TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

MASTER-/DIPLOMARBEIT

Formalisierung von Bauvorschriften für die automatisierte Überprüfung von BIM Modellen

AUSGEFÜHRT ZUM ZWECKE DER ERLANGUNG DES AKADEMISCHEN GRADES EINER DIPLOM-INGENIEURIN UNTER DER LEITUNG VON

AO.UNIV.PROF.DIPL.-ARCH.DR.PHIL
GEORG SUTER

E 259-1
ABTEILUNG FÜR DIGITALE ARCHITEKTUR UND RAUMPLANUNG
INSTITUT FÜR ARCHITEKTURWISSENSCHAFTEN

EINGEREICHT AN DER
TECHNISCHEN UNIVERSITÄT WIEN
FAKULTÄT FÜR ARCHITEKTUR UND RAUMPLANUNG

BRIGITTE BLANK-LANDESHAMMER

00627085

WIEN, AM ____________________ ____________________________

Acknowledgements

First, I want to sincerely thank my supervisors, Prof. DI Dr. Georg Suter from the University of Technology in Vienna and DI Dr. Gerhard Zucker from the Austrian Institute of Technology for all their support, advice, feedback, guidance and encouragement throughout this journey.

I also want to thank the Austrian Institute of Technology, in particular the Competence Unit “Sustainable Thermal Energy Systems” at the Center for Energy and my colleagues there for providing me with the opportunity to write this thesis and supporting me in the process.

I am also very grateful to the company A-Null Bausoftware and especially Hannes Asmera, formerly of A-Null Bausoftware, and Tamás Kurucsó, for not only providing me with a license for Solibri Office, without which this work would not have been possible, but also for offering help and guidance when working with the software, sharing insights of previous projects and allowing me to use their demo house as one of the test models for this work.

Last but not least I want to thank my family and friends for their encouragement, their patience and for always supporting me.

Kurzfassung

Pläne und Gebäudemodelle in der Architektur und Bauindustrie werden in weiten Teilen der Welt manuell auf die korrekte Einhaltung der lokalen baulichen Bestimmungen und mögliche Fehler geprüft. Um diesen zeitaufwändigen Prozess zu automatisieren und somit auch zu beschleunigen, ist es notwendig, dass die relevanten baulichen Richtlinien und Bestimmungen nicht nur von Menschen, sondern auch von Computern verstanden und überprüft werden können. Um dies möglich zu machen, bedarf es einer Formalisierung der relevanten Bestimmungen.

Im Zuge dieser Arbeit wird zum einen darauf eingegangen, wie eine solche Formalisierung von Baurichtlinien in Bezug auf Maschinenlesbarkeit aussehen kann, und welche Möglichkeiten bisher erforscht wurden, vergleichbare Rechtstexte für Computer verständlich zu machen. Auch die historische Entwicklung der automatisierten Überprüfung von Plänen und Gebäudemodellen wird dargestellt, um den Fortschritt und Wissenszuwachs in diesem Gebiet in einen Kontext zu setzten. Außerdem werden Beispiele von bereits existierenden Versuchen und Lösungen zur automatisierten Überprüfung auf Konformität mit lokalen Richtlinien aus verschiedenen Ländern und Regionen recherchiert. Beispielhaft werden ausgewählte Paragraphen der österreichischen OIB Richtlinien formalisiert und dann mit Hilfe der Software Solibri Office als Prüfregeln implementiert, um auf die Möglichkeiten einzugehen, wie solche Regeln generell erstellt werden können. Durch die manuelle als auch automatische Überprüfung von Testmodellen mit Hilfe dieser Prüfregeln soll gezeigt werden, wie eine Automatisierung der Konformitätsprüfung in Österreich aussehen könnte und inwieweit diese derzeitig überhaupt möglich ist oder von technischen sowie sprachlichen Faktoren behindert oder erschwert wird.

Schlagwörter: Computational Design, BIM, Building Information Modeling, Automatisierte Überprüfung

Abstract

Plans and building models in the architecture and construction industry are manually checked in many parts of the world for correct compliance with local building regulations and for possible errors. In order to automate and thus accelerate this time-consuming process, it is necessary that the relevant building guidelines and regulations can be understood and verified not only by people but also by computers. In order to make this possible, a formalization of the relevant regulations is necessary.

In the course of this work, this thesis deals, on the one hand, with how such a formalization of building regulations can look like in terms of machine readability, and on the other hand, which possibilities have been explored so far to make comparable legal texts understandable for computers. The historical development of automated verification of plans and building models is presented in order to put progress and knowledge growth in this area into context. Furthermore, examples of existing experiments and solutions for automated checks for compliance with local guidelines from different countries and regions are researched. In order to explore the possibilities and challenges of creating code-checking rules, selected paragraphs of the Austrian OIB guidelines are formalized and then implemented as test rules using the software Solibri Office. The manual and automatic verification of test models supported by these test rules shows how automated conformity testing in Austria could look like and to what extent this is currently feasible or limited by technical and linguistic factors.

Keywords: Computational Design, BIM, Building Information Modeling, Automated Compliance Checking

Contents

1. Introduction

Architects, civil engineers, structural designers, and other technicians increasingly work with digital building information models in the design process of a building. Those models are not only used for better communication and information exchange between the designing parties, but could also be used to automatically check if the proposed buildings fulfil certain requirements with very little additional expenditures for them. This could also offer a significant decrease of time and effort needed for the authorities and designers when checking the projects for compliance with building codes when issuing or applying for building permits.

However, automated compliance checking requires, among other things, the translation of legislations and guidelines relevant for a certain building project to executable code. As Beach and Rezgui (2018) state, this is usually not only difficult because experts from various fields are required, including experts in building regulations, software engineers and experts in the most widely used data-storage formats in construction, but also because of the regulations themselves. They are usually not following one semantic standard, leave room for interpretation, and change regularly, which means the formalization has to be repeated or at least adjusted periodically (Beach and Rezgui, 2018).

While in some countries at least partial automatic compliance checks are already an intrinsic part in the application process for building permits, in many other countries the plans for proposed buildings are still checked manually (Doing Business 2019; Choi and Kim, 2017). This is changing however, as many of them are working on improving these processes. Overall, this offers new opportunities for the construction sector regarding faster processing time of applications by using already existing data material. One example is the current effort undertaken in Vienna, where the digital application for building permits ("digitale Baueinreichung") started in 2019.

The different approaches currently in existence are also reflected in the Doing Business report which is issued annually by the World Bank Group. It compares 190 economies regarding their regulations affecting eleven areas of life, including 'paying taxes', 'starting a business', 'getting electricity' as well as 'dealing with construction permits'. For the latter section building officers in those economies were interviewed in regards to steps, time and money required to get a building permit for a simple storage building in a local major city. Fiedler named Singapore as the fastest among the examined economies with a time of only 26 days until the building permit process was completed, based on the report from 2014 (Fiedler, 2015). This was still true for the report from 2016, but the following reports showed 48, 54 and 41 days respectively for 2017, 2018 and 2019. Denmark, Marshall Islands, Sweden, France, Germany and the UK are among the countries with the least number of steps required. Beside the actual step of applying for and getting the building permit itself, there are other steps which may include getting a soil test, hiring a qualified professional to apply for permits, obtain water and sewage connection or receive final inspection. In the latest report from 2019, the fastest is Korea with only 27.5 days (even though more steps are required), followed by Marshall Islands, Singapore, United Arab

Emirates and Malaysia. As the performance rank is not only based on those two factors, the overall top ranking looks slightly different, with Hong Kong SAR being number one, followed by Taiwan, Malaysia, Denmark, the United Arab Emirates, New Zealand, Lithuania, Singapore, Australia and Korea. Most of the mentioned economies already have a digital solution for the application for building permits in place or are currently working on one. Austria is currently ranked 42nd with 11 procedures and 222 days required without any such solution (Doing Business 2019).

As observed by Fiedler (2015), planners as well as planning officers think that the building permit process in Austria has been getting more complex in the last years. Especially the planners think that it takes too long, and a modernization would be very welcome. A majority of planning officers are also willing to learn new software tools if they made their work easier (Fiedler, 2015).
Additionally, the communication between planners and planning officers as well as with other authorities is an issue. Among other thing, the majority of both parties agree that pieces of legislation are not interpreted the same way by different planning officers, which leads to confusion and insecurity (Fiedler, 2015). If those pieces of legislation could be translated to automatically checkable code, the number of possible interpretations would be drastically reduced in the process of making natural language text interpretable by computers, which would lead to more certainty and stability for all parties.

In Austria about 30% of the OIB guidelines no. 2, 3 and 4 are currently checkable using Solibri Office (formerly Solibri Model Checker) with the pre-defined checking rules of the program according to Fiedler (2015). The remaining 70% are either not checkable due to vague phrasings in the guidelines or due to limitations of the predefined rules in Solibri Office (Fiedler, 2015). According to Hannes Asmera from A-Null Bausoftware, who implemented the rules in Solibri Model Checker (now Solibri Office) together with Fiedler in 2015, an update was made by A-Null covering about 50% of the guidelines already. With the newly released API Solibri provides, individual implementations of new rules are possible without relying solely on the predefined rules already in the software. This should lead to an increase of checkable content.

In a study regarding the potential of digitalization in the construction sector in Austria, Goger, Piskernik and Urban (2017) also identified possible benefits of using a digital building permit process such as faster application times, a potential harmonization of Austrian building codes, as well as non-ambiguous interpretations of pieces of legislation. They also identified possible issues, such as the act of implementing the building codes, possible adjustments to these codes, as well as receiving models of sufficient quality from the designers that have all the information necessary to check for compliance (Goger et al., 2017).

2. Aim of the Work

This work focuses on the following research questions:

- What is the state of the art regarding the current efforts for formalizing building codes and automatically checking models for compliance against those codes?
- How can selected paragraphs of OIB guidelines be formalized to facilitate automated compliance checks in existing code-checking software? More specifically, the following questions are investigated for the commercial code-checking software Solibri Office:
    - Are pre-defined rules available in Solibri Office sufficient for such checks?
    - To what extent does the recently released API from Solibri change the amount of paragraphs from the OIB guidelines that can be checked automatically with the Solibri Office application?

This thesis aims at giving an overview of the current efforts regarding the formalization process of building codes and automated compliance checking in different parts of the world. Furthermore, attempts are made to formalize paragraphs of OIB guidelines as an example for building code formalizations in general. Exemplary rules based on those formalizations are implemented using Solibri Office. These rules are then used to automatically check BIM models for compliance with parts of those guidelines. For further details regarding the software see chapter 3.3. Rules which are more difficult to formalize than others or currently not verifiable by computers, the reasons behind these difficulties, as well as possible solutions, are identified and discussed.

To answer the first research question, a literature review is conducted and the findings are presented in chapters 4 and 5:
Chapter 4 aims at providing a short introduction to Automated Compliance Checking and a historic overview of past efforts made. Additionally, current government initiatives and the utilized software frameworks to prepare for and conduct automated checks are researched, introduced and discussed.
Chapter 5 deals in more detail with the different methods of formalizing building codes or other similarly structured legal texts for a later automation of compliance checks. A focus is placed on formalizations of regulatory documents in the building and construction industry. Literature from related disciplines such as computer science is also researched to gain a better understanding of the methods, but the approaches discussed are limited to formalization methods previously used in connection with building codes.

For the second research question and sub questions, exemplary paragraphs of OIB guidelines are selected and formalized using formal logic. These formalizations are then used to implement rules in Solibri Office. To ensure the general validity of the approach, these examples are selected based on their structure and wording as well as previously found difficulties regarding checkability. The selection includes requirements in the form of a table with distinct values as well as requirements in natural language. Additionally, an example paragraph is selected for the ambiguous wording used, which makes it difficult to verify. The selection of exemplary paragraphs is described in detail in chapter 6.1.
In chapter 6.2 the method used for formalizing the selected examples is presented and in

chapter 6.3 the process of formalizing the example clauses is illustrated. Chapter 7 then discusses the further translation of these formalized examples to executable rules in Solibri Office. While some requirements can be checked with the pre-defined rules in the software, others need a custom implementation. This is also illustrated in this chapter. These custom implementations are generated using the JAVA API included in the software.

To validate the correctness of the implementations, example models are checked for their compliance with the generated rules in Solibri Office. In terms of data-storage formats, the implementation in this thesis focuses on IFC models. IFC was chosen as an open standard that serves as an exchange format and is implemented directly in many proprietary software suites or available as an additional module. It is also the primary import format for Solibri software applications. The models used for testing are presented in chapter 8. They were generated with different proprietary software frameworks to ensure that the implementation works with IFC models from different sources.
In chapter 9 the results of the checks are presented and explained.

3. Basic Concepts

3.1. BIM

BIM is short for Building Information Modeling, which describes a process focused on handling and managing the digital information of a building throughout its lifecycle. The idea behind BIM is to reuse the digital models created in different stages of this lifecycle, from the conceptual phase to the construction phase and then to the operation phase, instead of reducing complex 3D models to 2D drawings in order to pass them on, so that the next professionals can create their own 3D models based on these drawings again. A lot of information is lost in this process and a lot of time spent on redoing work that was already there before (Borrmann et al., 2018).

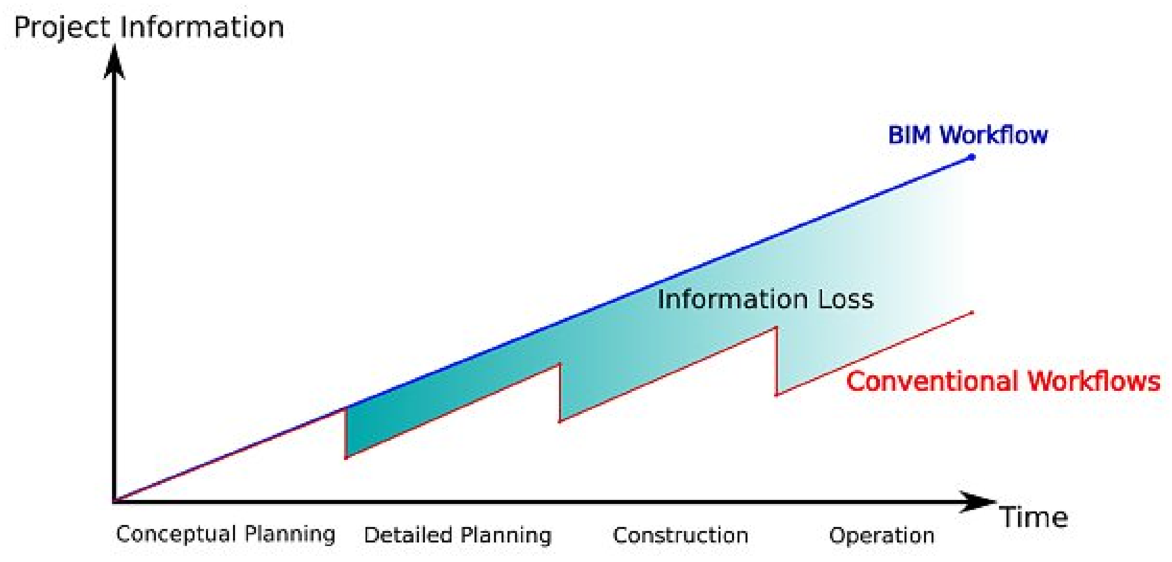


Figure 1: Information loss in conventional workflows
(based on Figure 1.1 from Borrmann et al., 2018, p. 3)

However, the number of different software solutions used in a building's lifecycle makes simply reusing a model difficult, as there are often no interfaces defined that let users easily convert a model from one software's format to another (Borrmann et al., 2018). This is a problem already within such software, like applications solely used for the conceptual phase, but taking the needs of other phases into account, no single software can yet handle all the information of a building at once and even software families still have their limitations. To address this problem, an exchange format was developed to ease the transfer of a model's data from one software to the next.

BIM was introduced already at the end of the last century. In the last 20 years a steady increase in its use and distribution could be observed. While some countries – like Singapore or Taiwan – are already quite far in their implementation of BIM throughout their construction and facility management sector, some have only started in recent years to pursue the goal of increasing the use of building information modeling (Holte Consulting, 2014).

3.2. IFC

IFC is short for Industry Foundation Classes and is an open standard to digitally describe a building and its environment. This can include the physical components used to construct a building as well as information regarding relations between different elements in a building and also information about how a facility is used (Industry Foundation Classes (IFC) - buildingSMART Technical, 2019).

The modeling language EXPRESS serves as the basis for IFC. Entities are used to describe individual object categories. An entity can have both subclasses and superclasses, making generalizations or inheritances possible. In addition, entities can also contain constraints, in addition to attributes, which act as a restriction for all elements of that class (Hudeczek, 2017).

Historically the standard goes back to an industry consortium formed by various companies including Autodesk in 1994, which evolved over the following few years and was reconstituted as a non-profit industry-led organisation in 1997. Since 2005 this organization is called buildingSMART and it still develops and maintains the specifications for IFC standards today. The current standard at the time of writing this thesis is IFC 4.1.0.0, which was published in June 2018. The predecessor of this, IFC 4.0.2.1, was also published as ISO publication ISO 16739-1:2018 (IfcWiki, 2019; buildingSMART - The Home of BIM, 2019).
IFC is also known as a file extension for STEP Physical File Format, which is the format most widely used for IFC in practice. Other formats used for IFC include XML, ZIP and JSON (buildingSMART, 2020).

3.3. Solibri Office and Solibri API

Solibri Office, formerly Solibri Model Checker (SMC), is a software developed by the Finnish company Solibri to view IFC models and check the model quality using pre-defined rules (Solibri Inc., 2020). This includes rules to check if certain elements or properties exist in the model, rules to check the size of or distance between elements, but also more complex rules such as an escape route analysis to validate if an exit can be reached from every room within a certain user-defined distance. The current version 9.10 of Solibri Office contains 55 pre-defined rules and nearly all of them can be customized by adjusting parameters or settings. These rules can also act as gatekeepers for other rules, so only the elements – called *components* in Solibri Office – that passed the first rule are then checked in the following one. Using this mechanism as well as adjusting the rule parameters are the only ways to build custom rules from inside the software itself. In 2019 Solibri released the Beta version of an API (Application Programming Interface) which can be used to realize custom rules outside the limits of the software's pre-defined rules. The API allows the programming of custom rules with additional features, settings or parameters using the JAVA programming language. These rules can then be imported into the software and used like any of the pre-defined rules. The API allows the implementation of completely new program code, so custom rules are not limited to the software's user interface or scope, the only limitation is that these pieces of software have to be run using the checking mechanism in Solibri Office. This means it is also possible to incorporate additional features into Solibri Office by using such rules, like an export functionality of the IFC model or any defined subsystem, but also to add user interactions to the checking process such as asking the user for additional information or verification during the checking process if this information cannot be found automatically in the model.

4. Automated Compliance Checking

There is a large number of guidelines and regulations in the architecture and construction industry, ranging over multiple lifecycle phases and covering multiple disciplines, to assure, among other things, a building's technical and structural soundness, accessibility, and safety measures. To make sure a building fulfils all the relevant requirements, in a conventional checking process, 2D drawings like floor plans and sections are checked manually. This often happens multiple times due to changes in the plans based on earlier feedback or due to adjustments from other disciplines. In general this process is time-consuming and error prone as it requires a detailed knowledge of the guidelines and regulations as well as a high level of attentiveness of the person checking the plans. However, this process can be improved and – at least partially – automated utilizing digital methods like Building Information Modeling to achieve a higher degree of compliance as well as to reduce the checking effort (Preidel and Borrmann, 2018).

In Austria the city of Vienna started accepting copies of building plans used to apply for a building permit in PDF or PNG/JPG format in 2019. One set of plans still has to be handed in on paper though. For Vienna this is a first step on the way to a completely digital building permit process (Baueinreichung - Mein Wien; Digitales Wien - Wien startet Digitale Baueinreichung). But while the replacement of conventional printed plans with digital copies is still rather new for Austria's authorities, other countries have had solutions in place for more than 20 years. They are used not only to accept digital copies but also to automatically check them for compliance with some of their building codes. The following chapter gives an overview of what automated compliance checking means, how the process is structured and which research efforts were made in the past. Different solutions currently in place around the world are introduced and discussed to give an idea of what is already possible and what is planned in the future.

4.1. Definition

Automated Compliance Checking refers to an automated process of verifying whether a model fulfils certain predefined rules based on relevant regulations or guidelines. In order for the automation of this process to work, all checked models need to conform to the same basic data structure so the required information can always be found at a standardized location (Tulke, 2018). To check if a building model is compliant with a certain building code each object in the model is systematically compared to a set of relevant constraints and the result of these comparisons is a list of objects not meeting one or more of the requirements (Dimyadi and Amor, 2013).

4.2. Process

The process of Automated Code Compliance Checking consists of four main steps, also shown in Figure 2:

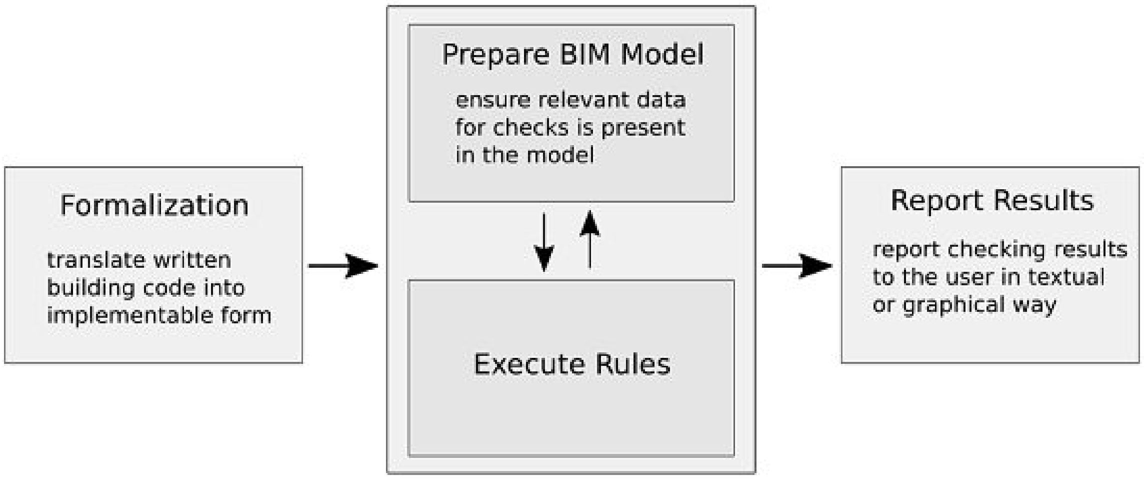

Formalization
translate written building code into implementable form
Prepare BIM Model
ensure relevant data for checks is present in the model
Execute Rules
Report Results
report checking results to the user in textual or graphical way

Figure 2: Process of Automated Compliance Checking (based on Figure 1 from Eastman et al., 2009, p. 6)

The first step is the translation of the rules into a computer-readable format. According to Preidel and Borrmann (2018) this is the core task as it serves as the starting point for automated code compliance checks. The two different approaches used here are so-called ”Black-Box methods” and ”White-Box methods”. In the former, only the inputs and outputs are visible to the user, the checking methods are directly integrated (”hard-coded”) into the software used for the checks, usually hidden from the user. This leads to a relatively low error rate on the one hand, but also to an increased difficulty when modifying rules as this is only possible with the help of the software vendor. Users may also distrust the implementation as they cannot verify what is actually happening in the software. In White-Box methods the internal workings are visible to the user and therefor also verifiable. To achieve this verifiability, the translations of the regulations must be both human- and computer-readable, which can require more effort from a developer’s side than hard-coding the rules (Preidel and Borrmann, 2018), especially if they should also be understandable by non-programmers.
Different methods used to formalize building codes are discussed in detail in chapter 5.

The second and third step are the execution of the implemented rules and the preparation of the building information model. In order to get meaningful results, not only the implementations of the translated guidelines must be correct, but also the information in the model has to be accurate, correct and consistent. This can be ensured by utilizing a pre-processing step to check the data in the model and prepare it accordingly (Preidel and Borrmann, 2018).

The final step is the preparation and representation of the results in a human-readable way so that the person responsible can easily understand the results in order to deal with any part of the model not compliant with the checked building codes (Preidel and Borrmann, 2018).

4.3. Rule Classifications

In order to further understand the difficulties and to solve some of the problems when formatting rules to check against, Solihin and Eastman (2015) classified the different kinds of rules in four general categories, according to their processing complexity: Class 1 rules which check for explicit attributes in a model, Class 2 rules which require derived attributes to check against, Class 3 rules which require an extended data structure for complex requirement checks, and Class 4 rules which require the finding of a specific solution in order to decide whether or not a model is compliant with a regulation (Solihin and Eastman, 2015).

Class 1 rules are usually the easiest to check against. They include simple building code checks such as “Elevators must exist in buildings with more than a certain number of floors.”, in which case the rule checks for the existence of certain entities – in this case an elevator – in the model. Other checks are aimed at the quality of models to be handed over from one profession to another according to previously agreed upon requirements. This may include something like “Every IFCSpace must have the attribute “HandicapAccessible” set to either true or false.”, in which case the rule checks for correct values of certain attributes of building elements in the model. Many class 1 rules act as a base for more complex rules, such as checking for the existence of values of certain parameters which are then needed in a further step to derive other information on the model (Solihin and Eastman, 2015).

In class 2 rules data is derived either by the user or by the logic in the rule itself instead of being available as a value in the model. This can include checks like whether or not an element is contained within a certain area or the shortest distance between two elements in the model, which can be calculated based on the available geometric information (Solihin and Eastman, 2015).

Class 3 rules often involve the need for a higher level of semantics for building model elements and advanced topological modeling and knowledge of spatial relations within the model. Rules in this category include checks for the proper spacing of certain elements such as smoke sensors or sprinkler systems, which need not only be present in rooms, but also achieve a certain coverage and therefore require minimum and maximum distances between each other. In order to check these distances, a knowledge of the nearest neighbours for each of those detectors is required. This category also includes rules such as escape route calculations, which do not simply need to check for a certain attribute of an element, but to calculate a specific solution for a problem. In the case of escape routes, an exit door must be reachable from every point in each room in the building within a certain distance (Solihin and Eastman, 2015).

Class 4 rules are not significantly more complex than the rules in class 3, but they involve the need of additional information in order to find a specific solution for a given problem that needs to be solved. For example, this can include the calculation of an access path to certain room in order to move a large object into or out of this room, which is not necessarily possible through every door in the building. Checks based on performance based building codes are typically also considered class 4 rules (Solihin and Eastman, 2015).

4.4. Historic Overview of Past Efforts

Research concerning formalization of building codes and automated compliance checking goes back as far as the 1960ies, when Fenves introduced decision tables as a method of structuring pieces of building regulation (Eastman et al., 2009; Macit İlal and Günaydın, 2017; Solihin and Eastman, 2015). Based on this research, AISC (American Institute of Steel Construction) specifications were described utilizing a network of decision tables in 1969 and SASE (Standard, Analysis, Synthesis and Expression) was developed in 1984 as a management and maintenance system for decision tables. It served as base for two compliance checking applications: SICAD (Standards Interface for Computer Aided Design) to help a user extract information from standards, and SPEX (Standards Processing Expert), a knowledge-based system to check component structure, geometry and material (Dimyadi and Amor, 2013).

In the 1980ies hyper-document modeling was introduces as another approach to make building codes computer readable (Dimyadi and Amor, 2013).

Expert systems and knowledge-based systems became popular during the 1990ies as a first working implementation of artificial intelligence (AI) approaches. During this time many design checking and analysis efforts used hard-coded interpretations of building regulations. Although still used today, many such solutions allow a minimum of user customization through adjustable parameters (Dimyadi and Amor, 2013). In recent years, research focused on the use of Artificial Intelligence and Natural Language Processing techniques or semantic modeling (Dimyadi and Amor, 2013).

