, and the number of followed “-” denotes the hierarchy level of the “prop”; the occurrence of “?” after “-” indicates that the requirement is applicability; otherwise, it is a constraint. In the proposed RCTree, the default Boolean relation between requirements is “AND”, i.e., if all applicabilities are satisfied, all constraints shall be satisfied. The “OR” relation between requirements is addressed by: (1) element merging, which merges consecutive OR-related elements of the same label into one union element; (2) requirement association, which explicitly indicates a requirement is OR-related to the previous requirement, denoted by changing the last leading “-” to “+” (e.g., “|--” to “|-+”) in the text-based format; and (3) sentence splitting, which splits a sentence into multiple sentences and uses multiple RCTrees to represent.

Table 2. The proposed seven semantic elements.

| Name | Definition |
|---|---|
| prop | The element can be linked to a requirement. A prop can be a child node of an obj or another prop. E.g., in Figure 4, prop1/2/3 are properties of obj1, and prop4 is a property of prop3. |
| obj | The parent node of one or more props. An RCTree has only one obj. E.g., obj1 in Figure 4. |
| sobj | The parent node of the obj or another sobj. E.g., sobj1 in Figure 4. |
| cmp | The comparative/existential relation between a prop and a requirement. E.g., “greater than”, “has”. |
| Rprop | The constraint requirement linked to a prop that shall be satisfied. |
| ARprop | The applicability condition linked to a prop that if satisfied then other constraints will be checked. |
| Robj | The parent node of a Rprop or ARprop, usually optional. |




a) Graph-based representation

```
[sobj1]-[obj1]
|-?[prop1] cmp1 [ARprop1]
|-[prop2] cmp2 [Rprop1]-[Robj1]
|-[prop3]
|--[prop4] cmp3 [Rprop2]
```

b) Text-based representation

```
If   (sobj1.obj1.prop1 = ARprop1)
Then (sobj1.obj1.prop2 = Robj1.Rprop1) AND
     (sobj1.obj1.prop3.prop4 = Rprop2)
```

c) If-then representation (assume all cmp nodes are “=”)

Figure 4. Illustration of the proposed rule check tree (RCTree).

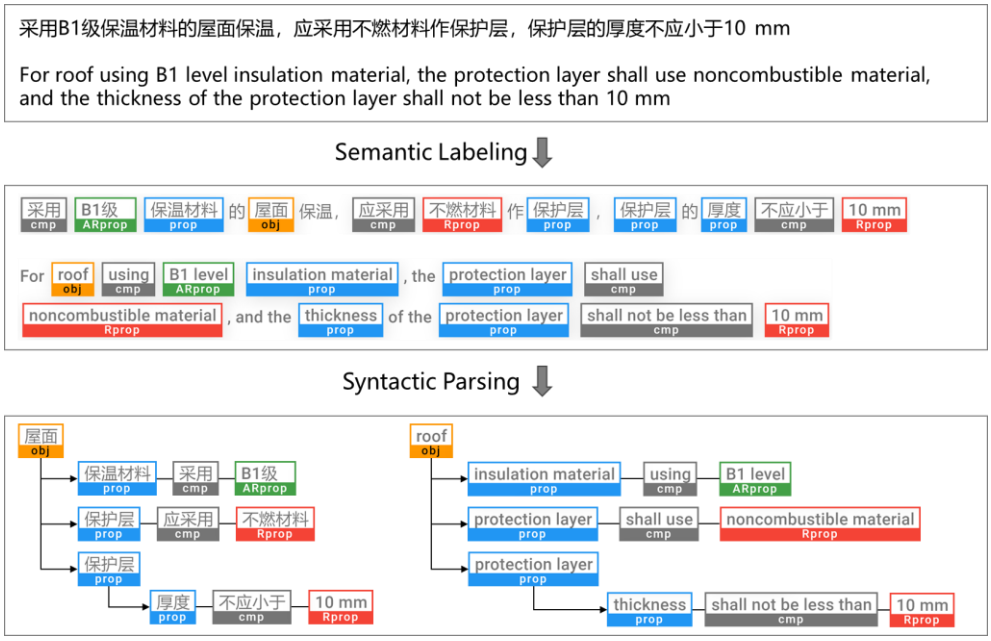



Figure 5. Illustrative example of semantic labeling and syntactic parsing (the English sentence is translated from Chinese).

## 3.2 Semantic labeling

Semantic labeling is the process of assigning semantic labels to words or phrases in a sentence, where a labeled word or phrase is called a semantic element. The labeling process (e.g., POS tagging) has been widely involved in the information extraction stage of existing studies for ARC. Common methods include gazetteer lookup, the hidden Markov model (HMM), conditional random fields (CRF), and so on. However, one of the best-known weaknesses of these methods is the lack of semantic awareness; i.e., they have difficulty recognizing semantic information. In addition, few researchers have studied machine or deep learning-based information extraction methods for ARC [37].

In recent years, DNN models (e.g., recurrent neural networks and transformers) have been introduced for sequence labeling [38] and have shown huge improvements over traditional methods. Meanwhile, these DNN models promote a new methodology for NLP, that is, transfer learning. For example, the state-of-the-art DNN model BERT [39] can be first pretrained in a large general corpus and then applied to a specific domain by training it with a small quantity of supervised data. This transfer learning technique can greatly reduce the amount of domain-specific training data and improve performance.

Thus, to enhance the ability of semantic recognition, this research uses a deep learning model with the transfer learning technique for semantic labeling. BIO tagging format is adopted for the labeling, which annotates the belonging of a word to a semantic element by labeling whether the word is at the beginning, inside, or outside of that element, as explained in Figure 6. An illustration of the labeling process by a DNN model (i.e., BERT) is shown in Figure 7. The DNN model first embeds all tokens of the input sentence and then uses a bidirectional transformer to encode the input embeddings to contextual representations. Finally, the representations are transformed into the BIO format for output.

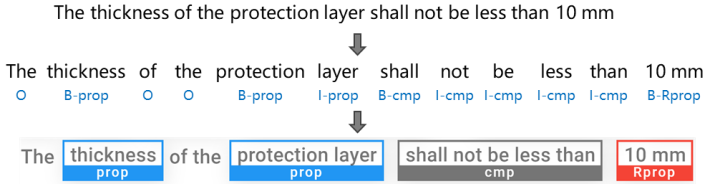



Figure 6. Illustration of the BIO format labeling.

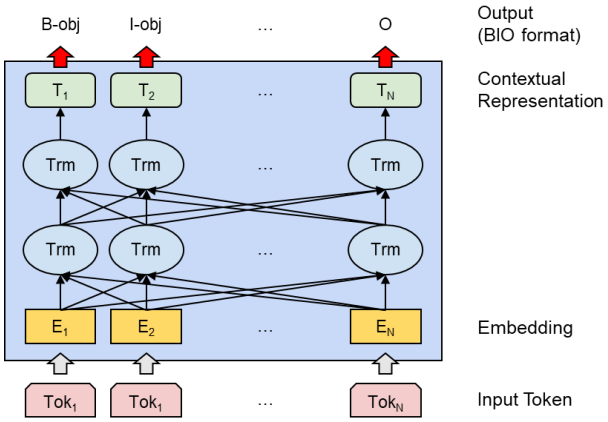



Figure 7. Illustration of the labeling by a DNN model.

## 3.3 Syntactic parsing

In this research, syntactic parsing is the process of analyzing the structure of a labeled sentence and parsing it into the RCTree. In programming languages, a parser is a component of a compiler that parses the source code of a programming language to create an internal representation (e.g., parsing tree). In the case of domain-specific regulatory text, it is more regular (e.g., less ambiguity and fewer homonym conflicts) than general nontechnical text and thus is more suitable for NLP [13]. This nature of regulatory text brings the possibility of parsing it with semantic labels like a domain-specific language (DSL).

Programming languages tend to be specified in terms of CFG. CFG is a formal grammar that could be used to derive the sentences of a language, in which every production rule is of the form $A \rightarrow \alpha$, where $A$ is a single nonterminal symbol and $\alpha$ is a sequence of terminals (symbols that cannot be further broken down). There are two commonly used grammars according to Chomsky hierarchy: CFG (type-2) and regular grammar (type-3) [40]. CFG has higher expressiveness since it can represent recursively defined concepts and can be parsed by an LL or LR parser, whereas regular grammar generates a subset language of CFG (e.g., lack of recursion) and can be obtained by simpler regular expressions.

