

---

# Use of “Reasoner” for automated building code compliance assessment

---

Eilif Hjelseth, [eilif.hjelseth@ntnu.no](mailto:eilif.hjelseth@ntnu.no)

Norwegian University of Science and Technology, Norway

Beidi Li, [beidi.li@ece.au.dk](mailto:beidi.li@ece.au.dk)

Aarhus University, Denmark

## Abstract

Despite significant research effort, there is still a lack of complete solutions for automated compliance checking of regulatory, normative, and contractual requirements pertaining to buildings. This is reflected in numerous studies and prototype implementations targeting exclusively prescriptive requirements with numeric, unambiguous data, rather than descriptive, vague information in descriptive, performance-based building codes. We argue that a formal, systematic, logic-based approach to tackle qualitative performance goals and design requirements is not only valuable, but also necessary for enabling traceable, transparent code checking applications, and reducing laborious, error-prone manual processing. To this end we investigate a dedicated spatial reasoner, ASP4BIM, specifically designed for tackling implicit, uncertain, and fallible human knowledge about building-specific rules and constraints, such as objects' functions and structures, and occupants' experiences and behavior. Through empirical evaluations with large-scale buildings and real-world code provisions, we show that our approach increases the number of requirements eligible for digital processing.

Keywords: Automated Building Code Compliance Assessment, BIM-based Model Checking, Building Information Modelling, Performance-based Building Code, Declarative Spatial Reasoning, Answer Set Programming, ASP4BIM

## 1 Challenges in implementing complete BIM-based model checking solutions

BIM-based Model Checking (BMC) is an excellent demonstration of atomization of numerous, disparate requirements and sporadic, competing implementation approaches. Despite a significant increase in BMC solutions and code checking applications, there is still a lack of systematic consideration of descriptive, performance-based building codes. As a result, many software systems tend to claim prematurely full-scale automation while they do not support the interpretation of qualitative codes.

Study by (Dimyadi et al., 2016) shows the limited number of relevant rulesets for model validation is a major bottleneck in enabling automated code compliance assessment. From a practitioner's perspective, rules-of-thumb are indispensable for assessing a building's compliance but are rarely *codified*. An automated BMC tool should also be able to detect and correct minor modelling inconsistencies and discrepancies that are hard to discover by humans, but this is not possible when a software system presumes a building model is perfectly accurate and precise. The fundamental issue here is the seemingly incompatibility between human commonsense that is implicit and fallible, and formal, procedural algorithms that are unable to deal with fuzziness and uncertainty. As a result, state-of-the art BMC software systems mainly tackle simple geometry problems such as clash detection with Boolean logic formulas. There is also a considerable focus on de facto reasoning and computational efficiency, so that the underlying algorithm is often riddled with ad-hoc assumptions, magic numbers, and numerical tricks to speed up runtime.

There are two types of rules: prescriptive (such as the minimum clearance for wheelchair users is 1500 mm) and descriptive (such as well illuminated, close to parking, large enough, etc.). Most prescriptive requirements can be directly implemented in BMC solutions. If the metrics or dimensions elicited can be mapped to a property or attribute in BIM, the requirement can be translated to predicate logic propositions based on Boolean Algebra.

In contrast, performance-based requirements describe abstract design goals, rather than concrete design solutions. They evoke human-centric concepts such as “easy to understand” and “line of sight” (Li, Dimyadi, et al., 2020) that are inherently ambiguous and elusive. This paper aims to reconcile the gap between qualitative performance goals and individual compliance paths with a declarative spatial “reasoner” based on non-monotonic logic.

A study by (Hjelseth, 2012) shows that 43% of the ISO 21542:2011 standard cannot be directly converted into computable rules. This study further proposes methods TX3 and TIO for identifying patterns in performance goals, which increases the number of requirements eligible for digital processing up to 83%. However, the residual part still requires extensive expert knowledge and specific competencies to enable full-scale building code checking applications.

The research question is therefore: **RQ** - *Can we use a declarative spatial reasoner to improve the implementation of performance-based codes in BIM-based Model Checking software?*

## 2 Methods for interpreting requirements into computable rules

### 2.1 Challenges in assessing qualitative building codes

Consider the following building code from New Zealand: “Accessible route shall give direct access to the principal entrance to the building where practical”. Terms “direct”, “principal”, and “practical” are ambiguous. A building can be missing some key information (such as the primary function or activity of a space) to derive “accessible routes”. Knowledge about a building’s structure and usage can be undocumented but are necessary to determine “direct access”. These aspects are essential for enabling complete code checking applications, and crucial for providing clear arguments and explanations for a particular checking result (*Pass, Fail, or Undetermined*).

In Figure 1, we illustrate the role of “Reasoner” to integrate various processes and knowledge related to automated building code compliance assessment. Specifically, the reasoner negotiates the uncertainty and fuzziness of real-world data, information, and human practices, by providing a uniform and systematic approach to encode vague and elusive natural language statements, inaccurate and incomplete building models, and implicit and fallible human commonsense about objects, structures, functions, and behavior.

