Li B., Dimyadi J., Amor R. and Schultz C. (2020). "Qualitative and traceable calculations for Building Codes" In: *Proc. 37th CIB W78 Information Technology for Construction Conference* (CIB W78), São Paulo, Brazil, pp. 69-84. DOI: http://dx.doi.org/10.46421/2706-6568.37.2020.paper006

# QUALITATIVE AND TRACEABLE CALCULATIONS FOR BUILDING CODES

**Beidi Li[1], Johannes Dimyadi[2], Robert Amor[3], and Carl Schultz [4]**

**Abstract:** We present a methodology and prototype software framework to formalise normative provisions related to building design and construction to support automated compliance audit processes of ISO-standard building information models (BIM). Our framework is based on the declarative Answer Set Programming (ASP) logic programming language that we extend to support optimised geometric and spatial reasoning. We address three key challenges in formalising building codes:

(1) Ambiguity and qualitative aspects of modern performance-based building codes, e.g. "Essential information on wayfinding must be easy to see, read and understand (BR15)"; "There shall be no direct line of sight between an access route and a WC (NZBC-G1)". Our framework supports integrated layers of abstraction (in the context of ontologies and knowledge engineering) in order to ground how behavioural and human-centred terms such as "easy to see" and "access route" are ultimately defined based on results from cognitive psychology and declarative spatial reasoning.

(2) Managing a relatively large code base that can support maintainability, extensibility, traceability, and transparency, i.e. a particular code definition can be easily traced back to research literature used for its formalisation, and indeed multiple alternative definitions can be implemented and checked simultaneously.

(3) Keeping the computational runtime efficient for practical applications, despite processing a large number of codes and a large size of real-world building models.

We present empirical results (including real building models from New Zealand) to demonstrate that formalised codes definitions can focus exclusively on the semantics without mixing in "tricks" to make them computationally faster for handling a large-scale BIM, which is often at the cost of clarity and code base maintainability, etc. Thus, we leverage the declarative character of ASP to achieve a total separation of (a) semantic definition from (b) computational runtime efficiency of conformance checking.

**Keywords:** Building Information Modelling, Automatic Code Checking.

## 1 PERFORMANCE-BASED BUILDING CODES

Performance-based building codes aim to reduce unnecessary costs of conservatism in complying with a limited range of prescriptive design solutions but are challenging from both interpretation and implementation perspectives (Meacham et al. 2005). In addition to being conveyed in natural language intended for human interpretation, provisions in performance-based codes are often qualitative and descriptive in nature. In order to

[1] PhD student, Aarhus University, Aarhus, Denmark, beidi.li@eng.au.dk
[2] Honorary Academic, University of Auckland, Auckland, New Zealand, j.dimyadi@auckland.ac.nz
[3] Professor, University of Auckland, Auckland, New Zealand, trebor@cs.auckland.ac.nz
[4] Assistant Professor, Aarhus University, Aarhus, Denmark, cschultz@eng.au.dk

translate a natural language statement into computer-readable codes, domain experts need to formalise them into logical propositions, deduce implicit properties from a building model, and apply formalised constraints to derived model parameters (Eastman et al. 2009). The manual process is not only time-consuming and fallible, but also hinders knowledge sharing and transfer in multi-platform, cross-disciplinary cooperation and collaboration in the Architecture, Engineering, Construction (AEC) industry (Dimyadi and Amor 2013).

Our primary contribution in this research is a building code formalisation approach that facilitates automated compliance checking, as follows:

- To separate the (formal) representation of a building code from its numerous disambiguating interpretations and enables every disambiguating interpretation to be traceable back to research literature. We achieve this by augmenting Building Information Modelling (BIM) models with *spatial artefacts* (Bhatt et al. 2012a; Bhatt et al. 2010);
- To separate the semantics of a formalised building code from computational (spatial) optimisations necessary for checking the code set against a large-scale BIM in practice. We achieve this by extending the code checking engine with in-built spatial processing optimisation features that are applied automatically "under the hood".

## 2 CHALLENGES IN AUTOMATIC CODE CHECKING APPLICATIONS

Consider the workflow illustrated in Figure 1 which reflects a common approach for formalising building codes (Yang and Xu 2004). This workflow firstly requires a systematic approach for mapping qualitative terms in a building code to instantiated building properties, and a systematic approach for executing the formalised rule and reporting the checking results.

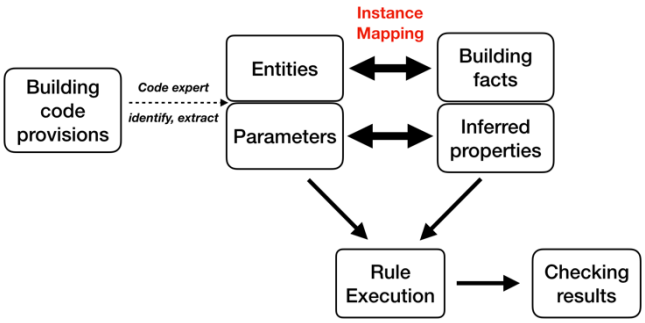


Figure 1. A common workflow for automated compliance checking of building codes.