This research treats a regulatory sentence with semantic labels as a DSL and uses CFGs to parse it to obtain the RCTree. Figure 8 illustrates the proposed syntactic parsing method, which consists of three processes: sentence standardization, CFG-based parsing, and RCTree conversion. Note that "[word/tag]" is a text-based notation to represent a semantic element, and "cmp" elements are simplified to symbols in the text-based RCTree representation in Figure 8d.

### 3.3.1 Sentence standardization

This process extracts "sobj" and "obj" elements in a sentence, adds them to the RCTree, and standardizes the remaining elements. First, all "sobj" elements are extracted, and their occurrence order in the sentence is regarded as the hierarchies from high to low. Note that this decision can be further optimized by utilizing an ontology (e.g., if a "sobj" element is the property or child node of another "sobj" that occurred latter, which can be recognized by an ontology, their hierarchies can be adjusted to make the latter one higher). Second, all "obj" elements are extracted, and if there are multiple "obj", their union is considered as one "obj". As discussed in Section 3.1, after preprocessing such as sentence splitting, it is assumed that the input sentence has only one "obj". Finally, the order of the remaining elements in the sentence is standardized. For example, in the occurrence of "A of B", the order of words can be switched to "B's A" for consistency. Similarly, in Figure 8, "[using/cmp] [B1 or B2 level/ARprop] [insulation material/prop]" is standardized into "[insulation material/prop] [using/cmp] [B1 or B2 level/ARprop]". To achieve this, regular expression-based rules can be used, but the number of rules should be small, and the meanings should be explicit to retain the maintainability of the method. In this research, four types of sentence standardization rules are developed, and their descriptions are provided in Table 3. It is worth noting that sentence standardization may be language-specific. In this paper, Chinese building codes are considered, and the developed sentence standardization rules may need to be modified when applied to English sentences.

### 3.3.2 CFG-based parsing

This is the core process of syntactic parsing, and it uses CFGs as a parser to parse standard labeled sentences. Figure 9 shows the parsing CFGs in the form of ANTLR4 grammar [41]. In Figure 9, uppercase words represent lexical rules, and the first five ones represent the corresponding five semantic elements as defined in Table 2 while "sobj" and "obj" have

been extracted and removed after sentence standardization. Note that content of green font in Figure 9 represents comments.

Lowercase words in Figure 9 represent grammatical rules that parse semantic elements in a sentence. According to the proposed RCTree and semantic elements, this research defines P-R (Prop-Req) pattern as the basic semantic unit, and the grammatical rules are based on the recognition of P-R patterns and their recursive occurrences. The P-R pattern represents elements with corresponding requirements, where "P" means the "prop" element and "R" means "CMP? ROBJ? (RPROP|ARPROP)" pattern as the "req" rule shown in Figure 9 ("?" means optional, "|" means OR-relation). For instance, in Figure 8, "[protection layer/prop] [shall use/cmp] [noncombustible material/Rprop]" is a basic P-R pattern; and "[protection layer/prop] [thickness/prop] [shall not be less than/cmp] [10 mm/Rprop]" is a $P_1$-$P_2$-R pattern, where $P_2$ has the requirement of R and is the child node (property) of $P_1$. In Figure 9, "pr" is defined to recognize balanced P-R patterns that have the same number of "P" and "R", and "prs" recognizes all possible and rational P-R patterns based on "pr". For example, "The insulation layer of a pipeline shall use noncombustible materials with a melting point greater than 1000 ℃" will be labeled and standardized into "[pipeline/obj] [insulation layer/prop] [melting point/prop] [greater than/cmp] [1000 ℃/Rprop] [shall use/cmp] [noncombustible material/Rprop]", and it is a "pr" pattern consists of $P_1$-$P_2$-$R_2$-$R_1$, where $P_2$ is linked to $R_2$ and $P_1$ is linked to $R_1$. Finally, the "tree" in Figure 9 is defined to combine all the recognized "prs" into a single parsing tree, as shown in Figure 8c.

### 3.3.3 RCTree conversion

This process takes a parsing tree as the input and converts it into the RCTree. First, it performs postprocessing work, such as rectifying the parsing result and removing duplications. Then, it converts a parsing tree into an RCTree by recursively visiting subtrees of the parsing tree and creating corresponding RCTree elements, as shown in Figure 8d.

Table 3. Developed sentence standardization rules.

| Rule name | Description | Example |
|---|---|---|
| Label merge | Merges elements with the same label into one in some cases | "[A/obj] and [B/obj]" to "[A\|B/obj]" |
| Label order change | Changes the order of consecutive elements in some cases; language-specific | "[A/prop] of [B/prop]" to "[B/prop]'s [A/prop]" |
| Subsentence order change | Changes the order of two subsentences (e.g., separated by comma or colon) in some cases | "A shall meet the requirements: when B, C" to "A shall meet the requirements: C, when B" |
| Default element completion | Completes the missing elements in a sentence | "A shall be greater than B and less than C" to "A shall be greater than B, and A shall be less than C" |

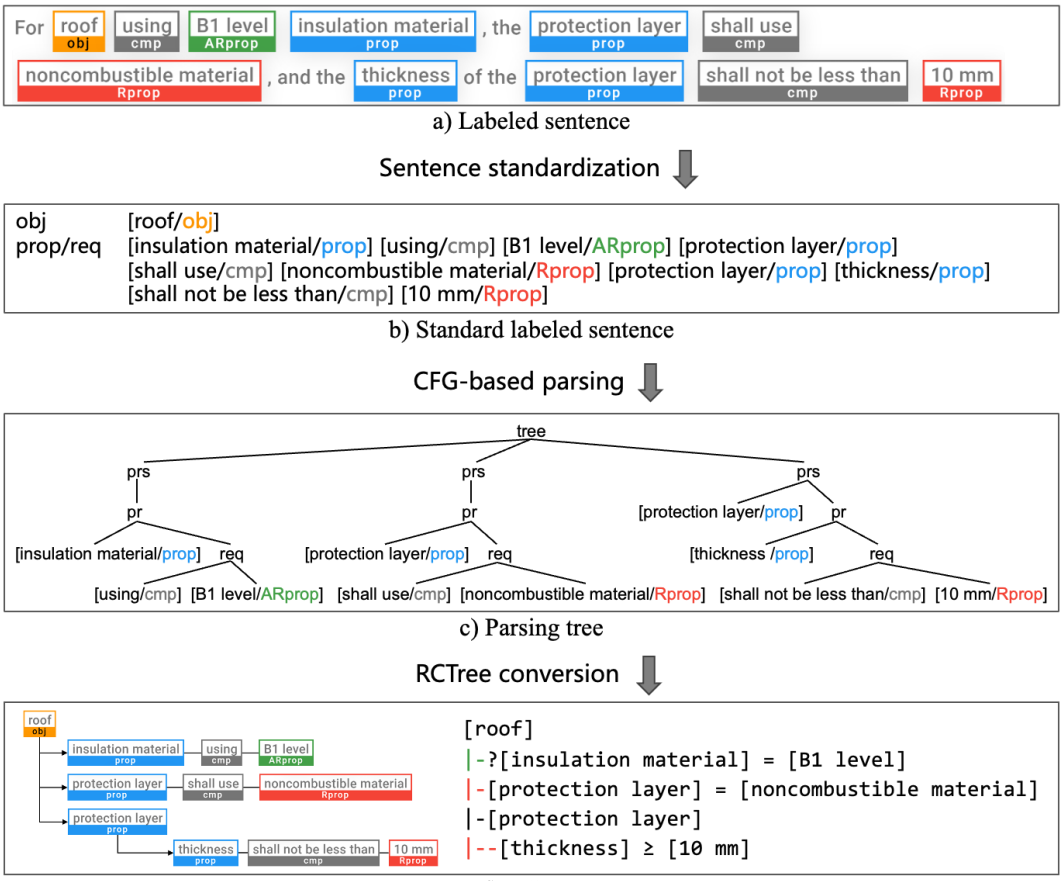



Figure 8. Workflow of the syntactic parsing method.

```
// Grammatical rules
tree:   prs+;
prs:    pr            // All valid P-R patterns
    |   PROP+ pr
    |   pr? req
;
pr:     PROP pr+ req  // Balanced P-R patterns
    |   PROP req
;
req:    CMP? ROBJ? (RPROP|ARPROP); // Requirement

// Lexical rules
PROP:   '[' CHAR*? '/prop]';
CMP:    '[' CHAR*? '/cmp]';
ROBJ:   '[' CHAR*? '/Robj]';
RPROP:  '[' CHAR*? '/Rprop]';
ARPROP: '[' CHAR*? '/ARprop]';
OTHERS: CHAR+? -> skip;
CHAR:   ~[[\r\n];
NEWLINE:'\r'?'\n' -> skip;
```

Figure 9. CFGs for syntactic parsing.