Some countries developed national solutions to achieve a larger level of automation in the checking process. This includes Singapore, which started developing their CORENET (Construction and Real Estate Network) project in the 1990ies. It was developed as a building plans expert system to check 2D plans for compliance and got updated to work with 3D representations in 2002. Another example is Norway, which developed the Express Data Manager (EDM) Suite in 1998 as a collaboration tool. EDM later also received additional modules, including a model checker. Australia also had an approach called BCAider (Building Code of Australia Aider), an expert system developed by the Commonwealth Scientific and Industrial Research Organisation (CSIRO) that was released in 1991 to check for compliance with Australia's building code. It was available until 2005 (Dimyadi and Amor, 2013).

In the United States SMARTCodes was developed in 2006 and maintained until 2010 by the International Code Council (ICC). It is based on a mark-up concept and provides official representations of a number of standards as well as an authoring tool to manage alterations to the regulations (Dimyadi and Amor, 2013).

A rough timeline of the efforts made since 1980 is displayed in Figure 3. For reference, the time of release of widely used frameworks in architecture and construction, such as AutoCAD, ArchiCAD and Revit, are also included.

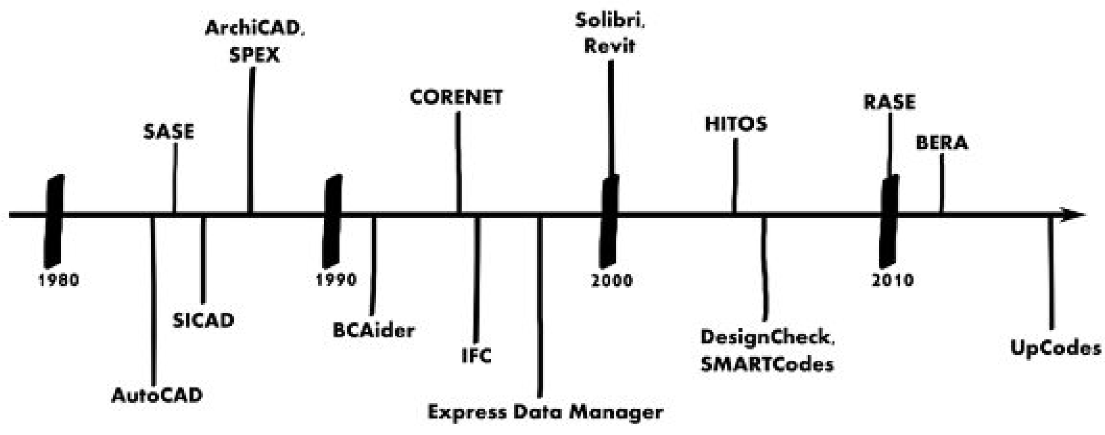


Figure 3: Timeline of mentioned Automated Compliance Checking research and relevant technology since 1980

4.5. Government Initiatives

The following examples should illustrate the state of automated compliance checking solutions in different regions around the world and highlight the efforts in the field of automation in the building and construction industry.

4.5.1. Korea

Lee et al. (2016) conducted a study in the course of which 15.000 building-permit related sentences were translated into computer executable form to use for compliance checking. In a first step sentences from the Korean Building Act were analysed and the objects and predicates refined and stored in a "logic meta database". Then the objects and their properties as well as the methods for verifying their correctness were classified and the natural language sentences in the building code were translated following a logical process. The sentences in the logic meta database are referred to as "KBimCode", with the application "KBimLogic" used to manage these sentences. It also allows users to define new content and modify existing entries. "KBimAssess" uses the sentences to check designs for compliance with the given requirements. KBimCode sentences are reusable for multiple projects and there is also a test system for already implemented building codes which was set up in the Korean government network (Lee et al., 2016). In another project the generation of KBimCode was expanded to work with a visual language approach, which should act as a more natural and intuitive way for users to generate KBimCode (Kim et al., 2017; Kim et al., 2019). In order to ensure a good quality of BIM models a quality evaluation system was tested in a study using KBimCode rules and KBimAssess for checking, to enable designers to test their models before an actual application for a permit (Park et al., 2019).

4.5.2. Singapore

Singapore's CORENET (Construction and Real Estate Network) started as a research group in 1993 as part of a nation-wide study examining the use of information technology to enhance the quality of life in Singapore, aiming for the country's technological advancement. The goal of this research group was to re-think the business processes in the construction industry in order to make them better and faster (Goh, 2008). The CORENET project was officially launched in 1995 and the system was progressively developed and implemented throughout the years (Foo Sing and Zhong, 2001). The initial research goals were accomplished by providing infrastructure and services in order to help stakeholders in the construction industry with information gain and communications, both between businesses and the government, as well as between different businesses. New standards for business communications were introduced and additional trainings helped professionals adapt to the new processes. One such communication service between businesses and government is an automated check of submitted plans (Goh, 2008). This was done using 2D plans at the start, but with the rise of 3D modeling in the architecture and construction industry, CORENET services were adapted to check 3D models already as early as 2002 (Dimyadi and Amor, 2013). In 2016 the electronic submission was extended to include complete BIM models, first for architectural models only and a year later also for structural as well as mechanical, electrical and plumbing models. Codes of Practice, templates for widely used software suites and additional guidelines for the handling and submission of BIM models are also found on the CORENET website (CORENET Building Information Modeling (BIM) e-Submission).

The whole system is based on PAVOT, a Java Enterprise application. The checking module of CORENET, called e-planCheck, is built on FORNAX (Thompson, n.d.). In order to add additional information and semantics relevant to the checking, building elements are encapsulated in FORNAX objects with additional attributes (Martins and Monteiro, 2013).

Today the CORENET system acts as a so-called "one-stop-shop" for building information and communication with various different public sector organisations participating (Thompson, n.d.).

4.5.3. Norway

The ByggSøk Project was initiated in 2000 and aimed at improving nationwide standardization of building applications in Norway (Martins and Monteiro, 2013). A first version, which allowed filling out building permit applications, was launched in 2003 and further features, such as adding plans and drawings as attachments, were added over time. The software itself is based on EXPRESS Data Manager (EDM), and acts as an object database. For further digitalization efforts a new strategy, ByggNett, was developed, which aims at providing a solution for processing building applications and not just managing them (Hjelseth, 2015).

For a Norwegian housing project, "Munkerud", an adapted version of Singapore's e-PlanCheck was tested regarding the possibility of automated compliance checks (Lee et al., 2016). In another project, the compliance checking of spatial and accessibility requirements was executed using Solibri Model Checker. It was called the HITOS project for the Tromsø University College, which was used as a pilot project in the efforts to extend IFC to the whole project lifecycle. As no IFC with additional data can be generated from the implemented systems, the generation and adaption of the IFC files with all relevant data are left to proprietary software solutions (Greenwood et al., 2010). For the compliance checks methods for translating legislations to a format usable by rule-checking software were developed. The process of this translation ranges from the identification of scope and computability assessments to logic notations and finally the implementation in the software (Lee et al., 2016).

4.5.4. Australia

BCAider was an expert system to check for compliance with Australian building codes until 2005. It was developed in the 1990ies by the Commonwealth Scientific and Industrial Research Organisation (CSIRO) as a commercial tool (Holte Consulting, 2014). DesignCheck succeeded BCAider in 2006. It is based on EXPRESS Data Manager (EDM), a collaboration software solution which uses the EXPRESS language, and extends IFC, which is based on the EXPRESS language also, with additional information using an internal model based on the IFC model (Holte Consulting, 2014; Preidel, 2014). This allows the use of information other than that implemented in proprietary software generating the IFC file (Greenwood et al., 2010). Building codes are implemented using the EXPRESS language and written into the model as objects. The translation of legislation to executable rules is often done with an intermediate pseudo-code step and then implemented manually, which requires great deal of expertise (Preidel, 2014). Rule schemas for different design stages are available, allowing compliance checks during these design phases already, but viewing 3D models or generating graphical reports is not possible in the software (Greenwood et al., 2010). According to Holte Consulting (2014) a commercial version of DesignCheck never became available, however.

4.5.5. UK

As part of the UK government Construction Strategy from 2011, the requirement of 3D BIM by 2016 was planned. COBie (Construction Operations Building information exchange), developed in the United States of America since 2005, is adapted to UK standards and used. The COBie format is a requirement for BIM models in the United Kingdom since 2016 (Holte Consulting, 2014).

4.5.6. Other European Countries

Electronic application for building permits should have been implemented in all European countries as part of the European Commission’s primary e-Government services by as early as 2005. This date was then extended to 2010 and later to 2020 as many countries failed to provide solutions to this challenge (Bellos et al., 2015). Such solutions often act as a first step in the direction of automated compliance checking of local building codes.

The European e-Government Benchmark (European Commission, 2020) acts as an indicator for quality and implementation of e-Government services in the European Union. While the 2020 reports don’t speak of building permits and compliance checks specifically for each country, they give a general overview over the availability of similar services.

Austria is rated among the top providers for e-Government services (European Commission, 2020). The legal aspect of such a service for building permits in Austria was researched and discussed by Sonntag and Wimmer (2003).

In Greece a case study regarding an electronic building permission system was conducted in 2015, which discussed a project named "e-Poleodomia" as a Greek solution. The study identified a number of challenges that need to be addressed in the project and the splitting into three main parts, dealing with management and control of permits, searching and indexing for laws and regulations and the development of a Geographic Information System (GIS) respectively. While the first part was eventually finished, the project as a whole was not completed for various reasons such as insufficient design effort of the platform, but also the impossibility of easily processing projects possibly not compliant with local law (Bellos et al., 2015).

In Spain the relevant forms for applying for a building permit are available online for most regions, and some cities already offer a complete application for building permits electronically (Bellos et al., 2015). The Netherlands did not offer any electronic building permit applications at the time of the study from Bellos et al. (2015), but it is mentioned that the progress of a permit application can be tracked in some regions (Bellos et al., 2015). The 2020 e-Government Benchmark Background Report mentions the availability of dealing with construction permits online in Croatia but does not give any further details (European Commission, 2020). However, in those countries no automated compliance checking system known to the author is currently in place.

Portugal offers the possibility to check compliance of projects with the Portuguese domestic water systems regulations using an application called LicA, developed at the University of Porto. While the scope may be limited, this application uses IFC as an exchange format and acts as an example of future possibilities when implementing checks for other regulations (Martins and Monteiro, 2013). The application relies on a SQL Database in the background with input tables, output tables and tables containing textual descriptions for report generating. A graphical user interface was developed for easier interactions as a separate project. Before code compliance checking, a preliminary check has to be passed to ensure model integrity and quality. Additionally, hydraulic analysis computations can be requested from the system, with the results including flow and pressure information (Martins and Monteiro, 2013).

A research paper by Bus et al. (2018) discussed the possibilities of French building code compliance checking using semantic Web technologies on the example of fire safety and accessibility regulations (Bus et al., 2018). As an intermediate step between the legal text and the actual query to check for compliance, the regulations are translated to a semi-formal representation understandable by expert groups, construction experts and data scientists alike. Bus et al. (2018) point out that their approach cannot generate correct results for all constraints in the regulations, sometimes due to approximations, for example using bounding boxes, and sometimes due to a lack of specification in IFC for some needed information (Bus et al., 2018).

4.6. Discussion

While automated compliance checking can save a lot of time and effort, the responsibility of deciding whether a building is compliant with effective legislation still lies in the hand of humans and cannot be handed to machines. This means that even given the possibility for automated compliance checking, the results have to be treated with caution and additional manual checks may still be needed. Accordingly, Preindl and Borrmann advise the implementation of semi-automatic compliance checks using the White-Box method in favour of fully automated checks (Preidel and Borrmann, 2018).

Building permit application processes differs a lot between regions (Holte Consulting, 2014) and the building codes themselves are also highly local and can differ greatly between cities, regions and nations. For this reason, the solutions for automated compliance checking may use the same underlying principles and methods, but the actual implementations still have to be done on a local level.

5. Formalization of Building Codes

The formalization of building codes refers to the first step in the automated code compliance checking process (as introduced in chapter 4): the translation of building legislations to a machine-readable format. In the following chapter a number of methods are introduced and discussed that can be used to achieve this.

The constraints imposed by different regulations can be represented in a graphical way, as a parameter table or as a text (Preidel and Borrmann, 2018). The tables are easier to translate in a computer-readable format in most cases as there is usually very little room for interpretation in comparison to a textual description and they mostly contain discrete values for the parameters that can be verified.

5.1. Logic-Based Interpretation

In classical logic, statements can be formalized by means of mathematical operators. The statements are first broken down into elementary statements that cannot be further simplified. These elementary statements should always be formulated positively, a negation is achieved using the logical operator *not*. Several elementary statements can be combined to form more complex statements using various logical operators, such as *and* or *or*. Every formula thus formed is either *true* (it is valid) or *false* (it is not valid) and the result of every complex statement is unambiguously derived from the results of its partial statements. While propositional logic deals only with elementary propositions and their relations, in predicate logic so-called quantifiers are added, with the help of which statements like ‘for all’ or ‘for some’ can be formalized.

However, the suitability of logic for the formalisation of regulations in the building industry is limited according to Hudeczek (2017). While the verification of the truth content of logical statements is very simple, the logical structures can become very hard to understand if the statements are more complex. This is especially relevant for the maintainability of formalized rules as opposed to re-formalizing them after every small change (Hudeczek, 2017).

To address this issue, Zhang (2017) investigated the connection of a logic-based representation of building regulations with a tree-based visualization method. The visualization proposed in this research significantly improves the readability and understandability of logic-based building code representations, achieving a reading speed comparable to the original text. However, the requirements formalized in this research were still relatively simple, the readability of more complex requirements using this method of visualization still need further investigation (Zhang, 2017). A comparison between the traditional representation as logical formula and the proposed tree-based representation is shown in Figure 4.

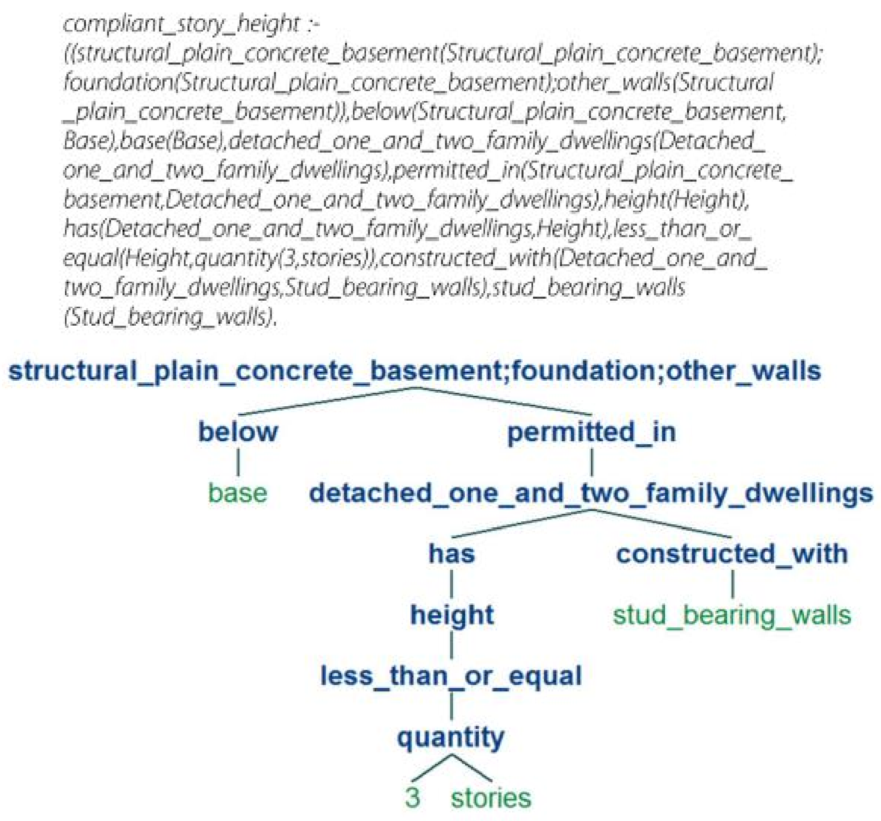

compliant_story_height :-
((structural_plain_concrete_basement(Structural_plain_concrete_basement);
foundation(Structural_plain_concrete_basement);other_walls(Structural
_plain_concrete_basement)),below(Structural_plain_concrete_basement,
Base),base(Base),detached_one_and_two_family_dwellings(Detached_
one_and_two_family_dwellings),permitted_in(Structural_plain_concrete_
basement,Detached_one_and_two_family_dwellings),height(Height),
has(Detached_one_and_two_family_dwellings,Height),less_than_or_
equal(Height,quantity(3,stories)),constructed_with(Detached_one_and_
two_family_dwellings,Stud_bearing_walls),stud_bearing_walls
(Stud_bearing_walls).

Figure 4: Textual logic representation versus tree-based visualization (Figure from Zhang, 2017, p. 7)

5.2. Semantic Encoding

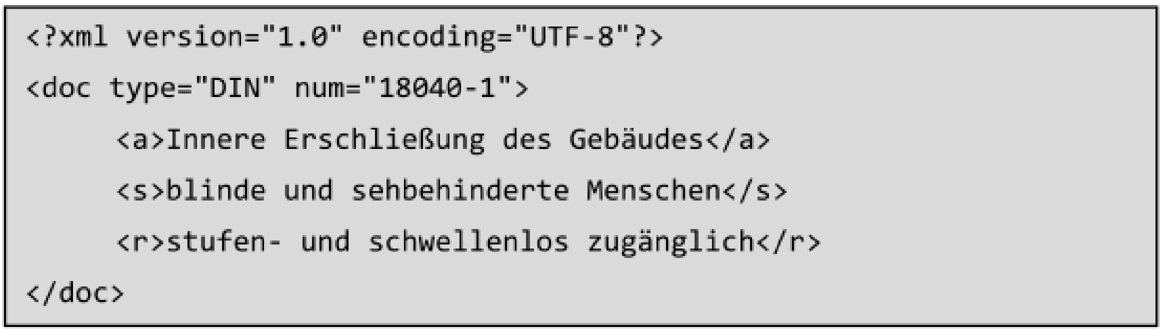
The language-based approaches include XML (eXtensible Markup Language), on the basis of which approaches such as RASE (Requirement, Application, Selection and Exception) (Hjelseth and Nisbet, 2010), and BERA (Building Environment Rule and Analysis) (Lee, 2011) are also created.

Markup languages belong to the category of formal languages in computer science and structure data using a strict syntax. They were initially developed for text documents but are not limited to textual data. As the name suggests, content is annotated through inserting *marks* or *tags* into the content itself (Hudeczek, 2017; Liu and Özsu, 2009).

The first version of XML was published in 1998 and is still widely used today as an open standard. As a metalanguage, XML can be used not only to structure data, but also to define application-specific languages based on the same grammatical rules. The information in XML-based languages is put into so-called *tags*, which can have any defined name (Hudeczek, 2017). While some applications require tags for personal information such as *last name* or *address*, others use tags such as *component* or *date of manufacture*. The information contained in the tag is placed between a start tag and an end tag and can be formatted as desired. Further elements with their own tags can also be included as part of the information. This creates a tree-like hierarchical structure witch the enclosing tags acting as parent elements to the contained ones.

Additionally, tags can be extended using attributes, which are written into the tag itself to make a descriptive statement about the element. In some cases it makes sense to replace child elements with attributes or vice versa, depending on the context and preference (Hudeczek, 2017).

An example of such a notation is shown in Figure 5.



```
<?xml version="1.0" encoding="UTF-8"?>
<doc type="DIN" num="18040-1">
     <a>Innere Erschließung des Gebäudes</a>
     <s>blinde und sehbehinderte Menschen</s>
     <r>stufen- und schwellenlos zugänglich</r>
</doc>
```

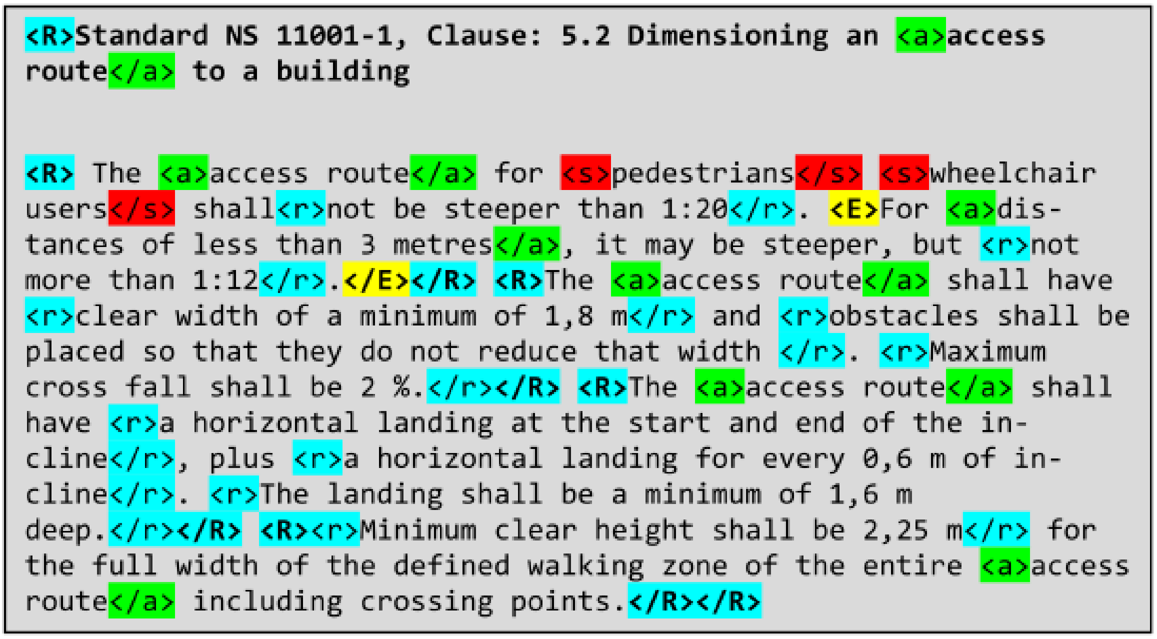
Figure 5: Example of an XML-based notation to structure data (Figure from Hudeczek, 2017, p. 20)

BERA is designed as a domain-specific programming language to deal with building information models while still remaining relatively easy to use for non-programmers. It also acts as a rule-checking language for these models, even from an early design phase on (Lee, 2011; Lee et al., 2015).

SMARTCodes was launched in 2006 as a project by the ICC (International Code Council) and the company AEC3. The developed markup language enables the users and not only very few domain experts to formalize rules and regulations themselves after a short training period. In SMARTCodes, passages in a regulation are marked and highlighted in colour. This highlighted text can be used to generate a requirements model and check it against IFC models. Based on the four main operators available – Requirement, Application, Selection and Exception – this type of highlighting has been named RASE. The current version of the syntax is developed by AEC3 (Hudeczek, 2017).

In order to formalize a guideline, in a first step the regulations are structured into a hierarchy. This hierarchy looks similar to the original document structure in most cases. Simple paragraphs result in true or false, more complex paragraphs have their results defined through the contribution of their sub-paragraphs (Beach and Rezgui, 2018).

Regarding the encoding, one approach is to tag each paragraph using RASE. For each part the Application tag identifies the scope in which the rule should apply (Hjelseth and Nisbet, 2011). The Selection tag is used to increase the scope which was limited by the application tag according to Beach and Rezgui (2018). Hjelseth and Nisbet (2011) and Hudeczek (2017) use this tag to describe objects or persons affected by the regulation. While a whole paragraph acts as a requirement for a certain domain or building type, the requirement tags within a paragraph highlight the values that must be met in order to comply with the given regulation. When there is an exception to any given paragraph, the exception tag is used. It may also contain additional application, selection and requirement tags (Hjelseth and Nisbet, 2011; Hudeczek, 2017). An example of a regulation tagged in the described way is shown in Figure 6.



```
<R>Standard NS 11001-1, Clause: 5.2 Dimensioning an <a>access
route</a> to a building


<R> The <a>access route</a> for <s>pedestrians</s> <s>wheelchair
users</s> shall<r>not be steeper than 1:20</r>. <E>For <a>dis-
tances of less than 3 metres</a>, it may be steeper, but <r>not
more than 1:12</r>.</E></R> <R>The <a>access route</a> shall have
<r>clear width of a minimum of 1,8 m</r> and <r>obstacles shall be
placed so that they do not reduce that width </r>. <r>Maximum
cross fall shall be 2 %.</r></R> <R>The <a>access route</a> shall
have <r>a horizontal landing at the start and end of the in-
cline</r>, plus <r>a horizontal landing for every 0,6 m of in-
cline</r>. <r>The landing shall be a minimum of 1,6 m
deep.</r></R> <R><r>Minimum clear height shall be 2,25 m</r> for
the full width of the defined walking zone of the entire <a>access
route</a> including crossing points.</R></R>
```

Figure 6: RASE example (Figure from Hudeczek, 2017, p. 46)

In the research by Beach and Rezgui (2018), a graphical user interface is used to help the user with managing the data when encoding new regulations as well as maintaining already encoded regulations. This user interface also prompts the user to specify additional metadata for objects which is then used to automatically build an ontology based on the given semantics. Rather than following an existing building ontology, users are supposed to specify the semantics explicitly themselves. This is due to the many subtle differences in semantics for different regulations. The authors found the only reliable way to correctly map the semantics was to let the user specify them (Beach and Rezgui, 2018).

As an additional step, the semantics of a regulation have to be aligned to a BIM data format. This is done by mapping the data in the selected format to the objects and properties defined in the ontology. First the general classes are mapped and then properties of those classes are mapped (Beach and Rezgui, 2018).

In case the data is not present in the model, it is either calculated from other data available or, if a calculation based on existing data is not possible, the user is asked to manually specify the data. Unit conversions use defined functions to convert between different units, however, in IFC units for each property are not always specified, so the conversion relies on the current ifcOWL implementation (Beach and Rezgui, 2018).

5.3. Visual Languages

A visual language distinguishes itself from other languages in the sense that syntax and semantics of a language rely on a system of visual signs and rules instead of a textual representation. They often show the flow of information, making even complex structures easier to understand and to follow (Preidel and Borrmann, 2015).

Visual languages exist for a multitude of different domains. The best-known ones in the building sector are VCCL and KBVL.

The Visual Code Checking Language (VCCL) tries to define elements as generically as possible and break them down to their lowest level, which leads to an increase of flexibility in its usage. The main goal of this approach is to make the language usable also to non-programmers due to its intuitive connection of easily understandable base elements, as well as to make the flow of information and decision making in a compliance checking process visible and understandable to the user (Preidel and Borrmann, 2015).

In VCCL elements of the language are represented as nodes with a name and a datatype, which may also include attributes. A collection of nodes of the same type can be combined in a set node. Operator nodes are used to describe an operation with input and output as a process between nodes. Directed lines between the nodes symbolize the flow of information and build processing chains. With these elements a checking procedure for regulatory requirements can be visualized and implemented using a VCCL graph (Preidel and Borrmann, 2015).