We argue that the following two properties are essential for the body of formalised codes to be usable in practice, on a large community-wide scale, over a long period of time. We will use the following provision from the New Zealand Building Code (NZBC) as an illustrative example:

"*There shall be no direct line of sight between an access route or accessible route and a WC, urinal, bath, shower or bidet*".

**Desired property 1**: Whenever any (natural language) term is disambiguated, the rationale should be justified, and the justification should be readily traceable (e.g. to research literature in psychology, ergonomics, etc.).

Terms such as "direct line of sight" are qualitative and require disambiguation. Consider, for example, the range of eye heights of various occupants such as wheelchair users, walking children, walking adults, etc. Moreover, consider the possibility of visual reflections, the relevance of peripheral vision, whether a person using a bathroom facility should also be occluded, the visual system of a stationary versus a moving person, the precise definition of an access route, etc.

A comprehensive approach would be to provide numerous alternative (disambiguated) definitions of what "direct line of sight" means, with respect to a range of user groups, and maintain meta-data on the provenance of each definition. Compliance and non-compliance are then justified and traceable back to research literature. In contrast, ad hoc definitions, algorithms, and "magic numbers" used in disambiguation that are not clearly justified result in obscured deontic constraints and arbitrary evaluation criteria.

Our approach for systematic disambiguation, rather than to capture it at the code formalisation stage, is to extend instances of BIM with spatial artefacts, i.e. regions of empty space that carry information about human behaviour and experiences (Bhatt et al. 2012a; Bhatt et al. 2010; Bhatt et al. 2012b; Schultz and Bhatt 2013; Bhatt et al. 2014).

Example: Consider the region of empty space around an object such as a washbasin - this region is meaningful because a person must be located in that region to perform a particular act (e.g. washing hands). The geometry of this *functional space* region depends on properties of the person (consider wheelchair users, children, etc.), the task, and the object. Doors have an *operational space* required for opening and closing; people and sensors have *range spaces* (which can be further refined: visibility space, hearing space, etc.), and so on. These are examples of *spatial artefacts* (Bhatt et al. 2012a): regions of empty space that are rich with perceptual-locomotive semantics.

In our approach, the geometry of spatial artefacts is directly based on research literature in psychology and ergonomics (Kondyli et al. 2017; Kondyli et al 2018). This also enables the formalised code itself to remain closely aligned with the original natural language code.

**Desired property 2**: The formalised rule should, as closely as possible, reflect the source provision in the building code with no additional relational clauses added to facilitate the computational task of checking the code.

The computational task of checking a code provision against a BIM should not influence how the provision is represented formally in the code set. For example, suppose a first-order formalisation of the above NZBC provision is as follows, stating that it is a violation if bathroom object B is visible from corridor C:

Exists B,C in Objects :

bathroom_object(B) and corridor(C) and visible_from(B,C) →

violation(code(nzbc, privacy))

Now suppose this provision is checked against a large multi-storey BIM consisting of **n** corridor sections and **m** bathroom facilities. A naive code checking engine implementation will result in **n·m** visibility checks which is prohibitively expensive to run on large, real-world BIMs.

One solution is to include additional information in the formalisation itself, e.g. that the corridor and bathroom object should be on the same floor, and in the same rectangular visibility quadrant from a top-down floor plan perspective, i.e. a quad-tree cell division of building objects, where cells are merged if they are mutually visible:

```
Exists C,B in Objects :
  corridor(C) and bathroom_object(B) and
  same_floor(B,C) and same_2d_vis_quad_cell(B,C) and
  visible_from(B,C) →
  violation(code(nzbc, privacy))
```

While this provision can now be checked much faster, it has two serious drawbacks:

- The intended purpose of the provision (its semantics) is obscured by the added clauses making it significantly less comprehensible and extensible;
- The code set itself now depends on certain spatial data structures, and thus is significantly less portable, transparent, and maintainable.

The solution we advocate is to keep the initial, simple formalised code as is, and instead enhance the code checking engine with spatial data structure optimisations that are applied automatically.

## 2.1 Proposed framework and workflow

In contrast with state-of-the-art code checking practices, our framework proposes to map semantic code definitions to augmented building objects at the domain level, i.e. from regulatory ontology domain (*access route*, *line of sight*) to building ontology domain (*movement space*, *visibility space*). The separation of code semantics from enhanced model generation ensures that the code interpretation is accessible to regulation experts, and the code compliance assessment is verifiable by building experts independently. Figure 2 presents our workflow of translating textual requirements into formal rules that can be checked with respect to building instances.