## 4 Results

This section describes the experiments and applications for validating the proposed method. First, a dataset of regulatory sentences and its gold standard for semantic labeling is developed, and the sentences are classified by complexity for measuring the application scope of rule interpretation methods. Second, a deep learning method is implemented for semantic labeling and validated by the gold standard. Third, all labeled regulatory sentences in the gold standard are parsed into RCTrees and manually evaluated for correctness. These experiments are used to validate performance of the proposed

method and demonstrate its high accuracy and wide application scope.

An actual rule checking case is also presented, where a specific rule generation method is proposed to convert RCTrees of several regulatory sentences into executable rules and check them in a Revit BIM model. This is used to demonstrate the applicability of the proposed method for interpreting and checking rules in a real case.

## 4.1 Dataset development

In this research, Chinese building codes for fire protection are selected for validation. The proposed method is applied in Chinese text and then the result is translated into English. First, regulatory text in building codes is collected and split into sentences by semicolons, periods, and newlines. Next, the sentences are filtered to select those have quantitative requirements (e.g., have keywords of "no less than" or "greater than") since they are mostly used in building codes for describing requirements and it is a reasonable choice adopted by many researches [13]. Then, these sentences are manually reviewed by two of the authors to ensure the applicability for rule checking in BIM while removing unsatisfactory sentences. Finally, 611 regulatory sentences distributed in 46 building codes are selected, forming the dataset.

To provide criteria for validating semantic labeling, these 611 sentences are manually annotated by two of the authors, and the result is regarded as the gold standard. In the annotation, approximately 15% of the sentences were first labeled by both authors to reach a consensus. The remaining parts were separately labeled, and the labeling results were cross-validated by the two authors. Finally, the gold standard was developed, and the dataset including 611 sentences has 4336 semantic elements. An open repository containing the dataset with annotation was established on GitHub at https://github.com/Zhou-Yucheng/auto-rule-transform. Note that it also contains codes used in the following semantic labeling and syntactic parsing steps.

To measure and compare the application scope of rule interpretation methods, regulatory sentences in the dataset are classified into two categories according to complexity:

- Single-requirement sentence. A sentence (after standardization) that has only one requirement and thus has at most one instance for all of the following labels: "prop", "cmp", and "Rprop".
- Multi-requirement sentence. A sentence (after standardization) that has at least two requirements and thus has at least two instances of any of the following labels: "prop", "cmp", and "Rprop" (i.e., a sentence that is not a single-requirement sentence).

Having multiple requirements in a sentence is common but significantly increases the complexity of parsing. A single-requirement sentence has a simple flattened structure; however, the structure of a multi-requirement sentence has branches. An apparent difference is that a single-requirement sentence has only one subtree of the "obj" node in the transformed RCTree, whereas a multi-requirement sentence has at least two subtrees. In our dataset of 611 regulatory sentences, 224 (36.7%) and 387 (63.3%) of them are single- and multi-requirement sentences, respectively, which suggests that multi-requirement sentences are more common.

## 4.2 Semantic labeling result

Semantic labeling is implemented in Python language with the PyTorch deep learning package, and BERT [39] for Chinese is chosen as the pretrained DNN model, which is provided by "bert-base-chinese" in the transformers package. The dataset is randomly split into training and validation datasets at a 0.8:0.2 ratio, where the training dataset is used to train and update the DNN model, and the validation dataset is used to test the model's performance. To measure the result, the model predictions are compared with those from the gold standard, and the precision ($P$), recall ($R$), and F1-score ($F_1$) are calculated for each semantic label:

$$P = N_{correct}/N_{labeled} \quad (1)$$

$$R = N_{correct}/N_{true} \quad (2)$$

$$F_1 = 2PR/(P+R) \quad (3)$$

where $N_{\{correct,labeled,true\}}$ denotes the number of {model correctly labeled, model labeled, true} elements. Note that the measurement is based on the BIO tags, and B- and I-tagged elements are summed for each semantic label (e.g., both the B-obj and I-obj tagged elements are regarded as the "obj" element). Finally, the weighted (micro) average F1-score is calculated to represent the overall performance of the model ($n_i$ denotes the number of the $i$-th semantic labels):

$$\text{Weighted } F_1 = \left(\sum_i n_i F_{1,i}\right)/\sum_i n_i \quad (4)$$

In the model training, two key hyper-parameters are tested and optimized: learning rate (LR, for how fast a model adjusts and updates its parameters) and batch size (BS, for how many samples are used in one training iteration) [42], the others use the recommended default settings in [39]. Since [39] suggested using LR and BS of about 3e-5 and 32 for training BERT, we test LR on 1e-5/3e-5/5e-5/7e-5/9e-5 and BS on 4/8/16/32/64. Results show that using LR=7e-5 and BS=16 in our case achieves the best performance, and the model training can reach convergence in 30 epochs consuming 5 minutes. The experiment was performed on a GPU with 16 GB memory and 65 TFLOPS of FP16 performance.

The performance of semantic labeling on the validation dataset is shown in Table 4. Note that the "O" tag is used to label words or phrases not belonging to any semantic label. The DNN model achieves 86.2%, 86.3%, and 86.2% overall precision, recall, and F1-score on the validation dataset, respectively. This is a promising result of semantic labeling as a proof-of-concept in an early stage because it can recognize different semantic labels in long and complex sentences with relatively high accuracy. The confusion matrix of the semantic labeling is shown in Figure 10. It can be seen that "sobj" is likely to be misclassified as "obj" label, and "ARprop" and "prop" are also likely to be misclassified.

The analysis of the varying and some low accuracies suggests two main reasons. First, there exist difficulties in differentiating labels with close semantic meanings. For example, "sobj" and "obj" have similar usages, and a word labeled "obj" could be labeled "sobj" in another sentence. Second, there is ambiguity when representing a concept using semantic labels. For instance, "vertical storage tank" can be regarded as a single "obj" but can also be "[storage tank/obj]" and "[vertical/ARprop]". To address these problems, concepts and relations should be clearly defined, and a promising approach to achieve this is utilizing an ontology, which can be used to assign or adjust semantic labels. In addition, the amount of training data is relatively small in this research. Therefore, the semantic labeling process has great potential to be substantially improved in the future.

Table 4. Semantic labeling result on the validation dataset.

| Label | Number | Precision | Recall | F1-score |
|---|---|---|---|---|
| sobj | 153 | 82.5% | 64.0% | 72.1% |
| obj | 727 | 81.4% | 83.8% | 82.6% |
| cmp | 641 | 93.9% | 96.3% | 95.1% |
| prop | 859 | 89.1% | 86.4% | 87.7% |
| Rprop | 687 | 84.9% | 96.1% | 90.1% |
| ARprop | 360 | 80.9% | 78.1% | 79.4% |
| Robj | 170 | 83.2% | 90.0% | 86.4% |
| O | 978 | 90.2% | 80.7% | 85.2% |
| Total | 4575 | 86.2% | 86.3% | 86.2% |

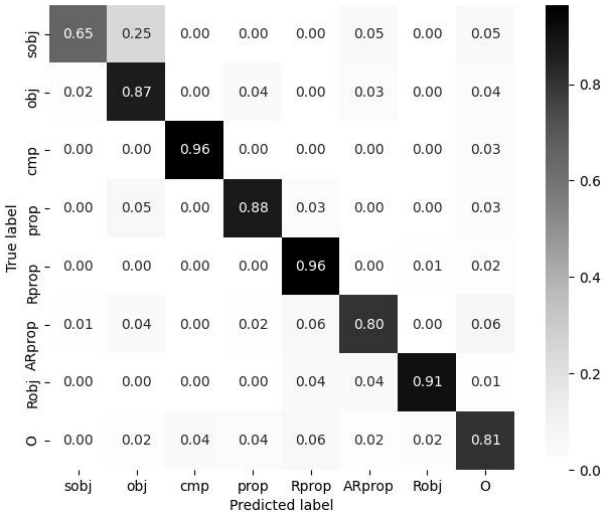



Figure 10. Confusion matrix of semantic labeling result on the validation dataset.