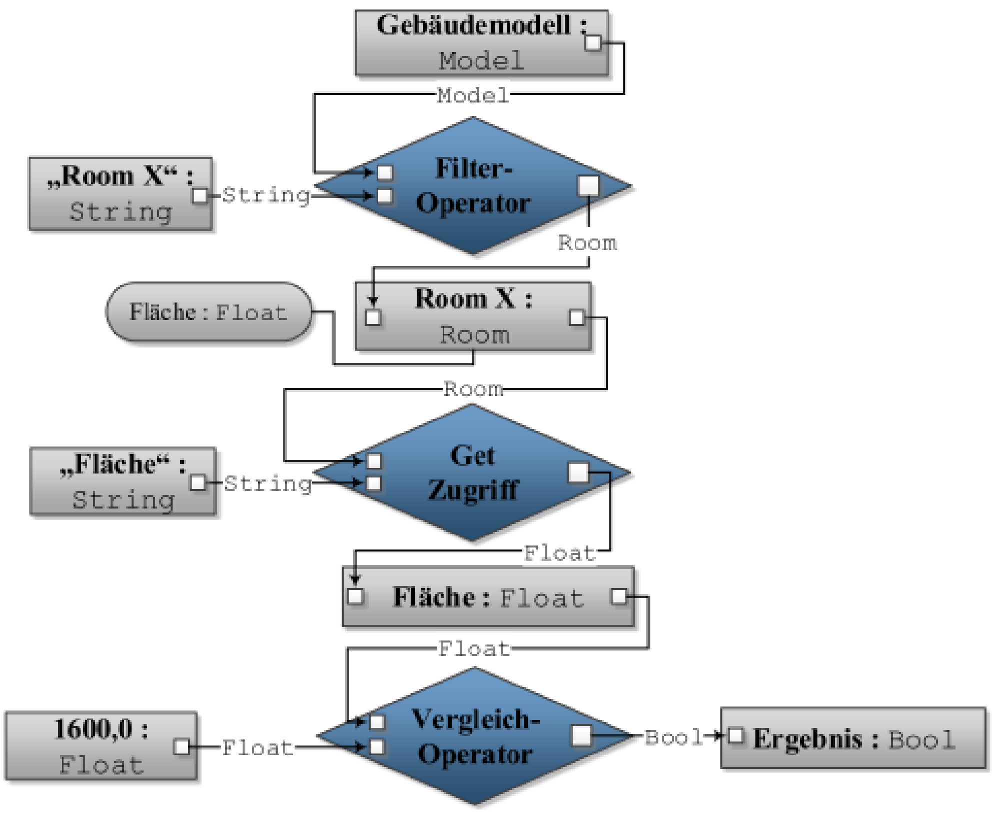


Figure 7: VCCL example (Figure from Preidel, 2014, p. 87)

KBim Visual Language (KBVL) was developed for similar reasons as VCCL, to create a computer-readable building code that can still be understood by users without programming knowledge, but also to automate the process of creating KBimCode in the future. KBimCode refers to the computer-readable interpretation of the Korean natural language building code. It is based on KBimLogic, a logic rule-based mechanism, which was used for the translation. KBVL uses vocabulary of the building domain which users already know and not only acts as management system for the information, but also as interactive programming solution. The usage is similar to that of VCCL – or any other visual language – with the representation of basic building blocks as nodes and connecting them based on their input and output parameters and pre-defined functions. The different node types that exists in KBVL are the rule node, which must contain at least one method node and at least one object node, a condition node defining the relationship between Boolean values, and two types of components acting as connections between nodes or used to control the level of information displayed by arranging nodes in a nested structure than can be collapsed and expanded. While an object node is used for querying objects in the model, a method node contains the function to be executed given an object node's result. A connection component acts as the connecting line between output and input of nodes. An additional nesting component is used to control the level of detail visible to the user by expanding or collapsing such components (Kim et al., 2019).

Visual Bim Query Language (VBQL) and Visual Query Language for BIM (vQL4BIM) are two other examples of the use of visual language approaches in the field of BIM to help the users deal with the complex and information-heavy BIM models without additional programming skills in order to extract information (Kim et al., 2019).

The discussed visual languages are very similar in their structure and usage, but each still focuses on the building regulations of their own country. Also, the usage of visual language is not suitable for all kinds of building regulations but is usually focused on checks that need to query data and information (Kim et al., 2019).

5.4. Machine Learning and Artificial Intelligence Approaches

Machine Learning is an application of artificial intelligence in which systems learn to derive their own rules based on large amounts of data they are given. The main objective of machine learning is to enable computers to learn without the need of human assistance and to adapt to new situations by themselves. The algorithms used to achieve this are generally categorized in supervised and unsupervised learning (Louridas and Ebert, 2016).

For supervised machine learning algorithms, a learning set is prepared by human experts in advance. The data in the learning set is labelled and tagged in order to provide additional information to the algorithm based on pre-calculated results of time-consuming computations such as complex simulations, or human knowledge such as facial recognition or identification of emotions in a person's facial expression or a written statement. Based on the analysis of this first set of data, the algorithm derives rules in order to predict the results of similar calculations on similar sets of data. A second set of data is then used to verify the correctness of the predictions. If the prediction is wrong, the algorithm can also compare the estimated value to the correct result in order to improve the previously inferred function, thus advancing the accuracy of predictions with the amount of data used (Kwok et al., 2015).

Unsupervised machine learning works without the additional work of labelling and tagging the learning data set. A function derived from this data is used to find hidden structures in the data instead of predicting a certain result of a calculation (Kwok et al., 2015).

Semi-supervised machine learning falls between the two previously described categories, using both labelled and unlabelled data in the training set. Due to the large amount of work required to correctly label data manually, the largest part of the used data sets typically consist of unlabelled data (Kwok et al., 2015).

Another learning method is reinforced machine learning in which the system interacts with its environment and is given reward feedback for correct behaviour in order to reinforce this behaviour, while incorrect behaviour is not rewarded. This way the system determines the ideal behaviour within a certain context (Kwok et al., 2015).

Machine learning approaches for legal texts have already been discussed in multiple publications, for example Ashley (2017) discussed ways to extract structure and semantics from legal documents, Bach et al. (2013) proposed a framework for extracting the logical structure of legal sentences, and Nay (2018) researched Natural Language Processing together with machine learning for processing laws and policies.

Natural Language Processing (NLP) is primarily concerned with understanding human language and formulating correct responses. Typically, logic or statistics are used to analyse natural language, but with the development and improvement of deep learning and machine learning methods, NLP methods were also adapted (Song et al., 2018).

Lu et al. (2012) reviewed artificial intelligence in the construction industry prior to 2012. In a more recent research project Song et al. (2018) discussed a deep learning-based natural language processing approach to transform building regulations in a computer-readable form, which was then analysed based on the example of Korean building regulations. The approach enables computers to learn semantics of a language based on text data without manually extracting training sets first. In a first step, training sentences are decomposed to a word-level and the semantics of the words are learned. This includes a pre-processing step focussed on grammatical analysis of the given sentences in order to remove unnecessary symbols for a semantic understanding of the words which occur in the Korean language. For the semantic analysis, the semantics of words are learned by analysing the words that occur in close proximity to them and are therefore more likely to be closely related. Then numeric vector values are assigned to each word, enabling further calculations. For the given example a 400-dimensional vector space was used. The vector values then help classifying sentences based on previous tags to identify the ones relevant to a building as well as the ones only dealing with administrative procedures (Song et al., 2018).
In the second step sentences are analysed regarding words and topics based on the training set from the previous step. Then the sentences are classified according to the analysis (Song et al., 2018).

5.5. Discussion

Each of the different methods discussed above has their own advantages and disadvantages when it comes to formalizing building codes. While logic-based representations can be easily translated to computer-readable formats, they tend to be difficult to understand for humans trying to grasp their meaning at a glance. A visual tree-based representation as opposed to a purely textual one can increase the readability of logic constructs, but this has only been researched for fairly short and easy building codes.

For long and complex codes, the problem at its core is expected to remain, even though it is handled better than before.

Visual Languages can be handled even by non-programmers. Predefined elements representing objects or functions that are already known in the domain can be connected visually in an intuitive way. However, while less expert knowledge is required for the usage of a visual language, the effort to define a visual language is much higher than the effort needed to use a logic-based approach for the formalization. Adding additional node elements to the language is only done by experts and in many cases the codebase for the visual language is closed, so even though many languages break the nodes down to very small pieces, the functions behind the nodes still act as a black box for the users and the contents can neither be observed nor changed by them.

For building regulations, machine learning approaches may be used in order to correctly extract information from a legal text and formalize a rule automatically. Machine learning is also the only discussed approach that may not require the explicit formalization of building codes altogether. Given a large enough number of training models, an algorithm could be trained to predict whether a model is compliant with a regulation or not based on the models deemed correct or incorrect before. While this seems like an approach saving a lot of effort for experts, one main problem is still the number of models needed to train the algorithm. Another problem is the lack of explicit knowledge about the deductions of the algorithm. Depending on the number and complexity of rules as well as the number, quality and differentiability of the training models, the outcome of a prediction may not at all be what was expected. For example, when the correct models in the training set have their location, their number of floors or even their amount of walls in common by chance, the algorithm may deduct that this is the reason for their being compliant with the regulations. Additionally, changes in the regulations may require an additional training set or adjustments in the first one as some models that were compliant before may no longer be.

So manual checks are still required in most cases to make sure the prediction of the algorithm is indeed correct, which means one of the reasons to use automated compliance checks in the first place would be lost.

One possible approach to reduce the complexity would be to train different algorithms for different parts of building code and to use the algorithms not on the whole building models but just the parts relevant to the checks. However, this requires the extraction of certain elements of models depending on the rule to check, which is not always trivial due to the possibility of identifying elements by different means. For example, a separating wall between two adjacent flats (“Wohnungstrennwand” in German) is not currently modelled in IFC. A custom parameter may be used to identify it as such, but it is also possible to use a certain code in its name that distinguishes such a wall from others not separating two flats. Without a high degree of standardization in the models, the preparation of training models as well as the checking itself may not be feasible. But to compile a large set of standardized models from unstandardized input sources is very challenging as well.

One problem for the formalization of building codes that still remains throughout all formalization methods discussed is the difficulty to formalize codes that leave room for interpretation for human readers. In some cases, this can be necessary, as it is not always feasible for the institution responsible for the building code to include every single possibility of how certain requirements can be fulfilled. However, in order for a computer to check automatically if a requirement is fulfilled, the knowledge of how it can be fulfilled, which a human expert manually checking for compliance possesses, is necessary. One example for such a case is part of requirement 3.8.3 for fire walls in OIB guideline 2.1: They should be extended to 50 cm above the roof. However, they may end at the roof covering if "*the transmission of fire is impeded by other means of equal value*" (Österreichisches Institut für Bautechnik, 2019b). An expert is expected to be able to identify such other means while a computer needs additional information to do so.

6. Formalizing OIB Guidelines

The OIB Guidelines were chosen as representative nationwide set of building codes in Austria. They are issued by the Austrian Institute of Construction Engineering and declared binding by all federal states, though, in different versions. While in Lower Austria the Guidelines 1 to 5 are still in effect in their 2011 version and in Vienna the newest version from 2019 came into effect in early 2020, the other federal states currently have the version of 2015 in effect. Generally, deviations to those guidelines are explicitly allowed if an equivalent level of security and protection is demonstrably achieved through other means as those stated in the guidelines.

The OIB guidelines are separated in different guidelines based on covered topics, as seen in Table 1.

Table 1: OIB Guidelines

Guideline	Title
OIB Guideline 1	Mechanical resistance and stability
OIB Guideline 2	Safety in case of fire
OIB Guideline 2.1	Safety in case of fire in operational structures
OIB Guideline 2.2	Safety in case of fire in garages, roofed parking spaces and multi-storey car parks
OIB Guideline 2.3	Safety in case of fire in buildings with a fire escape level in excess of 22 m
OIB Guideline 3	Hygiene, health and preservation of the environment
OIB Guideline 4	Safety in use and accessibility
OIB Guideline 5	Protection against noise
OIB Guideline 6	Energy saving and heat insulation

OIB Guideline 1 (Österreichisches Institut für Bautechnik, 2019a) is left out in in this thesis as it is very limited in the number of rules and its included paragraphs are only generally talking about what kinds of loads have to be considered while referring to other norms for reliability requirements.

OIB Guidelines 5 (Österreichisches Institut für Bautechnik, 2019h) and 6 (Österreichisches Institut für Bautechnik, 2019i) are left out for two main reasons: On the one hand, Solibri Office is primarily focussing on checking certain parameters in an IFC model or calculating values such as minimum distance between elements based on the given geometry. Simulations for noise propagation or energy simulations are not part of the software and implementing such solutions is not the focus of this thesis. On the other hand, only checking for already existing noise or energy parameters in the building elements is currently possible using pre-defined rules in Solibri Office and works the same way as checking for any other parameter values of an element. As such checks of element parameters are demonstrated in this thesis using other examples, additional examples working the same way would not bring any additional benefit.

Therefore, the formalization and implementation in this thesis focusses on OIB Guidelines 2 (Österreichisches Institut für Bautechnik, 2019b, 2019c, 2019d, 2019e), 3 (Österreichisches Institut für Bautechnik, 2019f) and 4 (Österreichisches Institut für Bautechnik, 2019g).

6.1. Selection of Example Clauses

For the selection of example clauses, different types of requirements should be included, to ensure the general validity of the approach. On the one hand there are requirements in the form of tables with values for different objects, building types or other constraints. On the other hand, there are requirements written in natural language, which again fall in two categories: those which include discrete values to check for and those which lack those values. In the strict sense, the latter cannot be checked for compliance without additional knowledge of how the requirements can be achieved exactly. However, while automated compliance checking may not be possible, assisted compliance checking is proposed in those cases. By providing additional textual and visual information, such as the number or distribution of certain elements in the building, which can be analysed quickly by a professional checking the model manually, the time required for the manual check can be drastically reduced.

Additionally, paragraphs which could not be automatically checked previously due to the restrictions posed by Solibri's pre-defined rulesets are selected to test whether a custom implementation using the provided API can increase the amount of checkable content.

An example from OIB Guideline 2 is part 1 and 2 (without 2.4) of table 1b (in Figure 8). This example includes distinct values for different building elements and building classes in the form of a table. This table contains information on the required fire resistance of building elements. Part 1 addresses requirements for load-bearing elements, excluding slabs and walls which form fire compartments. Part 2 addresses requirements for partition walls, excluding walls of stairwells. The following parts 3, 4 and 5 address walls and slabs forming fire compartments, slabs and roof slopes with an inclination smaller than 60°, and balcony panels respectively. Parts 1 and 2 were chosen as example because the requirements themselves are rather easy for anyone to understand, however, 1.1, 1.2 and 2.1 could not be automatically checked previously due to referring to the "topmost floor" of a building, whose identification is not trivial. 2.4 is left out because it deals with a very specific kind of partition wall: a wall between two apartments or units in terraced houses. This would require either very specific test models for terraced houses, or at least additional information in the models regarding which spaces belong to which apartment. While this is possible, it is not necessary or feasible in order to show how such requirements can be formalized or implemented generally.

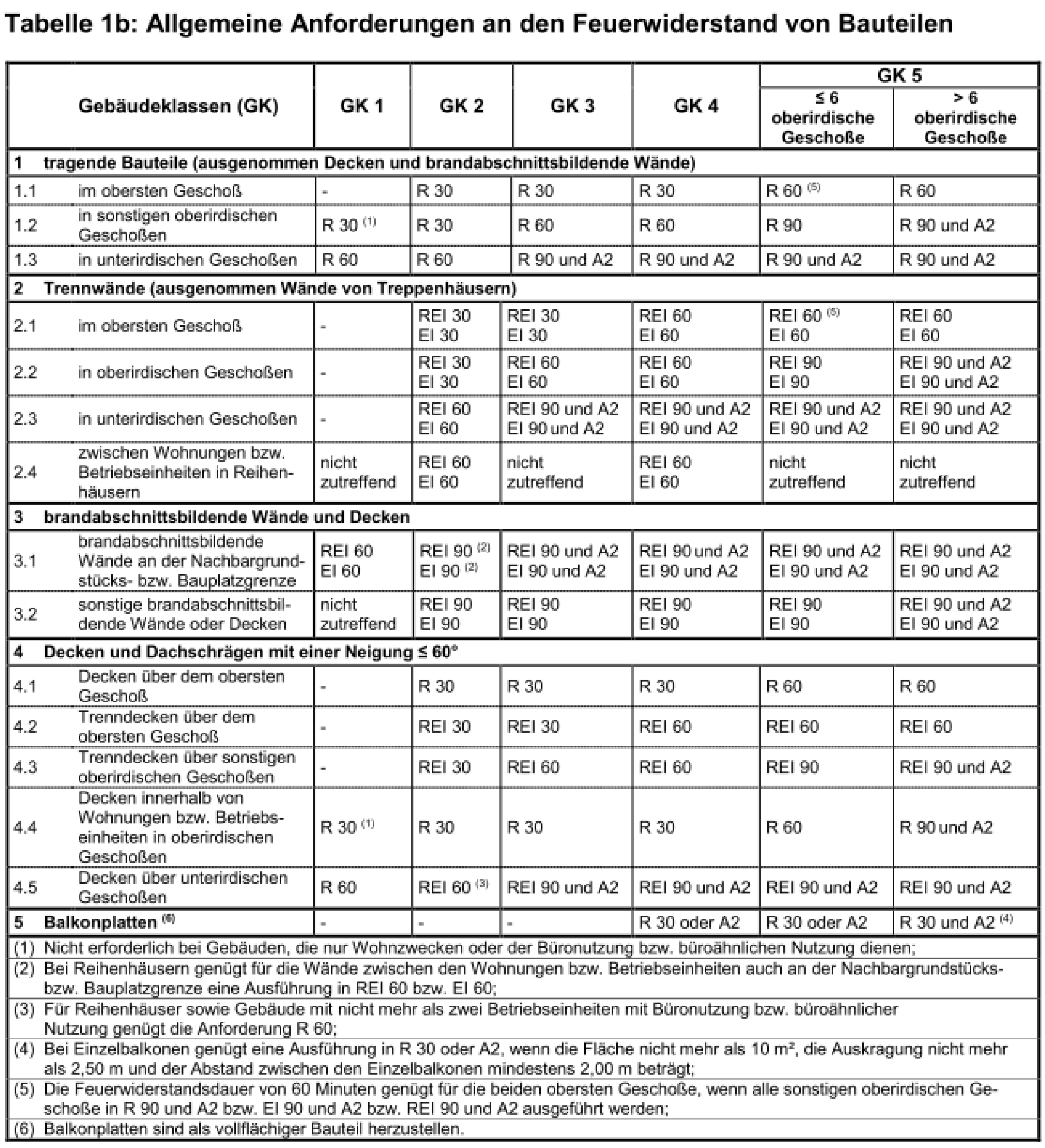


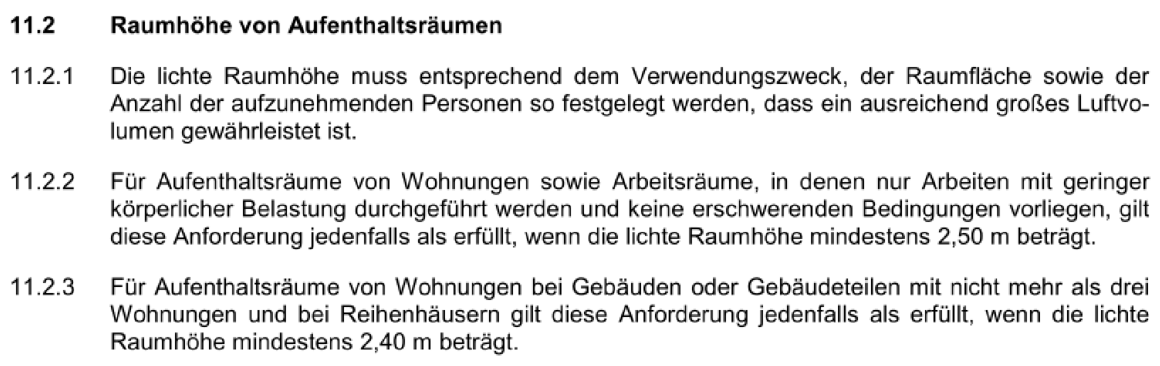
Tabelle 1b: Allgemeine Anforderungen an den Feuerwiderstand von Bauteilen

Gebäudeklassen (GK)		GK 1	GK 2	GK 3	GK 4	GK 5	
						≤ 6 oberirdische Geschoße	> 6 oberirdische Geschoße
1	**tragende Bauteile (ausgenommen Decken und brandabschnittsbildende Wände)**						
1.1	im obersten Geschoß	-	R 30	R 30	R 30	R 60 (5)	R 60
1.2	in sonstigen oberirdischen Geschoßen	R 30 (1)	R 30	R 60	R 60	R 90	R 90 und A2
1.3	in unterirdischen Geschoßen	R 60	R 60	R 90 und A2	R 90 und A2	R 90 und A2	R 90 und A2
2	**Trennwände (ausgenommen Wände von Treppenhäusern)**						
2.1	im obersten Geschoß	-	REI 30 EI 30	REI 30 EI 30	REI 60 EI 60	REI 60 (5) EI 60	REI 60 EI 60
2.2	in oberirdischen Geschoßen	-	REI 30 EI 30	REI 60 EI 60	REI 60 EI 60	REI 90 EI 90	REI 90 und A2 EI 90 und A2
2.3	in unterirdischen Geschoßen	-	REI 60 EI 60	REI 90 und A2 EI 90 und A2	REI 90 und A2 EI 90 und A2	REI 90 und A2 EI 90 und A2	REI 90 und A2 EI 90 und A2
2.4	zwischen Wohnungen bzw. Betriebseinheiten in Reihenhäusern	nicht zutreffend	REI 60 EI 60	nicht zutreffend	REI 60 EI 60	nicht zutreffend	nicht zutreffend
3	**brandabschnittsbildende Wände und Decken**						
3.1	brandabschnittsbildende Wände an der Nachbargrundstücks- bzw. Bauplatzgrenze	REI 60 EI 60	REI 90 (2) EI 90 (2)	REI 90 und A2 EI 90 und A2	REI 90 und A2 EI 90 und A2	REI 90 und A2 EI 90 und A2	REI 90 und A2 EI 90 und A2
3.2	sonstige brandabschnittsbildende Wände oder Decken	nicht zutreffend	REI 90 EI 90	REI 90 EI 90	REI 90 EI 90	REI 90 EI 90	REI 90 und A2 EI 90 und A2
4	**Decken und Dachschrägen mit einer Neigung ≤ 60°**						
4.1	Decken über dem obersten Geschoß	-	R 30	R 30	R 30	R 60	R 60
4.2	Trenndecken über dem obersten Geschoß	-	REI 30	REI 30	REI 60	REI 60	REI 60
4.3	Trenndecken über sonstigen oberirdischen Geschoßen	-	REI 30	REI 60	REI 60	REI 90	REI 90 und A2
4.4	Decken innerhalb von Wohnungen bzw. Betriebseinheiten in oberirdischen Geschoßen	R 30 (1)	R 30	R 30	R 30	R 60	R 90 und A2
4.5	Decken über unterirdischen Geschoßen	R 60	REI 60 (3)	REI 90 und A2	REI 90 und A2	REI 90 und A2	REI 90 und A2
5	**Balkonplatten (6)**	-	-	-	R 30 oder A2	R 30 oder A2	R 30 und A2 (4)
(1)	Nicht erforderlich bei Gebäuden, die nur Wohnzwecken oder der Büronutzung bzw. büroähnlichen Nutzung dienen;						
(2)	Bei Reihenhäusern genügt für die Wände zwischen den Wohnungen bzw. Betriebseinheiten auch an der Nachbargrundstücks- bzw. Bauplatzgrenze eine Ausführung in REI 60 bzw. EI 60;						
(3)	Für Reihenhäuser sowie Gebäude mit nicht mehr als zwei Betriebseinheiten mit Büronutzung bzw. büroähnlicher Nutzung genügt die Anforderung R 60;						
(4)	Bei Einzelbalkonen genügt eine Ausführung in R 30 oder A2, wenn die Fläche nicht mehr als 10 m², die Auskragung nicht mehr als 2,50 m und der Abstand zwischen den Einzelbalkonen mindestens 2,00 m beträgt;						
(5)	Die Feuerwiderstandsdauer von 60 Minuten genügt für die beiden obersten Geschoße, wenn alle sonstigen oberirdischen Geschoße in R 90 und A2 bzw. EI 90 und A2 bzw. REI 90 und A2 ausgeführt werden;						
(6)	Balkonplatten sind als vollflächiger Bauteil herzustellen.						

Figure 8: General requirements regarding fire resistance of building elements from OIB Guideline 2 (Österreichisches Institut für Bautechnik, 2019e)

Regarding paragraphs in natural language, four examples from OIB Guidelines 2, 3 and 4 were chosen:

Figure 9 shows part of paragraph 11.2 from OIB Guideline 3. While 11.2.1 states that a sufficiently large air volume must be guaranteed in all rooms according to their intended use, size and number of persons to be admitted generally, the following sub-paragraphs specify this in more detail. According to 11.2.2 this requirement is fulfilled if lounge rooms of apartments and workspaces for work with little physical stress reach a height of at least 2.5 meters. The next sub-paragraph further states that for lounge rooms of apartments in buildings (or parts of buildings) with no more than three apartments, as well as in terraced houses, a height of 2.4 meters is already enough to satisfy the requirement.



11.2 Raumhöhe von Aufenthaltsräumen

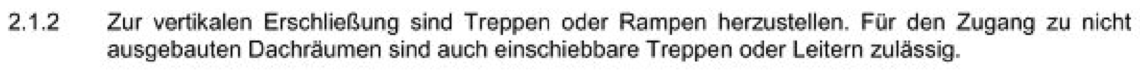
11.2.1 Die lichte Raumhöhe muss entsprechend dem Verwendungszweck, der Raumfläche sowie der Anzahl der aufzunehmenden Personen so festgelegt werden, dass ein ausreichend großes Luftvolumen gewährleistet ist.

11.2.2 Für Aufenthaltsräume von Wohnungen sowie Arbeitsräume, in denen nur Arbeiten mit geringer körperlicher Belastung durchgeführt werden und keine erschwerenden Bedingungen vorliegen, gilt diese Anforderung jedenfalls als erfüllt, wenn die lichte Raumhöhe mindestens 2,50 m beträgt.

11.2.3 Für Aufenthaltsräume von Wohnungen bei Gebäuden oder Gebäudeteilen mit nicht mehr als drei Wohnungen und bei Reihenhäusern gilt diese Anforderung jedenfalls als erfüllt, wenn die lichte Raumhöhe mindestens 2,40 m beträgt.

Figure 9: Requirement regarding the height of lounge rooms in apartments and workspaces for work with little physical stress from OIB Guideline 3 (Österreichisches Institut für Bautechnik, 2019f)

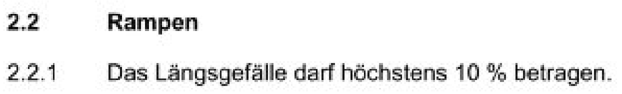
OIB Guideline 4 states in paragraph 2.1.2 (as seen in Figure 10) that for vertical access in buildings stairs or ramps have to be used. An exception is made for undeveloped attic spaces, in this case a ladder or retractable stairs are permitted as well.



2.1.2 Zur vertikalen Erschließung sind Treppen oder Rampen herzustellen. Für den Zugang zu nicht ausgebauten Dachräumen sind auch einschiebbare Treppen oder Leitern zulässig.

Figure 10: General requirement for vertical access in buildings from OIB Guideline 4 (Österreichisches Institut für Bautechnik, 2019g)

Figure 11 shows a requirement for ramps, as stated in OIB Guideline 4, paragraph 2.2.1: The slope of a ramp must not exceed 10%.

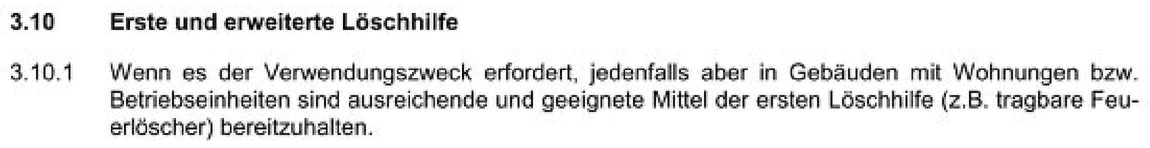


2.2 Rampen

2.2.1 Das Längsgefälle darf höchstens 10 % betragen.

Figure 11: Requirement regarding the steepness of ramps from OIB Guideline 4 (Österreichisches Institut für Bautechnik, 2019g)

An example in natural language with ambiguous wording is shown in Figure 12. This paragraph in OIB Guideline 2 states that, depending on the intended use of the building, but at least in buildings containing apartments or operating units, sufficient and appropriate means of first fire extinguishing aids such as portable fire extinguishers, must be available. This requirement neither states which types of buildings it exactly refers to or explicitly doesn't refer to, nor does it give a clear number of elements required or their distribution in the building. As a portable fire extinguisher is only used as an example, other means to extinguish fires may also be valid according to this requirement, but it is not stated, which means exactly are entailed by this OIB Guideline.