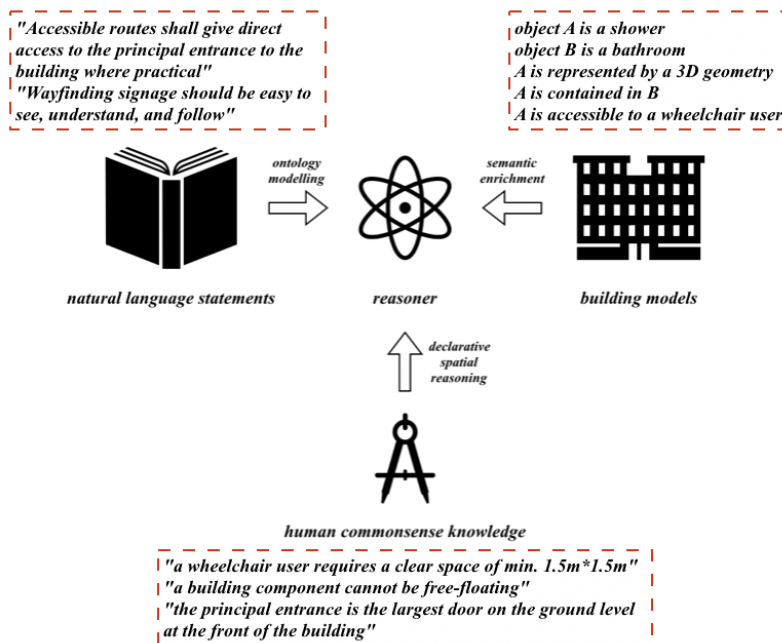


Figure 1. The role of “Reasoner” to integrate different code checking processes and knowledge.

Several practical difficulties arise when a building model undergoes increasing levels of detailing, disparate multi-disciplinary modifications, and continuous revisions and updates in light of as-built information. This makes deductive-only query languages unsuitable as they presume deterministic and complete information and thus, fail to capture the *dynamism* and *uncertainty* intrinsic to building models. As a result, adding or retracting information can make the program increasingly brittle and obscure, limiting the overall modifiability and extensibility.

In the following, we formulate three desired properties in checking intricate, qualitative building regulations against large, complex buildings:

**P1** – Executing formalized codes in a computational optimized way without altering code semantics.

**P2** – Introducing new information and knowledge without changing the control flow of the underlying checking approach

**P3** – Verifying and validating checking results with transparent rulesets, explicated assumptions, and motivating literatures that document and explain choices of interpretation

A common approach for addressing **P1** consists of incorporating execution concerns into rule formalization. Code experts supplement natural language provisions with additional assumptions to speed up runtime, e.g. a patient’s room is visible from a nurse station bounded by a common wall (Solihin & Eastman, 2015). Programmers encode knowledge about best practices in the form of magic numbers (“*close to parking*” is translated into 20m). Building experts augment models based on ad-hoc rules and complex schema mapping, e.g. an IfcWall element touching a concrete IfcSlab element from above implies structural support (Bloch et al., 2019). Such an approach, however, creates deeply entangled rule definition and rule execution, which tend to prioritize easy and practical software implementation over preserving code semantics.

## 2.2 Enhancing a logic-based reasoning engine with space

We propose to address the limitations of current approaches by *pushing down execution concerns to a dedicated logic-based reasoner* that natively understands spatial axioms and automatically applies spatial optimizations. We argue that such a reasoner generates solutions that are, by construct, logically sound and spatially consistent, and is able to deal with numerous code provisions and large-scale building models in an efficient and scalable way<sup>1</sup>. In this way, regulation experts can focus exclusively on code semantics without worrying about whether the formalization is easy and practical to implement.

To this end, we elicit a knowledge-driven AI subfield, Declarative Problem Solving (DPS), where a user is only concerned with describing a problem, rather than how the solver finds solutions. Historically, DPS is inspired by various forms of commonsense reasoning, particularly explanation where a person fills in the gap in a way that is justified by background knowledge (*a hinged door can open either to the left or right*) and consistent with observations (*two doors do not clash during operation*). In practice, DSP applications extend Boolean Satisfiability (B-SAT) problems with domain-specific rules and constraints, e.g., a space cannot be *above* and *under* a slab at the same time.

Figure 2 illustrates a typical DPS software where the solver is enhanced with background knowledge and human commonsense, so that a user can express their high-level queries and problems in a natural language way that the solver can natively interpret and execute. In the context of assessing building compliance, code engineers formalize natural language statements as logic rules and constraints (*bathrooms shall have privacy*). A building model is translated into a set of facts, and we fill in incomplete and missing information by default assumptions and what-if scenarios. We use inference rules to derive implicit model properties, supported by expert knowledge about affordance and behavior (*an opaque screen of 1700mm provides visual occlusion*) and human understanding about object, space, relationships, and hierarchy (*the entrance of a building is contained in the ground level*).

---

<sup>1</sup> Our reasoner of choice, Answer Set Programming (ASP), is specifically designed for commonsense reasoning and NP-hard problems using modern Boolean SAT solving techniques such as DPLL (Calimeri et al., 2019; Gebser et al., 2016).