## 4.3 Syntactic parsing result

Syntactic parsing is implemented in Python, and the ANTLR4 package which provides an efficient LL parser for CFG [43] is utilized. In the experiment, the method takes each gold standard labeled sentence as input and outputs the corresponding RCTree. The result is manually evaluated at two levels. (1) Sentence level: whether the RCTree fully correctly expresses the meaning of the original sentence. (2) Element level: for each element in the RCTree, whether the element is generated and located correctly (note that if the RCTree is correct at the sentence level, all elements are correct). The correctness at the sentence level indicates a perfect parsing result that can be used in downstream tasks such as rule execution, whereas the correctness at the element level shows the accuracy for each semantic element and is adopted by many existing studies. The experiment is performed on a 4.6 GHz CPU and consumes 6.9 seconds in total (0.011 s/sentence on average). All result output by the method with manual evaluation is published at https://github.com/Zhou-Yucheng/auto-rule-transform/blob/main/src/logs/ruleparse-eval.log.

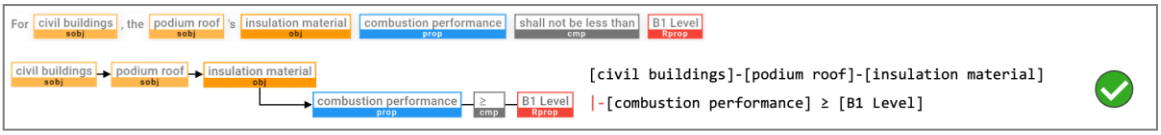
Figure 11 shows five example parsing results, including one single-requirement and four multi-requirement sentences, where the English sentences are translated from Chinese results and the order of words is preserved. For each sentence, its graph-based and text-based representations of the parsed RCTree are given below, and the latter is the direct output of the algorithm. In Figure 11, the single-requirement sentence and the first three multi-requirement sentences are correctly parsed (at the sentence level), but the last multi-requirement sentence is not. In the parsed RCTrees, all “cmp” elements are simplified to symbols (e.g., “=”, “>”); some omitted properties are completed by the method (e.g., “Type” is the default “prop” if not specified); and the unified property merged from multiple properties is connected by “|”.

In Figure 11e, the parsed RCTree is incorrect because the “evacuation corridor” is not applied to “minimum illumination at floor level”, and the correct RCTree is given below. This error occurs because the occurrence orders of semantic elements are not regular, so the meaning is misunderstood by the method. One way to eliminate this error is by adding more sentence standardization rules or CFGs to enhance the parsing ability of the method. Another way to resolve this issue quickly is rewriting the original sentences to a more regular form. In this case, if we change the sentence to “for evacuation corridor, emergency lighting's minimum illumination at floor level shall not be less than 5.0 lx” (i.e., move the parenthesis “for evacuation corridor” to the beginning of the sentence), the method will yield the correct result. As we discussed above, our parsing method is based on the recognition of P-R patterns, and it is evident that the “evacuation corridor” in the parenthesis breaks the connection between “minimum illumination at floor level” and “5.0 lx”. Therefore, we can perform this intuitive modification as a quick fix, which is very useful in practical applications.

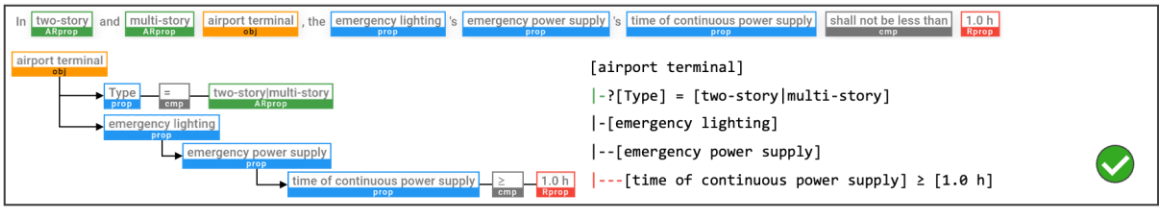
Table 5 shows a comparison of the parsing performances of our proposed method and two state-of-the-art methods at the element level. In this table, Regex-E is a rule transformation method that uses a regular expression-based pattern-matching approach with six essential semantic information tags, and Regex-ES is an enhancement of Regex-E by utilizing more secondary information tags and achieves state-of-the-art performance [12]. The experiment for Regex-E and Regex-ES is performed based on 62 sentences from chapter 19 of International Building Code (IBC) 2009 [12], and all of the sentences are quantitative requirements and extracted in [13], where each of them is a single-requirement sentence as discussed above. It is worth noting that, by using approximately twice as many elements to describe the patterns in a sentence, the enhancement of Regex-ES from Regex-E increases the complexity and specificity of the patterns [12], which may decrease its generalisability. The datasets used in these studies have not been published.

The result in Table 5 demonstrates our method outperforms the state-of-the-art method: for single-requirement sentences, our method achieves 99.57% element-level parsing accuracy, which is higher than the state-of-the-art method Regex-ES; more importantly, for multi-requirement sentences, which are more complex and common but not suitable to be interpreted by pattern matching-based methods and not considered by existing studies, our method achieves high performance of 95.26% parsing accuracy. This result suggests that our method can achieve a high level of automation and wide scope of application because: (1) this method can be applied to both single- and multi-requirement sentences with high accuracy; (2) the dataset used in this research is larger than those used in most existing studies (e.g., tens of sentences used for validation in [12,31]); and (3) the method uses a small number of labels (tags) to represent different elements in sentences, suggesting less specificity and higher generalisability.

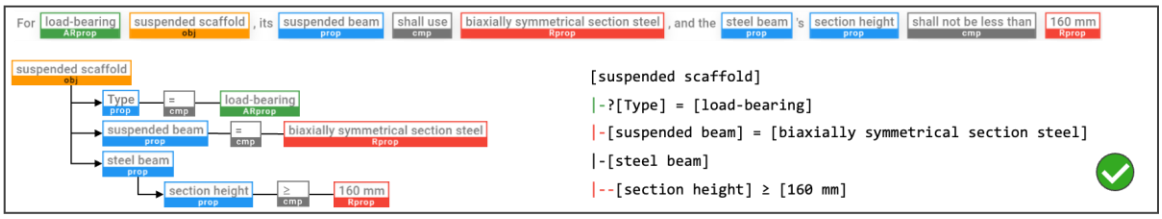
Table 6 shows the parsing performance of our method at the sentence level. It achieves 99.6% and 91.0% parsing accuracies for single- and multi-requirement sentences, respectively, and a 94.1% overall parsing accuracy. This research suggests that the sentence-level accuracy is more suitable to represent the parsing performance because this stricter criterion can better indicate the applicability for supporting downstream task rule execution in ARC. In the entire rule interpretation process, the total correctness is related to the sentence-level accuracies. Therefore, this result of high sentence-level parsing accuracies also demonstrates the high performance and applicability of our method.



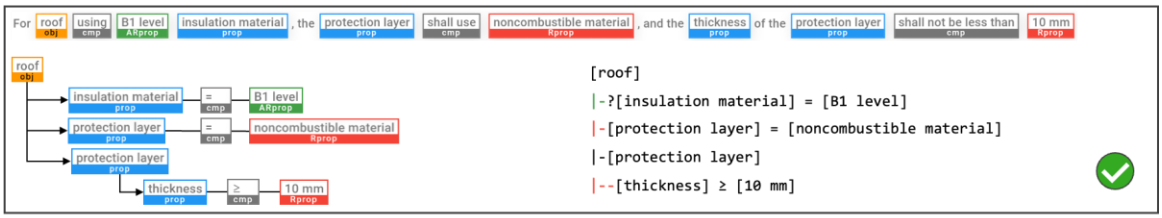

a) Single-requirement sentence, correctly parsed



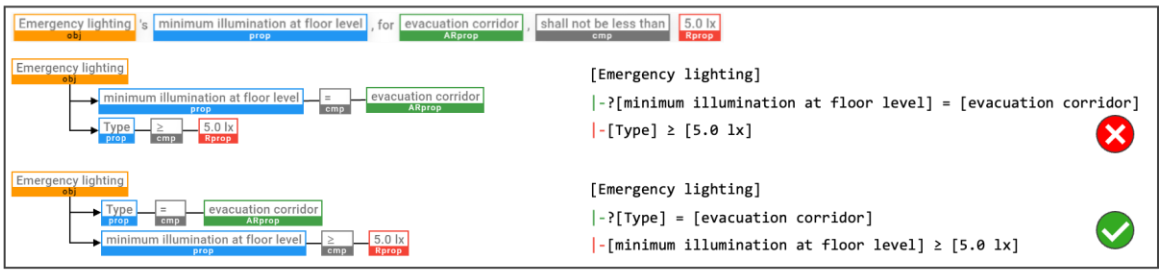

b) Multi-requirement sentence, correctly parsed




c) Multi-requirement sentence, correctly parsed




d) Multi-requirement sentence, correctly parsed




e) Multi-requirement sentence, incorrectly parsed

Figure 11. Example parsing results (English sentences are translated from Chinese preserving the order of words).