3.10 Erste und erweiterte Löschhilfe

3.10.1 Wenn es der Verwendungszweck erfordert, jedenfalls aber in Gebäuden mit Wohnungen bzw. Betriebseinheiten sind ausreichende und geeignete Mittel der ersten Löschhilfe (z.B. tragbare Feuerlöscher) bereitzuhalten.

Figure 12: Requirements regarding the availability of first fire extinguishing aid in buildings from OIB Guideline 2 (Österreichisches Institut für Bautechnik, 2019e)

Additional information regarding the methods and amount of first fire extinguishing aids can be found in other legal documents, such as TRVB 124 /17 (F) “Erste und erweiterte Löschhilfe” (first and extended extinguishing aid), which includes requirements concerning the selection, number and distribution of extinguishing equipment in a building. According to this guideline the amount of portable fire extinguishers depends on the fire risk category (low, medium or high), the extinguishing capacity of the portable fire extinguisher, as well as the net floor area in each storey and the length of the way to reach the fire extinguisher (ÖBFV, 2017). However, for this thesis only the text in the OIB Guideline is considered, as an example of a not clearly defined requirement, as the two are not directly linked or related, and merely happen to complement each other.

For the following chapters the selected examples are referred to as examples A-E:

- Example A: Fire resistance classes according to OIB Guideline 2 as shown in Figure 8
- Example B: Ramp steepness according to OIB Guideline 4 as shown in Figure 11
- Example C: Vertical access to buildings according to OIB Guideline 4 as shown in Figure 10
- Example D: Height requirements of lounge rooms according to OIB Guideline 3 as shown in Figure 9
- Example E: Amount of first fire extinguishing aids according to OIB Guideline 2 as shown in Figure 12

6.2. Method

Based on the previous research in chapter Formalization of Building Codes, a formalization approach using logic is chosen for the formalization process. While there are other available methods, a formalization using logic is possible without any additional software by only using text and is therefore widely available.

As a representation, formulas in first-order logic, or “predicate logic”, are used. First-order logic uses propositional logic as foundation, but unlike propositional logic, first-order logic allows the use of quantifiers. Formulas in first-order logic can consist of predicates or other formulas connected by logical connectives such as *and* ($\wedge$) or *or* ($\vee$). The use of non-logical symbols is also allowed. For example:

$$isValid(formulaX)$$

isValid is used as a predicate symbol. If the element *formulaX* is valid, then *isValid(formulaX)* is *true*, otherwise it is *false*. In this case *formulaX* is a constant referring to a specific element in the universe this formula resides in. A predicate is a statement that always validates to either true or false and is often used to refer to properties of elements. A function looks similar, but instead of giving a Boolean value, it gives an element of the universe. So for example

$$author(book)$$

gives the author of the element *book*. Predicates and functions can take variables, constants or functions as parameters. The number of parameters is defined for each predicate or function individually, but there is no overall limit to the amount of parameters that may be used.

With quantifiers, statements like “for all” or “for some” can be formalized. For example the constant *formulaX* is valid, so

$$\exists x(isValid(x))$$

is true, as at least one element – namely *formulaX* – is known which is valid. However,

$$\forall x(isValid(x))$$

is not true if there is any another element defined in the universe which is not valid.

Requirements are often formalized as implications, such as

$$building(x) \rightarrow hasDoor(x)$$

If the element *x* is a building, it needs to have a door. So for every building that has a door this formula validates to *true*. However, *x* could be any element in the defined universe. If *x* is not a building, then this formula also validates to *true*, even if x doesn’t have a door, as *hasDoor(x)* is only relevant to elements which are buildings. This implication can also be written as

$$\neg building(x) \lor hasDoor(x)$$

So it validates to *true* if *x* is not a building or – if *x* is a building – *x* has a door.

Using an implication is important for the following reason: As this formula should only check if buildings have doors, it should ignore objects that don’t meet the criteria of being a building. This is done by validating to *true* for all those other objects. This way, multiple formulas can be combined to one large formula using *and* as connector without interfering with each other.

To achieve better readability of the logical representation, the names of predicates and functions in a formula are chosen as short descriptions of what the predicate or formula does.

In order to formulize statements, the answers to five questions, similar to RASE tags, are extracted from a paragraph in the building code:

- Which types of elements is the requirement about?
  The types of elements refer to different types of object such as walls, slabs, buildings or spaces.
- Which constraints apply to the elements?
  The constraints applicable to the elements should further limit this selection based on a specific parameter or location of an element – such as “only concrete walls on the ground floor” should be checked.
- Which element property decides compliance or non-compliance?
  The property can be a geometric parameter, such as the height of the element, but also a property like *loadBearing* or *fireRating*.

- Which value or threshold and operator are needed?
  This question aims at understanding if the value of the deciding property is a distinct value such as a Boolean value, which should either be true or false, or if the value represents a minimum or maximum, such as minimum volume for a space.
- What are exceptions to this rule?
  In case there are any exceptions when the requirement is not applicable, this exception is also part of the formalization.

In this context, “elements” refer to objects of the universe. These can be building elements such as walls or slabs, but they can also refer to zones or spaces or building storeys.

After the necessary information is extracted from the building code, the formalization is executed in multiple steps, adding constraints or values and changing the formula in smaller portions. Each step and change is described in order to help the reader understand the general approach and not just see the finished formula.

6.3. Formalization of Examples

6.3.1. Example A

For the formalization of the examples from the table in OIB guideline 2, in a first step individual statements are extracted for each field in the table containing a value for fire resistance. Figure 13 shows the relevant part of the table with color highlights to easier identify the different sections.

Generally the required information can be found in the table as follows:
Types of element: highlighted green in the figure
Constraints: highlighted red (element location in the building) and blue (building class) in the figure
Element property: *fireResistance* as text
Value/threshold and operator: highlighted grey in the figure
Exceptions: highlighted purple in the figure

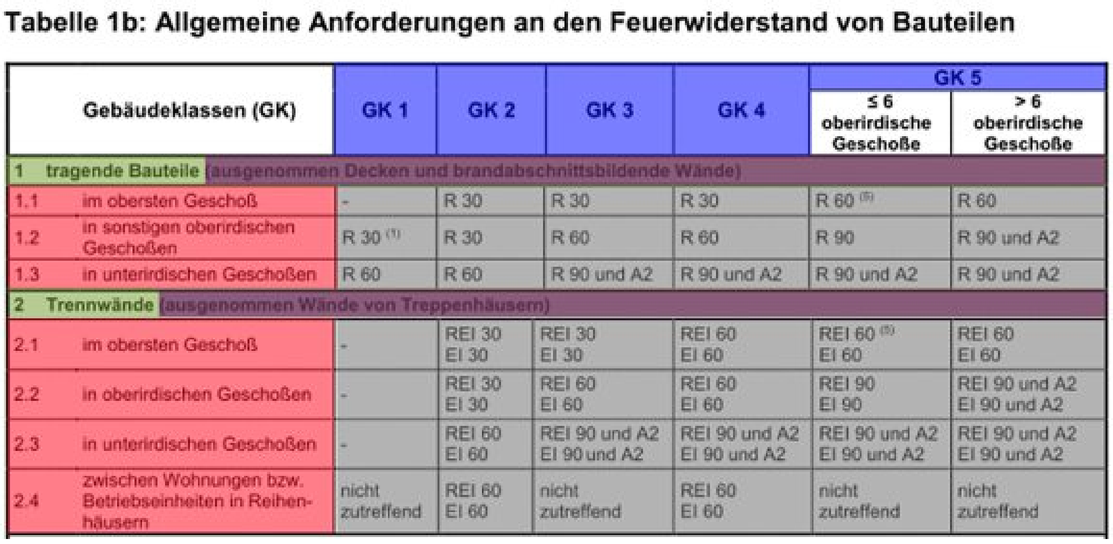


Tabelle 1b: Allgemeine Anforderungen an den Feuerwiderstand von Bauteilen

Gebäudeklassen (GK)		GK 1	GK 2	GK 3	GK 4	GK 5	
						≤ 6 oberirdische Geschoße	> 6 oberirdische Geschoße
1	tragende Bauteile (ausgenommen Decken und brandabschnittsbildende Wände)						
1.1	im obersten Geschoß	-	R 30	R 30	R 30	R 60 (5)	R 60
1.2	in sonstigen oberirdischen Geschoßen	R 30 (1)	R 30	R 60	R 60	R 90	R 90 und A2
1.3	in unterirdischen Geschoßen	R 60	R 60	R 90 und A2	R 90 und A2	R 90 und A2	R 90 und A2
2	Trennwände (ausgenommen Wände von Treppenhäusern)						
2.1	im obersten Geschoß	-	REI 30 EI 30	REI 30 EI 30	REI 60 EI 60	REI 60 (5) EI 60	REI 60 EI 60
2.2	in oberirdischen Geschoßen	-	REI 30 EI 30	REI 60 EI 60	REI 60 EI 60	REI 90 EI 90	REI 90 und A2 EI 90 und A2
2.3	in unterirdischen Geschoßen	-	REI 60 EI 60	REI 90 und A2 EI 90 und A2	REI 90 und A2 EI 90 und A2	REI 90 und A2 EI 90 und A2	REI 90 und A2 EI 90 und A2
2.4	zwischen Wohnungen bzw. Betriebseinheiten in Reihenhäusern	nicht zutreffend	REI 60 EI 60	nicht zutreffend	REI 60 EI 60	nicht zutreffend	nicht zutreffend

Figure 13: Part of table 1b of OIB guideline 2 with color highlights (Österreichisches Institut für Bautechnik, 2019e)

For the first value on the top left in the table a formula is constructed like this:

<u>Extracting the requirements:</u>
Types of element: load-bearing building elements
Constraints: building elements in the topmost floor of the building, if the building has a building class “GK1”
Element property: *fireResistance* as text
Value/threshold and operator: “-“, so any fire resistance class fulfils the requirement
Exceptions: slabs and walls which separate fire compartments

First a basic formula is identified which is then extended to include all constraints. For the table there are formulas in the form “if an element meets certain constraints, then a certain value must be present”. This “if-then” relationship is formalized with an implication.

$$\forall e\big(isBuildingElement(e) \rightarrow hasValue(e, value)\big)$$

This reads as follows: “For all building elements *e*, *e* must have a certain value.” For better readability, variables referring to building elements or spaces are typically named “*e*”, while the variables referring to buildings and floors are named “*b*” and “*f*” respectively. The predicate *hasValue(x,y)* just acts as a placeholder for now and is replaced later.

The element that must have this value is described in more detail by adding the desired type of element to the formula. For the top left value in the table this means that the element should be a load-bearing element. In order to check if an element is load-bearing, the predicate *isLoadBearing(x)* is introduced, which evaluates to *true* if an element x is load-bearing and to *false* if it is not. Similar checks, such as whether or not an element forms a fire compartment, are formalized in the same way.

$$\forall e\big(isBuildingElement(e) \land isLoadBearing(e) \rightarrow hasValue(e, value)\big)$$

Next the exceptions are added to the formula. Exceptions define that the desired element is neither a slab nor a wall which forms a fire compartment. The formula can be extended to include this information as follows:

$$\begin{aligned}\forall e\Big(\Big(&isBuildingElement(e) \;\land\; isLoadBearing(e)\\ &\land \neg hasType\big(e,\text{'SLAB'}\big) \land \big(hasType(\mathrm{e},\text{'}WALL\text{'}\big)\\ &\rightarrow \neg separatesFireCompartments(e)\Big) \rightarrow hasValue(e, value)\Big)\end{aligned}$$

For excluding elements of type “*SLAB*”, a predicate *hasType(x,y)* is used which evaluates whether or not the type of a given element, *x*, is the same as the second parameter given, *y*. In this case the function is negated as the formula should check elements of all types except slabs. Without this negation, only elements of type *slab* would be checked. Excluding walls which separate fire compartments requires another implication as not all elements of type *wall* should be excluded, but only specific ones. As load-bearing elements can also be of type *column* for example, a connection with *and* would exclude those elements as the formula would then only concern load-bearing elements of type *wall* which do not separate fire compartments. However, all other walls which are not separating fire compartments, are still required to have the value from the table.

In the next step constraints are added to the formula. For this example, the elements to check should be part of the topmost floor of the building. In order to formalize this, another variable representing the building has to be introduced and the relationship between the building and the building elements has to be formalized.

$$\begin{aligned}\forall b \forall e \Big(\big(&isBuilding(b) \land isBuildingElement(e) \land partOf(e,b)\\ &\land isLoadBearing(e) \land \neg hasType(e,\text{'SLAB'}) \land (\text{hasType}(\text{e},\text{'}WALL')\\ &\rightarrow \neg separatesFireCompartments(e))\\ &\land partOf\big(e, topmostFloor(b)\big)\big) \rightarrow hasValue(e, value)\Big)\end{aligned}$$

All buildings *b* and building elements *e* where the elements are part of the building are evaluated. This is important because elements only belong to one building, but the formula evaluates all values in the given universe. If there are multiple buildings in the universe, all building elements are evaluated for each building, even though they are only relevant to one. In this specific case, however, *partOf(e,b)* in the first line of the formula could be left out as the elements need to be part of the topmost floor of a building, which in turn is part of the building, so all the elements in this floor are by definition part of the building as well. To get the topmost floor of a building, a function *topmostFloor(x)* is introduced which returns the topmost floor of building *x*, which is then used to check whether or not an element is part of this extracted topmost floor.

What is still missing in the current formula is the building class, which is another constraint. As a variable for the building was already added, another predicate is introduced to evaluate the building class. It is assumed that “GK1” is the only allowed notation for building class 1 in order to keep the formula shorter and easier to read. Without this assumption, every possible notation would have to be included in the formula as a different possibility.

$$\begin{aligned}\forall b \forall e \Big(\big(&isBuilding(b) \land isBuildingElement(e) \land partOf(e,b)\\ &\land hasBuildingClass(b,'GK1') \land isLoadBearing(e)\\ &\land \neg hasType(e,\text{'SLAB'}) \land (\text{hasType}(\text{e},\text{'}WALL')\\ &\rightarrow \neg separatesFireCompartments(e))\\ &\land partOf\big(e, topmostFloor(b)\big)\big) \rightarrow hasValue(e, value)\Big)\end{aligned}$$

With this all the constraints are formalized and as a final step the actual value to check is added to the formula. In the given table fire resistance classes are checked, so a building element should have a specific fire resistance class. For this another predicate, *fireResistanceClass(x,y)*, is introduced.

$$\begin{aligned}\forall b \forall e \Big(\big(& isBuilding(b) \wedge isBuildingElement(e) \wedge partOf(e,b) \\ & \wedge hasBuildingClass(b,'GK1') \wedge isLoadBearing(e) \\ & \wedge \neg hasType(e,'\mathrm{SLAB}') \wedge (\mathrm{hasType}(\mathrm{e},'WALL') \\ & \rightarrow \neg separatesFireCompartments(e)) \\ & \wedge partOf\big(e, topmostFloor(b)\big)\big) \rightarrow fireResistanceClass(e,'-')\Big)\end{aligned}$$

However, for the given example, there is no value in the table. This does not mean that no fire resistance class should be defined, but rather that any fire resistance class is acceptable, including the option that none is defined. One way to formalize this is to add every known fire resistance class to the formula as any one of them is a desired value. However, this can make the formula very long and hard to read. Another way is to assume that there are no undesired values, so the formula just evaluates to *true* for all objects with the given constraints. This leads to a much shorter formula, however, if only certain values are allowed to define a fire resistance class, an additional check to guarantee this constraint would still be necessary. The formula then looks like as follows in a long and a short version:

$$\begin{aligned}\forall b \forall e \Big(\big(& isBuilding(b) \wedge isBuildingElement(e) \wedge partOf(e,b) \\ & \wedge hasBuildingClass(b,'GK1') \wedge isLoadBearing(e) \\ & \wedge \neg hasType(e,'\mathrm{SLAB}') \wedge (\mathrm{hasType}(\mathrm{e},'WALL') \\ & \rightarrow \neg separatesFireCompartments(e)) \\ & \wedge partOf\big(e, topmostFloor(b)\big)\big) \\ & \rightarrow (fireResistance(e,'R\ 30') \vee fireResistance(e,'R\ 60') \\ & \vee fireResistance(e,'R\ 90') \vee fireResistance(e,'R\ 120') \\ & \vee fireResistance(e,'R\ 180') \vee fireResistance(e,'REI\ 30') \\ & \vee fireResistance(e,'REI\ 60') \vee fireResistance(e,'REI\ 90') \\ & \vee fireResistance(e,'REI\ 120') \vee fireResistance(e,'REI\ 180') \\ & \vee fireResistance(e,'EI\ 30') \vee fireResistance(e,'EI\ 60') \\ & \vee fireResistance(e,'EI\ 90') \vee fireResistance(e,'EI\ 120') \\ & \vee fireResistance(e,'EI\ 180') \vee fireResistance(e,'E\ 30') \\ & \vee fireResistance(e,'E\ 60') \vee fireResistance(e,'E\ 90') \\ & \vee fireResistance(e,'E\ 120') \vee fireResistance(e,'E\ 180') \\ & \vee fireResistance(e,'EI2\ 30') \vee fireResistance(e,'EI2\ 60') \\ & \vee fireResistance(e,'\ EI2\ 30-C') \vee fireResistance(e,'EI2\ 60-C') \\ & \vee fireResistance(e,'EI2\ 90-C') \\ & \vee fireResistance(e,'EI2\ 30-C-Sm') \\ & \vee fireResistance(e,'EI2\ 60-C-Sm') \\ & \vee fireResistance(e,'EI2\ 90-C-Sm') \\ & \vee fireResistance(e,'E\ 30-C') \vee fireResistance(e,'E\ 60-C') \\ & \vee fireResistance(e,'E\ 90-C') \vee fireResistance(e,'ND'))\Big)\end{aligned}$$

This formula includes all fire resistance classes according to an Employer's Information Requirement by Eichler (2019). When also considering "R30" and "R 30" as two different possible values due to the space before the number of minutes, the overall number of possible values for this case doubles. For the sake of simplicity, the different notations are therefore treated as equal in the given formulas.

$$\begin{aligned}\forall b \forall e \Big(\big(&isBuilding(b)\ \wedge\ isBuildingElement(e)\ \wedge\ partOf(e,b)\\ &\wedge hasBuildingClass(b,'GK1')\wedge\ isLoadBearing(e)\\ &\wedge\ \neg hasType(e,'\mathrm{SLAB}')\wedge(\mathrm{hasType}(\mathrm{e},'WALL')\\ &\rightarrow\ \neg separatesFireCompartments(e))\\ &\wedge\ partOf\big(e, topmostFloor(b)\big)\big)\rightarrow\ true\Big)\end{aligned}$$

For all buildings *b* and elements *e*, if *e* is load-bearing and *e* is neither a slab nor a wall forming a fire compartment and *e* is part of the topmost floor of *b* and the building class of *b* is “GK1”, then the formula is always true. There can be no wrong values in the model according to the formula, and therefor this specific formula never evaluates to false.

For the second value for building class “GK1”, a formula is found as follows:

Extracting the requirements:
Types of element: load-bearing building elements
Constraints: building elements in a floor above ground which is not the topmost floor, if the building has a building class “GK1”
Element property: *fireResistance* as text
Value/threshold and operator: “R30“, so structurally sound for at least 30 minutes
Exceptions: slabs and walls which separate fire compartments, building usage residential or office

$$\begin{aligned}\forall b \forall e \exists f\big((&isBuilding(b)\ \wedge\ isBuildingElement(e)\ \wedge\ isFloor(f)\ \wedge\ partOf(f,b)\\ &\wedge\ partOf(e,f)\wedge\ isLoadBearing(e)\ \wedge\ \neg hasType(e,'\,SLAB')\\ &\wedge\ (\mathrm{hasType}(\mathrm{e},'WALL')\\ &\rightarrow\ \neg separatesFireCompartments(e))\ \wedge\ isFloorAboveGround(f)\\ &\wedge\ \neg isTopmostFloor(f)\ \wedge\ hasBuildingClass(b,'GK1')\big)\\ &\rightarrow\ fireResistanceClass(e,'R30'))\end{aligned}$$

The steps to reach this formula are the same as in the first example, with the only difference being that the building elements on floors above ground are considered, instead of elements on the topmost floor. The topmost floor in a building is one of the floors above ground, but as in this formula *f* is a floor above ground but not the topmost floor, this has to be explicitly stated. Given that each element can only be part of one distinct floor, the table divides the floors above ground into a topmost floor and all other above-ground floors.

This formula is not correct yet, however: A fire resistance class of R30 is given in the table, but this is just a minimum requirement. R30 means that this element is structurally sound for at least 30 minutes during a fire. If the element had a fire resistance class of R60, being structurally sound for 60 minutes at least, would also include being structurally sound for 30 minutes. However, in the current formula this would be considered invalid. So all fire resistance classes that guarantee structural soundness for at least 30 minutes should be included here. For better readability and without limitation of generality only fire resistance classes R 30/60/90 and REI 30/60/90 are considered in the formula.

The new formula after these changes looks like this:

$$\begin{aligned}\forall b \forall e \exists f \big(((& isBuilding(b) \land isBuildingElement(e) \land isFloor(f) \land partOf(f,b) \\ & \land partOf(e,f) \land isLoadBearing(e) \land \neg hasType(e,'SLAB') \\ & \land (\text{hasType}(\text{e},'WALL') \\ & \rightarrow \neg separatesFireCompartments(e)) \land isFloorAboveGround(f) \\ & \land \neg isTopmostFloor(f) \land hasBuildingClass(b,'GK1')\big) \\ & \rightarrow (fireResistance(e,'R30') \lor fireResistance(e,'R60') \\ & \lor fireResistance(e,'R90') \lor fireResistance(e,'REI30') \\ & \lor fireResistance(e,'REI60') \lor fireResistance(e,'REI90'))\end{aligned}$$

For this part of the table there is also a footnote, stating that the value in the table is not a requirement if the building is only used for residential or office use. This is an additional exception that has not yet been formalized, but can also be included in the formula:

$$\begin{aligned}\forall b \forall e \exists f \Big(((& isBuilding(b) \land isBuildingElement(e) \land isFloor(f) \land partOf(f,b) \\ & \land partOf(e,f) \land isLoadBearing(e) \land \neg hasType(e,'SLAB') \\ & \land (\text{hasType}(\text{e},'WALL') \rightarrow \neg separatesFireCompartments(e)\big) \\ & \land isFloorAboveGround(f) \land \neg isTopmostFloor(f) \\ & \land hasBuildingClass(b,'GK1') \\ & \land \neg\big(usage(b,'residential') \lor usage(b,'office'))\big) \\ & \rightarrow (fireResistance(e,'R30') \lor fireResistance(e,'R60') \\ & \lor fireResistance(e,'R90') \lor fireResistance(e,'REI30') \\ & \lor fireResistance(e,'REI60') \lor fireResistance(e,'REI90')))\end{aligned}$$

As a last example from the table in Figure 8, load-bearing elements in floors below ground of buildings with building class “GK5” are considered. As this building class has two columns in the table, depending on the number of floors above ground, the rightmost column with a number of floors above ground larger than 6 is selected for this example.

Extracting the requirements:
Types of element: load-bearing building elements
Constraints: building elements in a floor below ground, if the building has a building class “GK5” and more than 6 above-ground floors
Element property: *fireResistance* as text, *materialClass* as text
Value/threshold and operator: “R90“, so structurally sound for at least 90 minutes, material class “A2”
Exceptions: slabs and walls which separate fire compartments

Floors below ground can be formalized in two ways: Either a new predicate is introduced, similar to *isFloorAboveGround(x)* in the formula above, or the predicate is simply negated, assuming all floors have to be considered either above ground or below ground, not both or either of these options. For this formalization a negation of the predicate is chosen to reduce the amount of overall predicates used for the whole formalization.

The value in the table consists of a fire resistance class as well as a building material requirement in this case. To formalize this, new functions to handle building materials and the needed property of the materials are required.

Without considering the number of floors, a formula could look like this:

$$\begin{aligned}\forall b \forall e \exists f \Big(\big((&isBuilding(b) \land isBuildingElement(e) \land floor(f) \land partOf(f,b) \\ &\land partOf(e,f) \land isLoadBearing(e) \land \neg hasType(e,'SLAB') \\ &\land (\text{hasType(e,'}WALL') \rightarrow \neg separatesFireCompartments(e)\big) \\ &\land \neg isFloorAboveGround(f) \land hasBuildingClass(b,'GK5')\big) \\ &\rightarrow (fireResistance(e,'R90') \\ &\land isMaterialClass(buildingMaterial(e),'A2')))\end{aligned}$$

The number of floors can then be added directly into the formula, or by using another formula for better readability. The two options would look as follows:

Option 1:

$$\begin{aligned}\forall b \forall e \exists f, f1, f2, f3, f4, f5, f6, f7 \Big(\big((&isBuilding(b) \land isBuildingElement(e) \\ &\land isFloor(f) \land partOf(f,b) \land partOf(e,f) \land isLoadBearing(e) \\ &\land \neg hasType(e,'SLAB') \land (\text{hasType(e,'}WALL') \\ &\rightarrow \neg separatesFireCompartments(e)\big) \land \neg isFloorAboveGround(f) \\ &\land hasBuildingClass(b,'GK5') \land isFloor(f1) \land isFloor(f2) \\ &\land isFloor(f3) \land isFloor\ (f4) \land isFloor(f5) \land isFloor(f6) \\ &\land isFloor(f7) \land partOf(f1,b) \land partOf(f2,b) \land partOf(f3,b) \\ &\land partOf(f4,b) \land partOf(f5,b) \land partOf(f6,b) \land partOf(f7,b) \\ &\land isFloorAboveGround(f1) \land isFloorAboveGround(f2) \\ &\land isFloorAboveGround(f3) \land isFloorAboveGround(f4) \\ &\land isFloorAboveGround(f5) \land isFloorAboveGround(f6) \\ &\land isFloorAboveGround(f7) \land f1 \neq f2 \land f1 \neq f3 \land f1 \neq f4 \land f1 \\ &\neq f5 \land f1 \neq f6 \land f1 \neq f7 \land f2 \neq f3 \land f2 \neq f4 \land f2 \neq f5 \land f2 \\ &\neq f6 \land f2 \neq f7 \land f3 \neq f4 \land f3 \neq f5 \land f3 \neq f6 \land f3 \neq f7 \land f4 \\ &\neq f5 \land f4 \neq f6 \land f4 \neq f7 \land f5 \neq f6 \land f5 \neq f7 \land f6 \neq f7\big) \\ &\rightarrow (fireResistance(e,'R90') \\ &\land isMaterialClass(buildingMaterial(e),'A2')\ ))\end{aligned}$$

Option 2:

$$\begin{aligned}\forall b \forall e \exists f \Big(\big((&isBuilding(b) \land isBuildingElement(e) \land isFloor(f) \land partOf(f,b) \\ &\land partOf(e,f) \land isLoadBearing(e) \land \neg hasType(e,'SLAB') \\ &\land (\text{hasType}(\text{e},'WALL') \rightarrow \neg separatesFireCompartments(e)) \\ &\land \neg isFloorAboveGround(f) \land hasBuildingClass(b,'GK5') \\ &\land numberOfFloorsAboveGroundLargerThanSix(b) \big) \\ &\rightarrow (fireResistance(e,'R90') \\ &\land isMaterialClass(buildingMaterial(e),'A2')))\end{aligned}$$

With *numberOfFloorsAboveGroundLargerThanSix(b)* defined as:

$$\begin{aligned}\forall b \exists f1, f2, f3, f4, f5, f6, f7(&isBuilding(b) \land isFloor(f1) \land isFloor(f2) \\ &\land isFloor(f3) \land isFloor(f4) \land isFloor(f5) \land isFloor(f6) \\ &\land isFloor(f7) \land partOf(f1,b) \land partOf(f2,b) \land partOf(f3,b) \\ &\land partOf(f4,b) \land partOf(f5,b) \land partOf(f6,b) \land partOf(f7,b) \\ &\land isFloorAboveGround(f1) \land isFloorAboveGround(f2) \\ &\land isFloorAboveGround(f3) \land isFloorAboveGround(f4) \\ &\land isFloorAboveGround(f5) \land isFloorAboveGround(f6) \\ &\land isFloorAboveGround(f7) \land f1 \neq f2 \land f1 \neq f3 \land f1 \neq f4 \land f1 \\ &\neq f5 \land f1 \neq f6 \land f1 \neq f7 \land f2 \neq f3 \land f2 \neq f4 \land f2 \neq f5 \land f2 \\ &\neq f6 \land f2 \neq f7 \land f3 \neq f4 \land f3 \neq f5 \land f3 \neq f6 \land f3 \neq f7 \land f4 \\ &\neq f5 \land f4 \neq f6 \land f4 \neq f7 \land f5 \neq f6 \land f5 \neq f7 \land f6 \neq f7)\end{aligned}$$

As counting floors is not possible in first-order logic, to ensure there are more than 6 distinct floors above ground, at least 7 floors need to be presented in the formula. As variables can have the same values, neither of these floors must be equal to any other.