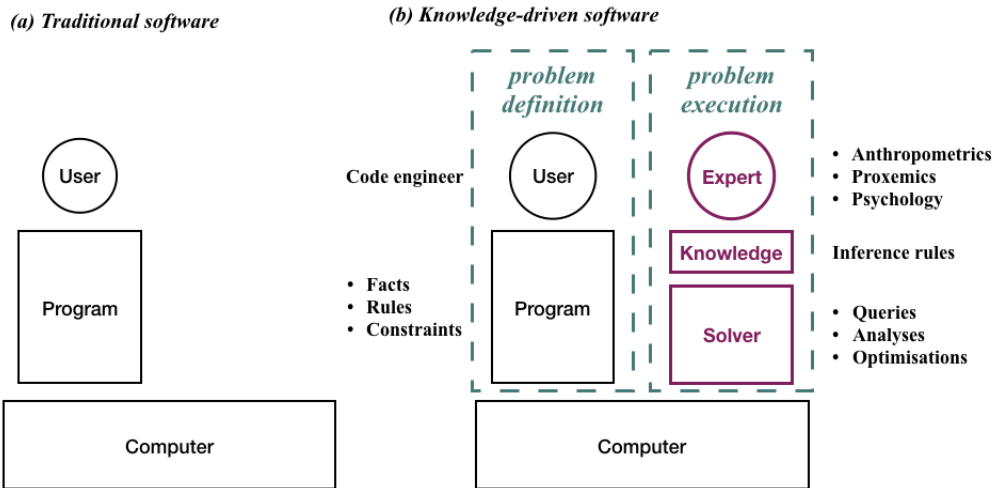


Figure 2. System architecture of knowledge-driven DPS software.

### 3 Ontology mapping between regulation terms and building models

We argue that a reasoner bridges the gap between high-level performance goals such as navigability and individual compliance paths such as well-placed signage systems. The reasoner documents the mapping between qualitative concepts and low-level discrete metrics, by explicating a code provision’s context and scope. The reasoner also maintains the provenance of *disambiguating* alternative definitions so that the parametric models used for quantifying concepts such as visibility can be traced back to reference documents, e.g., the visual field of a normal adult spans 114° in the horizontal plane (Kondyli et al., 2017).

Figure 3 shows the role of a reasoner in separating rule definition from rule execution. Legal experts supplement building codes with implied knowledge about their purposes and functions, e.g., to ensure bathroom privacy or structural validity. This knowledge is motivated by reference documents in forms of metadata or hyperlinks, e.g., the movement pattern of a building occupant is computed based on supporting *evidence* from user studies, experiments, or behavioral theories. In this way, rule interpretation is *traceable* and *configurable*. Building experts augment building models with their usage and behavior, e.g., routine activities and normal operations. This knowledge can be asserted or hypothesized based on prior experiences in the form of inference rules, e.g., a room is assigned property *HandicapAccessible* if it has a minimum width of 1200mm. In this way, rule formalization is *transparent* and *modifiable*.

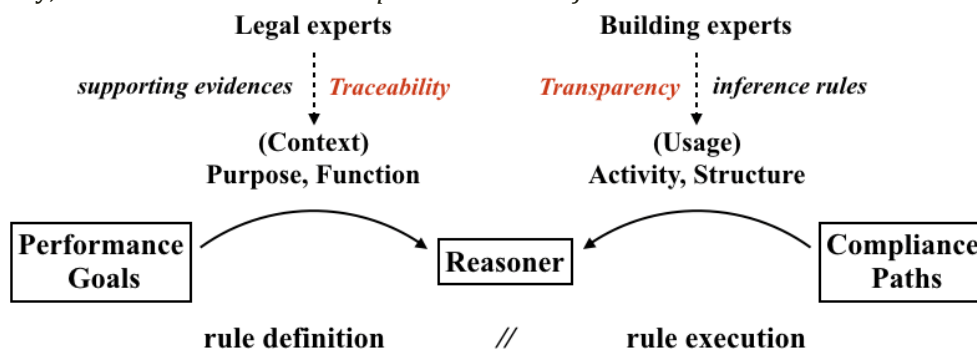


Figure 3. The role of reasoner to bridge performance goals and compliance paths.

#### 3.1 Explicating design intents in rule interpretation

Consider the following code provision from the Swedish National Building Regulation (Boverkets byggregler): “Ramps shall be able to be used by people with limited mobility. The maximum slope shall be no more than 1:12”. This code consists of a descriptive part eliciting a user-centric requirement of ramps (usable to people with limited mobility) and a prescriptive part dictating normative dimensions (1:12). However, such combination is inherently incongruous. We argue

that qualitative goals are explicated into quantitative metrics, but the underlying assumptions and scope of validity still remain tacit. For instance, the Norwegian Building Regulations (Byggteknisk Forskrift) requires a maximum slope of 1:20, but permits 1:12 for sections up to 3.0 m, which shows that compliance paths are far from unique and exhaustive.

In fact, the intention of this code falls within two major design concerns, *accessibility* and *safety*, to ensure that a ramp is not too steep for a wheelchair user to easily and safely roll up and down. Other factors influencing the ramp’s usability include a person’s weight and forces applied on the hand rims, and the ramp’s width and surface friction factors (Bennett et al., 2009). In contrast, a single value 1:12 fails to capture the multitudes and interdependencies among the physical and perceptual characteristics of ramps, but merely provides one *static* layout with little justification. As a result, a compliant design does not guarantee usability and comfort for all users in all building use scenarios, e.g., a hospital patient with walking aids descending a crowded ramp.