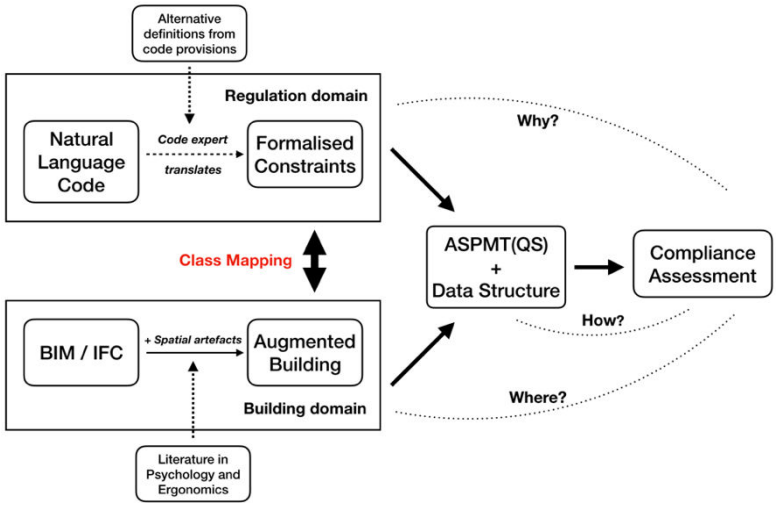


Figure 2. Proposed code checking workflow.

Firstly, code engineers enrich code provisions with meta-data, to capture the underlying semantics. This provides a unified, coherent way of interpreting qualitative

terms, based on the chosen quantification of human experiences (vision, access) from legislative documents or literature research.

Secondly, the provision is translated into a predicate rule in the following form stating that all objects X satisfying preconditions P must demonstrate properties Q:

$$For\ all\ X\ in\ Objects: P(X) \rightarrow Q(X)$$

Thirdly, object geometries, properties, and relationships are extracted from the building instance and declared as facts in the knowledge base. We then apply evidence-based deductive rules from literature (cognitive psychology, ergonomics) to enhance the building's representation with numerous modalities of actions permitted in the environment (hearing, viewing, manipulating, moving), captured through spatial artefacts.

Finally, we check the formalised rule against enhanced building objects using logic satisfiability solver extended with space, ASPMT(QS), and assess regulatory compliance (satisfiable, unsatisfiable).

Independent formulations of model facts, regulatory constraints, and deductive rules provide traceable rationales ("why"), transparent numerical computations ("how"), and rapid identification of code violations ("where").

In this paper we utilise the logic programming language Answer Set Programming extended to natively support spatial reasoning, ASPMT(QS) (Wałega et al. 2017; Wałega et al. 2015; Schultz et al. 2018), to implement our code compliance checking engine via automated non-monotonic reasoning. The declarative character of ASP allows the code formalisation to be open to alternative interpretations and user-defined rules, so that the checking system is portable and customisable.

In order to reduce runtime, we have further developed the core functionality of ASPMT(QS) with a general theory and framework of integrated spatial data structures for pre-processing building models and subsequently compute volumetric and topological relations of comparable objects. Examples of pre-processed spatial data structures include quad trees, R-trees, total orderings, containment hierarchies, etc. Spatial data structures are exploited in the form of additional constraints and optimisation techniques that are employed automatically by ASPMT(QS) as an integral part of its numerical and geometric computations used to determine spatial relationships. This enables building code experts to focus exclusively on the semantics of rules during code formalisation, and to ignore any computational runtime issues.

Our framework captures the general, abstract concept of spatial data structures (i.e. we do not commit to any particular data structure at this general level), and the general way in which they are exploited for more efficient geometric processing. Through this general framework we then define "instances" of spatial data structures that are actually employed during runtime – that is, our framework provides an extensible, flexible mechanism for incorporating a range of new spatial data structures in a uniform, mathematically rigorous manner.

## 3 EXTENDING BIM WITH SPATIAL ARTEFACTS

While BIM provides an annotated, relational structure representing a building's physical manifestation, building ontologies deal with physical, perceptual, and cognitive affordances of objects, e.g. a door is a place-transitioning object, a glazed panel permits mutual visibility, a wall is a place-delimiting object, etc. Such information is reflected in empty spaces, regions demarcated by theoretical limits of a user being able to interact

with a given object. Driven by the NZBC code, we demonstrate the creation of three types of artefacts: movement space, visibility space, and functional space.

**Movement spaces** are series of rooms, corridors, and stairways connected by place-transitioning objects (doors, openings), conditioned by user-specific body schema, e.g. a door with a clear opening less than 760mm is considered non-transitable to a wheelchair user (DBH 2015). Movement spaces further depend on intrinsic properties of building components (slab inclination, floor-intersecting obstacles) that can be derived systematically and unequivocally from IFC-compliant components (Figure 3).