Table 5. Comparison of the parsing performance at the element level.

| Sentence category | Method | Sentence number | Element number | Correctly parsed elements | Accuracy |
|---|---|---|---|---|---|
| Single-requirement | Regex-E[12] | 62 | 1083 | 1030 | 95.11% |
| | Regex-ES[12] | 62 | 1919 | 1901 | 99.06% |
| | **CFG (ours)** | **224** | **1174** | **1169** | **99.57%** |
| Multi-requirement | **CFG (ours)** | **387** | **3162** | **3012** | **95.26%** |

Table 6. Parsing performance of the proposed method at the sentence level.

| Sentence category | Sentence number | Correctly parsed sentences | Accuracy |
|---|---|---|---|
| Single-requirement | 224 | 223 | 99.6% |
| Multi-requirement | 387 | 352 | 91.0% |
| Total | 611 | 575 | 94.1% |

## 4.4 Rule generation and application

To demonstrate the real application of our method, this section implements an actual rule checking case in a Revit BIM model using executable rules generated from the parsed RCTrees by developing a rule generation algorithm. Revit Model Checker [44], a tool that can automatically check Revit models based on a set of BIM requirements and generate a compliance report, is utilized. The executable rules for Revit Model Checker are specified in XML format and called Checkset [45]. The XML Checkset is composed of checks that consist of many filters. An individual check describes configurations for validating a rule in a model and can be converted from an RCTree, and a filter inside a check defines a condition or requirement. For more information about the XML Checkset, the readers are referred to [45].

Figure 12 illustrates a rule generation from RCTree to XML Checkset. In the rule generation, RCTree elements are mapped to property names specified in the XML Checkset. For example, in Figure 12, the "window" element is mapped to "OST_Windows", and the "width" element is mapped to "Width". In other cases, for instance, the "width" of a stair will be mapped to "Minimum Run Width", and the default property "Type" of walls will be mapped to "Structural Material". To achieve this, a dictionary is built to store the mapping information. To represent the If-Then format rule, the result condition of a check is defined as "FailMatchingElement", which means if matching elements are found, it will result in a failure. For example, "if A then B" can be represented by the fail-matching of "A and (not B)", i.e., if both "A" and "not B" are satisfied, the result fails and the matched elements will be reported. Therefore, the "cmp" for "Rprop" in the RCTree will be negated in this rule generation. Additionally, unit conversion should also be processed. Revit uses imperial units by default when processing the XML Checkset, so a length value of a metric unit will be converted to an imperial unit (e.g., 800 mm to 2.62467 ft).

Figure 13 shows the rule generation result in the experiment. First, four example regulatory sentences in English are labeled and parsed into RCTrees, and they are two single-requirement and two multi-requirement sentences which are applicable for rule checking in Revit. Then, the RCTrees are generated to an XML Checkset. Figure 14 shows a rule checking application in a BIM model using Revit Model Checker and the four example rules. As shown in the figure, rules are executed to check the BIM model of an actual hotel building, and the result is reported which has a 50% pass rate because the first and fourth checks fail. As illustrated in Figure 14, in the first check, a floor has a thickness of 25 mm which is less than the requirement of 30 mm. Two walls fail to pass the fourth check, and it is found that both walls have a thermal resistance of 0.0923 $m^2K/W$ less than the requirement of 0.1 $m^2K/W$. The BIM model has 10 floors and 94 concrete walls with a width less than 200 mm. Thus, this rule checking, which identifies 1 floor and 2 walls that fail to meet the requirements, successfully detects the issues and can significantly improve the efficiency of design.

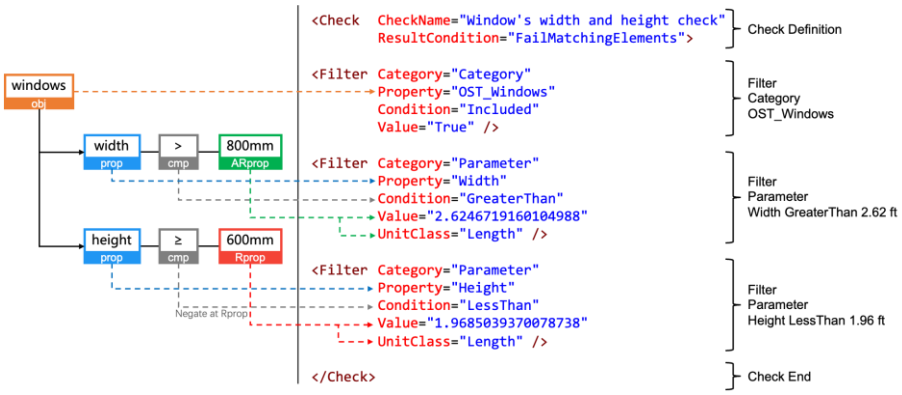



Figure 12. Illustration of a rule generation from RCTree to XML Checkset.

(1) In [public buildings/sobj], the [thickness/prop] of [floors/obj] [shall not be less than/cmp] [30 mm/Rprop]

```
[public buildings]-[floors]
|-[thickness] ≥ [30 mm]
```

(2) In [public buildings/sobj], the [width/prop] of [stairs/obj] [shall not be less than/cmp] [0.8 m/Rprop]

```
[public buildings]-[stairs]
|-[width] ≥ [0.8 m]
```

(3) For [windows/obj] that having a [width/prop] [greater than/cmp] [800 mm/ARprop], they shall have a [height/prop] [no less than/cmp] [600 mm/Rprop]

```
[windows]
|-?[width] > [800 mm]
|-[height] ≥ [600 mm]
```

(4) To meet the thermal insulation requirement, [concrete/ARprop] [walls/obj] that having a [width/prop] [less than/cmp] [200 mm/ARprop] shall have a [thermal resistance/prop] [no less than/cmp] [0.1 m2K/W/Rprop]

```
[walls]
|-?[Type] = [concrete]
|-?[width] < [200 mm]
|-[thermal resistance] ≥ [0.1 m2K/W]
```

Rule Generation ⇩

```
<?xml version='1.0' encoding='utf-8'?>
<MCSettings AllowRequired="False" Author="Zyc" Name="CheckSet-Zyc">
    <Heading HeadingText="Test" IsChecked="True">
        <Section SectionName="Test">
            <Check CheckName="In public buildings, the thickness of floors shall not be less than 30 mm" ResultCondition="FailMatchingElements">
                <Filter Category="TypeOrInstance" Condition="Equal" Property="Is Element Type" Unit="Default" UnitClass="None" Value="0" />
                <Filter Category="Category" Condition="Included" Property="OST_Floors" Value="True" />
                <Filter Category="Parameter" Condition="LessThan" Property="Default Thickness" Unit="Default" UnitClass="Length" Value="0.09842519685039369" />
            </Check>
            <Check CheckName="In public buildings, the width of stairs shall not be less than 0.8 m" ResultCondition="FailMatchingElements">
                <Filter Category="TypeOrInstance" Condition="Equal" Property="Is Element Type" Unit="Default" UnitClass="None" Value="0" />
                <Filter Category="Category" Condition="Included" Property="OST_Stairs" Value="True" />
                <Filter Category="Parameter" Condition="LessThan" Property="Minimum Run Width" Unit="Default" UnitClass="Length" Value="2.6246719160104988" />
            </Check>
            <Check CheckName="For windows that having a width greater than 800 mm, they shall have a height no less than 600 mm"
            ResultCondition="FailMatchingElements">
                <Filter Category="TypeOrInstance" Condition="Equal" Property="Is Element Type" Unit="Default" UnitClass="None" Value="0" />
                <Filter Category="Category" Condition="Included" Property="OST_Windows" Value="True" />
                <Filter Category="Parameter" Condition="GreaterThan" Property="Width" Unit="Default" UnitClass="Length" Value="2.6246719160104988" />
                <Filter Category="Parameter" Condition="LessThan" Property="Height" Unit="Default" UnitClass="Length" Value="1.9685039370078738" />
            </Check>
            <Check CheckName="To meet the thermal insulation requirement, concrete walls that having a width less than 200 mm shall have a thermal resistance
            no less than 0.1 m2K/W" ResultCondition="FailMatchingElements">
                <Filter Category="TypeOrInstance" Condition="Equal" Property="Is Element Type" Unit="Default" UnitClass="None" Value="0" />
                <Filter Category="Category" Condition="Included" Property="OST_Walls" Value="True" />
                <Filter Category="Parameter" Condition="WildCard" Property="Structural Material" Unit="Default" UnitClass="None" Value="Concrete" />
                <Filter Category="Parameter" Condition="LessThan" Property="Width" Unit="Default" UnitClass="Length" Value="0.6561679790026247" />
                <Filter Category="Parameter" Condition="LessThan" Property="Thermal Resistance (R)" Unit="Default" UnitClass="None" Value="0.1" />
            </Check>
        </Section>
    </Heading>
</MCSettings>
```

(1) (2) (3) (4)

Figure 13. Rule generation result for Revit Model Checker.