This example motivates the separation of *general* performance goals from *specific* compliance paths, to allow designers to focus on user-centric design requirement, rather than rigid, obscure dimensions and properties.

### 3.2 Framework overview

The above examples provide the basis for a uniform, coherent framework for interpreting performance-based codes, as design goals and intentions can be qualitatively interpreted using terminologies from human ergonomics and psychology, and later quantitatively computed using discrete metrics and parameters from research *evidence*. In this way, we enable a modular, evidence-based interpretation of performance objectives, so that code semantics sustain the minimum amount of alteration, but every deliberate choice of interpretation is documented and can be traced back to motivations and rationales.

Figure 4 illustrates our proposed mapping between building domains and regulation domains with explicated types of *occupants*, *actions*, and *situations*. From left to right, spatial experiences of a person performing a certain task in a particular situation are qualitatively described using parametric design principles from reference studies, e.g., an observer behind a crowd w.r.t a visual cue will have reduced visibility (Li, Fitzgerald, et al., 2020b). From right to left, performance criteria are delineated based on specific user profiles, e.g., the accessibility of a walking adult in a wayfinding scenario. We explicitly do not intend to be comprehensive, but aim to provide an extensible and configurable approach for content-based transformation of descriptive codes, e.g., a user can customize it with their choice of parametrization, and adapt it to specific use cases (patients with dementia, urban transportation, fire evacuation, etc.).

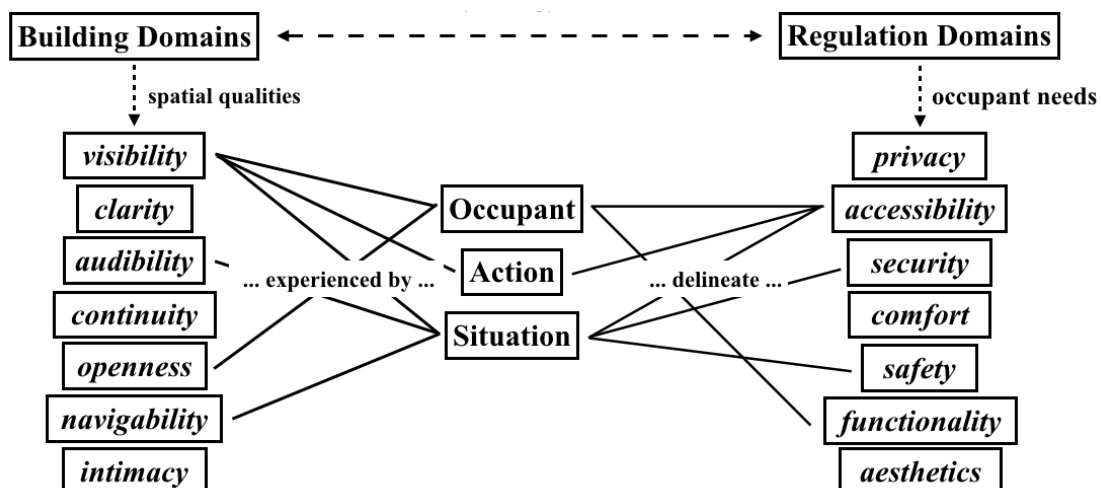


Figure 4. An ontological mapping between building domains and regulation domains

#### 4 ASP4BIM – a declarative spatial reasoner

ASP4BIM is our logic-based spatial query language for representing and reasoning about large, complex building models<sup>2</sup>. ASP4BIM handles quantitative spatial functions (*area, union*) and qualitative spatial relations (*size, topology*) in a reliable and efficient way with state-of-the-art geometry libraries. ASP4BIM is seamlessly integrated into our proposed workflow as a dedicated spatial reasoner for checking formalized rules and constraints against real-world building models as shown in Figure 5. ASP4BIM natively supports spatial ontologies (e.g., a window is embedded in its hosting wall) and building-specific rules and constraints (e.g., a window cannot be constructed before its hosting wall). In this way, every inference drawn from an ASP4BIM adheres to human commonsense about space, people, objects, and buildings.

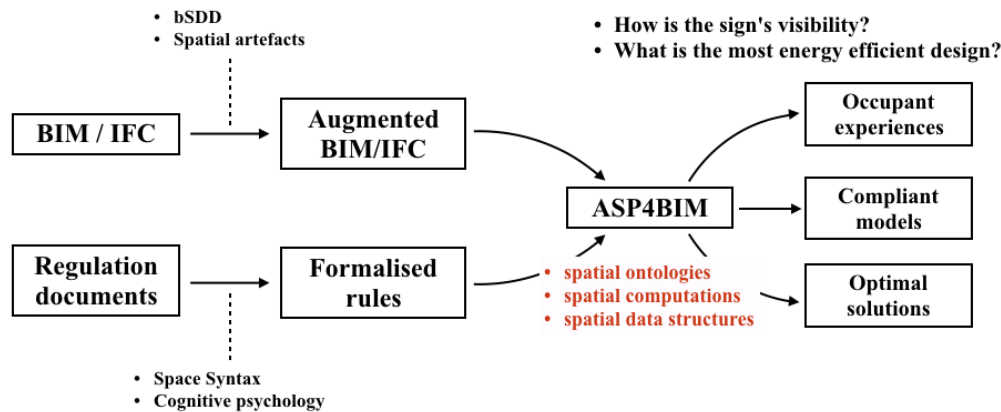


Figure 5. Our proposed workflow for checking qualitative building codes with ASP4BIM