Other values in the table can be formalized in the same way as the given examples.

6.3.2. Example B

For formalizing natural language rules, the sentences are broken down to smaller pieces.

In the case of the example in Figure 11 this step is not necessary as the sentence is already a very simple one. The statement to formalize in this case is the following: “A ramp has a steepness of at most 10%”.

Extracting the requirements:

Types of element: ramps
Constraints: none
Element property: *steepness* as number
Value/threshold and operator: “10%“, maximum
Exceptions: none

The elements to check are ramps, there are no additional constraints for this requirement. The property to check is the steepness of a ramp and it should have a value of less than or equal 10%. Using this information, a formula would look like this:

$$\forall e\left(\left(isBuildingElement(e)\ \wedge hasType(e,'RAMP')\right) \rightarrow steepness(e) \leq 10\%\right)$$

This reads as ‘For all elements *e*, if *e* is a ramp, the steepness of *e* is less than or equal 10%’.

6.3.3. Example C

For the example in Figure 10, a formula already looks more difficult.

“For vertical access in buildings, stairs or ramps have to be used. An exception is made for undeveloped attic spaces, in this case a ladder or retractable stairs are permitted as well.” (Österreichisches Institut für Bautechnik, 2019f)

Extracting the requirements:
Types of element: stairs, ramps, floors
Constraints: connecting floors
Element property: *none*
Value/threshold and operator: none
Exceptions: undeveloped attic spaces can use ladder or retractable stairs

The first sentence gives the requirement in a simple form while the second sentence adds an exception to the requirement. So first the basic requirement is formalized:

$$\begin{aligned}\forall b((isBuilding(b)\ \wedge\ needsVerticalAccess(b))\\ \rightarrow (hasElementOfType(b,'STAIRS')\\ \vee\ hasElementOfType(b,'RAMP')))\end{aligned}$$

This reads as “For all buildings *b*, if in *b* vertical access is needed, *b* has stairs or *b* has ramps.” Having both stairs and ramps is also permitted. However, this formula would be valid for a building with 5 floors and a ramp only between the first and the second floor, so a further detailing is needed. First a definition has to be found, what “vertical access” in buildings means and in which cases it is required. One such case is accessing the floors above or below the current one, so each floor needs to be connected to the floor above and the floor below through stairs or a ramp. However, also on the same storey vertical access to a different part of the storey may be necessary in case there are elevated platforms that need to be accessed.

For the first part, a formula could look like this:

$$\begin{aligned}\forall f \exists e((isFloor(f) \wedge \neg isTopmostFloor(f))\\ \rightarrow (\ isBuildingElement(e) \wedge connectedTo(e,f)\\ \wedge\ connectedTo(e,floorAbove(f)) \wedge (hasType(e,'RAMP')\\ \vee hasType(e,'STAIRS')))\end{aligned}$$

For all floors f that are not the top most floor, there exists at least one building element *e* which is connected to *f* and the floor above *f* and *e* is a stair or a ramp. It should be noted that only one element is needed to fulfil the formula due to the existential quantifier, so even with elements like for example a column that is connected to both floors but is not a stair or a ramp the formula still evaluates to true as long as one such element exists. As stated before, elements can only be part of a single floor, which is why a different predicate, *connectedTo(x,y),* is used to express the connection between an element and a floor. If an element is part of a floor, it is also connected to this floor, but elements not part of the floor may still be connected to it.

The second part of the requirement adds an exception to this formula when dealing with undeveloped attic spaces. The first formula is changed so that an undeveloped attic is treated differently.

$$\begin{aligned}&\forall f \exists e((isFloor(f)\ \wedge\ \neg isTopmostFloor(f))\\ &\qquad \to \Big(isBuildingElement(e) \wedge connectedTo(e,f)\\ &\qquad \wedge\ connectedTo\big(e, floorAbove(f)\big) \wedge\\ &\qquad \wedge \big(hasType(e,'RAMP') \vee hasType(e,'STAIRS')\big)\Big)\\ &\qquad \vee\ ((isFloor(f) \wedge\ isUndevelopedAttic(floorAbove(f)))\\ &\qquad \to\ (isBuildingElement(e) \wedge connectedTo(e,f)\\ &\qquad \wedge\ connectedTo\big(e, floorAbove(f)\big)\\ &\qquad \wedge \big(hasType(e,'RAMP') \vee hasType(e,'STAIRS')\\ &\qquad \vee hasType(e,'LADDER')\ \vee hasType(e,'RETRACTABLESTAIRS')\big))\end{aligned}$$

For dealing with staircases and ramps to connect elements on the same floor, a function to compute the difference in elevation between those elements is needed and a threshold to indicate when a ramp or a staircase is necessary. In this case it can also be assumed that the elements that need connecting through a ramp or staircase are of type slab. Without this restriction, for example a missing ramp between a chair and a shelf due to their differences in elevation may invalidate the formula.

$$\begin{aligned}&\forall s \exists f \exists a \exists e((isBuildingElement(s) \wedge hasType(s,'SLAB'))\\ &\qquad \to (isFloor(f)\ \wedge buildingelement(a) \wedge buildingelement(e)\\ &\qquad \wedge partOf(s,f) \wedge partOf(a,f) \wedge partOf(e,f) \wedge hasType(a,'SLAB')\\ &\qquad \wedge a\ \neq s\ \wedge\ absoluteDifferenceInElevation(a,s)\\ &\qquad \geq V \wedge connectedTo(e,s) \wedge connectedTo(e,a)\\ &\qquad \wedge \big(hasType(e,'RAMP') \vee hasType(e,'STAIR')\big)))\end{aligned}$$

V is the threshold and needs to be defined before a validation is possible.

6.3.4. Example D

For formalizing the height requirements of lounge rooms from Figure 9, the starting point is again the most general case and exceptions are added to the formula later.

Extracting the requirements:
Types of element: space
Constraints: space used as lounge or office
Element property: *height* as number
Value/threshold and operator: 2,5m, minimum
Exceptions: buildings with less than 4 apartments and terraced houses only need 2,4m minimum

$$\forall s((isElement(s) \land hasType(s,'SPACE') \\ \land \left(usage(s,'lounge') \lor usage(s,'office')\right) \to (heigth(s) \geq 2{,}5m))$$

For all spaces *s* which are used as lounge or office, a height of at least 2,5m is required.

To add the exception, 4 elements referring to distinct apartments need to be added to the formula in order to formalize the requirement of there being no more than 3.

$$\forall b \forall w_1, w_2, w_3, w_4 \exists s((isBuilding(b) \land\ isElement(s) \land partOf(s,b) \\ \land\ isElement(w_1) \land isElement(w_2) \land isElement(w_3) \\ \land isElement(w_4)\ \land hasType(w_1,'APARTMENT') \\ \land hasType(w_2,'APARTMENT')\ \land hasType(w_3,'APARTMENT') \\ \land hasType(w_4,'APARTMENT')\ \land partOf(w_1,b)\ \land partOf(w_2,b) \\ \land partOf(w_3,b) \land partOf(w_4,b) \land\ (hasType(b,'TERRACEDHOUSE') \\ \lor \neg(w_1 \neq w_2 \land w_1 \neq w_3 \land w_1 \neq w_4 \land w_2 \neq w_3 \land w_2 \neq w_4 \land w_3 \neq w_4)) \\ \to ((hasType(s,'SPACE') \land \left(usage(s,'lounge') \lor usage(s,'office')\right) \\ \land heigth(s) \geq 2{,}4m))$$

For all buildings *b* which are terraced houses, 2,4m height are enough for lounges and offices. This is also true for buildings with no more than 3 apartments. In the formula this is expressed by looking for apartments w_1 to w_4 which must all be distinct apartments and then negating this part. This ensures there are at most three apartments in the building.

6.3.5. Example E

To deal with ambiguous wording in a formula, it is possible to formalize it on a high level first and then break it down slowly. However, to validate the formula, the ambiguities still need to be resolved.

Extracting the requirements:
Types of element: first fire extinguishing aids
Constraints: residential buildings and buildings with a usage that requires first fire extinguishing aids
Element property: *none*
Value/threshold and operator: “enough”
Exceptions: none

For the given example in the OIB Guidelines regarding sufficient and appropriate means of first fire extinguishing aids, this could look as follows:

$$\forall b \exists x((isBuilding(b) \wedge ((usage(b,x) \ \wedge requiresFirstFireExtinguishingAids(x)) \\ \vee \ usage(b,'residential'))) \\ \rightarrow hasEnoughFirstFireExtinguishingAids(b))$$

For all buildings *b*, if the purpose of use of *b* is residential or is any purpose of use *x* which requires first fire extinguishing aids, then *b* must have enough suitable first fire extinguishing aids. How to define which fire extinguishing aids are suitable and how many are enough for a building *b* is not defined in this formula but needs to be known in order to validate it in practice. For the formula, *hasEnoughFirstFireExtinguishingAids(x)* is simply another predicate that validates to true or false, it is in short treated as a parameter of a building that has to be defined.

7. Implementation of Example Rules

For the implementation of example rules, the JAVA API provided by Solibri version 9.10.8.34 was used together with Eclipse version 2019-06 M3 as an integrated development environment. Compiling the code and moving the finished file to the Solibri library folder in order for Solibri Office to find and execute the implemented rules was done semi-automatically by utilizing a maven command.

To create a new rule for Solibri Office there are two main steps: The actual programming logic behind the rule has to be implemented in a java class and a graphical user interface for the rule parameters has to be defined based on which input is required from the user. To this end, the "Rule" interface is implemented in the JAVA class that contains the custom checking code. This interface enforces the implementation of the "check" and "preCheck" functions. The pre-check ensures among other things that the intended components are selected, and whether the check is applicable to begin with. Check, as the name suggests, executes the main checking logic and generates the results. It is important to note that the Solibri API is not a standalone API whose functions can be called by separate, external programs to receive results. Instead it makes many internal functions available to the user, which however are ultimately called from within Solibri once the written custom rule class is supplied to it by the user.

7.1. Example A

$$
\begin{aligned}
\forall b \forall e \exists f \Big(\big((&isBuilding(b) \land isBuildingElement(e) \land isFloor(f) \land partOf(f,b) \\
&\land partOf(e,f) \land isLoadBearing(e) \land \neg hasType(e,'SLAB') \\
&\land (\mathrm{hasType}(\mathrm{e},'WALL') \rightarrow \neg separatesFireCompartments(e)\big) \\
&\land isFloorAboveGround(f) \land \neg isTopmostFloor(f) \\
&\land hasBuildingClass(b,'GK1') \\
&\land \neg\big(usage(b,'residential') \lor usage(b,'office')\big)\Big) \\
&\rightarrow (fireResistance(e,'R30') \lor fireResistance(e,'R60') \\
&\lor fireResistance(e,'R90') \lor fireResistance(e,'REI30') \\
&\lor fireResistance(e,'REI60') \lor fireResistance(e,'REI90')))
\end{aligned}
$$

Starting the implementation, first the checks for load bearing elements in buildings of class GK1 are implemented and then the implementation is extended to contain other checks as well.

All load bearing elements are filtered by checking which components in the model have the property "LoadBearing" and for which of these components the value is set to true. Elements of type "Slab" are then removed from this set of elements. Components of type "Wall" which separate fire compartments need to be removed as well. There is a Boolean IFC property called "Compartmentation" which indicates whether an element is designed to be part of a fire compartment or not. However, as this property is typically not set in IFC models, an alternate way to identify those elements is added by giving the user the option to choose elements which form fire compartments by adding an additional element filter to the rule parameters. The elements selected with this filter are then excluded from the list of load bearing components considered for this part of the rule.

As the building class of the building is needed to check for this rule, the user is required to select the correct building class manually before starting the check.

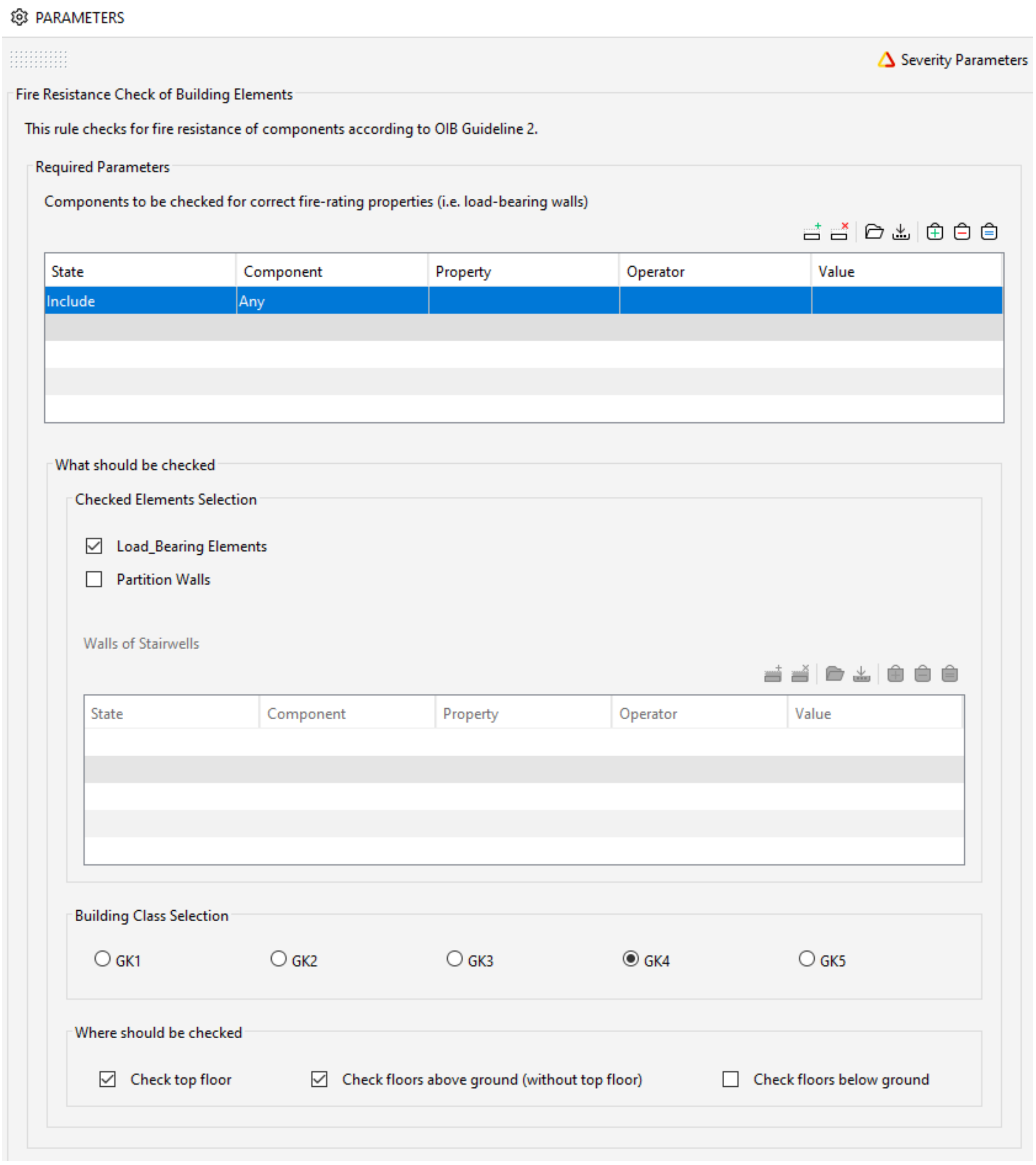
The usage of the building could be extracted from the “OccupancyType” property or the “MarketCategory” property of the building. Again, an option to manually select the usage is added as these properties typically do not contain any values in IFC models. Any manual input from the user will override any values stored in the model. In the case the user does not input a value and there is none stored in the model either, a message to tell the user additional input is required is displayed.

In a next step the floors of the building are divided into floors above ground and floors below ground. For this the “AboveGround” property of IFCBuildingStorey elements could be used, however, in this implementation the bottom elevation of the floor is utilised and floors with a negative bottom elevation are considered below ground while the others are considered above ground.

To find the topmost floor, the floor with the highest bottom elevation is chosen first. However, in architectural models this floor frequently contains only the roof of the building, but not the topmost spaces in the building. To correct this, the floor with the highest bottom elevation which also contains an IFC-Space is chosen as the topmost floor. As this is not necessarily always the actual topmost floor the user wishes to check, an additional option to enter the topmost floor manually is added, as well as the option to select elements which are treated as part of the topmost floor even though they are not part of the computed topmost floor.

The required parameters of the rule are displayed in Figure 14. Additional parameters giving the user the possibility to add supplementary information not currently in the model can be seen in Figure 15. If these are left empty, the corresponding values in the IFC model are checked if those values are available.

This implementation only considers the fire resistance class. The value is located in the “FireRating” property. For the additional material class of A2, which is required in some cases, there is currently no value specified in the IFC model. The name of the material itself can be stored in IFCMaterial, however, the classification cannot be extracted from the model. As material names are not standardized and can contain any text, a limitation to certain values as well as a database with the material classes for these values would be necessary to gain the information required for an automation. As maintaining such a database for any given model without material name restrictions would not be feasible, the checked elements are split into multiple categories in the checking result. One category contains elements without a compliant fire resistance class, which needs to be corrected for the model to pass a compliance check. Another categories contains elements which would pass the check according to their fire resistance class, but still need to be checked for their material class manually. Elements with a sufficient material class can be defined in the rule parameters and are then treated as compliant if their fire resistance class is compliant with the regulation and they are part of this selection.



PARAMETERS

Severity Parameters

Fire Resistance Check of Building Elements

This rule checks for fire resistance of components according to OIB Guideline 2.

Required Parameters

Components to be checked for correct fire-rating properties (i.e. load-bearing walls)

State	Component	Property	Operator	Value
Include	Any			

What should be checked

Checked Elements Selection

☑ Load_Bearing Elements

☐ Partition Walls

Walls of Stairwells

State	Component	Property	Operator	Value

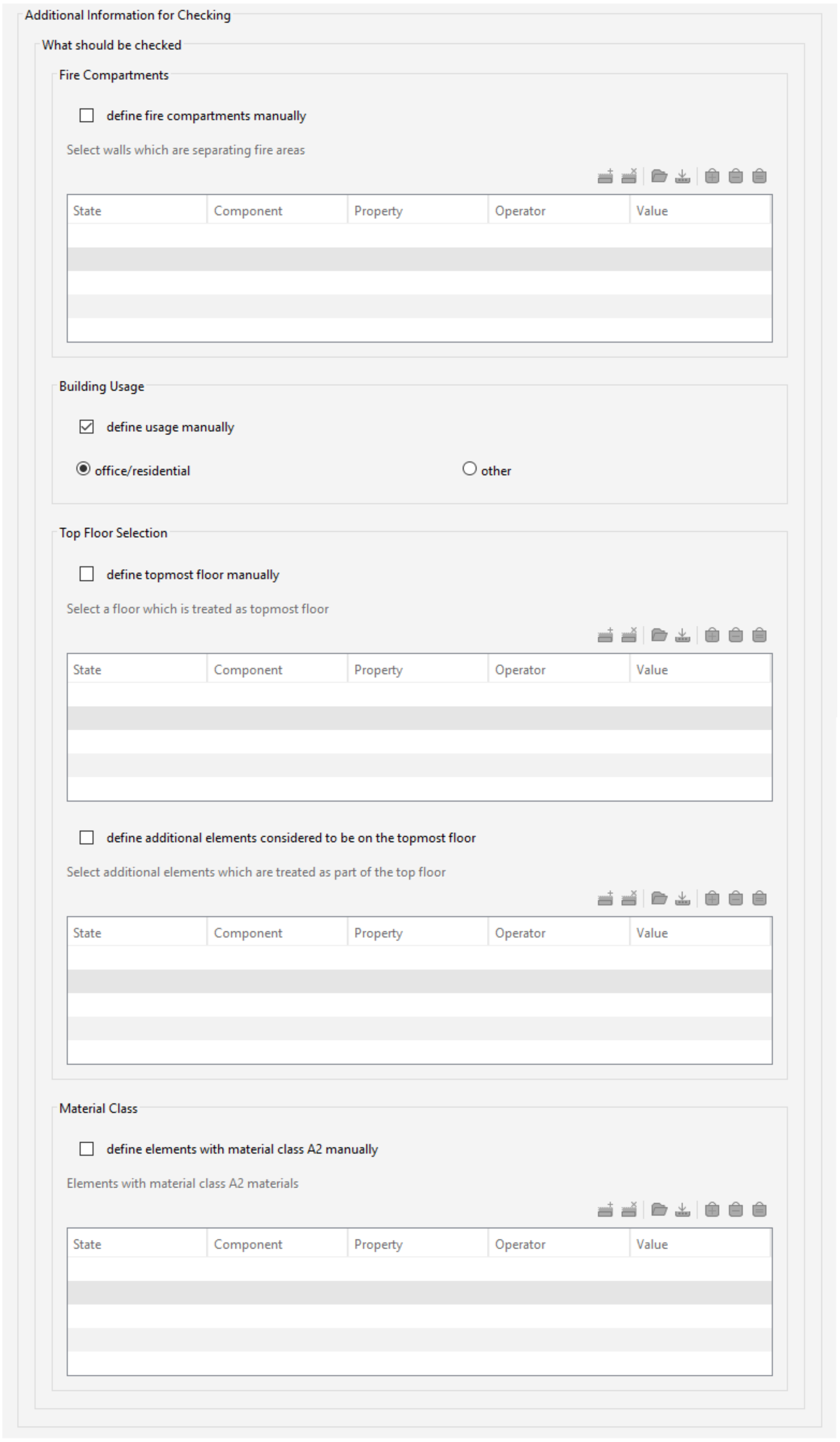
Building Class Selection

○ GK1 ○ GK2 ○ GK3 ● GK4 ○ GK5

Where should be checked

☑ Check top floor ☑ Check floors above ground (without top floor) ☐ Check floors below ground

Figure 14: Parameters of example A rule – required user input to check for fire resistance values



Additional Information for Checking

What should be checked

Fire Compartments

☐ define fire compartments manually

Select walls which are separating fire areas

State	Component	Property	Operator	Value

Building Usage

☑ define usage manually

[✓ office/residential] other

Top Floor Selection

☐ define topmost floor manually

Select a floor which is treated as topmost floor

State	Component	Property	Operator	Value

☐ define additional elements considered to be on the topmost floor

Select additional elements which are treated as part of the top floor

State	Component	Property	Operator	Value

Material Class

☐ define elements with material class A2 manually

Elements with material class A2 materials

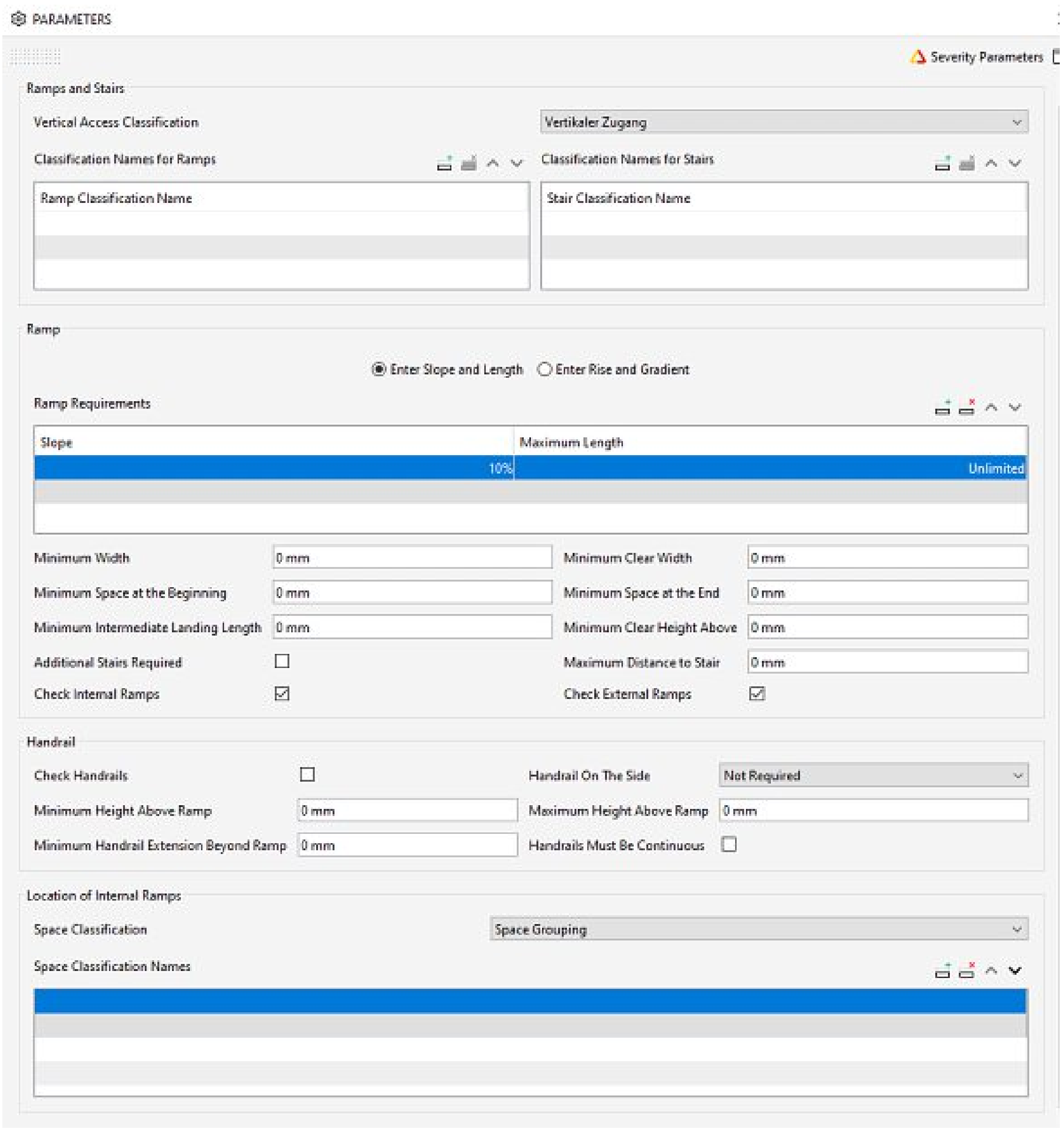
State	Component	Property	Operator	Value

Figure 15: Additional parameters for example A rule

7.2. Example B

$$\forall e\left(\left(isBuildingElement(e)\ \wedge hasType(e,'RAMP')\right) \rightarrow steepness(e) \leq 10\%\right)$$

Solibri Office already offers a pre-defined rule to check the accessibility of ramps: Rule 207, Accessibility Ramp Rule. This rule is more extensive than simply checking the slope of a ramp, it also includes, among others, checks for the width of the ramp, availability of sufficient space at the beginning and end of the ramp, a check for handrails as well as a check for the existence of ramps in certain rooms. Details concerning this rule can be found on the rules's help site (Solibri Inc.). For the given example only the slope is checked, by setting the other parameter values to 0, as seen in Figure 16, they are ignored in the checking.



