2023 European Conference on Computing in Construction
40th International CIB W78 Conference
Heraklion, Crete, Greece
July 10-12, 2023

A LEGALRULEML EDITOR WITH TRANSFORMER-BASED AUTOCOMPLETION

Stefan Fuchs[1], Johannes Dimyadi[12], Aryan Sharma Ronee[1], Rishaan Gupta[1]
Michael Witbrock[1], and Robert Amor[1]
[1]The University of Auckland, New Zealand
[2]CAS Limited, Auckland, New Zealand

Abstract

The construction industry has pursued automated compliance checking for decades, but legal requirements conveyed in natural language are not intended for machine processing. There have been numerous attempts to translate these requirements into computable representations, progressing from manual to fully-automated approaches. However, it is unclear if fully-automated translation will become reliable and interpretable enough for legal matters. We propose a LegalRuleML Editor with Transformer-based Autocompletion to facilitate a semi-automated workflow with minimal manual effort. A deep learning model generates initial translations and contextualised autocompletion options. This strategy offers experts a superior translation process, including continuous improvements approximating full automation.

Introduction

Computer-readable building codes lie at the root of automated compliance checking (ACC), but most codes and standards are exclusively conveyed in natural language. The translation of codes into a computer-readable format faces several challenges: 1) Cross-domain expertise is required (i.e. law, construction, knowledge engineering) (Amor and Dimyadi, 2021). 2) Convoluted sentences, domain-specific terminology and complex legalese cause difficulties for automated translation (Fuchs and Amor, 2021). 3) Since codes and standards change frequently, the computer-readable representation becomes obsolete if not updated in parallel (Zhang et al., 2023). 4) There are many different representations and no agreement on what representation is most suitable or what requirements a specific representation has to meet (buildingSMART, 2017). 5) The need for exceptionally accurate translations (Salama and El-Gohary, 2016) raises the requirement to correctly extract all relevant information and specify logical connections, negations, and deontic effects. So, manual tasks such as reviewing automatically translated clauses and specifying test cases remain.

Many of these issues could be addressed by simplifying the translation of clauses into a semantic representation and moving this task closer to the rule-authoring process. Therefore, we examine the following research questions:

- RQ1: How can we enable rule authors to specify and maintain a logical representation of regulations efficiently?
- RQ2: What functionality is necessary and most helpful for a seamless tool-supported workflow?

We first examine existing tools and translation requirements in the literature to address those research questions. Consistent translation can be supported by utilising domain knowledge bases such as dictionaries and ontologies. Niemeijer et al. (2014) proposed a database which is used to automatically translate design constraints via term matching. KBimCode aims for a consistent and user-friendly translation process by searching for entities and predicates in the clause to translate and the KBimCode Database (Song et al., 2018). LIME (Palmirani et al., 2023) and RAWE (Palmirani et al., 2013) are visual editors to annotate legal documents in Akoma Ntoso and generate a logical representation in LegalRuleML (LRML), respectively. Dimyadi et al. (2020) used Excel proformas to extract information from sixteen Acceptable Solutions of the New Zealand Building Codes and a LRML converter to produce LRML files and ensure the validity of generated rules. RASE (Requirements, Application, Selection, Exception) is a natural language markup technique to support the translation process (Hjelseth and Nisbet, 2011). The accompanying tool 'Require1' offers a user interface to annotate these high-level constructs and concept types (i.e. object, property, predicate, value, and unit).

Rule-based fully-automated approaches perform well in narrow domains but need further rule development to scale to new regulation types (Zhang and El-Gohary, 2019). I-SNACC (Wu et al., 2023) integrates a rule-based translation approach (Zhang and El-Gohary, 2016) into an ACC framework. It includes a user interface for developer-users to fix generated logic rules.

Machine-Learning (ML) based approaches have been used to extract information at different levels of granularity. For example, Zhang and El-Gohary (2019) segmented clauses into Requirement Units, Zhou et al. (2022) performed semantic role labelling, and Li et al. (2020) extracted Entity-Relationship triplets. In contrast, Fuchs et al. (2022) proposed a deep learning-based full translation of clauses into LRML using transformer models. Transformer refers to the underlying neural network architecture proposed by Vaswani et al. (2017).

GitHub Copilot (Chen et al., 2021) shows the potential of human-machine interaction for complex tasks such as writing computer code. To reduce the manual effort of semi-automated building regulation translation, this paper proposes a direction similar to Copilot: A LRML Editor powered by a transformer-based model for translation and autocompletion based on Fuchs et al. (2022). The LRML Editor allows quick translation of regulation clauses into LRML by supplying full translations and context-dependent autocompletion. In addition, we integrated a semantic search mechanism to retrieve terms from dictionaries and query translations of related regulatory

clauses following Niemeijer et al. (2014) and KBimCode, but with a transformer model to calculate semantic similarity. Compared to previous attempts, a full translation in combination with autocompletion provides greater support and promises further improvements over time. Newly translated regulations and fixed autocompletion suggestions can be used to retrain the transformer model to improve future translations and autocompletion suggestions. Rule authors could use this editor to create and review a computable representation of building regulations semi-automatically, and domain experts and data scientists could use it as a data collection platform to produce new training data more efficiently.