ASP4BIM has its theoretical foundation in Answer Set Programming, a prominent DSP paradigm. ASP4BIM combines the expressiveness of ontology language and the verifiability of logic programming (Eiter, 2007), to support queries and analyses about occupant’s experiences and behavior in the built environment, e.g., “Does a person at entrance A have direct movement access to lecture hall B?” and “How is the visibility of sign S to an observer at P?”. ASP is specifically designed to handle combinatorial search, thus ASP4BIM is well suited for navigating a vast landscape of disparate, numerous performance-based codes, supporting evidence from research literature, and expert knowledge about occupants, actions, and situations.

ASP4BIM is distinct from other spatial query languages based on deductive databases such as GeoSPARQL (Battle & Kolas, 2012) and SHACL (Stolk & McGlenn, 2020) in that:

- it is a three-valued logic system suitable for commonsense reasoning tasks such as default reasoning (by default, assume P is True, unless we can prove P is False) and abductive reasoning (if hypothesis H contradicts observation O, then retract H)
- its declarative<sup>3</sup> character enables a clean, modifiable, and extensible rule encoding
- it is integrated with advanced search space pruning techniques<sup>4</sup> to navigate a large search space of requirements, conditions, scope, and criteria

Therefore, ASP4BIM is able to handle uncertainties and incompleteness in code requirements and building models, as knowledge can be continuously updated and revised without changing the program structure (P2). As rulesets can be easily modified, ASP4BIM can support various design applications such as construction safety control, operational management, value-based renovation, etc. with empirical presets or custom parameters.

ASP4BIM is designed to be a configurable and modular software system, that maintains the motivation and rationale behind rule interpretation in forms of reference documents (code definitions, guidebook, research findings, experimental data, etc.). In this way, compliance results

<sup>2</sup> ASP4BIM enhances previous declarative spatial reasoning systems CLP(QS) (Bhatt et al., 2011), ASPMT(QS) (Watega et al., 2017), and λProlog(QS) (Li et al., 2019) by providing non-monotonic reasoning capabilities about real-world buildings with numerous, complex semantic and geometric details.

<sup>3</sup> That is, the order in which ASP rules are defined does not matter.

<sup>4</sup> Such as back-jumping, lazy grounding (Bogaerts & Weinzierl, 2018), and conflict-driven constraint learning (Eiter et al., 2012).

can be explained and argued about based on supporting evidence, and can be verified by checking the correctness of explicated assumptions and hypotheses about code semantics and building data (P3).

#### 4.1 ASP semantics

An ASP4BIM program is composed of facts, rules, and constraints. In the context of reasoning about buildings, semantic and geometric information is extracted from a BIM and transformed into ASP facts, e.g., objects `d13` and `d57` are hinged doors, `d13` serves as a fire exit, and `d57` is contained in the ground level (Line 1-2). A (deductive) rule, denoted as  $P:-Q_1,\dots,Q_n$ , states that  $P$  must hold if  $Q_1,\dots,Q_n$  hold, such as a space is an accessible route if it has property `handicapAccessible` (Line 3). A choice rule, denoted as  $1\{P_1;\dots;P_m\}1:-Q$ , states that exactly one  $P_i$  such that  $1\leq i\leq m$  can be deduced if  $Q$  holds, e.g., a hinged door can either swing inwards or outwards (Line 4). A constraint, denoted as  $:-Q_1,\dots,Q_n$ , states that  $Q_1,\dots,Q_n$  cannot be jointly *True*, e.g., a door cannot open inwards if the door is a fire exit due to safety concerns (Line 5).

```

1   door(d13). door(d57). type(d13, hinged). type(d57, hinged).
2   fireExit(d13). containedIn(d57, groundLevel).
3   accessibleRoute(S) :- space(S), property(S, handicapAccessible).
4   1{swings (D, inwards); swings (D, outwards)}1 :- door(D), type(D, hinged).
5   :- swings(D, inwards), door(D), fire_exit(D).
```

Solving an ASP4BIM program consists of computing the logical implications of facts and rules that are consistent with constraints. Considering the above program, door `d13` must swing inwards due to the constraint in Line 4, so ASP4BIM generates two models, a first one where door `d57` swings inwards and a second one where door `d57` swings outwards.