PARAMETERS

Severity Parameters

Ramps and Stairs

Vertical Access Classification	Vertikaler Zugang
Classification Names for Ramps	Ramp Classification Name
Classification Names for Stairs	Stair Classification Name

Ramp

(● Enter Slope and Length) ○ Enter Rise and Gradient

Ramp Requirements

Slope	Maximum Length
10%	Unlimited

Minimum Width	0 mm	Minimum Clear Width	0 mm
Minimum Space at the Beginning	0 mm	Minimum Space at the End	0 mm
Minimum Intermediate Landing Length	0 mm	Minimum Clear Height Above	0 mm
Additional Stairs Required	☐	Maximum Distance to Stair	0 mm
Check Internal Ramps	☑	Check External Ramps	☑

Handrail

Check Handrails	☐	Handrail On The Side	Not Required
Minimum Height Above Ramp	0 mm	Maximum Height Above Ramp	0 mm
Minimum Handrail Extension Beyond Ramp	0 mm	Handrails Must Be Continuous	☐

Location of Internal Ramps

Space Classification	Space Grouping
Space Classification Names	

Figure 16: Parameters of example B rule – selecting ramps with a steepness of more than 10%

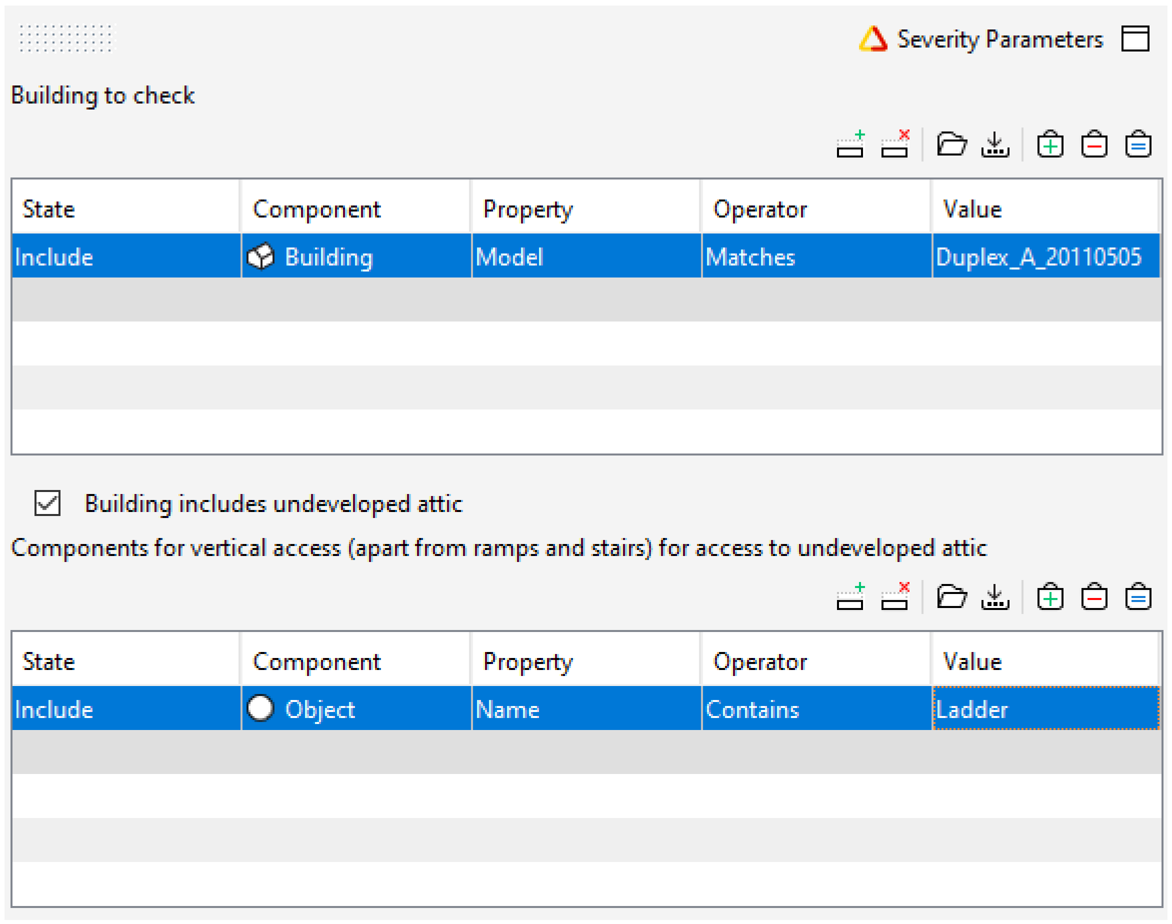
7.3. Example C

$$
\begin{aligned}
\forall f \exists e ((&isFloor(f)\ \wedge\ \neg isTopmostFloor(f)) \\
&\rightarrow \Big(isBuildingElement(e) \wedge connectedTo(e,f) \\
&\wedge\ connectedTo\big(e, floorAbove(f)\big) \wedge \\
&\wedge \big(hasType(e,'RAMP') \vee hasType(e,'STAIRS')\big)\Big) \\
&\vee\ \ ((isFloor(f) \wedge\ isUndevelopedAttic(floorAbove(f))) \\
&\rightarrow\ \ (isBuildingElement(e) \wedge connectedTo(e,f) \\
&\wedge\ connectedTo\big(e, floorAbove(f)\big) \\
&\wedge \big(hasType(e,'RAMP') \vee hasType(e,'STAIRS') \\
&\vee hasType(e,'LADDER')\ \vee hasType(e,'RETRACTABLESTAIRS')))
\end{aligned}
$$

For this implementation all floors in the model are extracted and sorted according to their elevation. For each floor the one directly above it as well as the one directly below it is stored. For the bottom floor there is no floor below and for the top floor there is no floor above. Then the floors are considered as pairs, checking if there is a ramp or stair element on each floor that connects to the floor directly above. Ramps and stairs are identified according to their type: IFCRamp and IFCStair. For ladders and retractable stairs there is no concrete type available in IFC, so the user is required to define which elements in the model can be considered as ladders or retractable stairs. This is only necessary if the model contains an undeveloped attic. As this identification is also not standardized in IFC, an option to decide whether or not the model contains an undeveloped attic is included in the parameters for the check. In order to deal with the possibility of multiple buildings in the model, the user is asked to select which building to check. As this was not yet possible to achieve with a drop-down menu at the time of implementation, a component filter is used to select the building. The parameters of the rule are shown in Figure 17.

To verify if a ramp or stair is connected to two consecutive floors, it is assumed that the ramp or stair element is part of the lower floor in the IFC model, the bottom elevation of the element corresponds to the bottom elevation of the floor and the top elevation of the element corresponds to the bottom elevation of the floor above. A range of 30cm around the elevation value is included in the implementation, inside which the rule still identifies the element as connected. If the building contains an undeveloped attic, the rule first checks if there is a ramp or stair connecting to this attic. If not the elements selected in the second component filter, defining additional elements to access the attic, are considered. It is assumed that those elements share the bottom elevation with the floor below and the top elevation with the top floor, where the attic is located, as any ramp or stair element would, or the elements itself are part of the topmost floor, which would be the case for retractable stairs. Unlike in example A, the topmost floor is not required to have a space element, as rooftop terraces without a specified space should be reachable as well.

For checking accessibility within a floor, the slab elements in each floor are compared regarding their elevation. In the implementation a difference of more than 30cm, which is more than the height of a normal step in a stair, is used as threshold to identify the need of ramps or stairs for accessibility.



Building to check

State	Component	Property	Operator	Value
Include	Building	Model	Matches	Duplex_A_20110505

☑ Building includes undeveloped attic

Components for vertical access (apart from ramps and stairs) for access to undeveloped attic

State	Component	Property	Operator	Value
Include	Object	Name	Contains	Ladder

Figure 17: Parameters of example C rule – checking vertical access of a building

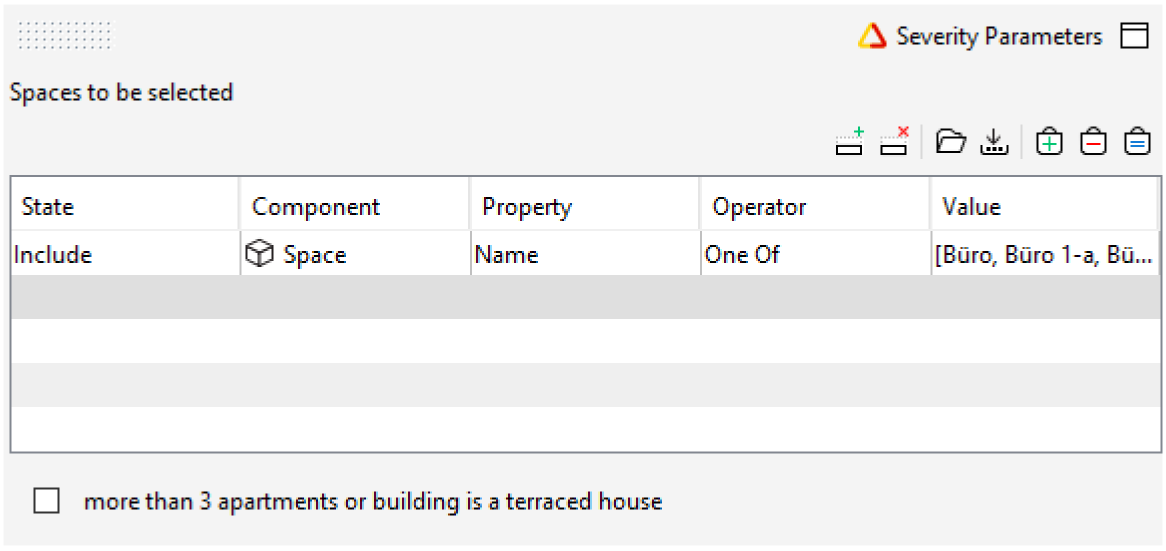
This implementation only checks whether or not each floor is accessible through a stair or ramp, other requirements such as the width of those elements or the availability of handrails is not part of this rule. This rule also does not check the horizontal accessibility and the placement of walls for each room in each floor.

7.4. Example D

$$\begin{aligned}
&\forall b \forall w_1, w_2, w_3, w_4 \exists s((isBuilding(b) \wedge isElement(s) \wedge partOf(s,b) \\
&\quad \wedge isElement(w_1) \wedge isElement(w_2) \wedge isElement(w_3) \\
&\quad \wedge isElement(w_4) \wedge hasType(w_1, 'APARTMENT') \\
&\quad \wedge hasType(w_2, 'APARTMENT') \wedge hasType(w_3, 'APARTMENT') \\
&\quad \wedge hasType(w_4, 'APARTMENT') \wedge partOf(w_1, b) \wedge partOf(w_2, b) \\
&\quad \wedge partOf(w_3, b) \wedge partOf(w_4, b) \wedge (hasType(b, 'TERRACEDHOUSE') \\
&\quad \vee \neg(w_1 \neq w_2 \wedge w_1 \neq w_3 \wedge w_1 \neq w_4 \wedge w_2 \neq w_3 \wedge w_2 \neq w_4 \wedge w_3 \neq w_4)) \\
&\quad \rightarrow ((hasType(s, 'SPACE') \wedge (usage(s, 'lounge') \vee usage(s, 'office')) \\
&\quad \wedge heigth(s) \geq 2{,}4m))
\end{aligned}$$

For this rule the height of lounges and office spaces has to be considered. IFCSpace contains a quantity “AverageHeight”, which is calculated as difference between the top of the slab below the space and the bottom of the slab above the space. To simply check if the calculated value is above the required minimum, a similar pre-defined rule as in example B can be used.

To additionally identify if the building is a terraced house, the “OccupancyType” property or the “MarketCategory” property of the building may again be used, as in example A. However, a possibility for the user to add this information via the rule properties is again added. Additionally, a checkbox in the properties lets the user define whether or not there are more than three apartments in the building. IFCZone or IFCSpatialZone may be used to identify apartments which consist of multiple spaces each, though, in practice these elements are usually not part of the IFC model, which makes it necessary to provide a possibility for the user to enter this information manually. The parameters for checking lounge and office spaces are displayed in Figure 18.



Spaces to be selected

State	Component	Property	Operator	Value
Include	Space	Name	One Of	[Büro, Büro 1-a, Bü...

☐ more than 3 apartments or building is a terraced house

Figure 18: Parameters of example D rule – required room height in lounge and office spaces

7.5. Example E

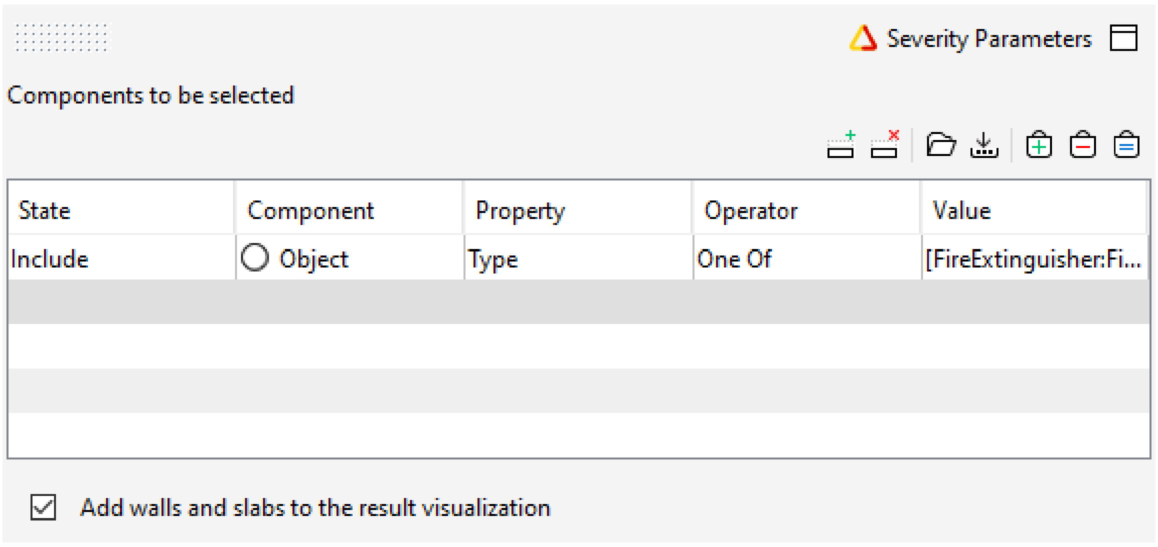
$$\forall b \exists x((isBuilding(b) \wedge ((usage(b,x) \wedge requiresFirstFireExtinguishingAids(x)) \\ \vee usage(b,'residential'))) \\ \rightarrow hasEnoughFirstFireExtinguishingAids(b))$$

As already stated when trying to formalize the requirement of “sufficiently many first fire extinguishing aids”, this is not possible to check without further information, such as a clear number or number range or a possibility to acquire the missing information through other means such as a known relation between the number of fire extinguishers and the number of storeys or a storey’s gross area, from which the required information can be derived.

It is possible to implement a checking rule which requires a distinct number from the user and then checks if more first fire extinguishing aids than the given number are present in the model. However, in the case that no actual decision whether or not the model is compliant with this requirement is expected from the checking, giving the user the overall number of such elements in the model and letting them decide if those elements are enough may be easier to handle than having to define an absolute minimum. As additional information the elements are highlighted in the model, giving the user an idea of their location and the distances between them.

As there is currently no IFC element type for first fire extinguishing aids, it is not known which elements are considered first fire extinguishing aids in the model. For this reason, the user is required to select the types of elements which fall in this category, as well as decide whether or not the usage of a building actually requires the check for such elements at all.

The parameters of the implemented rule give the user the option to select the elements considered first fire extinguishing aids as well as the option to display the surrounding walls and slabs of each floor for better orientation. The parameters are displayed in Figure 19.



Components to be selected

State	Component	Property	Operator	Value
Include	Object	Type	One Of	[FireExtinguisher:Fi...

☑ Add walls and slabs to the result visualization

Figure 19: Parameters of example E rule – first fire extinguishing aids in a building

8. Test Models

Four different models were chosen to test the implemented rules. Due to the fact that most IFC models in practice do not contain all the relevant parameters to test for compliance with the selected paragraphs of the OIB guidelines, such as fire resistance classes or fire compartments, a possibility to edit the test models was essential. For this reason, three of the test models are demo models. One already contains the needed information and the other two are used to demonstrate an iteration of testing the model, then correcting the mistakes and then checking it again. The last test model is an IFC file which is used as-is, without any changes.

Additionally, models generated with different proprietary software suites are used, namely ArchiCAD and Revit, in order to ensure the implementations work with IFC files from different sources.

8.1. A-Null Example House

This example building was designed and created as a BIM model using ArchiCAD by A-Null Bausoftware to be used in the course of the company’s training classes. It was thankfully made available to the author by A-Null Bausoftware for the purpose of testing and validating the correctness of the implemented rules in Solibri Office.

The exemplary building is a mixed-use building with four above-ground and one underground floor. On the ground floor there are offices and meeting rooms, on each of the other above-ground floors there are one or two residential units, and on the second floor there is also a doctor's office. In the basement there are storage areas and a parking garage as well as the boiler room. The building site is in an urban environment, the neighbouring buildings and streets are included in the model as base geometries without details, as seen in Figure 20. The model also includes furniture.

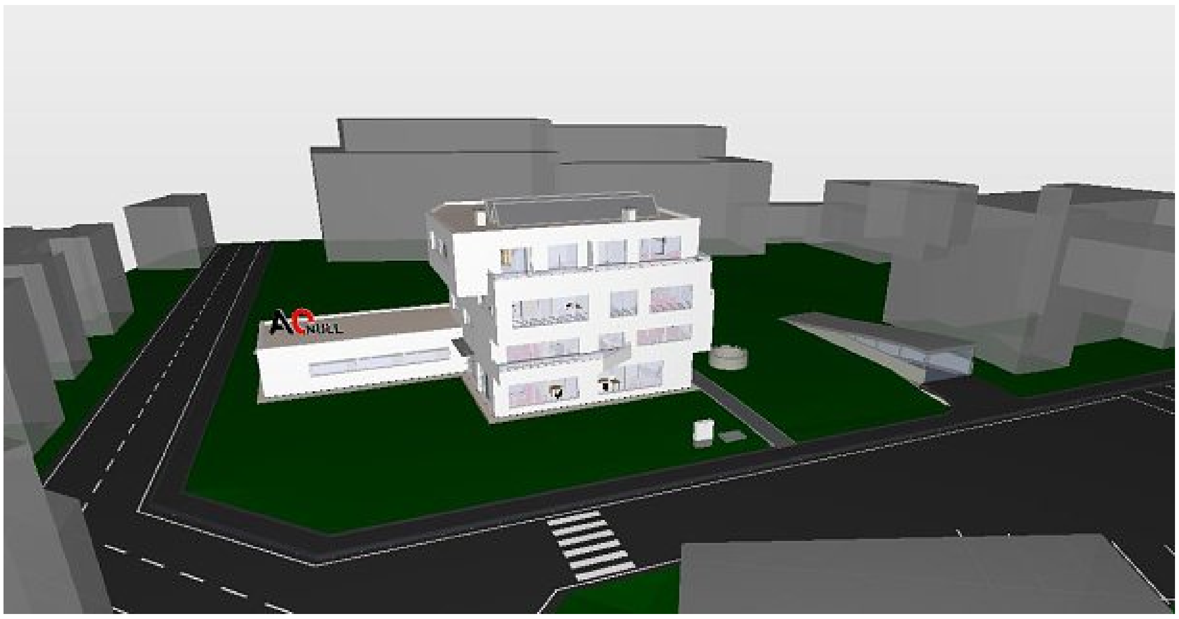


Figure 20: A-Null example house with surrounding urban environment in Solibri Office

8.2. Revit Sample Building Single Family House

Autodesk provides publicly available sample project files for Autodesk Revit. The used example is a single-family home with two storeys in a hillside location. The model includes surroundings, such as terrain and trees as seen in Figure 21, as well as furniture.

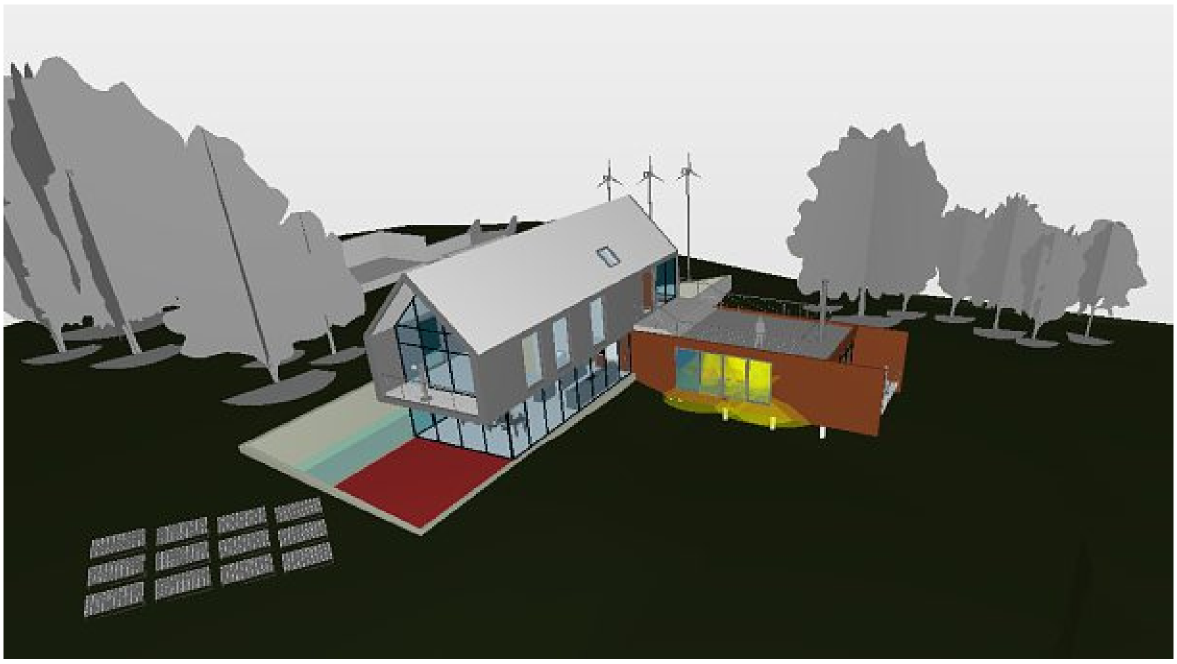


Figure 21: Revit sample building single-family house in Solibri Office

8.3. Revit Sample Building Golden Nugget

The Golden Nugget Project is a mixed-use Revit sample building in an urban location provided with the Revit 2021 student version. The building is located in Graz, between Schönaugasse and Grazbachgasse and consists of one below-ground floor with a garage and a meeting room and six above-ground floors with apartments and offices. The model includes furniture, as well as surrounding environment as base geometries, visible in Figure 22. The building consists of a two-storey part in the middle and an added 6-storey part to the right as seen in Figure 23.

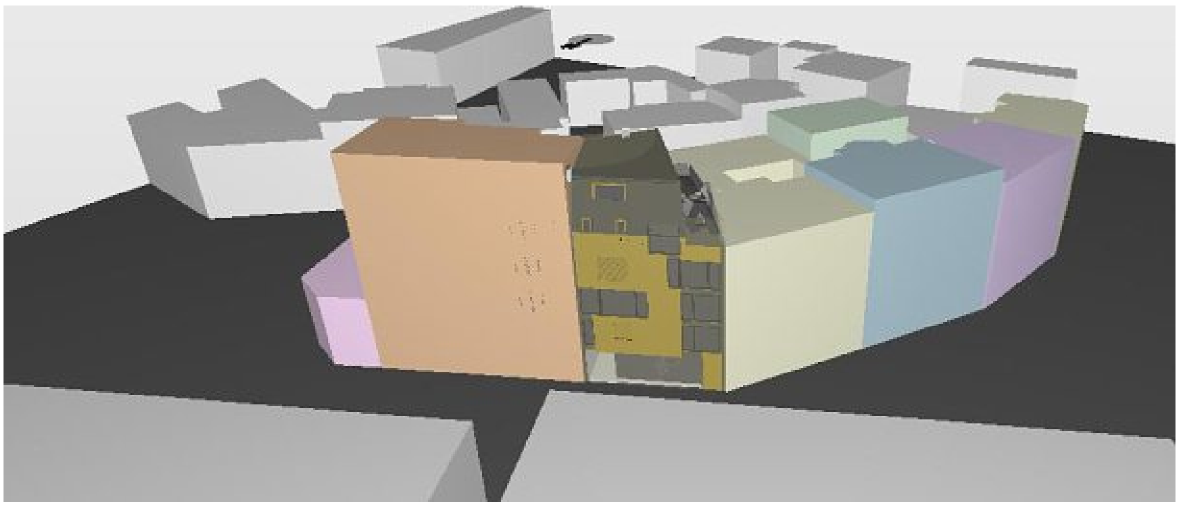


Figure 22: Revit Sample Golden Nugget with Surroundings in Solibri Office

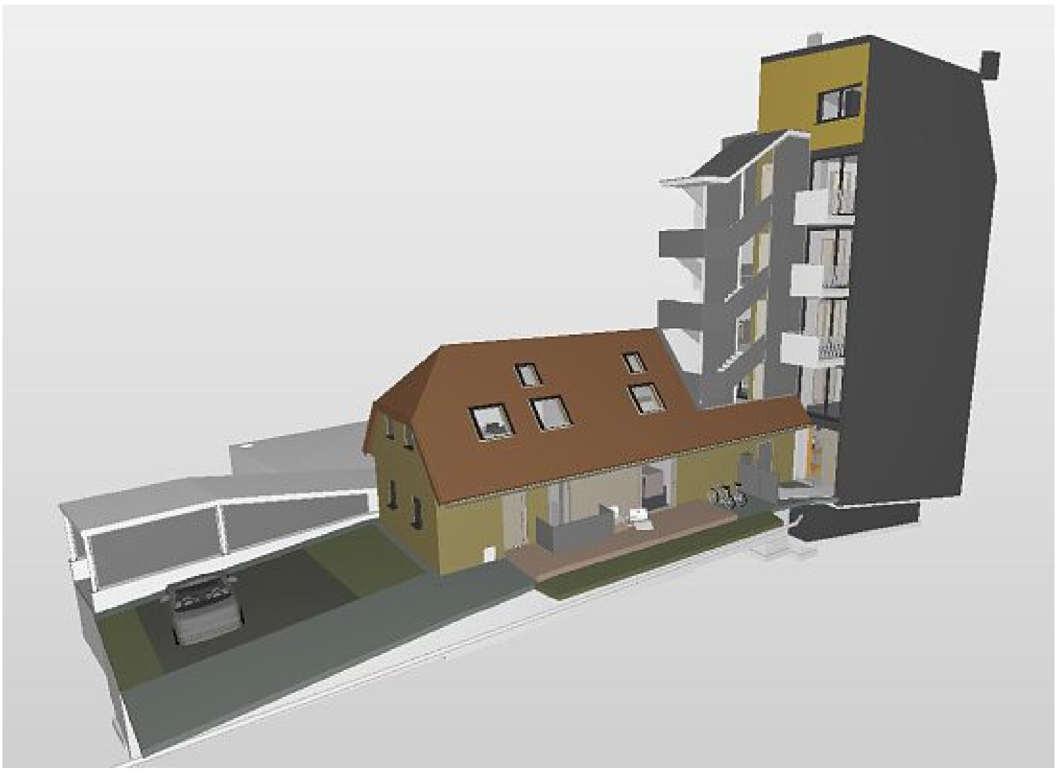


Figure 23: Revit sample building Golden Nugget in Solibri Office

8.4. Duplex Apartment Building

The Duplex Apartment model was produced for a competition in Weimar, Germany. The model was updated and distributed under Creative Commons license cc-by-sa-3.0 by the University of Auckland, New Zealand in 2011 and is available as part of the University's open IFC model repository. The building consists of two storeys with two apartments in a box-shaped geometry, as seen in Figure 24. Furniture is included, but no surrounding area is part of the model.