Tool-supported building code translation

In this section, we will first collect the requirements for tool-supported translation of building regulations by investigating previous literature on manual, tool-supported and automatic translation. We then evaluate the main requirements with the developed LRML editor.

Requirements

To collect the requirements, we investigate requirements for rule representations, typical translation workflows and re-occurring problems during translation. We will refer to existing tool support and how those aspects were addressed. While we cover the major functional requirements, we do not claim the completeness of this list. The collections helped us make informed decisions on the most important requirements for a proof-of-concept. Further requirements will be identified in the Evaluation section.

Document context

Ilal and Günaydın (2017) described the need to have translated clauses connected to their origin. Representations for legal documents such as ISO-STS and Akoma Ntoso can address such needs. For example, LIME and RAWE are interconnected visual editors for Akoma Ntoso and LegalRuleML. These representations are designed to work well together and were also used by Dimyadi et al. (2020). Annotating legal documents (e.g. PDF or HTML) was addressed by Lau and Law (2004), and governments have started to publish digitalised versions of legal documents.

Clause context

Zhang et al. (2023) suggest considering the overall set of regulations rather than individual clauses. Fuchs and Amor (2021) identified that the context of clauses within the regulatory documents is mostly neglected during the translation. KBimCode partly addresses this issue by searching for similar clauses and classifying clauses by topic. Also, LRML allows one to group statements and indicate defeasibility constraints between them.

Consistency

Yurchyshyna et al. (2010) showed the need for consistent translations. While the KBimCode's clause search can help with a more consistent translation, using dictionaries or ontologies in combination with the semantic representation is considered essential (Zhang et al., 2023). This principle was followed in various ways: RAWE allows one to link terms to an ontology. Dimyadi et al. (2020) introduced the Legal Knowledge Management (LKM) Dictionary. Niemeijer et al. (2014) stores objects and entities added by the user in a database. Kbimcode's translation interface lets one insert new translations with all constituents into their database.

Granularity

Fuchs et al. (2023) showed it is not straightforward to decide the required granularity of the extracted information. Dimyadi et al. (2020) addressed this by aligning the translation process to the buildingSMART Data Dictionary (bSDD). Shi and Roman (2017) mapped clauses to ontologies and IFC to support experts in translating them to SWRL and XSLT, respectively. Zhang et al. (2023) suggested the need to keep the dictionary independent from rule engines and building information models (BIM) since regulatory concepts might not be available in BIM.

User friendliness

Ilal and Günaydın (2017) and Zhang et al. (2023) identified the need for the representations to be human readable, flexible and easy to use. RAWE addressed this with a visual interface, RASE with an annotation tool, and KBimCode with extracting objects and predicates and retrieving related clauses and concepts.

Conciseness

Zhang et al. (2023) suggest conciseness and low repetition levels as another important factor. We consider it an important factor that the user-facing representation is easy to comprehend. But the final representation can have additional meta-data required for reasoning or versioning. For example, Dimyadi et al. (2020) converted user-readable Excel proformas into LRML.

Validity

Requirements for credibility (Zhang et al., 2023) and quality (Salama and El-Gohary, 2016) were addressed by Dimyadi et al. (2020) with a review process and sanity checks in the LRML Converter (i.e. is the syntax correct and are terms in the LKM dictionaries). Especially in NLP-related studies, the ground truth is often established by multiple experts in parallel, aiming for high inter-annotator agreement (Zhou and El-Gohary, 2017). Finally, Wu et al. (2023) checks the logic rules for grammar, format, and validity during generation and review.

Maintainability

Building codes and standards tend to be amended frequently, causing the need to modify the translations (Zhang et al., 2023). While some representations like LRML support versioning, most described approaches do not have a simple way to update clauses and must undergo a similar process to the initial translation. ML-

based approaches seem to have better prospects by keeping the manual effort to a minimum (e.g. I-SNACC (Wu et al., 2023)). Nevertheless, the better solution might be to have the translation process within the legislative body (Dimyadi and Amor, 2013).

Soundness

A specific focus should be on logical relationships (*and*/*or*/*not*) (Zhang et al., 2023) and deontic effects (Zhang and El-Gohary, 2016). These aspects decide the validity of the translation and are frequently incorrect in ML-based methods, which struggle with negations and underrepresented classes. So, semi-automated or rule-based strategies might have advantages in this aspect.