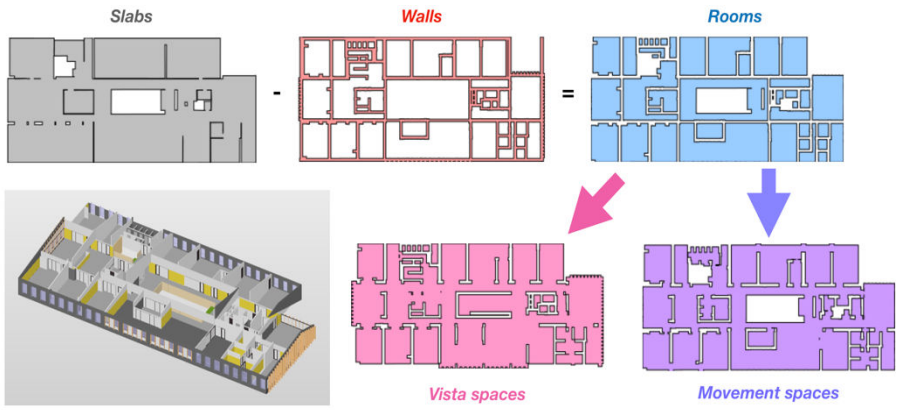


Figure 3. Construction of movement spaces from standardized physical representation of a real building.

The **visibility space** of an object is a closed, simple polyhedral region from which the object is viewable, depending on the observer's visual abilities (similar to the notion of isovists and viewsheds). We compute the unobstructed lines-of-sight from the object's boundaries as if seen by a person with perfect visual acuity and clip the region by an actual observer's visual range (Figure 4.a). Alternatively, Gibson's ecological theory suggests that visual perception depends on locomotion, i.e. the direction the observer is facing and the speed at which the observer is moving (Gibson 1954). This latter definition can be used to construct visibility spaces from an egocentric perspective, emphasising the effects of peripheral vision and vertical visual field.

The **functional space** of an object refers to the minimum clearances required for a human's physical activities in relation to the object (using a washbasin, grabbing handrails, opening a drawer, etc.). We infer the shape and position of functional spaces based on the object's geometry, the occupant's anthropometric properties, universal design guidelines, and ergonomics studies (Neufert and Neufert 2002). Such artefacts are represented as transparent blobs surrounding bathroom objects in Figure 4.e.

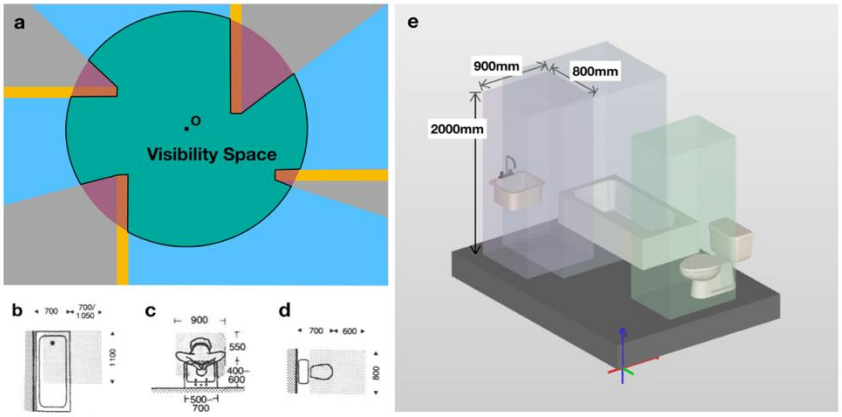


Figure 4. a) Visibility space of object O intercepted by the observer's visual range (green region); b-e) Spatial requirements for bathtub, basin, and toilet, figures taken directly from (Neufert and Neufert 2000); e) Visualization of functional spaces of bathroom objects.

Eliciting these spatial artefacts, the NZBC code is now translated into:

"*The visibility space of the functional space of a WC, urinal, bath, shower or bidet must be (topologically) disconnected from the movement space.*"

Such formalisation focuses exclusively on the semantics of the code provision, and systematically maps qualitative terms to classes of augmented building objects (*IfcSpatialArtefact*, a subtype of *IfcSpace*). Instances of *IfcSpatialArtefact* are then uniformly derived from a building's geometry based on empirical and experiential findings from the literature. In this way, we ensure a clear separation of rule interpretation, rule implementation, and rule execution.

## 4 Extending Logic programming with native spatial optimisation support

In prominent collision detection algorithms such as Q-tree and R-tree (Winter 1999), the algorithm recursively restricts the conditions for interferences so that at each step, the algorithm retains false positives (potential clashes) and discards all true negatives (no clashes).

Using the same principles, we present a general spatial data structure framework that automatically generates necessary and sufficient conditions to a given "target" relation R, respectively denoted as $R_{NEC}$ and $R_{SUFF}$, that are less costly to evaluate than the exact definition $R_{DEF}$. In terms of material implication:

(1) $R_{SUFF} \rightarrow R_{DEF}$ (a sufficient relation implies the target relation)

(2) $R_{DEF} \rightarrow R_{NEC}$ (a necessary relation is implied by the target relation)

(3) $\neg R_{NEC} \rightarrow \neg R_{DEF}$ (contrapositive of necessity)

Therefore, $R_{SUFF}$ gives a shortcut to determine that $R_{DEF}$ holds, and the negation of $R_{NEC}$ gives a shortcut to determine that $R_{DEF}$ does not hold. From a set theoretic perspective, $R_{SUFF}$ represents all true positives, the complement of $R_{NEC}$ represents all true negatives, and the difference between $R_{NEC}$ and $R_{SUFF}$ represents undecidable cases that require a

series of more refined sufficient and necessary conditions, the difference of which eventually, finally requires numerical evaluation with $R_{DEF}$ (Figure 5).