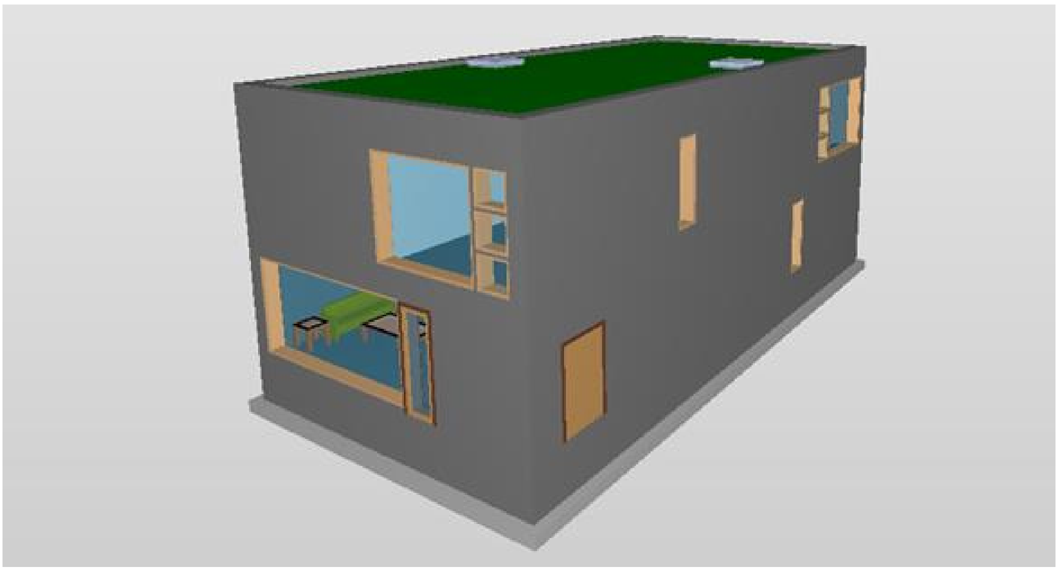


Figure 24: Duplex Apartment Building model in Solibri Office

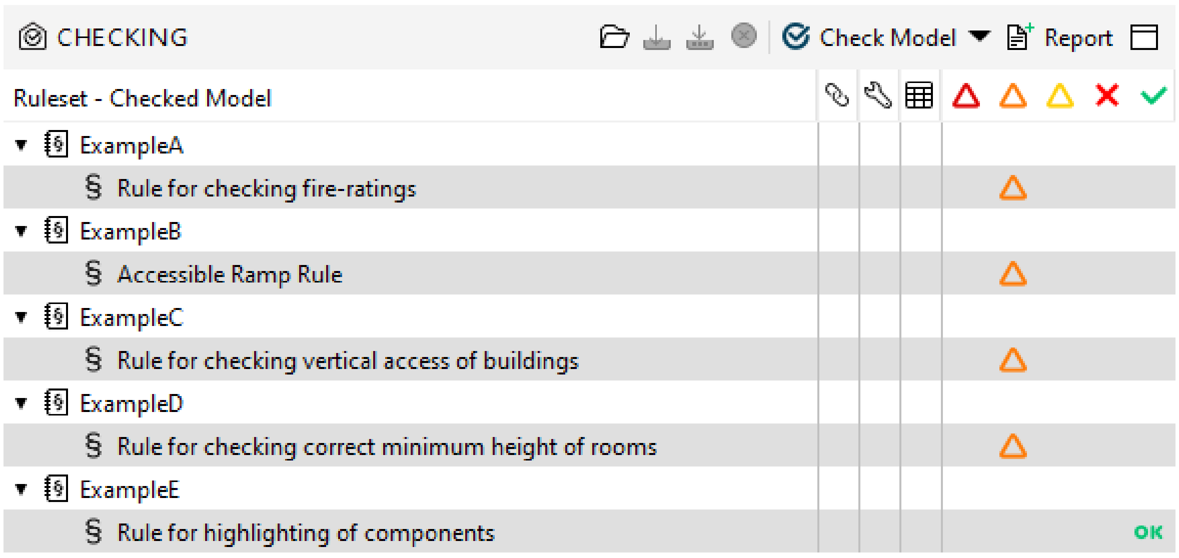
9. Validation and Results

To validate the correct functionality of the implemented rules, the example buildings introduced in chapter 8 are checked using Solibri Office with different rule parameters, and the results are then verified through additional manual checks.

As an additional step, the test models were checked for certain prerequisites before the actual tests in a so-called pre-check, without which a validation of the implemented rules would not be meaningful. For example, it was determined whether certain objects are present in the IFC model or whether values for certain parameters of objects are available, so that checking those objects or parameters would yield useful results. While the implementations of the example rules provide some basic checks regarding the availability of used parameters, in order to avoid errors during the execution, it is by no means their primary focus.

9.1. A-Null Example House

The results of the checks for the A-Null Example House look as follows:



Ruleset - Checked Model	Result
ExampleA	
Rule for checking fire-ratings	△
ExampleB	
Accessible Ramp Rule	△
ExampleC	
Rule for checking vertical access of buildings	△
ExampleD	
Rule for checking correct minimum height of rooms	△
ExampleE	
Rule for highlighting of components	OK

Figure 25: Checking results A-Null Example House

Looking at the results in more detail, a number of elements not compliant with the implemented OIB examples were found. This is to be expected as the model is used to teach users how to use Solibri Office and to help them find mistakes in their own building models.

Regarding example A, all the components in the model were part of the check and the model was checked for the compliance of load-bearing elements as well as partition walls on all floors. The building was treated as building class 4. The top-most floor was identified correctly, it is the “3.OG” (third floor) in the model as the building storey above it contains only the roof of the building. As it is common practice to draw the roof on a separate layer, which then commonly leads to being identified as a separate storey in the model, the top-most floor is not necessarily the floor with the highest elevation. To still check any elements which might be on this floor above the top-most one, it is treated as a regular floor above the ground for checking purposes. The errors in the model which were found when executing the implemented rule include beams and columns in the basement which are load-bearing elements but do not contain any value in the fire rating property. As in below ground floors of GK4 buildings the material class A2 is required but not checked in the implementation, elements in the basement which meet the expected fire resistance values are also part of the result, requiring the user to manually check the material class. The largest part of the non-compliant elements is due to limiting the implementation to fire ratings R30/60/90 as well as REI30/60/90 and EI30/60/90. However, some components in the model have a fire resistance class of RE90, which is compliant with the regulations, but not currently part of the implementation. One such example is visible in Figure 26. This case acts as a good example to the difficulties of implementation regarding the amount of possible values that have to be considered.

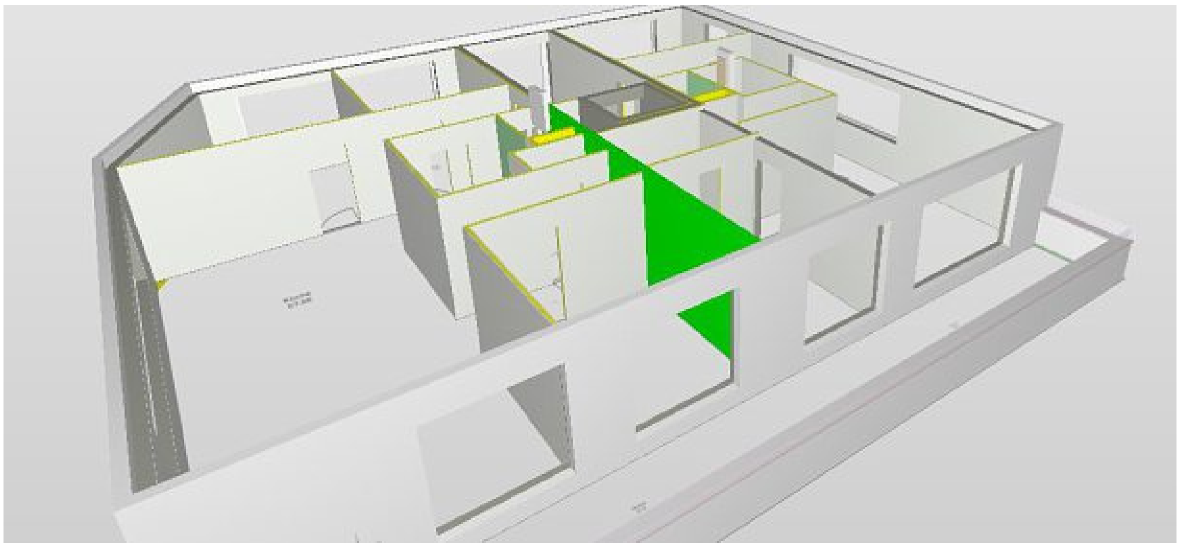


Figure 26: Non-compliant wall element to fire resistance requirements with fire rating RE90

Example B shows one ramp in the model with a steepness of more than 10%. This ramp is part of the garage entrance, as shown highlighted in Figure 27.

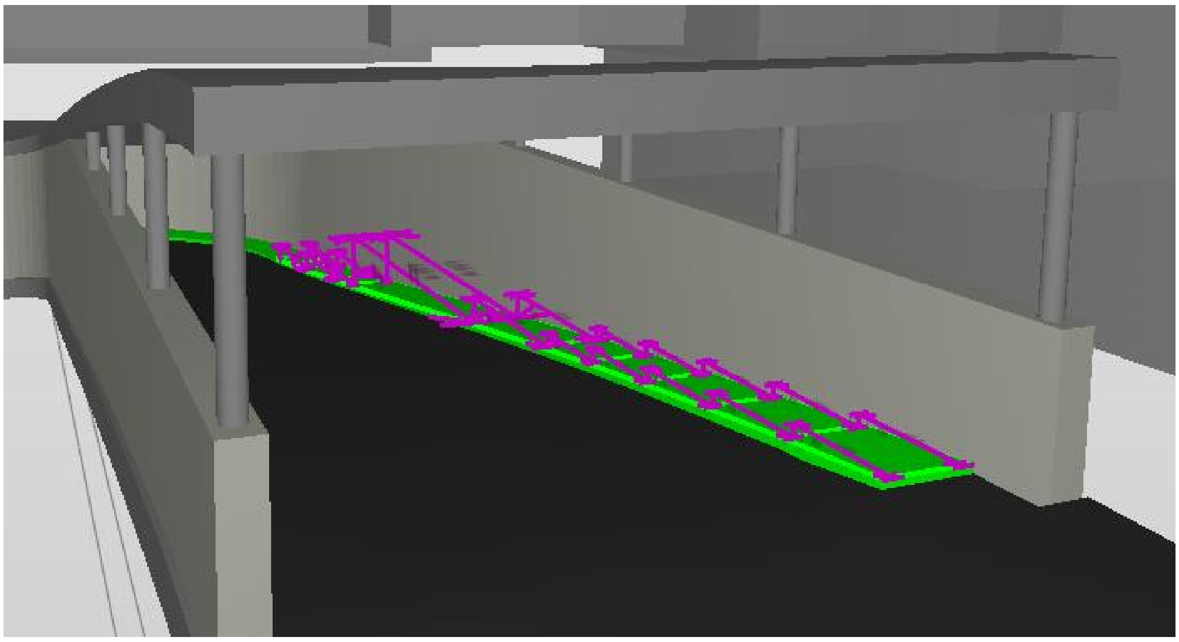


Figure 27: Ramp with a steepness of more than 10%

The check for vertical access of a building in example C also found mistakes in the model. While looking at the model manually, all floors – except the storey containing the building foundations – seem to be accessible through stairs. The errors found stem from the way the position of the stairs is checked in the implementation: A stair is assumed to be part of the floor from which it rises to the next floor. However, in the model the stairs between the second and third floor are modelled as part of the first floor for example, and thus not correctly identified by the implementation. While it is possible to identify stairs and ramps between floors according to their elevation instead of the floor they are related to, referring to the relation was chosen as a more meaningful check for a model, also in terms of model quality overall.
Another reason for detected mistakes in the model was the check of accessibility inside each floor, for which slab elements farther apart than 30cm in height are checked for connecting stairs or ramps. As there was no maximum distance specified in the implementation, this includes slabs for floors and ceilings on the same storey, which do not need vertical access necessarily. A change in the implementation to include only slabs which form the bottom boundary of spaces may be advisable for similar checks.

Example D checks spaces for office or lounge use for a required minimum height. As the building contains more than 3 apartments, a height of 2,5m is needed for these spaces. There is one living room on the third floor which does not fulfil this requirement, however, when looking more closely at the model, the height of the room itself is more than 2,5m, but the space element is elevated from the floor by about 15cm, as seen in Figure 28. This leads to a wrong value in the calculation, making the space less than 2,5m high.

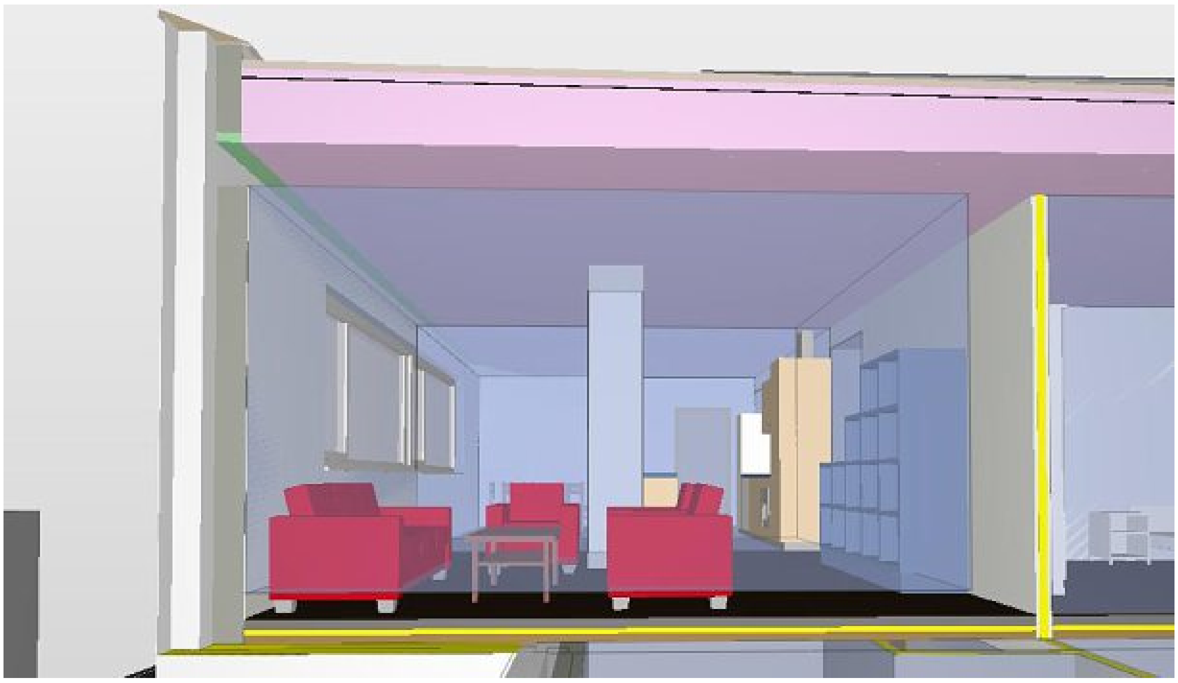
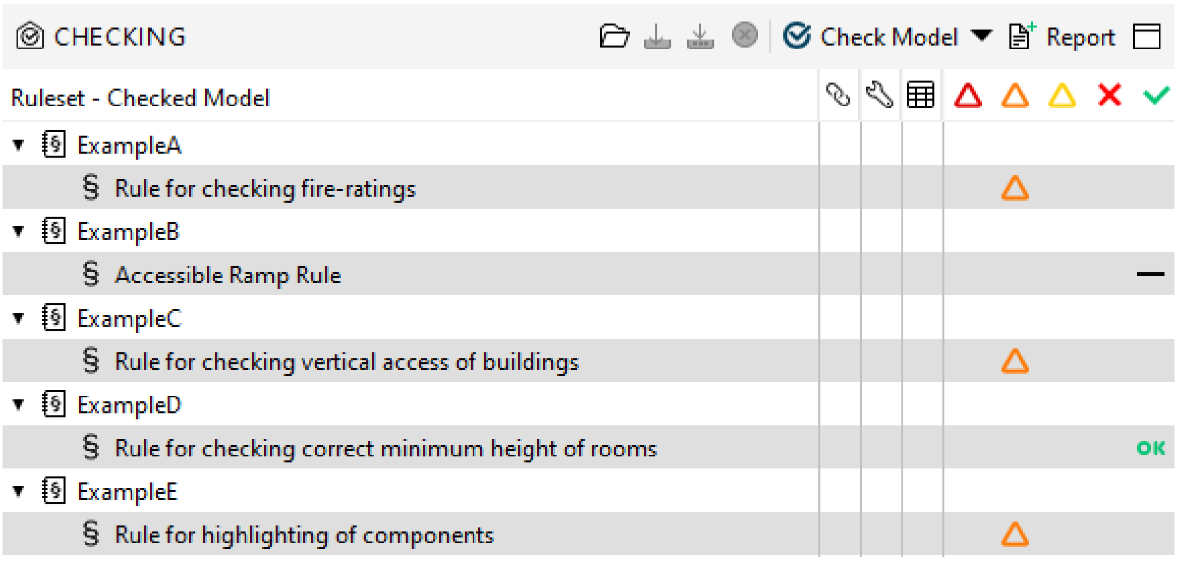


Figure 28: Space with incorrect height

For example E there are no fire extinguishers in this demo building, so the rule does not generate any results. The implementation itself can be tested with other elements, as the rule simply counts and highlights elements in the model. However, as fire extinguishers were added to other test models, example E is discussed in more detail then.

9.2. Revit Sample Building Single Family House



Ruleset - Checked Model	Result
ExampleA	
§ Rule for checking fire-ratings	△
ExampleB	
§ Accessible Ramp Rule	—
ExampleC	
§ Rule for checking vertical access of buildings	△
ExampleD	
§ Rule for checking correct minimum height of rooms	OK
ExampleE	
§ Rule for highlighting of components	△

Figure 29: Checking results Revit sample model 1

Regarding example A, the top-most floor was again correctly identified. As the building is considered as building class GK1, and used as residential building, there are no requirements regarding fire resistance for all floors above the ground. For below ground floors a fire resistance of "R60" for load-bearing elements is required. Two floors are considered below ground, the floor containing the foundations, as well as part of the lower part of the building, visible in brown in Figure 21. The rest of the model is considered above ground, according to the building storey information. However, as all walls have a fire resistance of “R60” in the model, the requirements for building class GK1 still hold. In a manual check it becomes clear that the elements in the model are not always assigned to the correct building storeys. Figure 30 shows the elements of storeys “Level 1 Living Rm.” (on the bottom) and “Level 1” (on top), which are on top of each other in the model, but the elements are on the same level in the building. This makes checking difficult, as the implementation relies on correct relation between floors and elements on the floors.

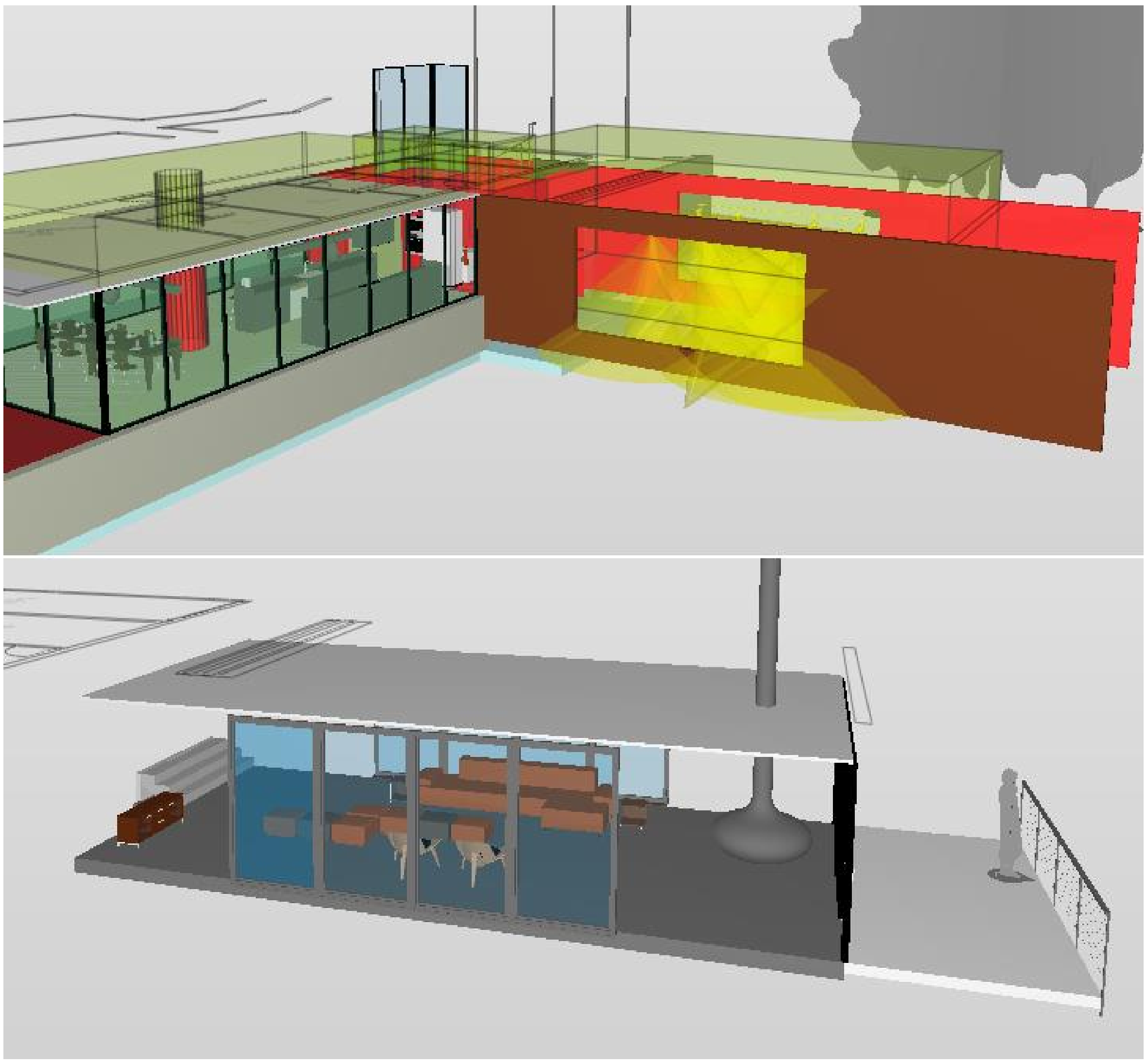


Figure 30: Assignment of elements to building storeys

When using GK3 as building class, errors are detected. For testing purposes only walls were given a fire resistance value, columns are therefore detected as not compliant due to a lack of this value. Additionally, some walls of above-ground floors were given a fire resistance value of “ND”, which is also detected as not compliant by the rule.

Example B cannot be tested as there are no ramps in the model.

The vertical access of buildings in example C is not given for the storey containing the building foundations as well as for the upper level of the building, lacking a stair or ramp to the roof, which is part of a separate floor again. Access within floors is again not computed correctly due to slabs used as ceilings also being checked for accessibility.

The height of spaces is compliant with the regulation according to the implemented rule. However, the boundaries of spaces are not modelled correctly. This is also visible in Figure 30, where the transparent green cubes reach above the ceilings of the rooms they are meant to represent. A manual check shows that the height of the rooms is indeed compliant, but this example also shows that computed values are not always reliable and depend greatly on the quality of the model.

To show fire extinguishers in the building, highlighting lines are displayed and act as beacons to easily find the relevant elements. Additionally, the overall number of elements is displayed in the checking results and the results are categorized by floor. The result for the upper floor is displayed in Figure 31.

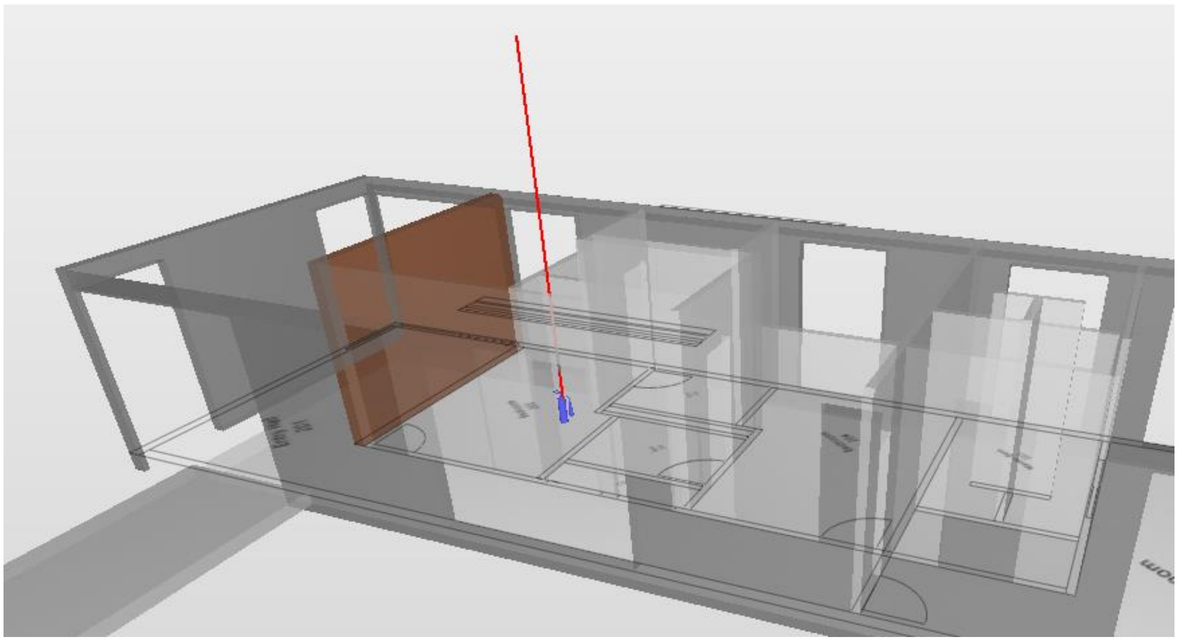
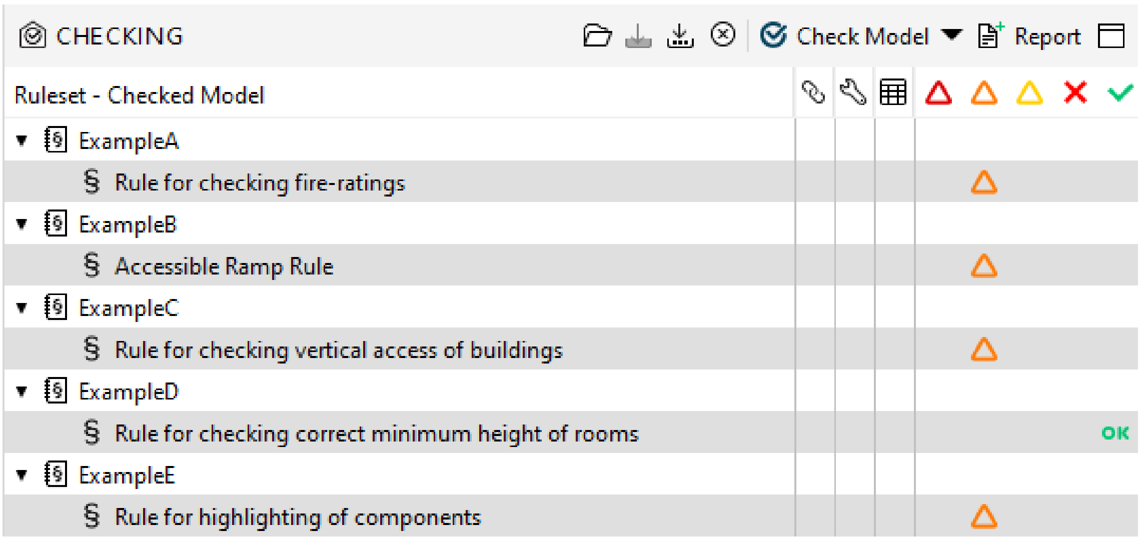


Figure 31: Highlighted fire extinguisher in the upper floor of the model with semi-transparent surrounding walls

9.3. Revit Sample Building Golden Nugget


CHECKING Check Model Report

Ruleset - Checked Model	Result
▼ ExampleA	
§ Rule for checking fire-ratings	⚠
▼ ExampleB	
§ Accessible Ramp Rule	⚠
▼ ExampleC	
§ Rule for checking vertical access of buildings	⚠
▼ ExampleD	
§ Rule for checking correct minimum height of rooms	OK
▼ ExampleE	
§ Rule for highlighting of components	⚠

Figure 32: Checking result Revit model 2

The results for this check were similar to that of the other models before:

For example A building class GK5 was used. While the model has less than 6 floors when manually looking at the building, the number of building storeys in the model is again larger, leading to additional requirements regarding the material class of elements, which would not be necessary had the model been drawn differently. As there is currently no parameter to enter the number of above-ground floors manually in the rule, the correct values of GK5 with less than 6 floors cannot be checked in this model.