Handling uncertainty

Ambiguity, contradiction and vagueness in legal clauses must be resolved during the translation process (Eastman et al., 2009; Soliman-Junior et al., 2020). Barraclough et al. (2021) addressed these issues by documenting all expert interpretations made during the translation process. Moreover, some user input is expected to remain necessary within the rule engine (Dimyadi et al., 2016).

Translation strategy

There are many strategies to translate regulations. For example, Bhuiyan et al. (2019) used a bottom-up strategy (i.e. define atoms, determine norms, generate if-then structure, encode rules). Such a strategy is supported by tools like KBimCode (Song et al., 2018), where concepts and predicates can be automatically extracted and mapped to KBimCode objects, properties and predicates. Also, the automated information extraction and relation extraction approaches (Zhang and El-Gohary, 2016; Zhou et al., 2022; Li et al., 2020) can be helpful for this strategy.
In comparison, a top-down strategy was used by Nazarenko et al. (2016), who first extracted coarse-grained information, and then split it into medium- and fine-grained entities. Similar strategies are followed by RASE and the requirement unit extraction by Zhang and El-Gohary (2019).

Editor Design

Based on our previous experiences and data availability, we selected LRML as the representation for building regulations. For this proof-of-concept, we focus on the requirements that allow a seamless and efficient translation workflow while demonstrating the benefits of integrating the editor with transformer-based translation. Table 1 gives a detailed description of the proposed and future functionality in relation to the previously described requirements.
The main components are two text editor views, one for the legal clause to be translated and one for the LRML representation of the clause. The LRML representation was shortened, automatically formatted, and syntax highlighted. The user can retrieve an initial translation from the deep learning model and get further autocompletion suggestions during the translation process. A file viewer gives access to translated clauses. Additionally, dictionary terms and translations related to the current clause or search terms are displayed in two separate views.

Editor Implementation

This section will give an overview of the user interface and provide implementation details on the relevant elements. The editor was developed as a React Web App to utilise existing code editor libraries and simplify collaborative work. A Python backend allows us to persist data and access the deep learning models. Figure 1 shows the full user interface:

- Views 1, 2, and 3 utilise the code editor library described in the section: LRML Editor View.
- View 1 is the LRML Editor used to enter new LRML rules. This view supports the user with syntax highlighting, formatting, warnings if terms are not in the LKM dictionary and autocompletion.
- View 2 allows viewing and editing legal clauses that should be translated.
- View 3 shows search results for related translations.
- View 4 shows search results from the LKM dictionaries and classification systems.
- View 5 allows one to navigate between existing translations of clauses and add new translations.

LRML Editor View

A feature-rich code editor library can provide a suitable starting point to ease the creation of LRML rules. The following are the criteria by which we selected a library: Extensibility, community support, and support for new languages, autocompletion, formatting and syntax highlighting. After an initial screening, three libraries were closely inspected: ACE, Monaco, and CodeMirror. All three editors fulfil these requirements. ACE is the code editor powering the AWS Cloud9 Online IDE, Monaco is directly generated from the Visual Studio Code source code, and CodeMirror is used by a wide range of online editors, including the Chrome DevTools, Github and JSFiddle.
We selected CodeMirror[1] for this work since it is well-documented, designed with extensibility in mind, and perfectly suited to run multiple views with different configurations. The integrated Lezer Parser System lets one easily define the grammar of a new language, and the syntax tree can be used for formatting and autocompletion.
The Lint extension for static code analysis loads the LKM dictionaries and shows a warning if functions, objects or properties are not in the dictionaries. New terms can be added to the dictionary to resolve the warning. Such terms would need to be reviewed by other authors in a separate workflow. In addition, the editor supports undo, bracket matching and closing, code folding, and a range of keymaps, allowing quick navigation within the editor.
Figure 2 shows an example LRML rule in the LRML Editor View, including the shortened LRML representation, auto-formatting, and syntax highlighting. The keywords