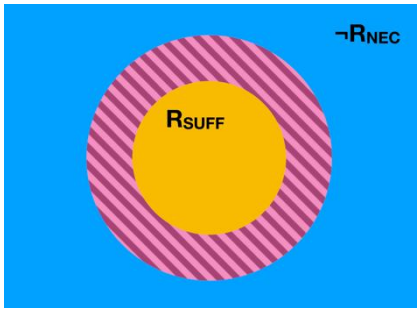


Figure 5. Boolean representation of relations $R_{NEC}$ and $R_{SUFF}$.

**Answer Set Programming**: ASP is a logic-programming paradigm developed within the artificial intelligence community that has its foundations in first-order logic (Brewka 2011). We use ASP by encoding a building as ASP facts, encoding building code compliance rules as ASP rules and constraints, and code compliance checking is implemented via ASP answer set search[5]. Our general formulation for spatial data structures is expressed in ASP as the follow logic programming rules and constraints:

```
R :- R_suff.

-R :- not R_nec.

R :- R_def.

:- R, -R.
```

## 4.1 Empirical test with circles

Now we test the operational aspects of our proposed spatial data structure framework with n randomly generated circles in a delimited rectangular region of the 2D plane. The goal is to enumerate all pairs of circles that overlap with each other as fast as possible.

The following is a simple demonstrative example of a *particular* spatial data structure that is used by following the *general* structure defined in our framework. We partition the region into k equisized clusters that are jointly exhaustive and pairwise disjoint, so that each circle is assigned to exactly one cluster containing its centre. If two circles are contained in non-adjacent clusters, they are necessarily disconnected. If two circles are concentric, they necessarily overlap.

In fact, checking cluster IDs (i.e. comparisons of numbers) is computationally much less costly than directly evaluating the exact definition of circle-circle intersection (i.e. a polynomial inequality of degree 2), this provides a shortcut to symbolically identify circle pairs that display trivially topological relations. Having run a series of tests, we

[5] Similar to Prolog, ASP has a knowledge base of facts and rules of the form: "Head :- Body." meaning that if the *Body* is true, then the *Head* must also be true. Rules with no *Head* are ASP constraints, written: ":- Body." meaning that the *Body* must not be true (i.e. as a logical expression: *Body* implies False). *Head* and *Body* expressions consist of literals, representing propositions that can be either True or False, and ASP reasoning engines are specifically designed to rapidly find combinations of deduced facts that are consistent with all given domain rules (referred to as models or answer sets). We have extended the base language of ASP beyond propositions so that a set of consistent facts must also be spatially consistent, e.g. a 2D point P can never be both inside, and outside, of a given circle C (Wałega et al. 2017; Walega et al. 2015; Schultz et al. 2018). In this paper we have further extended core functionality to automatically incorporate spatial data structures for efficient spatial reasoning.

empirically determine that the computational complexity decreases from $C_1 \cdot n^2$ to $C_2 \cdot n^2/k$, where $C_1$ and $C_2$ are some arbitrary constant factors. The ASP implementation is as follows:

**Sufficient condition**: The following rule states that concentric circles overlap:

```
overlap(C1, C2) :-
  circle(C1), circle(C2),
  centre(C1, (X1, Y1)), centre(C2, (X2, Y2)),
  X1 = X2, Y1 = Y2.
```

**Negated necessity**: The following rule states that circles in different clusters do not overlap:

```
-overlap(C1, C2) :-
  circle(C1), circle(C2),
  cluster(C1, K1), cluster(C2, K2),
  not adjacent(K1, K2).
```

**Relation definition**: The following rule implements the exact inequality that decides the overlap relation between two circles:

```
overlap(C1, C2) :-
  circle(C1), circle(C2),
  centre(C1, (X1, Y1)), centre(C2, (X2, Y2)),
  radius(C1, R1), radius(C2, R2),
  (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2) < (R1+R2)*(R1+R2).
```

We implement the simple cluster partitioning strategy in ASP (Figure 6). Our empirical experiments show that average runtimes rapidly decrease with large number of clusters[6] (Figure 7). For $10^4$ circles, clingo runtime drops from 17.1 seconds with no clusters (K = 1) to 4.2 seconds with 10 clusters (K = 10); for $2 \cdot 10^4$ circles, runtime drops from 60.5 seconds with no clusters to 18.7 seconds with 10 clusters; for $5 \cdot 10^4$ circles, runtime drops from 457.6 seconds with no clusters to 133.6 seconds with 10 clusters, resulting in an average decrease by factor 3.