Similar to the other Revit sample model, the elements in the model are not all on the correct floor, leading to undesired results in the checks for vertical access within each floor. Additionally, some storeys are split into two floors in the model, with ramps or stairs in only one of them.

Regarding the steepness of ramps, there was one ramp in the model which could not be checked by the rule. No further description to the problem was given. Another ramp was compliant with the requirement.

For the other examples, the height of spaces checked in example D is compliant with the requirements in all cases and spaces seem to be correctly modelled regarding their dimensions and locations. Six fire extinguishers were added to the model and displayed as seen in Figure 33.

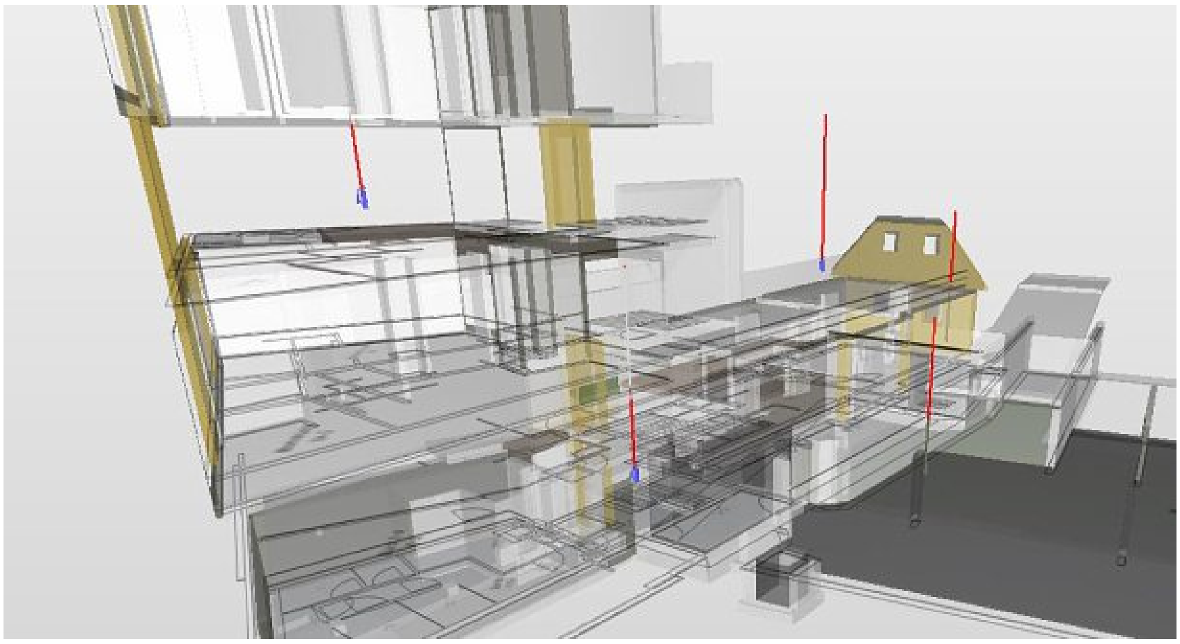
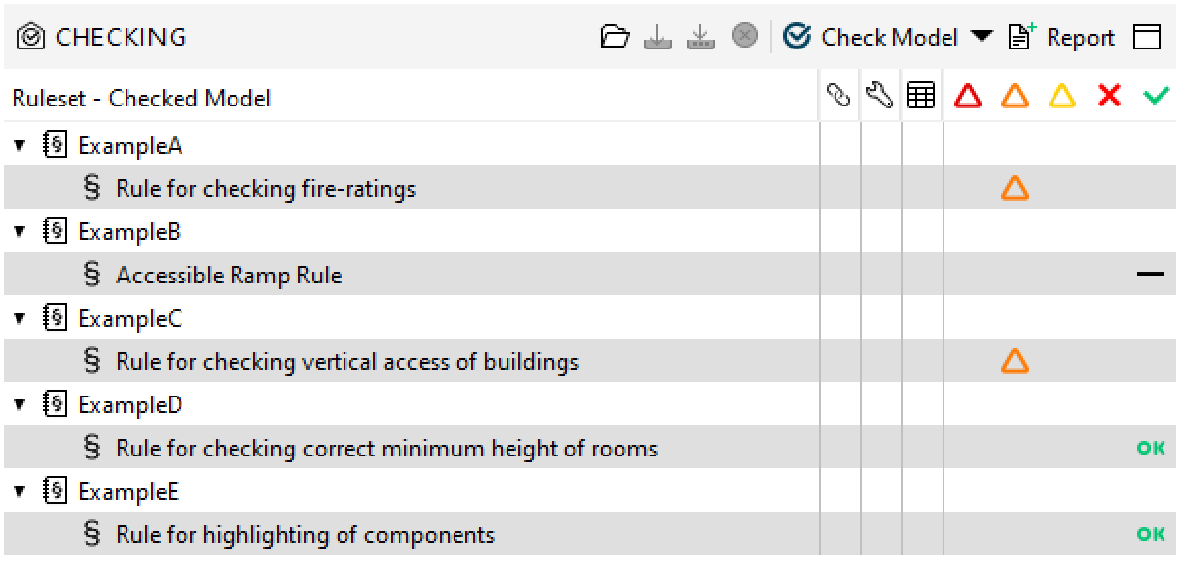


Figure 33: Multiple fire extinguishers highlighted throughout a model

9.4. Duplex Apartment Building


CHECKING — Check Model — Report

Ruleset - Checked Model								
ExampleA								
Rule for checking fire-ratings					△			
ExampleB								
Accessible Ramp Rule								—
ExampleC								
Rule for checking vertical access of buildings					△			
ExampleD								
Rule for checking correct minimum height of rooms								OK
ExampleE								
Rule for highlighting of components								OK

Figure 34: Checking results Duplex building

Regarding the results of the checks, there are no ramps and fire extinguishers in the building, so checking steepness of ramps and distribution of fire extinguishers does not yield any results. The top floor has been correctly identified, but as there is no information regarding fire resistance in the element properties, therefore all further checks regarding required values lead to non-compliance. For vertical access checking, the storey containing the foundation elements as well as the storey containing the roof are not accessible, stairs from ground floor to first floor have been identified and no vertical access within a floor is necessary.

10. Discussion

When looking at these steps in their consecutive order, the benefits and drawbacks of the chosen process become visible, most importantly the interaction and dependencies between formalization and implementation. Formalization and implementation are often discussed separately but benefit greatly from each other. While an implementation without prior formalization is often unstructured and relies greatly on testing to improve the quality of the implementation, a formalization can give an underlying base with all the important parameters to consider. However, a strict separation between the two processes is not effective as a formalization may be correct in the way it is formalized but may lack implicit knowledge about important relations between parameters. For a guideline or regulation, as it is written is not necessarily the same as how it is applied in practice.

While the formulas are named in a way that aims at making it easily understandable what they try to describe, statements like “isConnectedTo”, “hasFloorAbove” and so on are meaningless as long as it is not exactly specified what they are expressing. On the other hand, the formalization allows to abstract away the details of the rule while still being able to build the basic decision process. This is especially important considering that building codes, as already mentioned, are sometimes even intentionally vague, and therefore relying on the specific implementations alone would result in little reusability of rules.

Through the implementation of rules and their validation using sample models it has become clear that the model quality is of utmost importance for any compliance check. Similar to the selection of colours and shadings, which are to be used for 2D submission plans, strict modeling rules and standards for BIM models are necessary to gain enough quality for automated checks.

Implementations of the same rule can be different from each other and may yield different results, depending on how exactly a translation into executable code is performed. One problem with the implementation of example C was how to handle the connection of building storeys and elements for accessibility like ramps or stairs. As this was not possible through simply looking up the value of a specific parameter, other solutions had to be found. In the end, a requirement was enforced, to define a stair leading from a floor to the floor directly above as an element of the lower floor. Another way to solve this is to use the geometric positon of the stair in the model to determine which floors it connects. Depending on personal preference and background, this rule would have been implemented differently by someone else, also leading to different results in the tests owed to the way the buildings were modelled. It is therefore imperative that standards and best-practices for modelling building elements and their relations are established and followed. Also, even implicit knowledge should be communicated and formalized in a way that makes the inner workings of such an implementation clear to the users who may otherwise fail to understand why a model can fulfil all the requirements in their eyes, but still fail to pass an automated compliance check.

11. Conclusion

Research in the direction of automated code compliance checking and the formalization of pieces of legislations to make them computer-readable has been conducted for a long time. The use of digital tools as well as the utilization of BIM models in the building sector has risen considerably all over the world in recent years, yielding new tools and methods. However, many of the underlying problems still remain: Building legislation is often written with a certain room for interpretation on purpose, which makes it difficult to extract distinct values for automated compliance checking. BIM models vary greatly in content and quality, so information needed for checking a model for code compliance may not be available in the model. Additionally, exchanging models between different software frameworks as well as between closed-BIM and open-BIM solutions requires a lot of expertise as well as additional work for the involved professionals.

With the possibility to implement custom rules in Solibri instead of having to rely on pre-defined ones, the amount of checkable content from building legislations, including the OIB guidelines, rises. However, in most regulatory documents there are still parts that remain not suited for rule-based checking. These cases still require manual checks, but the checking can at least be sped up by automatically extracting additional information from the model and presenting it to the user in an easily comprehensible way. Such an “assisted checking” process might be a good solution for increasing automation and reducing time required for users, while still leaving the final decision regarding compliance in the hands of humans and not machines.

For Solibri, the implementation of an API is a great step in the direction of improving the software as it gives users the tools to implement changes themselves to meet their specific needs. Allowing the creation of not just new rules, but whole add-ons for the software might enable users to one day not only check the models for compliance, but edit them in a way that leads to compliance.

This work was focussed on compliance checks for BIM models at the end of the planning phase. While such checks can be performed already during the design phase, the need to export the models as IFC from proprietary software solutions acts as a bottleneck in the given approach. As these exports can take a lot of time for some large models, it is not feasible to export a model probably multiple times each day when changes are made. Staying in closed BIM, there are already solutions underway for proprietary software frameworks, such as model checkers for Revit (Autodesk) and ArchiCAD (GRAPHISOFT). This also includes projects using artificial intelligence for compliance checking during the design process, such as UpCodes (2020).
In open BIM, further research in the direction of interchangeability of models between different software frameworks and the usability of such systems is necessary, along with the improvement of standards in such directions.

The focus of such solutions is also an important aspect. One the one hand artificial intelligence can be used for solving specific and complex problems that cannot be matched by simple rule based checking (like determining the topmost floor). One the other hand broadening the application of such solutions by trying to integrate a BIM workflow on mobile platforms and not just desktop computers might play an important role in the future.

Concerning formalization efforts for building legislation, open software tools for the formalization of building codes may be helpful to support smaller projects of formalizations which larger government initiatives can later build upon.

Regarding Austrian solutions in the direction of compliance checking, the city of Vienna launched the project BRISE (Digitales Wien), a research and development project which is currently being worked on by multiple institutions, to further digitalization in the building sector.

List of Figures

References

Ashley, K. D. (2017) ‘Machine Learning with Legal Texts’, in Ashley, K. D. (ed) *Artificial Intelligence and Legal Analytics,* Cambridge, Cambridge University Press, pp. 234–258.

Autodesk *Autodesk Model Checker for Revit* [Online]. Available at https://www.biminteroperabilitytools.com/modelchecker.php (Accessed 20 October 2020).

Bach, N. X., Le Minh, N., Oanh, T. T. and Shimazu, A. (2013) ‘A Two-Phase Framework for Learning Logical Structures of Paragraphs in Legal Articles’, *ACM Transactions on Asian Language Information Processing*, vol. 12, no. 1, pp. 1–32.

Baueinreichung - Mein Wien [Online]. Available at https://mein.wien.gv.at/Meine-Amtswege/Baueinreichung/ (Accessed 23 July 2019).

Beach, T. H. and Rezgui, Y. (2018) ‘Semantic encoding of construction regulations’, *Proceedings of the 6th Linked Data in Architecture and Construction Workshop (LDAC 2018).* London.

Bellos, C. V., Petroutsatou, K. and Anthopoulos, L. (2015) ‘Electronic Building Permission System: The Case of Greece’, *Procedia Engineering*, vol. 123, pp. 50–58.

Borrmann, A., König, M., Koch, C. and Beetz, J. (2018) ‘Building Information Modeling: Why? What? How?’, in Borrmann, A., König, M., Koch, C. and Beetz, J. (eds) *Building Information Modeling,* Cham, Springer International Publishing, pp. 1–24.

buildingSMART (2020) *IFC Formats* [Online]. Available at https://technical.buildingsmart.org/standards/ifc/ifc-formats/ (Accessed 16 March 2020).

buildingSMART - The Home of BIM (2019) [Online]. Available at https://www.buildingsmart.org/ (Accessed 23 July 2019).

Bus, N., Roxin, A., Picinbono, G. and Fahad, M. (2018) ‘Towards French Smart Building Code: Compliance Checking Based on Semantic Rules’, *Proceedings of the 6th Linked Data in Architecture and Construction Workshop (LDAC 2018).* London, pp. 6–15.

Choi, J. and Kim, I. (2017) ‘A Methodology of Building Code Checking System for Building Permission based on openBIM’, *Proceedings of the 34th International Symposium on Automation and Robotics in Construction (ISARC).* Taipei, Taiwan, 6/28/2017 - 7/1/2017, Tribun EU, s.r.o., Brno.

CORENET Building Information Modeling (BIM) e-Submission [Online]. Available at https://www.corenet.gov.sg/general/building-information-modeling-(bim)-e-submission.aspx (Accessed 2 April 2020).

Digitales Wien *BRISE-Vienna - Digitales Wien* [Online]. Available at https://digitales.wien.gv.at/site/projekt/brisevienna/ (Accessed 20 October 2020).

Digitales Wien - Wien startet Digitale Baueinreichung [Online]. Available at https://digitales.wien.gv.at/site/wien-startet-digitale-baueinreichung/ (Accessed 23 July 2019).

Dimyadi, J. and Amor, R. (2013) ‘Automated Building Code Compliance Checking - Where is it at?’, in *Proceedings of the 19th International CIB World Building Congress*.

Doing Business 2019: Reports 2004-2019 [Online]. Available at https://www.doingbusiness.org/en/reports/global-reports/doing-business-2019 (Accessed 24 July 2019).

Eastman, C. M., Lee, J.-m., Jeong, Y.-s. and Lee, J.-k. (2009) ‘Automatic rule-based checking of building designs’, *Automation in Construction*, vol. 18, no. 8, pp. 1011–1033 [Online]. DOI: 10.1016/j.autcon.2009.07.002.

Eichler, C. C. (2019) *Standard-AuftraggeberInformationsanforderungen für Hoch- und Tiefbau* [Online]. Available at https://cloud.buildingsmart.co.at/EmBt7dcJWnt8sCr.

European Commission (2020) *eGovernment Benchmark 2020: eGovernment that works for the people - Shaping Europe’s digital future - European Commission* [Online]. Available at https://ec.europa.eu/digital-single-market/en/news/egovernment-benchmark-2020-egovernment-works-people (Accessed 16 October 2020.743Z).

Fiedler, J. N. (2015) *Modernisierungszenarien des Baubewilligungsverfahrens unter Berücksichtigung neuer technologischer Hilfsmittel,* Technische Universität Wien [Online]. Available at https://resolver.obvsg.at/urn:nbn:at:at-ubtuw:1-84205.

Foo Sing, T. and Zhong, Q. (2001) ‘Construction and Real Estate NETwork (CORENET)’, *Facilities*, vol. 19, 11/12, pp. 419–428.

Goger, G., Piskernik, M. and Urban, H. (2017) *Studie: Potenziale der Digitalisierung im Bauwesen: Empfehlungen für zukünftige Forschung und Innovationen*.

Goh, B.-H. (2008) ‘E-Government for Construction: The Case of Singapore’s CORENET Project’, in Xu, L. D., Chaudhry, S. S. and Tjoa, A. M. (eds) *Research and Practical Issues of Enterprise Information Systems II Volume 1: IFIP TC 8 WG 8.9 International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2007) October 14-16, 2007, Beijing, China,* Boston, MA, International Federation for Information Processing, pp. 327–336.

GRAPHISOFT *Check your model on the go* [Online]. Available at https://graphisoft.com/try-archicad/check-your-model-on-the-go (Accessed 20 October 2020).

Greenwood, D., Lockley, S., Malsane, S. and Matthews, J. (2010) ‘Automated compliance checking using building information models’, *COBRA 2010 - Construction, Building and Real Estate Research Conference of the Royal Institution of Chartered Surveyors*.

Hjelseth, E. (2015) ‘Public BIM-based model checking solutions: lessons learned from Singapore and Norway’, *Building Information Modelling (BIM) in Design, Construction and Operations.* Bristol, UK, 9/9/2015 - 9/11/2015, WIT PressSouthampton, UK, pp. 421–436.

Hjelseth, E. and Nisbet, N. (2010) ‘Exploring Semantic Based Model Checking’, in *CIB W78 2010 - Applications of IT in the AEC Industry (ISSN: 2706-6568)*.

Hjelseth, E. and Nisbet, N. N. (2011) *CAPTURING NORMATIVE CONSTRAINTS BY USE OF THE SEMANTIC MARK-UP RASE METHODOLOGY*.

Holte Consulting (2014) *ByggNett Status Survey* [Online]. Available at https://dibk.no/globalassets/byggnett/byggnett_rapporter/byggnett-status-survey.pdf.

Hudeczek, D. (2017) *Formalisierung von Normen mithilfe vonAuszeich-nungssprachen für die automatisierte Konformitäts-überprüfung*, Masterthesis, Technische Universität München.

IfcWiki (2019) [Online]. Available at http://www.ifcwiki.org/index.php?title=IFC_Wiki (Accessed 23 July 2019).

Industry Foundation Classes (IFC) - buildingSMART Technical (2019) [Online]. Available at https://technical.buildingsmart.org/standards/ifc/ (Accessed 9 October 2019).

Kim, H., Lee, J.-k., Shin, J. and Choi, J. (2019) 'Visual language approach to representing KBimCode-based Korea building code sentences for automated rule checking', *Journal of Computational Design and Engineering*, vol. 6, no. 2, pp. 143–148.

Kim, H., Lee, J.-k., Shin, J. and Kim, J. (2017) 'Visual Language-based approach for the Definition of Building Permit Related Rules', *Proceedings of the 34th International Symposium on Automation and Robotics in Construction (ISARC).* Taipei, Taiwan, 28.06.2017 - 01.07.2017, Tribun EU, s.r.o., Brno.

Kwok, J. T., Zhou, Z.-H. and Xu, L. (2015) 'Machine Learning', in Kacprzyk, J. and Pedrycz, W. (eds) *Springer Handbook of Computational Intelligence,* Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 495–522.

Lee, H., Lee, J.-k., Park, S. and Kim, I. (2016) 'Translating building legislation into a computer-executable format for evaluating building permit requirements', *Automation in Construction*, vol. 71, pp. 49–61.

Lee, J.-k. (2011) *Building environment rule and analysis (BERA) language and its application for evaluating building circulation and spatial program*, Dissertation, Georgia Institute of Technology [Online]. Available at http://hdl.handle.net/1853/39482.

Lee, J.-k., Eastman, C. M. and Lee, Y. C. (2015) 'Implementation of a BIM Domain-specific Language for the Building Environment Rule and Analysis', *Journal of Intelligent & Robotic Systems*, vol. 79, 3-4, pp. 507–522.

Liu, L. and Özsu, M. T., eds. (2009) *Encyclopedia of database systems*, New York, NY, Springer.

Louridas, P. and Ebert, C. (2016) 'Machine Learning', *IEEE Software*, vol. 33, no. 5, pp. 110–115.

Lu, P., Chen, S. and Zheng, Y. (2012) 'Artificial Intelligence in Civil Engineering', *Mathematical Problems in Engineering*, vol. 2012, no. 6, pp. 1–22.

Macit İlal, S. and Günaydın, H. M. (2017) 'Computer representation of building codes for automated compliance checking', *Automation in Construction*, vol. 82, pp. 43–58.

Martins, J. P. and Monteiro, A. (2013) 'LicA: A BIM based automated code-checking application for water distribution systems', *Automation in Construction*, vol. 29, pp. 12–23.

Nay, J. (2018) 'Natural Language Processing and Machine Learning for Law and Policy Texts', *SSRN Electronic Journal*.

ÖBFV (2017) *TRVB 124 /17 (F) "Erste und erweiterte Löschhilfe"* [Online]. Available at https://www.bundesfeuerwehrverband.at/produkt/trvb-124-17-f-erste-und-erweiterte-loeschhilfe/ (Accessed 2 July 2020).

Österreichisches Institut für Bautechnik (2019a): *OIB-Richtlinie 1: Mechanische Festigkeit und Standsicherheit* [Online]. Available at https://www.oib.or.at/de/oib-richtlinien/richtlinien/2019/oib-richtlinie-1.

Österreichisches Institut für Bautechnik (2019b): *OIB-Richtlinie 2.1: Brandschutz bei Betriebsbauten* [Online]. Available at https://www.oib.or.at/de/oib-richtlinien/richtlinien/2019/oib-richtlinie-21.

Österreichisches Institut für Bautechnik (2019c): *OIB-Richtlinie 2.2: Brandschutz bei Garagen, überdachten Stellplätzen und Parkdecks* [Online]. Available at https://www.oib.or.at/de/oib-richtlinien/richtlinien/2019/oib-richtlinie-22.

Österreichisches Institut für Bautechnik (2019d): *OIB-Richtlinie 2.3: Brandschutz bei Gebäuden mit einem Fluchtniveau von mehr als 22 m* [Online]. Available at https://www.oib.or.at/de/oib-richtlinien/richtlinien/2019/oib-richtlinie-23.

Österreichisches Institut für Bautechnik (2019e): *OIB-Richtlinie 2: Brandschutz* [Online]. Available at https://www.oib.or.at/de/oib-richtlinien/richtlinien/2019/oib-richtlinie-2.

Österreichisches Institut für Bautechnik (2019f): *OIB-Richtlinie 3: Hygiene, Gesundheit und Umweltschutz* [Online]. Available at https://www.oib.or.at/de/oib-richtlinien/richtlinien/2019/oib-richtlinie-3.

Österreichisches Institut für Bautechnik (2019g): *OIB-Richtlinie 4: Nutzungssicherheit und Barrierefreiheit* [Online]. Available at https://www.oib.or.at/de/oib-richtlinien/richtlinien/2019/oib-richtlinie-4.

Österreichisches Institut für Bautechnik (2019h): *OIB-Richtlinie 5: Schallschutz* [Online]. Available at https://www.oib.or.at/de/oib-richtlinien/richtlinien/2019/oib-richtlinie-5.

Österreichisches Institut für Bautechnik (2019i): *OIB-Richtlinie 6: Energieeinsparung und Wärmeschutz* [Online]. Available at https://www.oib.or.at/de/oib-richtlinien/richtlinien/2019/oib-richtlinie-6.

Park, H., Gu, H., Lee, W., Kim, I. and Choo, S. (2019) 'A Development of KBIMS-based Building Design Quality Evaluation and Performance Review Interface', in Haeusler, M. H., Schnabel, M. A. and Fukuda, T. (eds) *Intelligent & Informed: Proceedings of the 24th International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2019) - Volume 1,* The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), pp. 747–756.

Preidel, C. (2014) *Code Compliance Checking: Entwicklung einer Methode zur automatisierten Konformitätsüberprüfung auf Basis einer graphischen Sprache und Building Information Modeling*, Masterarbeit, Technische Universität München.

Preidel, C. and Borrmann, A. (2015) 'Automated Code Compliance Checking Based on a Visual Language and Building Information Modeling', *Proceedings of the 32nd International Symposium on Automation and Robotics in Construction and Mining (ISARC 2015).* Oulu, Finland, 15.06.2015 - 18.06.2015, International Association for Automation and Robotics in Construction (IAARC).

Preidel, C. and Borrmann, A. (2018) 'BIM-Based Code Compliance Checking', in Borrmann, A., König, M., Koch, C. and Beetz, J. (eds) *Building Information Modeling,* Cham, Springer International Publishing, pp. 367–381.

Solibri Inc. *207 Accessibility Ramp Rule* [Online]. Available at https://solibri-community.force.com/know2/s/article/UUID-ad50d351-94b8-2f18-f856-0d698237f60f (Accessed 3 August 2020).

Solibri Inc. (2020) *Solibri | BIM software for architects, engineers and construction…* [Online]. Available at https://www.solibri.com/ (Accessed 23 March 2020).

Solihin, W. and Eastman, C. M. (2015) 'Classification of rules for automated BIM rule checking development', *Automation in Construction*, vol. 53, pp. 69–82.

Song, J., Kim, J. and Lee, J.-k. (2018) 'NLP and Deep Learning-based Analysis of Building Regulations to Support Automated Rule Checking System', *Proceedings of the 35th International Symposium on Automation and Robotics in Construction (ISARC).* Taipei, Taiwan, 6/28/2017 - 7/1/2017, International Association for Automation and Robotics in Construction (IAARC).

Sonntag, M. and Wimmer, M. (2003) 'Legal Aspects of One-Stop Government: The Case of Applying for a Building Permission', in Goos, G., Hartmanis, J., van Leeuwen, J. and Traunmüller, R. (eds) *Electronic Government,* Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 385–392.

Thompson, J. (n.d.) 'Singapore CORENET Case Study' [Online]. Available at https://buildingsmart.no/sites/buildingsmart.no/files/Case_studie_Singapore.pdf (Accessed 17 October 2020).

Tulke, J. (2018) 'BIM-Based Design Coordination', in Borrmann, A., König, M., Koch, C. and Beetz, J. (eds) *Building Information Modeling,* Cham, Springer International Publishing, pp. 317–327.

UpCodes *Searchable platform for building codes* [Online]. Available at https://up.codes/ (Accessed 20 October 2020).

Zhang, J. (2017) 'A logic-based representation and tree-based visualization method for building regulatory requirements', *Visualization in Engineering*, vol. 5, no. 1.

Appendix

Attached to this paper is a CD containing the following data:

- Digital version of the thesis
- Source code of the implemented rules