[1]https://codemirror.net/

Table 1: Requirements addressed by features

Requirements	LRML Editor Features	Future Possibilities
Document context	LRML supports integration with Akoma Ntoso. The editor's shortened representation can be converted to LRML files, including the necessary meta-data.	Load clauses and definitions from legal documents.
Clause context	File viewer and search mechanism to inspect related clauses.	Reference LRML rules and reuse expressions.
Consistency	Representation grounded in dictionaries. Search function for similar clauses. Dictionary- and transformer-based autocompletion to encourage using consistent vocabulary and translation styles. Linter to encourage dictionary use.	-
Granularity	Alignment with classifications (i.e. IFC, Uniclass, OmniClass) helps extract information in correct granularity without relying on a specific format.	Include entities defined within the legal document.
User friendliness	Syntax highlighting, formatting and short format for readability. Initial translation as a skeleton. Related translations as guide. Autocompletion as inspiration and support.	-
Conciseness	Shortened format to reduce repetition and writing effort. A reversible transformation was applied to Fuchs et al. (2022): *expr(fun(greaterThan),atom(rel(width),var(wall)),data(1m)) -> greaterThan(wall.width, 1 m)*.	Define reusable expressions.
Validity	Lint rules: Check if terms are in the dictionaries.	Syntax checks. Support review process.
Maintainability	LRML's versioning support. Possible integration with rule authoring. Efficiency enhancement (e.g. autocompletion).	Extensive versioning and improved text editing for clauses and legal documents.
Soundness	Semi-automated rather than fully automated. Logical and deontic keywords are highlighted.	Further sanity checks. Support review process.
Handling uncertainty	Integration with rule authoring could avoid or reduce ambiguities, contradictions and vagueness.	Commenting and decision tracking. Support user input.
Bottom-up strategy	Autocompletion model to suggest specific terms likely extracted from the current clause.	Entities and predicates extracted from the clause as autocompletion source.
Top-down strategy	Autocompletion model to suggest expressions depending on the completion context (e.g. *if*, *then*, *and*).	Intermediate translation steps to direct the model and simplify the supervision.

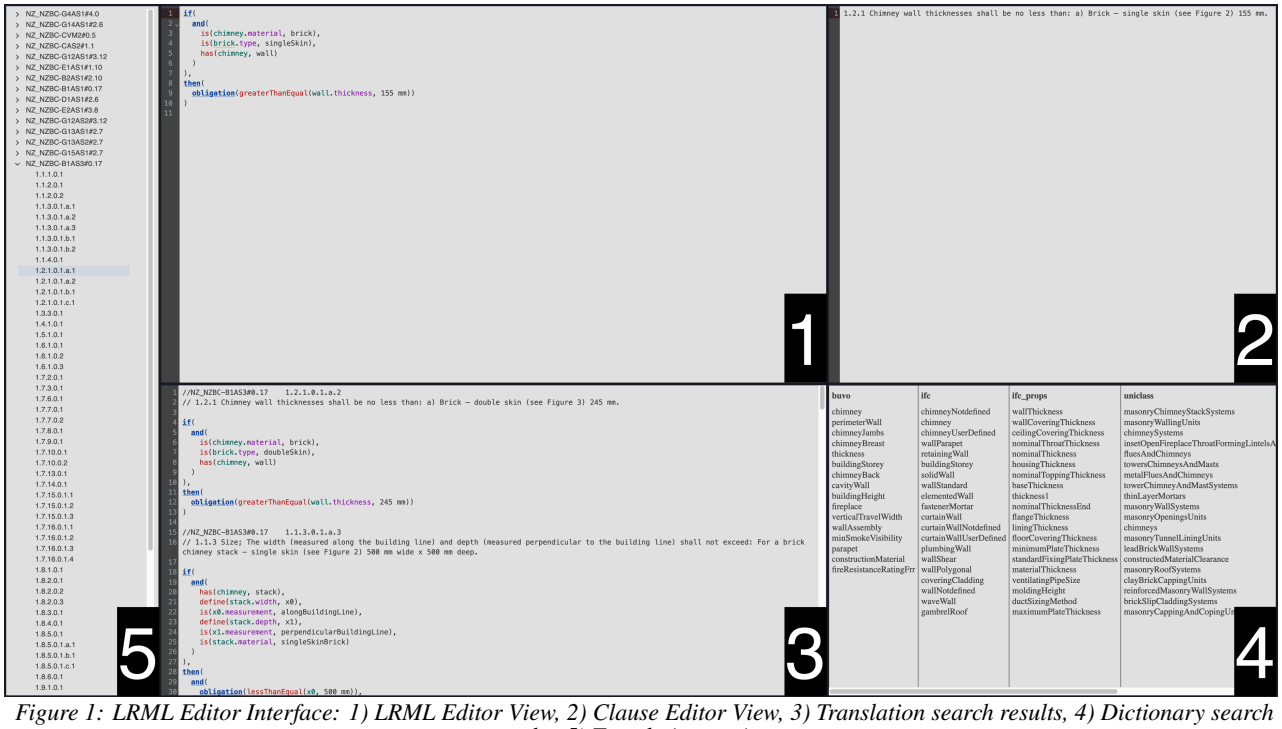

Figure 1: LRML Editor Interface: 1) LRML Editor View, 2) Clause Editor View, 3) Translation search results, 4) Dictionary search results, 5) Translation navigator

if, *then*, *and*, and *obligation* are underlined, emphasising their importance. The material *brick* has a warning because it is missing in the LKM dictionary.