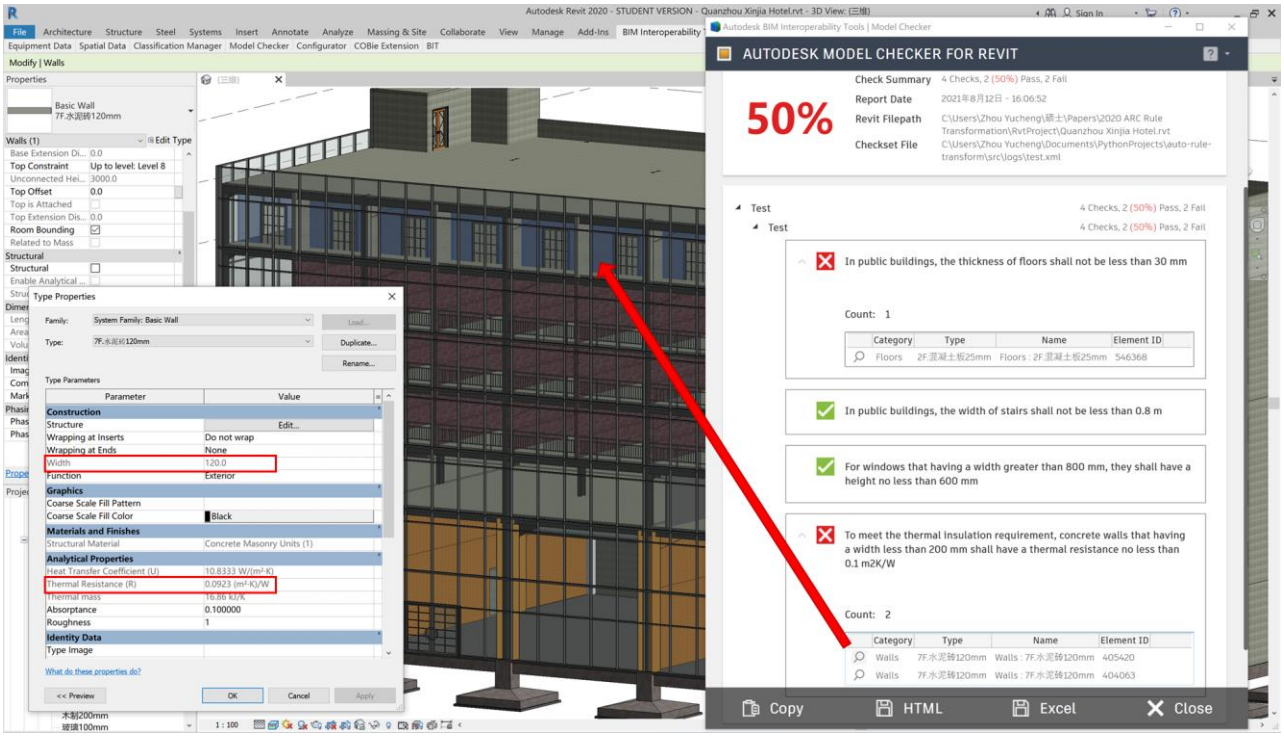



Figure 14. Screenshot of a rule checking application in a BIM model using Revit Model Checker.

# 5 Discussion

This research presents an automated rule interpretation method consisting of semantic labeling and syntactic parsing steps. Currently, existing rule interpretation methods rely on considerable manual effort or a hard-coded manner, and they cannot achieve both a high level of automation and wide scope of application. Compared to the state-of-the-art methods, the significant advances of our method are summarized as follows.

1. Wide application scope. In contrast to existing rule transformation methods focusing on single-requirement sentences, our method can interpret more complex multi-requirement sentences, which are also more commonly found in regulatory documents. Compared to the regular expression approach with low expressiveness (e.g., cannot represent recursion) used in existing methods, our proposed syntax tree structure RCTree and developed CFGs of high expressiveness significantly expand the application scope. Besides, the proposed RCTree is language-independent and can be transformed into many other languages for rule execution, such as XML, SWRL, and programming languages.
2. Automation with high accuracy. Our method achieves high accuracy in the automated rule interpretation process, especially for syntactic parsing. The syntactic parsing step achieves 99.6% and 91.0% accuracies for single- and multi-requirement sentences, respectively, outperforming the state-of-the-art methods. For semantic labeling, our deep learning method obtains a promising 86.2% overall F1-score, and the performance can be further improved by such as utilizing an ontology. Note that our dataset containing 611 sentences is much larger than that used in most existing studies, supporting the result.
3. Interpretable parsing process. In contrast to existing methods that use exhaustive enumerations of matching patterns to parse labeled sentences, our method mainly uses CFGs for parsing. Therefore, our syntactic parsing process, which treats a labeled sentence like a DSL, is interpretable and understandable. This advantage can promote the real application of the method, such as enabling semi-automated parsing and guiding regulatory sentence writing.

Interpretability is of great significance for rule interpretation, and this is a significant difference compared to regular expression-based methods. Natural language has high flexibility, ambiguity, and subjectivity, which challenges the rule interpretation and ARC [46]. Therefore, rule interpretation requires effort from two sides: a method with high performance and regulatory text written in a more regular style. Our method, which provides verifiable labeling results and a deterministic parsing procedure, can bridge this gap. It provides a standard for the syntax of the labeled sentence and thus can guide the writing of regulatory text and parsing irregularly written sentences semi-automatically.

The proposed method provides a generalized framework for automated rule interpretation. When working on other regulatory documents, once the document expresses requirements or rules in the same way (i.e., having object hierarchies and using quantitative requirements applied to some objects), our method can parse its structure in the same way based on the CFGs; the deep learning model for semantic labeling can be reused or re-trained to adjust to a different domain or improve the accuracy. Thus, for example, a designer can use the method to create computable rules from customized regulatory documents, which can be used to check the design proactively and filter out noncompliant designs.

The main limitations of this research are identified and will be studied in the future by the authors. (1) The proposed RCTree has limitations in expression. Currently, it can only represent tree structures where at most one node (“obj”) has multiple child nodes, which is designed as a trade-off between expressiveness and parsing complexity. The representation of more complex structures (e.g., a “sobj” or “prop” element has multiple child nodes) is addressed by sentence splitting. The same also represents complex Boolean logic between requirements in the RCTree. However, these approaches bring difficulties in sentence splitting. To resolve this limitation, more methods and techniques for sentence splitting and tree structure representation need to be studied. (2) The number of parsing CFGs is limited. Expanding the quantity and scope of the CFGs can help the parsing method better characterize the structure of regulatory sentences and thus increase the ability to parse more complex sentences. CFGs of higher expressiveness can also reduce the dependence on the regular expression-based standardization rules and improve maintainability and generalisability. (3) The dataset contains only sentences having quantitative requirements (some having both quantitative and existential requirements are also included). Although it is a legitimate and widely adopted choice [13], more kinds of regulatory rules should be considered to expand the method’s applicability, e.g., geometry/location-based and performance-based rules. We suggest that the end requirements of these rules need to be quantified by calculation or inference, and after that, it is possible to handle these rules by transforming them into attribute-based checking. Take “beam in front of the door” as an example, the “in front of” should be quantified and if the opening direction of the door is the positive direction of the X-axis, it can be transformed into “beam’s x-coordinate value greater than the door’s x-coordinate”. Another example that needs knowledge inference

[47] is “beams in the second floor”, where objects “beams” may not be stored as properties/attributes in “the second floor” in a BIM model, but the relations may be stored in “beams” or independently. Thus, these implicit relations should be inferred and then can be stored as properties. A promising method to achieve knowledge inference is utilizing ontology.