If we add an additional constraint that ground level doors cannot swing outwards in order to avoid collision with pedestrians and vehicles (Line 6), ASP4BIM generates only one model where `d57` swings inwards. Furthermore, if a space `s2` has property `handicapAccessible` and is bounded by `d57` (Line 7), constraint in Line 8 is checked, and we derive that `s2` is an access route from Line 3. This triggers Line 8 where we state that a door opening into an access route cannot intrude its minimum clear width of 800mm (required by NZBC). The `@-`-prefixed function `minimumWidth` calculates the minimum width of `s2` in the presence of a fully-open `d57` and returns it as an integer-valued constant in Line 9. If the minimum width of `s2` is less than 800mm, Line 9 concludes a violation of code provision `d1-7-0-3`. Otherwise, the building is compliant by Line 10.

```

6   :- swing(D, outwards), door(D), containedIn(D, ground_level).
7   space(s2). property(s2, handicapAccessible). boundedBy(s2, d57).
8   @minimumWidth(S, D) :-
9     accessRoute(S), door(D), bounded_by(S, D), swings(inwards, D).
9   violation(d1-7-0-3) :-
10    accessRoute(S), @minimumWidth(S, D) < 800.
10  compliant :- not violation(_).
```

ASP4BIM is a three-valued logic system with  $P$ ,  $\neg P$ , and  $\text{not } P$  respectively denoting proposition  $P$ , its classical negation, and failure to prove  $P$ . In other words, ASP4BIM embraces the open-world assumption, and postulates that knowledge is not exact, certain, and complete, but can be revised, rectified, and retracted in light of new observations and evidence.

#### 4.2 Reasoning about building codes in ASP4BIM

We provide a modular, flexible encoding with ASP4BIM with independent formulations of four stages from a Knowledge Representation and Reasoning (KR) perspective:

**Representing buildings in ASP4BIM.** We parse a BIM or IFC model into a symbolic knowledge base that stores semantic data and an external spatial database that stores geometric data. The knowledge base encodes BIM objects, properties, and relationships as ASP facts, identified by the underlying mapping schema (IFC4, bSDD, IfcOwl, etc.). The spatial database maintains a dynamic link between object IDs and their geometric representations (bounding boxes, 3D meshes, 2D footprints, etc.) and is continuously invoked from symbolic ASP using predicate `repr/2` to update the geometry database with newly derived objects, in sync with ASP

solving. In the following snippet, a bathroom object gives rise to a visibility space. The new object is added to the spatial database and produces a fresh object ID that is used to map it to its geometric representation in Line 14.

```

11     Repr(Id, GeomId) :- product(Id), GeomId = @geom_id(Id).
12     defineNewObject(P, R) :-
        bathroomObject(B), P = @newId(B), R = @visibilitySpace(B).
13     visibilitySpace(P) :- defineNewObject(P, _).
14     repr(P, R) :- defineNewObject(P, R).

```

**Encoding performance goals in ASP4BIM.** One major challenge in formalizing normative texts as logical propositions is that deontic operators (obligation, permission, prohibition, and exception) can lead to redundant, inconsistent rulesets. ASP4BIM tackles these issues with its native three-value logic encodings. For instance, a requirement upon *Entity* is encoded as a deductive rule that fires when *Entity* satisfies *condition* and is not an exception of *r12* (Line 15). Then it suffices to state that an instance *e1* is an exception to prevent *r12* (Line 16) from firing. In this way, knowledge can be asserted or retracted by adding or removing this simple fact, without changing the program structure.

```

15     applies(Entity, r12) :-
        condition(Entity), not except(Entity, r12).
16     except(e1, r12).

```

**Maintaining supporting evidences in ASP4BIM.** Rationales behind a particular rule interpretation and evidence behind a specific approach to parametrize qualitative concepts are maintained in ASP4BIM as *modules*. Consider the isovist of an object, we use *visibility space* to denote the region from which the object is visible to a person, computed as the isovist clipped by the person’s visual range. This range can take different values depending on the type of occupants and their visual acuity, based on ophthalmology studies. In the following, we use prefixed statement `#include` to indicate a specific reference literature (Line 17), which “grounds” predicate `visualRange/1` with a specific value *D*, so that the visibility space is computed by intersecting the isovist with a sphere of radius *D*, centered at object *B* (Line 18).

```

17     #include
18     visibilitySpace(B, S) :-
        bathroomObject(B), isovist(B, P), visualRange(D), @clip(P, D).

```

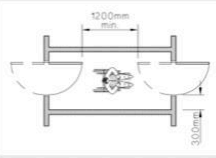
**Resolving spatial queries.** Querying-answering in ASP4BIM consists of deriving ASP answer sets (stable models). Each answer set is computed by assuming some hypotheses are True (such as a hinged door swings inwards) and checking if their implications lead to contradictions with other program rules (if the door is a fire exit and by definition must swing outwards). We have integrated ASP with spatial ontologies so that ASP4BIM predicates (e.g., *inside* and *overlap*) adhere to spatial axioms and calculi about form, orientation, topology, and mereology (Li, et al., 2020d). We have also enhanced ASP4BIM with (general) spatial data structures such as Q-tree and R-tree for reliable and efficient results.

**Operational performance of ASP4BIM on large BIMs.** Table 1 shows runtime statistics of ASP4BIM from our previous case studies of assessing building codes on real-world BIMs (Li et al., 2020a; Li et al., 2020c; Schultz et al., 2020). The codes demonstrate various degrees of descriptiveness. Our experiments showed that ASP4BIM is fast enough to run on large-scale BIMs and is compact<sup>5</sup> enough to be integrated into other applications. In all case studies ASP4BIM was able to generate hundreds of enhanced objects (such as visibility spaces) with clean, modifiable encodings, thus provides a modular, portable approach to building code checking by allowing numerous, competing interpretations of qualitative rules to run simultaneously and concurrently.