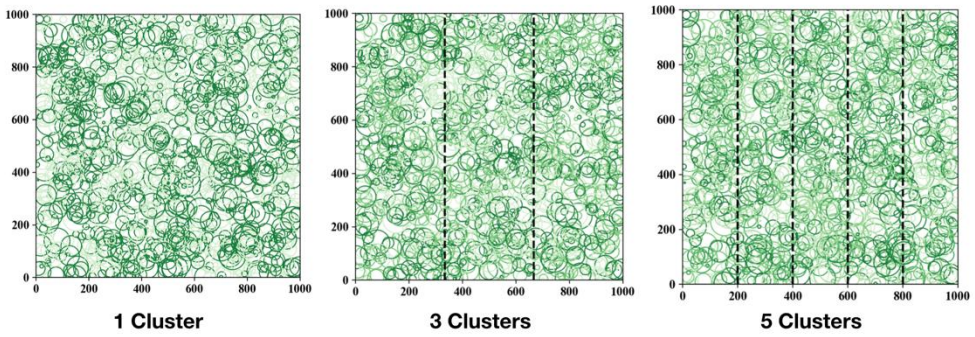


Figure 6. Horizontal clustering of circles with K = 1, 3, 5.

[6] We ran our empirical experiments on a Mac Book Pro with Mac OS 10.13.6, processor 2.2 GHz Intel Core i7, and 16GB RAM.

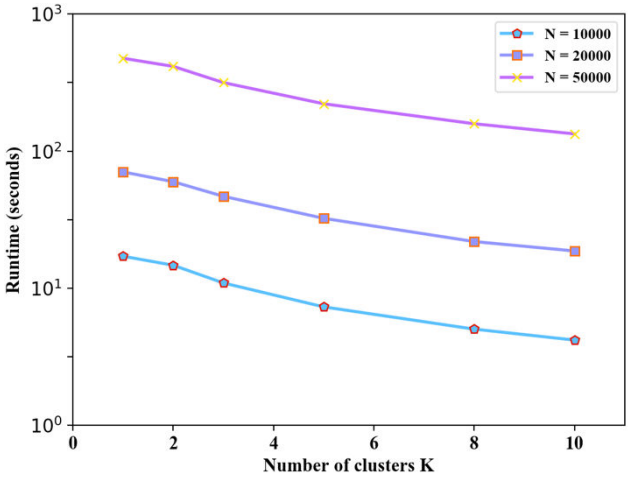


Figure 7. Runtimes of the all overlapping circles test for K = 1, 2, 3, 5, 8, 10 and N = 10000, 20000, 50000 with *clingo*.

## 4.2 Scalability test with a real building model

Now let's take another look at the formalised NZBC rule asserting that there should be no direct line of sight between an access route or accessible route[7] and the vicinity of a bathroom object that a user may occupy during its usage. Moreover, acceptable solutions allude to the procedural complexity in checking such requirement, e.g. a single use sanitary facility can be visually separated solely by an opaque door, whereas a space containing multiple cubicles should include additional screens for visual privacy.

As seen Section 2 and 3, ambiguous terms (line of sight, access, vicinity) in the new code formulation require diligent consideration of every combination of user, activity, and situation. As the code introduces accessible route that is applicable to people with disabilities, we are further to incorporate physiological evidence of one's body measurement (height, encumberment, etc.) into our semantic formalisation. Similarly, functional spaces and visibility spaces vary greatly depending on occupant's perceptual-locomotive abilities and are constrained by local spatial configurations (sloped roof, handrails, sanitary bins, etc.). We define the following predicates:

*Movement_Space(Occupant)*

One movement space predicate is defined for each Occupant value: "Children", "Adults", "People with dementia", "Elderly people", "Patients with crutches".

*Functional_Space(Sanitary_Fixture)*

One functional space predicate is defined for each instance of sanitary fixtures taking the following type values: "WC with external cistern", "Enclosed shower", "Open shower", "Bidet", "Urinal", "Wall-hung basin".

*Visibility_Space(Object, Observer)*

One visibility space predicate is defined for each instance of Object and Observer, where Observer can take the values: "Fully-sighted", "Children", "People with peripheral vision loss".

In relation to particular building instances, we employ psychology findings to semantically enrich the building representation with spatial artefacts.

[7] People whose ability to use buildings is affected by mental, physical, hearing or sight impairment.

In practice, we first identify bathroom objects labelled as *IfcFlowTerminal* and infer the shape of regions in which a user can engage in material contact with the object, characterised by action's modality, e.g. approach, attainment, operation, manipulation (Ginnerup 2009), denoted as *IfcFunctionalSpace*. We then compute the regions visible from the outer boundaries of I*fcFunctionalSpace*, depending on individual occupant's visual acuity, eye level, and head movement, denoted as *IfcVisibilitySpace*. After that, we extract access routes from the building's walkable surfaces, floor-intersecting obstacles, and place-transitioning objects, denoted as *IfcMovementSpace*. Finally, we test intersections between *IfcVisibilitySpace* and *IfcMovementSpace*.