In the future, the following improvements and extensions can be pursued. (1) Developing domain knowledge models such as ontology. In the rule interpretation, a concept can be represented in different ways which leads to ambiguity (e.g., in semantic labeling), and a relation between concepts can be implicit which limits the representation. To address this problem, concepts and their relations in a domain should be clearly defined, and thus ontologies or well-documented concept vocabularies can be developed. Furthermore, since ARC involves two aspects of both regulatory text and BIM model, their concepts should be consistently represented. To achieve this, exploring information alignment technique [48] and utilizing IFC which is friendly to ontology [49] could be studied. Consequently, instructions and standards can be formulated to guide writing regulatory documents in a regular form to substantially facilitate ARC. (2) Quantifying statistics of quantitative sentences and sentences applicable for rule interpretation, which can help understand regulatory documents’ quality and how wide the method’s real applicability is. In the current dataset development, we first collect 30000 regulatory sentences after sentence splitting, and then recognize 3660 sentences having quantitative requirements by keyword matching. Finally, we randomly select 1000 quantitative sentences as candidates and manually review their applicability, and obtain 611 sentences forming the dataset. However, these numbers and their ratios may be varied and not accurate because of the large and varied proportion of uninterpretable regulatory sentences we identified: (a) non-requirement sentences such as writing purpose, term definitions, titles, appendix, etc.; (b) subjective requirements such as “涂料应按说明书规定在现场调配且应充分搅拌” or “The paint should be prepared on-site according to the instructions and should be fully stirred”; and (c) incomplete sentences such as sentences are incomplete split or have incomplete meanings. Accurate measurement and classification of these sentences need future study [50]. (3) Including deontic logic in the rule representation. Some regulatory rules have terms of deontic logic such as “must”, “shall”, “could”, and “prohibit”, and the rule checking of them should result in a graded scale from pass to fail. This research limits this kind of rule to a binary kind of true or false to simplify the rule representation and checking. However, they need future studies including the identification, representation, and result explanation, and new semantic labels and/or specific algorithms can be introduced. (4) Improving the accuracy of semantic labeling. Currently, an overall F1-score of 86.2% is achieved by a DNN model. For improvements, there exist three promising methods: developing an ontology to help recognize concepts in sentences, utilizing unsupervised learning to pretrain DNN models in domain-specific corpora [51], and increasing the size of dataset. (5) Expanding the scope of the experiment. The proposed method is tested only on Chinese building codes for fire protection, and it is necessary to conduct more experiments on different types and languages of building codes. To achieve this, more DNN models (e.g., English BERT) could be utilized for semantic labeling and more language-specific sentence standardization rules could be developed. It is expected that the method can also achieve high performance on different building codes because of the large size of dataset used in this experiment. However, the results may vary due to the variety of expressions in the natural language. (6) Improving rule generation methods. Currently, RCTree is transformed into XML Checkset of Revit Model Checker for application. The XML Checkset has limited expressiveness for representing rules that have more than two hierarchy levels and implicit object relations. Therefore, improvements for the rule generation method and its application for more kinds of BIM such as IFC need to be studied.

# 6 Conclusion

This research contributes to the body of knowledge an automated rule interpretation method with a wide application scope and high accuracy, and an open regulation dataset for ARC. First, we propose seven semantic elements and the RCTree to represent BIM objects and rule checking requirements. Second, a deep learning model with the transfer learning technique is utilized for semantic labeling. Finally, CFGs and sentence standardization rules are developed to parse labeled sentences into RCTrees. Results show that in semantic labeling, our method is applicable to long and complex sentences and obtains a promising 86.2% overall F1-score, and has great potential for further improvement. In syntactic parsing, our method (1) achieves a high 99.6% accuracy for parsing single-requirement sentences, and (2) achieves a high 91.0% accuracy for parsing multi-requirement sentences, which is complex and common but existing methods are difficult to apply. This result suggests our method has a wider application scope than existing automated rule interpretation methods. The interpretable and understandable parsing process of our method can guide writing regulatory text in a more regular form and eliminate incorrect interpretation results easily to facilitate real applications. This research also contributes a generalized framework of automated rule interpretation for creating computable rules from various regulatory documents

and different languages. This research publishes the first regulation dataset, which contains a larger size of regulatory sentences with annotation than existing studies, for future exploration, validation, and benchmarking in the ARC area.

Future work remains to be done to enhance and expand the contributions of this research. (1) Expression of more complex structures by RCTree. (2) Expanding CFGs for higher expressiveness and performance. (3) Interpreting more kinds of regulatory sentences than quantitative sentences. (4) Developing ontologies to clearly define domain concepts and relations and exploring writing regulatory documents in a regular form. (5) Quantifying statistics about sentences applicable for rule interpretation to measure the overall applicability of the method. (6) Including deontic logic in the rule representation. (7) Increasing the size of dataset and utilizing unsupervised learning to enhance the performance of semantic labeling. (8) Experimenting on more types and languages of regulatory documents for validation. (9) Improving rule generation methods such as considering IFC model. These aspects will constitute our future work.

## References


[1] N.O. Nawari, Building information modeling: automated code checking and compliance processes, CRC Press, 2018. https://doi.org/10.1201/9781351200998.

[2] L. Jiang, J. Shi, C. Wang, Multi-ontology fusion and rule development to facilitate automated code compliance checking using BIM and rule-based reasoning, Advanced Engineering Informatics. 51 (2022) 101449. https://doi.org/10.1016/j.aei.2021.101449.

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

[4] T.H. Beach, J.-L. Hippolyte, Y. Rezgui, Towards the adoption of automated regulatory compliance checking in the built environment, Automation in Construction. 118 (2020) 103285. https://doi.org/10.1016/j.autcon.2020.103285.

[5] A.S. Ismail, K.N. Ali, N.A. Iahad, A Review on BIM-based automated code compliance checking system, in: Proceedings of the 5th International Conference on Research and Innovation in Information Systems (ICRIIS), IEEE, Langkawi, Malaysia, 2017: pp. 1–6. https://doi.org/10.1109/ICRIIS.2017.8002486.

[6] E. Hjelseth, A.K. Lassen, J. Dimyadi, Development of BIM-based model checking solutions – ongoing research and practitioners' demand, in: Proceedings of the 33rd CIB W78 Conference, Brisbane, Australia, 2016. http://itc.scix.net/paper/w78-2016-paper-018 (accessed August 7, 2021).

[7] X. Xue, J. Wu, J. Zhang, Semiautomated generation of logic rules for tabular information in building codes to support automated code compliance checking, Journal of Computing in Civil Engineering. 36 (2022) 04021033. https://doi.org/10.1061/(ASCE)CP.1943-5487.0001000.

[8] C. Eastman, J. Lee, Y. Jeong, J. Lee, Automatic rule-based checking of building designs, Automation in Construction. 18 (2009) 1011–1033. https://doi.org/10.1016/j.autcon.2009.07.002.

[9] D. Greenwood, S. Lockley, S. Malsane, J. Matthews, Automated compliance checking using building information models, in: The Construction, Building and Real Estate Research Conference of the Royal Institution of Chartered Surveyors, RICS, Paris, 2010. http://nrl.northumbria.ac.uk/6955/1/Automated_compliance_checking_using_building_information.pdf.

[10] N.O. Nawari, A generalized adaptive framework (GAF) for automating code compliance checking, Buildings. 9 (2019) 86. https://doi.org/10.3390/buildings9040086.

[11] E. Hjelseth, N. Nisbet, Capturing normative constraints by use of the semantic mark-up RASE methodology, in: Proceedings of the 28th CIB W78 Conference, Sophia Antipolis, France, 2011: pp. 1–10. http://itc.scix.net/paper/w78-2011-Paper-45.

[12] J. Zhang, N.M. El-Gohary, Automated information transformation for automated regulatory compliance checking in construction, Journal of Computing in Civil Engineering. 29 (2015) B4015001. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000427.

[13] J. Zhang, N.M. El-Gohary, Semantic NLP-based information extraction from construction regulatory documents for automated compliance checking, Journal of Computing in Civil Engineering. 30 (2016) 04015014. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000346.

[14] G. Tomassetti, The ANTLR mega tutorial, Federico Tomassetti - Software Architect. (2017). https://tomassetti.me/antlr-mega-tutorial/ (accessed August 19, 2020).

[15] Y. Ding, J. Ma, X. Luo, Applications of natural language processing in construction, Automation in Construction. 136 (2022) 104169. https://doi.org/10.1016/j.autcon.2022.104169.

[16] S.J. Fenves, Tabular decision logic for structural design, Journal of the Structural Division. 92 (1966) 473–490. https://doi.org/10.1061/JSDEAG.0001567.

[17] J.H. Garrett, S.J. Fenves, A knowledge-based standards processor for structural component design, Engineering with Computers. 2 (1987) 219–238. https://doi.org/10.1007/BF01276414.