<sup>5</sup> ASP4BIM is available as a standalone Python module at <https://github.com/libeidihuhuhu/ASP4BIM>.



Table 1. Runtime performance of ASP4BIM in assessing code requirements.

Code provisions	Model statistics	Runtime	References
There shall be no direct line of sight between an access route or accessible route and a WC, ...	2-storey building in NZ 8304 BIM objects	1.6 seconds	NZBC-G1-6.1.1*
Where doors open into a lobby, the clear space between open doors shall comply with Figure 27.		8.4 seconds	NZBC-D1/AS1-7.0.1*
An unobstructed wall space of not less than 300 mm shall be provided at the side of the door adjacent the handle.		8.5 seconds	
The clear width of an accessible route shall be no less than 1200 mm.		0.3 seconds	NZBC-D1/AS1-2.2.1*
Access route shall have a height clearance of minimum 2100 mm		0.2 seconds	NZBC-D1/AS1-1.4.1*
Each worker constructing a leading edge 6 feet or more above a lower level must be protected by guardrail systems, safety net systems, or personal fall arrest systems.	9-storey building in DE 5415 BIM objects	0.59 seconds	OSHA 29 CFR 1926.501(b)(2)*
	4-storey building in DK 707 BIM objects	0.24 seconds	
	2-storey fire station in DE 12173 BIM objects	0.2 seconds	

\*NZBC New Zealand Building Code  
\*OSHA Occupational Safety and Health Administration

## 5 Discussion and conclusion remarks

Implementing complete building code checking applications is not only a technical challenge per se, but is philosophically complex and intricate from a process and workflow perspective. This paper reveals several hidden caveats and bottlenecks in technology-driven solutions as we cannot presume that data and information are complete, certain, and deterministic. This is also reflected in different approaches adopted by human practitioners and software systems, the former is inherently implicit and fallible, while the latter requires formal, exact definitions and well-defined procedures to execute.

Therefore, we propose a dedicated spatial reasoner that is highly complementary to current approaches for encoding, documenting, and maintaining diverse knowledge related to regulations, buildings, and occupants. We do not advocate for the elimination of ambiguity and vagueness, but the co-existence of prescriptive and descriptive codes, and a systematic way of interpreting and motivating performance goals and design solutions using research evidence.

We opt for a transparent and traceable code checking solution with a focus on explanation and argumentation, in an effort to promote human-centric design and integrate compliance assessment with other domains such as decision support, assess management, building permit approval, compliance audits, etc.

As a next step, we are investigating the applicability of ASP4BIM with entire volumes of codes, regulations, and standards. We are also investigating the integration of ASP4BIM with existing code checking workflows and approaches such as Linked Data (Pauwels et al., 2011), Semantic Web (Bouzidi et al., 2012), BPMN (Häußler et al., 2021), VCCL (Preidel & Borrmann, 2015), and Machine Learning (Zhang & El-Gohary, 2019). The idea is to combine the strength of diverse methods in terms of availability, verifiability, portability, and scalability to support a holistic understanding of building requirements, and to tackle the specificity of multi-disciplinary knowledge and information.