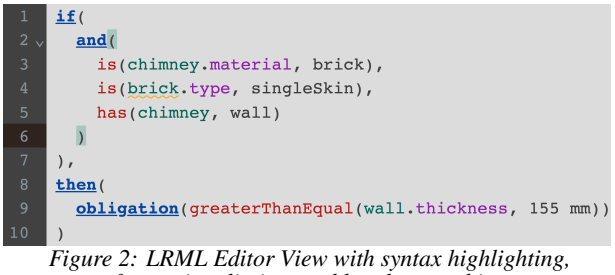

Figure 2: LRML Editor View with syntax highlighting, formatting, linting, and bracket matching

Autocompletion

The main functionality that promises efficient translation is autocompletion, which was also utilised to provide complete translations. While the LKM dictionary and logical and deontic keywords were added as autocompletion sources out-of-the-box, a major contribution of this paper is to provide deep learning-based building clause-specific autocompletion options (see Figure 3).

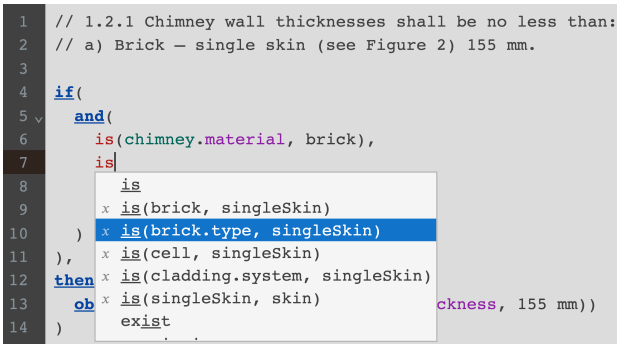

Figure 3: Autocompletion example

Fuchs et al. (2022) trained a T5-Model to generate the LRML representation for a given input clause. We train T5 similarly with the legal clause as input (e.g., *translate English to LegalRuleML: G13AS1 3.4.2 The floor waste shall have a minimum diameter of 40 mm.*) and the LRML rule as the label: *if(exist(floorWaste)), then(obligation(greaterThanEqual(floorWaste.diameter, 40 mm)))*.

While the same model could be used to provide autocompletion options by algorithmically extracting entities and expressions, this approach is not optimal in three regards: 1) The model needs to generate longer outputs than required, 2) the editor context, i.e. what parts were already translated, is disregarded, and 3) an algorithmic selection of the options introduces a new error source.

To avoid these issues, we generated training samples that require a completion starting from a certain point in the syntax tree. Therefore, we randomly selected 2 nodes for each training sample. Selected nodes act as pivot points. The LRML statement up to each node was appended to the clause. The training objective is to generate the LRML statement for the selected node, including all its children. The statement length depends on the depth of the syntax tree. E.g., Input: *translate English to LegalRuleML: G13AS1 [...] 40 mm.<sep>if(*, Label: *exist(floorWaste)*.

While this procedure addressed all three issues, the editor context is not yet fully integrated. In practice, autocompletion is mostly triggered by the user typing the beginning of a word. To accommodate this behaviour, the training data generation was adopted to include a sub-string of the pivot node's text, with shorter sub-strings being more likely selected than longer ones. E.g., Input: *translate [...] 40 mm.<sep>if(exist(fl*, Label: *floorWaste*.

With the described training procedure, a full translation can only be achieved by separately generating the *if*- and *then*-statements. To avoid loading a separate model into memory for full translations, we added the full translation data set to the generated completion data set. Accordingly, autocompletion without any context will trigger a full translation, which might only need minor corrections and is usable as a skeleton for the expert's translation.

Figure 3 shows an example of the resulting autocompletion options. *is* and *exist* are suggestions from the LKM relation dictionary. The remaining suggestions are generated by the deep learning model using beam search. It can be noted that the selected suggestion is correct according to the ground truth (see Figure 2). This is no surprise in this example since it was part of the model's training data. This brings one problem with it: The workflow of fixing existing translations. If a clause was included in the training data, the model would be heavily biased towards the existing translation. As part of the evaluation, we will investigate autocompletion suggestions for clauses not seen during training or testing.

Search mechanism

Similar to previously described approaches, we offer a mechanism for searching the dictionaries and related translations. In particular, we use two strategies depending on the search case. First, we calculate n-gram overlaps between the search query and the search index. Tokenization, stopword removal and lemmatization were used as preprocessing. This strategy works well for searching dictionary terms for an entire clause. For short search queries and to search related translations, we used the SentenceTranformer library to create high-dimensional sentence embeddings (Reimers and Gurevych, 2019). The similarity between the search terms and all options is then calculated using cosine similarity. Up to 20 suggestions per search index over a specified threshold are presented to the user.