[18] E.A. Delis, A. Delis, Automatic fire-code checking using expert-system technology, Journal of Computing in Civil Engineering. 9 (1995) 141–156. https://doi.org/10.1061/(ASCE)0887-3801(1995)9:2(141).

[19] C.S. Han, J.C. Kunz, K.H. Law, A hybrid prescriptive/performance based approach to automated building code checking, in: International Computing Congress, ASCE, 1998: pp. 537–548. http://eil.stanford.edu/publications/chuck_han/9810%20ICC.pdf.

[20] L. Ding, R. Drogemuller, M. Rosenman, D. Marchant, J. Gero, Automating code checking for building designs -DesignCheck, Clients Driving Innovation: Moving Ideas into Practice. (2006) 1–16.

[21] 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. https://doi.org/10.1016/j.eswa.2015.02.029.

[22] G. Lau, K. Law, An information infrastructure for comparing accessibility regulations and related information from multiple sources, in: Proceedings of the 10th International Conference on Computing in Civil and Building Engineering, Professur Informatik im Bauwesen, Weimar, Germany, 2004. http://eig.stanford.edu/publications/gloria_lau/icccbe.pdf (accessed August 7, 2021).

[23] J. Dimyadi, P. Pauwels, M. Spearpoint, C. Clifton, R. Amor, Querying a regulatory model for compliant building design audit, in: Proceedings of the 32nd CIB W78 Conference, Eindhoven, The Netherlands, 2015: pp. 139–148. http://itc.scix.net/paper/w78-2015-paper-014 (accessed August 7, 2021).
[24] 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. https://doi.org/10.1016/j.autcon.2009.07.008.
[25] J.-K. Lee, C.M. Eastman, Y.C. Lee, Implementation of a BIM domain-specific language for the building environment rule and analysis, Journal of Intelligent & Robotic Systems. 79 (2015) 507–522. https://doi.org/10.1007/s10846-014-0117-7.
[26] S. Park, H. Lee, S. Lee, J. Shin, J.-K. Lee, Rule checking method-centered approach to represent building permit requirements, in: Proceedings of the 32nd International Symposium on Automation and Robotics in Construction (ISARC), IAARC Publications, Oulu, Finland, 2015: pp. 1–8. https://doi.org/10.22260/ISARC2015/0049.
[27] C. Sydora, E. Stroulia, Rule-based compliance checking and generative design for building interiors using BIM, Automation in Construction. 120 (2020) 103368. https://doi.org/10.1016/j.autcon.2020.103368.
[28] C. Preidel, A. Borrmann, Automated code compliance checking based on a visual language and building information modeling, in: Proceedings of the 32nd International Symposium on Automation and Robotics in Construction (ISARC), IAARC Publications, Oulu, Finland, 2015: pp. 1–8. https://doi.org/10.22260/ISARC2015/0033.
[29] M. Häußler, S. Esser, A. Borrmann, Code compliance checking of railway designs by integrating BIM, BPMN and DMN, Automation in Construction. 121 (2021) 103427. https://doi.org/10.1016/j.autcon.2020.103427.
[30] P. Zhou, N. El-Gohary, Ontology-based automated information extraction from building energy conservation codes, Automation in Construction. 74 (2017) 103–117. https://doi.org/10.1016/j.autcon.2016.09.004.
[31] S. Li, H. Cai, V.R. Kamat, Integrating natural language processing and spatial reasoning for utility compliance checking, Journal of Construction Engineering and Management. 142 (2016) 04016074. https://doi.org/10.1061/(ASCE)CO.1943-7862.0001199.
[32] X. Xu, H. Cai, Semantic frame-based information extraction from utility regulatory documents to support compliance checking, in: Proceedings of the 35th CIB W78 Conference, Chicago, USA, 2018. http://itc.scix.net/paper/w78-2018-paper-027 (accessed August 7, 2021).
[33] X. Xu, H. Cai, Ontology and rule-based natural language processing approach for interpreting textual regulations on underground utility infrastructure, Advanced Engineering Informatics. 48 (2021) 101288. https://doi.org/10.1016/j.aei.2021.101288.
[34] D.M. Salama, N.M. El-Gohary, Semantic Text Classification for Supporting Automated Compliance Checking in Construction, Journal of Computing in Civil Engineering. 30 (2016) 04014106. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000301.
[35] J.R. Lin, Y.C. Zhou, J.P. Zhang, Z.Z. Hu, Classification and exemplary bim models development of design changes, in: Proceedings of the 36th International Symposium on Automation and Robotics in Construction (ISARC), IAARC Publications, Banff, Canada, 2019: pp. 122–127. https://doi.org/10.22260/ISARC2019/0017.
[36] J.-R. Lin, Y.-C. Zhou, Semantic classification and hash code accelerated detection of design changes in BIM models, Automation in Construction. 115 (2020) 103212. https://doi.org/10.1016/j.autcon.2020.103212.
[37] R. Zhang, N. El-Gohary, A machine learning approach for compliance checking-specific semantic role labeling of building code sentences, in: Proceedings of the 35th CIB W78 Conference, Chicago, USA, 2018. http://itc.scix.net/paper/w78-2018-paper-067 (accessed August 7, 2021).
[38] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, C. Dyer, Neural architectures for named entity recognition, in: Proceedings of NAACL 2016, San Diego, California, USA, 2016: pp. 260–270. https://doi.org/10.18653/v1/N16-1030.
[39] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: Proceedings of NAACL 2019, Minneapolis, Minnesota, USA, 2019: pp. 4171–4186. https://doi.org/10.18653/v1/N19-1423.
[40] N. Chomsky, Three models for the description of language, IRE Transactions on Information Theory. 2 (1956) 113–124. https://doi.org/10.1109/TIT.1956.1056813.
[41] T. Parr, The definitive ANTLR 4 reference, Pragmatic Bookshelf, 2013. https://dl.acm.org/doi/10.5555/2501720.
[42] F. He, T. Liu, D. Tao, Control Batch Size and Learning Rate to Generalize Well: Theoretical and Empirical Evidence, in: Proceedings of Advances in Neural Information Processing Systems 32 (NeurIPS 2019), Curran Associates, Inc., 2019. https://proceedings.neurips.cc/paper/2019/file/dc6a70712a252123c40d2adba6a11d84-Paper.pdf.
[43] T. Parr, K. Fisher, LL(*): the foundation of the ANTLR parser generator, ACM Sigplan Notices. 46 (2011) 425–436. https://doi.org/10.1145/1993316.1993548.
[44] Autodesk, Autodesk Model Checker for Revit, (n.d.). https://interoperability.autodesk.com/modelchecker.php (accessed April 14, 2021).
[45] Autodesk, Autodesk Revit Model Checker - XML schema and definitions, (n.d.). https://interoperability.autodesk.com/modelcheckerconfigurator/downloads/xmlschema.pdf (accessed April 14, 2021).
[46] J. Soliman-Junior, P. Tzortzopoulos, J.P. Baldauf, B. Pedo, M. Kagioglou, C.T. Formoso, J. Humphreys, Automated compliance checking in healthcare building design, Automation in Construction. 129 (2021) 103822. https://doi.org/10.1016/j.autcon.2021.103822.
[47] Z.-Z. Hu, S. Leng, J.-R. Lin, S.-W. Li, Y.-Q. Xiao, Knowledge Extraction and Discovery Based on BIM: A Critical Review and Future Directions, Archives of Computational Methods in Engineering. 29 (2021) 335–356. https://doi.org/10.1007/s11831-021-09576-9.
[48] P. Zhou, N. El-Gohary, Semantic information alignment of BIMs to computer-interpretable regulations using ontologies and deep learning, Advanced Engineering Informatics. 48 (2021) 101239. https://doi.org/10.1016/j.aei.2020.101239.
[49] R.K. Soman, M. Molina-Solana, J.K. Whyte, Linked-Data based Constraint-Checking (LDCC) to support look-ahead planning in construction, Automation in Construction. 120 (2020) 103369. https://doi.org/10.1016/j.autcon.2020.103369.
[50] Z. Zheng, Y.-C. Zhou, K.-Y. Chen, X.-Z. Lu, J.-R. Lin, Z.-T. She (2022). Text classification-based approach for automatically evaluating building codes' interpretability. (Under review)
[51] Z. Zheng, X.-Z. Lu, K.-Y. Chen, Y.-C. Zhou, J.-R. Lin, Pretrained domain-specific language model for natural language processing tasks in the AEC domain, Computers in Industry. (2022). https://doi.org/10.1016/j.compind.2022.103733.