A proof-of-concept implementation with a two-storey building in the city of Christchurch, New Zealand has identified 2 stairs, 1 atrium, and 2 corridors, 87 sanitary fixtures, of which 12 showers, 17 bathrooms, and 17 basins. We derive 245 functional spaces of bathroom objects, 490 visibility spaces of such functional spaces, and 100 movement spaces, based on the building geometry and user-specific data.

A naive approach for checking compliance attempts to compute the exact intersections between all pairs of comparable polyhedral meshes, which is laborious and subject to numerical instability issues.

Based on our previous empirical test results, we propose a series of increasingly constricting necessary and sufficient conditions that efficiently prune the search space. Concretely:

- Two meshes are necessarily disconnected if their axis-aligned bounding boxes (AABB) are disconnected;
- Two meshes are disconnected if their 2D projections are disconnected;
- Two polygons A and B are disconnected if A is a proper part of one of B's holes;
- Two polygons A and B overlap (not disconnected) if A contains one of B's holes, etc.

In the absence of obvious shortcuts, one may still apply advanced collision detection algorithms such as overlapping AABB (Luo et al. 2011) to shrink the region where interferences may occur. Exact calculations will be used as a last resort to decide if two meshes overlap. The ASP implementation is as follows:

Relation R1 is implemented as:

```
comparable(V, M) :-
  bathroom_object(O), functional_space(O, F),
  visibility_space(F, V), movement_space(M).
```

Relations R2 are implemented as:

```
overlap_(C1, C2) :-
  axis_aligned_bounding_box(C1, (Xmin1,Xmax1,Ymin1,Ymax1,Zmin1,Zmax1)),
  axis_aligned_bounding_box(C2, (Xmin2,Xmax2,Ymin2,Ymax2,Zmin2,Zmax2)),
  Xmax2 > Xmin1, Xmin2 < Xmax1,
  Ymax2 > Ymin1, Ymin2 < Ymax1,
  Zmax2 > Zmin1, Zmin2 < Zmax1.
```

```
part_of(C1, C2) :-
  axis_aligned_bounding_box(C1, (Xmin1,Xmax1,Ymin1,Ymax1,Zmin1,Zmax1)),
  axis_aligned_bounding_box(C2, (Xmin2,Xmax2,Ymin2,Ymax2,Zmin2,Zmax2)),
  Xmin1 >= Xmin2, Xmax1 <= Xmax2,
  Ymin1 >= Ymin2, Ymax1 <= Ymax2,
  Zmin1 >= Zmin2, Zmax1 <= Ymax2.

disconnected(A, B) :-
  comparable(A, B), not overlap_(BA, BB).

disconnected(A, B) :-
  comparable(A, B), hole(B, HB), part_of(A, HB).

-disconnected(A, B) :-
  comparable(A, B), hole(B, HB), part_of(HB, A).
...
disconnected(A, B) :-
  comparable(A, B), intersect(MA, MB, I),I = spatial_void.
```

Relation R3 is implemented as:

```
violation(NZBC, privacy) :- comparable(V, M), not disconnected(V, M).
```

We now have a clear formulation of code semantics R1 that can take alternative definitions, a set of rules R2 that automatically integrates optimisation in numerical evaluation of fully grounded variables, and a rule R3 that indicates facts in the actual building contradict regulatory requirements.

Table 1 shows runtime before and after introducing the data structure. The objects have been extracted from the IFC model and manually edited to resolve geometric discrepancies. For the sake of simplicity, we only show results from intersection checks between 2D projections of visibility spaces and movement spaces in a single storey. Figure 8 shows representative cases where visibility spaces and movement spaces match with necessary or sufficient conditions of “disconnected” so that exact computations of intersection can be avoided.

Table 1. Model specifications and runtimes in clingo of evaluating the building code visibility requirement on the two-story building in Christchurch.

| | |
|---|---|
| Average number of vertices in visibility spaces | 61 |
| Average number of vertices in movement spaces | 127 |
| Average runtime without spatial data structures | 9.297 seconds |
| Average runtime with spatial data structures | **1.579** seconds |

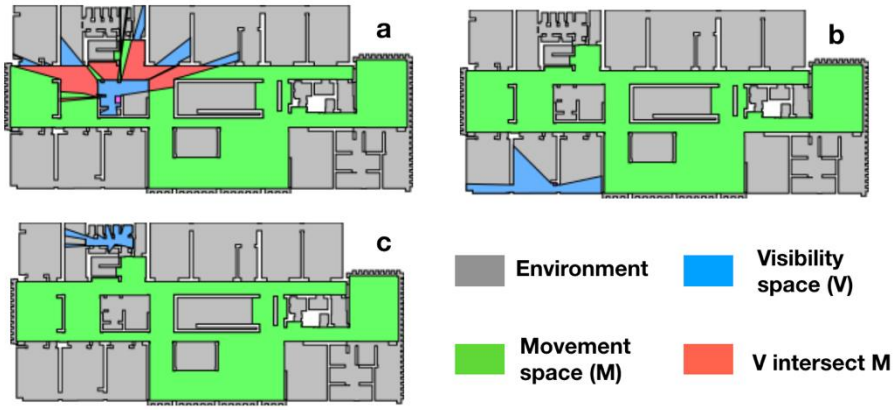


Figure 8. Necessary contact between visibility space V and movement space M: a) V overlaps with a hole in M. Necessary disconnectedness between V and M: b) M contains no vertices in the overlapping AABB of V and M; c) V and M have disconnected axis-aligned bounding boxes.