Evaluation

LRML Editor translation workflow

As part of the evaluation, we show an exemplary translation workflow for Clause 3.1.1.1 of the Acceptable Solution H1/AS1 for the New Zealand Building Code H1 Energy Efficiency (Ministry of Business, Innovation and Employment, 2022). This Building Code was not part of

the translated documents by Dimyadi et al. (2020) used as training data. Figure 4 shows the example clause plus target translation displayed in the LRML Editor View.

```
// Hot water systems for sanitary fixtures and sanitary appliances
// having a storage water heater capacity of up to 700 litres
// shall comply with NZS 4305.

if(
  and(
    for(hotWaterSystem, or(sanitaryFixture, sanitaryAppliance)),
    include(hotWaterSystem, storageWaterHeater),
    lessThanEqual(storageWaterHeater.capacity, 700 l)
  )
),
then(
  obligation(complyWith(hotWaterSystem, nzs_4305))
)
```

Figure 4: Example Clause H1/AS1 3.1.1.1 with the target LRML representation

The initial translation can be generated using the transformer model (see Figure 5). The model returns the top five beam search results. We select the first option that includes the disjunction.

```
x if(and(for(hotWaterSystem, and(sanitaryFixture, sanitaryAppliance)), has(sanitaryAppl…
x if(and(for(hotWaterSystem, and(sanitaryFixture, sanitaryAppliance)), has(sanitaryAppl…
x if(and(for(hotWaterSystem, and(sanitaryFixture, sanitaryAppliance)), has(sanitaryAppl…
x if(and(for(hotWaterSystem, or(sanitaryFixture, sanitaryAppliance)), has(sanitaryAppli…
x if(and(for(hotWaterSystem, or(sanitaryFixture, sanitaryAppliance)), has(sanitaryAppli…
```

Figure 5: Retrieve full translations

Most parts of the selected translation shown in Figure 6, such as the preconditions and the obligation, are already correct. Only the relation between the *storageWaterHeater* and the *hotWaterSystem* was not identified, and *greaterThanEqual* has to be changed to *lessThanEqual*. While *lessThanEqual* was correct in one of the autocompletion options, the *hotWaterSystem*-relation was continuously wrong.

```
if(
  and(
    for(hotWaterSystem, or(sanitaryFixture, sanitaryAppliance)),
    has(sanitaryAppliance, storageWaterHeater),
    greaterThanEqual(storageWaterHeater.capacity, 700 l)
  )
),
then(
  obligation(complyWith(hotWaterSystem, nzs_4305))
)
```

Figure 6: Initial translation

We lead the transformer model in the right direction by typing *inc* before triggering the autocompletion. The first autocompletion option in Figure 7 is correct.

```
if(
  and(
    for(hotWaterSystem, or(sanitaryFixture, sanitaryAppliance)),
    inc
                include
                x include(hotWaterSystem, storageWaterHeater)
  )             x include(sanitaryAppliance, storageWaterHeater)
),              x include(sanitaryAppliances, storageWaterHeater)
then            x includes(sanitaryAppliance, storageWaterHeater)
  ob            x including(sanitaryAppliance, storageWaterHeater)
)               intersect
```

Figure 7: Intentional autocompletion by typing "inc"

For the second error, we use the dictionary-based autocompletion to insert *lessThanEqual* (see Figure 8).

```
if(
  and(
    for(hotWaterSystem, or(sanitaryFixture, sanitaryAppliance)),
    include(hotWaterSystem, storageWaterHeater),
    l(storageWaterHeater.capacity, 700 l)
  )             lessThan
),              lessThanEqual
then            loop
  ob            x lessThanEqual(storageWaterHeater.capacity, 700 l)
)               x lessThanEqual(storageWaterHeater.capacity, 700l)
```

Figure 8: Dictionary completion

Since *hotWaterSystem* had a warning for not being in the LKM dictionaries, we searched for this term in the classification systems. Uniclass contains *waterHeaters* and *heatingSystems* while OmniClass includes *hotWaterHeaters* (see Figure 9). This indicates a suitable granularity, but more information, e.g. definitions for those classes, would help to decide whether to add *hotWaterSystem* or another term to the database. An information alignment step could be integrated with the dictionary management at this stage. Suggesting suitable mappings beyond the current search mechanism, e.g. by integrating Zhang and El-Gohary (2023), could support this task.

Uniclass	OmniClass
hotAndColdWaterSupplySystems	hotWaterHeaters
coldWaterSupplySystems	domesticHotWater
mediumTemperatureHotWaterHeatingSystems	residentialHotWaterDispensers
chilledWaterSystems	hotWaterTankHeaters
hotAndColdWaterSupply	commercialHotWaterDispensers
waterHeaters	hotWaterTankSteamHeaters
directHotWaterStorageSupplySystems	chilledWaterFacility
instantaneousHotWaterSupplySystems	domesticColdWater
waterSourceHeatPumpSystems	hvacSteamHotWaterConverters
waterCoolingSystems	instantaneousHotWaterHeaters
hotAndColdWaterSupplyDesignStrategy	waterSourceSplitSystemHeatPumps
lowTemperatureHotWaterHeatingSystems	fluidHeatExchangers
hotAndColdWaterSupplyPerformanceRequirements	waterSourcePackagedHeatPumps
indirectHotWaterStorageSupplySystems	heatExchangers
heatingSystems	heatPipes

Figure 9: Search for hotWaterSystem

Translator feedback

We collected user feedback from three researchers who used the LRML Editor to translate several clauses. The researchers had varying knowledge of the LRML syntax and the translation guidelines. Three main issues became evident, which should be resolved before conducting larger user studies.