## References

- Battle, R., & Kolas, D. (2012). GeoSPARQL: Enabling a Geospatial Semantic Web. *Semantic Web Journal*. <https://doi.org/10.3233/SW-2012-0065>
- Bennett, S., Lee Kirby, R., & MacDonald, B. (2009). Wheelchair accessibility: Descriptive survey of curb ramps in an urban area. *Disability and Rehabilitation: Assistive Technology*. <https://doi.org/10.1080/17483100802542603>
- Bhatt, M., Lee, J. H., & Schultz, C. (2011). CLP(QS): A declarative spatial reasoning framework. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. [https://doi.org/10.1007/978-3-642-23196-4\\_12](https://doi.org/10.1007/978-3-642-23196-4_12)
- Bloch, T., Katz, M., Yosef, R., & Sacks, R. (2019). Automated model checking for topologically complex code requirements – security room case study. *Proceedings of the 2019 European Conference on Computing in Construction*. <https://doi.org/10.35490/ec3.2019.157>
- Bogaerts, B., & Weinzierl, A. (2018). Exploiting justifications for lazy grounding of answer set programs. *IJCAI International Joint Conference on Artificial Intelligence*. <https://doi.org/10.24963/ijcai.2018/240>
- Bouzidi, K. R., Fies, B., Faron-Zucker, C., Zarli, A., & Thanh, N. Le. (2012). Semantic Web Approach to Ease Regulation Compliance Checking in Construction Industry. *Future Internet*. <https://doi.org/10.3390/fi4030830>
- Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., & Schaub, T. (2019). ASP-Core-2 input language format. In *arXiv*. <https://doi.org/10.1017/S1471068419000450>
- Dimyadi, J., Hjelseth, E., & Lassen, A. (2016). Development of BIM-based model checking solutions – ongoing research and practitioners’ demand. *CIB W78 2016 (33rd CIB International Conference on IT in Construction)*. <https://itc.scix.net/pdfs/w78-2016-paper-018.pdf>
- Eiter, T. (2007). Answer set programming for the Semantic Web. *Lecture Notes in Computer Science, 4670 LNCS*, 23–26. [https://doi.org/10.1007/978-3-540-74610-2\\_3](https://doi.org/10.1007/978-3-540-74610-2_3)
- Eiter, T., Fink, M., Krennwallner, T., & Redl, C. (2012). Conflict-driven ASP solving with external sources. *Theory and Practice of Logic Programming*. <https://doi.org/10.1017/S1471068412000233>
- Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., & Wanko, P. (2016). Theory solving made easy with clingo. *OpenAccess Series in Informatics*. <https://doi.org/10.4230/OASiCS.ICLP.2016.2>
- Häußler, M., Esser, S., & Borrmann, A. (2021). Code compliance checking of railway designs by integrating BIM, BPMN and DMN. In *Automation in Construction*. <https://doi.org/10.1016/j.autcon.2020.103427>
- Hjelseth, E. (2012). Converting performance based regulations into computable rules in BIM based model checking software. *EWork and EBusiness in Architecture, Engineering and Construction - Proceedings of the European Conference on Product and Process Modelling 2012, ECPPM 2012*. <https://doi.org/10.1201/b12516-73>
- Kondyli, V., Schultz, C., & Bhatt, M. (2017). Evidence-based parametric design: Computationally generated spatial morphologies satisfying behavioural-based design constraints. *Leibniz International Proceedings in Informatics, LIPIcs*. <https://doi.org/10.4230/LIPIcs.COSIT.2017.11>
- Li, B., Bhatt, M., & Schultz, C. (2019).  $\lambda$ Prolog(QS): Functional spatial reasoning in higher order logic programming. *Leibniz International Proceedings in Informatics, LIPIcs*. <https://doi.org/10.4230/LIPIcs.COSIT.2019.26>
- Li, B., Dimyadi, J., Amor, R., & Schultz, C. (2020). *Qualitative and Traceable Calculations for Building Codes*. <https://doi.org/10.46421/2706-6568.37.2020.paper006>
- Li, B., Fitzgerald, J., & Schultz, C. (2020). Modelling co-presence in the built environment - A spatio-temporal approach to human perception and movement. *EG-ICE 2020 Workshop on Intelligent Computing in Engineering, Proceedings*. <https://doi.org/10.1145/3414080.3414096>
- Li, B., Schultz, C., Melzner, J., Golovina, O., & Teizer, J. (2020). Safe and Lean Location-based Construction Scheduling. *Proceedings of the 37th International Symposium on Automation and Robotics in Construction (ISARC)*. <https://doi.org/10.22260/isarc2020/0195>
- Li, B., Teizer, J., & Schultz, C. (2020). Non-monotonic spatial reasoning for safety analysis in construction. *ACM International Conference Proceeding Series*. <https://doi.org/10.1145/3414080.3414096>
- Pauwels, P., Van Deursen, D., Verstraeten, R., De Roo, J., De Meyer, R., Van De Walle, R., & Van Campenhout, J. (2011). A semantic rule checking environment for building performance checking. *Automation in Construction*. <https://doi.org/10.1016/j.autcon.2010.11.017>
- Preidel, C., & Borrmann, A. (2015). Integrating Relational Algebra into a Visual Code Checking Language for Information Retrieval from Building Information Models. *16th International Conference on Computing in Civil and Building Engineering, Osaka, Japan*. [https://publications.cms.bgu.tum.de/2016\\_Preidel\\_ICCCBE.pdf](https://publications.cms.bgu.tum.de/2016_Preidel_ICCCBE.pdf)
- Schultz, C., Li, B., & Teizer, J. (2020). Towards a unifying domain model of construction safety: SafeConDM. *EG-ICE 2020 Workshop on Intelligent Computing in Engineering, Proceedings*. <https://doi.org/10.14279/depositonce-9977>
- Solihin, W., & Eastman, C. (2015). A Knowledge Representation Approach to Capturing BIM Based Rule Checking Requirements Using Conceptual Graph. In *CIB W78 Conference 2015*. <https://itc.scix.net/pdfs/w78-2015-paper-071.pdf>
- Stolk, S., & McGlinn, K. (2020). Validation of IfcOWL datasets using SHACL. *CEUR Workshop Proceedings*. <http://ceur-ws.org/Vol-2636/07paper.pdf>
- Wałęga, P. A., Schultz, C., & Bhatt, M. (2017). Non-monotonic spatial reasoning with answer set programming modulo theories. *Theory and Practice of Logic Programming*, 17(2), 205–225. <https://doi.org/10.1017/S1471068416000193>
- Zhang, R., & El-Gohary, N. (2019). A Machine Learning Approach for Compliance Checking-Specific Semantic Role Labeling of Building Code Sentences. In *Advances in Informatics and Computing in Civil and Construction Engineering*. [https://doi.org/10.1007/978-3-030-00220-6\\_67](https://doi.org/10.1007/978-3-030-00220-6_67)