1. Input validation beyond the dictionaries needs to be provided. In particular, the data entity (i.e. second function argument) does allow a variety of different datatypes and notations (e.g. value and unit, document reference, string value, and equations). The translator needs to receive stronger guidance and immediate feedback upon input.
2. The display of related translations was useful for users without knowledge of the LRML syntax. In contrast, for expert users, it can lead to an information overload. It is too time-consuming to comprehend a full translation. We plan to show translation snippets as an alternative setting.
3. Visualising the connection between clause phrases and LRML expressions was requested as a feature.

This would also be of use for translation snippets and cover a neglected requirement: Isomorphism (Palmirani et al., 2013). LRML allows for isomorphism by storing corresponding natural language phrases. Furthermore, this information could be used for training and evaluation purposes.

Discussion

The LRML editor's most important feature is the transformer-based model. In particular, the interplay between full translation and autocompletion promises efficiency. The generated initial translation can be used as a skeleton for the expert's translation. If changes are required, autocompletion can help with syntax and reduce manual efforts. Furthermore, the autocompletion options could inspire the translator on what relations should be included or how to continue the translation. Such inspirational effects could be increased significantly by utilising sampling strategies rather than beam search when retrieving additional options to increase diversity.

As shown in Figure 5, the current autocompletion options have high similarity, with only three distinctions. These distinctions indicate the aspects where the model was most unsure: 1) Use of disjunctions and conjunctions *and/or*, 2) Comparators *greaterThanEqual/lessThanEqual*, and 3) References *nzs_4305/nzs_4603*. Identifying the correct logical connections and relations is a critical and challenging task requiring further investigation to prevent the model from choosing the majority class. *nzs_4603* is referenced in one of the training samples. In contrast, *nzs_4305* was never seen before. This indicates the difficulty of deciding when to rely on learnt knowledge and when to copy entities from the current clause.

While the LRML Editor subjectively improved the translation workflow significantly over the proforma-based method, there is likely some researcher bias involved in this judgement. For more objectivity, we plan to conduct a user study with more objective measures. The user study will include three parts: 1) Translation interface baselines, 2) Translation and autocompletion improvements over time, and 3) Non-developer user feedback.

Especially the second part will be crucial to investigate the effect size of additional training data and to provide evidence for the gradual improvements achievable with transformer-based models and, with it, the advantage of our approach over rule-based methods, such as Wu et al. (2023).

Conclusion

In this paper, we proposed a novel LRML Editor, which supports efficient translation of building codes and standards into a semantic representation usable for automated compliance checking. At the heart of the editor lies a transformer-based model to provide both full translations and context-dependent autocompletion options. The LRML Editor stands out from previous semi-automated translation tools by providing more extensive translation support, thus reducing the manual workload. With the editor, rule authors have the means to create a computable representation of building regulations without much overhead, and data scientists could produce training data more efficiently, resulting in higher-quality fully-automated translation.

Acknowledgments

This research was funded by the University of Canterbury's Quake Centre's Building Innovation Partnership (BIP) programme, which is jointly funded by industry and the Ministry of Business, Innovation and Employment (MBIE).

References

Amor, R. and Dimyadi, J. (2021). The promise of automated compliance checking. Developments in the built environment, 5.

Barraclough, T., Fraser, H., and Barnes, C. (2021). Legislation as code for new zealand: Opportunities, risks, and recommendations. Technical report, Brainbox Ltd.

Bhuiyan, H., Olivieri, F., Governatori, G., Islam, M. B., Bond, A., and Rakotonirainy, A. (2019). A methodology for encoding regulatory rules. In MIREL@ JURIX.

buildingSMART (2017). Regulatory Room Report on Open Standards for Regulations, Requirements and Recommendations Content. buildingSMART Standards Summit 2017 in Barcelona.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

Dimyadi, J. and Amor, R. (2013). Automated building code compliance checking–where is it at. Proc. of CIB WBC, 6(1).

Dimyadi, J., Fernando, S., Davies, K., and Amor, R. (2020). Computerising the new zealand building code for automated compliance audit. In Proc. – New Zealand Built Environment Research Symposium (NZBERS).

Dimyadi, J., Pauwels, P., and Amor, R. (2016). Modelling and accessing regulatory knowledge for computer-assisted compliance audit. Journal of Information Technology in Construction (ITcon), 21(21):317–336.

Eastman, C., Lee, J.-m., Jeong, Y.-s., and Lee, J.-k. (2009). Automatic rule-based checking of building designs. Automation in construction, 18(8):1011–1033.

Fuchs, S. and Amor, R. (2021). Natural language processing for building code interpretation: A systematic literature review. In Proc. of the Conference CIB W78, volume 2021, pages 11–15.

Fuchs, S., Dimyadi, J., Witbrock, M., and Amor, R. (2023). Training on digitised building regulations for automated rule extraction. In eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2022. CRC Press.

Fuchs, S., Witbrock, M., Dimyadi, J., and Amor, R. (2022). Neural semantic parsing of building regulations for compliance checking. In IOP Conference Series: Earth and Environmental Science, volume 1101.

Hjelseth, E. and Nisbet, N. (2011). Capturing normative constraints by use of the semantic mark-up rase methodology. In Proc. of CIB W78-W102 Conference.

Ilal, S. M. and Günaydın, H. M. (2017). Computer representation of building codes for automated compliance checking. Automation in construction, 82:43–58.

Lau, G. and Law, K. (2004). An information infrastructure for comparing accessibility regulations and related information from multiple sources. In Proc. of International Conference on Computing in Civil and Building Engineering, ICCCBE, Weimar 10.

Li, F., Song, Y., and Shan, Y. (2020). Joint extraction of multiple relations and entities from building code clauses. Applied Sciences, 10(20):7103.

Ministry of Business, Innovation and Employment (2022). Acceptable Solution H1/AS1 For New Zealand Building Code Clause H1 Energy Efficiency, volume 5th edition, amendment 1.

Nazarenko, A., Levy, F., and Wyner, A. Z. (2016). Towards a methodology for formalizing legal texts in legalruleml. In JURIX, pages 149–154.

Niemeijer, R. A., de Vries, B., and Beetz, J. (2014). Freedom through constraints: User-oriented architectural design. Advanced Engineering Informatics, 28(1).

Palmirani, M., Cervone, L., Bujor, O., and Chiappetta, M. (2013). Rawe: a web editor for rule markup in legalruleml. In CEUR workshop proceedings, volume 1004.

Palmirani, M., Vitali, F., and Cervone, L. (2023). Lime: The language independent markup editor. http://lime.cirsfid.unibo.it/. Accessed: 2023-01-20.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084.

Salama, D. M. and El-Gohary, N. M. (2016). Semantic text classification for supporting automated compliance checking in construction. Journal of Computing in Civil Engineering, 30(1):04014106.

Shi, L. and Roman, D. (2017). From standards and regulations to executable rules: A case study in the building accessibility domain. In Proc. of the Industry Track @ RuleML & RR 2017.

Soliman-Junior, J., Formoso, C. T., and Tzortzopoulos, P. (2020). A semantic-based framework for automated rule checking in healthcare construction projects. Canadian Journal of Civil Engineering, 47(2):202–214.

Song, J., Kim, J., and Lee, J.-K. (2018). Nlp and deep learning-based analysis of building regulations to support automated rule checking system. In ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction, volume 35.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

Wu, J., Xue, X., and Zhang, J. (2023). Invariant signature, logic reasoning, and semantic natural language processing (nlp)-based automated building code compliance checking (i-snacc) framework. Journal of Information Technology in Construction (ITcon), 28(1).

Yurchyshyna, A., Faron-Zucker, C., Le Thanh, N., and Zarli, A. (2010). Adaptation of the domain ontology for different user profiles: Application to conformity checking in construction. In Web Information Systems and Technologies: 5th International Conference.

Zhang, J. and El-Gohary, N. M. (2016). Semantic nlp-based information extraction from construction regulatory documents for automated compliance checking. Journal of Computing in Civil Engineering, 30(2).

Zhang, R. and El-Gohary, N. (2019). A machine learning-based method for building code requirement hierarchy extraction. In Proc. - Annual Conference-Canadian Society for Civil Engineering, pages 1–10.

Zhang, R. and El-Gohary, N. (2023). Transformer-based approach for automated context-aware ifc-regulation semantic information alignment. Automation in Construction, 145:104540.

Zhang, Z., Nisbet, N., Ma, L., and Broyd, T. (2023). Capabilities of rule representations for automated compliance checking in healthcare buildings. Automation in Construction, 146:104688.

Zhou, P. and El-Gohary, N. (2017). Ontology-based automated information extraction from building energy conservation codes. Automation in Construction, 74.

Zhou, Y.-C., Zheng, Z., Lin, J.-R., and Lu, X.-Z. (2022). Integrating nlp and context-free grammar for complex rule interpretation towards automated compliance checking. Computers in Industry, 142:103746.