## 5 Related work and conclusions

Regulatory compliance, as a major concern to professionals in the AEC industry, provides statutory, contractual performance objectives that building design must provide for a variety of occupant types and use cases (Beach et al. 2015). The task of automatic code checking requires encoding textual regulations as a constraint that is systematically checkable and verifiable with respect to digital information acquired about a building.

However, the maintenance of a voluminous code base with respect to real world buildings is facing serious drawbacks in terms of insufficient deployment of industry standards and lack of adequate implementation support (Preidel and Borrmann 2016). Challenges in automatic and semi-automatic code checking include but are not limited to: conflicting shape representation, inconsistent object classification, rule abstraction and required tool complexity, obscure semantic mapping, continuous updates in legislative documents, etc. (Sacks et al. 2017; Dimyadi et al. 2016; Solihin and Eastman 2015). In addition, state-of-the-art automatic code checking software systems often embrace an ad hoc approach of deriving model views required to run an encoded query (Eastman 2009), which makes the checking results subject to obscured rule interpretation and untraceable numerical assumptions. Furthermore, most theoretical frameworks have been developed in a specific context targeting one single aspect of regulatory conformance (structural soundness, safety, thermal performance), this further prohibits rule sets to be reused and shared among different communities (Solihin and Eastman 2015; Zhang et al. 2013; Pauwels et al. 2011). Therefore, it has become evident that a modular, extensible, and

transparent system is needed for rule interpretation, and a reliable, coherent, operational logic-based system for rule checking.

In this paper, we proposed to leverage the strength of a (pure) semantic code formalisation to maintain a clean, modifiable code base. To achieve this, we extended ASPMT(QS) with a framework for spatial data structures to embed optimisations directly within the reasoning engine. We demonstrated the operational aspects of our framework with an industry-scale building and a qualitative, descriptive constraint on bathroom user's privacy from the NZBC. The ASP rule that was checked was a "pure" encoding of the original building code rule, containing no additional clauses concerning computational runtime and thus more closely aligns with the natural language code description. Our empirical results show a 4-factor speed up when the spatial data structure optimisations were built into the ASP search engine.

This paper belongs to a series of research efforts specifically aiming to provide a standard-compliant code checking system that allows a regulation expert to define a rule, execute the rule, and explain how the rule is applied and why the rule has passed, failed, or inconclusive based on available information. As we quickly touched upon in 4.2, rule semantics are a complex subject that display layers of abstraction including procedural complexity, defeasible principles, tolerances, etc. (Sacks et al. 2017; Dimyadi et al. 2017), and we intend to meticulously address these issues in coming studies.

## 6 References

Beach, T.H., Rezgui, Y., Li, H., and Kasim, T. (2015). A rule-based semantic approach for automated regulatory compliance in the construction sector. *Expert Systems with Applications*, 42 (12), pp. 5219-5231.

Bhatt, M., Hois, J., Kutz, O., and Dylla, F. (2010). Modelling functional requirements in spatial design. *International Conference on Conceptual Modeling (ER 2010)*, pp. 464-470. Springer-Verlag Berlin Heidelberg.

Bhatt, M., Schultz, C., and Huang, M. (2012). The shape of empty space: Human-centred cognitive foundations in computing for spatial design. *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 33-40. Innsbruck.

Bhatt, M., Hois, J., and Kutz, O. (2012). Ontological modelling of form and function for architectural design. *Applied Ontology*, 7(3), 233-267.

Bhatt, M., Schultz, C., and Freksa, C. (2014). The 'Space' in spatial assistance systems: conception, formalisation and computation. *Representing Space in Cognition: Interrelations of behaviour, language, and formal models*. Oxford Scholarship Online.

Brewka, G., Eiter, T., and Truszczynski, M. (2011). Answesr set programming at a glance. *Communication ACM*, 54(12), pp. 92–103.

Danish Transport and Construction Agency. (2015). Danish Building Regulations 2015.

Department of Building and Housing (DBH), Wellington, New Zealand. (2011). Compliance Document for New Zealand Building Code Clause G1 Personal Hygiene. Second Edition Amendment 6.

Dimyadi, J., and Amor, R. (2013). Automated Building Code Compliance Checking – Where is it at? *Proceedings of CIB World Building Congress*. Brisbane, Australia, pp.172-185.

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

Dimyadi, J., Governatori, G., and Amor, Robert. (2017). Evaluating LegalDocML and LegalRuleML as a Standard for Sharing Normative Information in the AEC/FM Domain. *Proceedings of Joint Conference on Computing in Construction (JC3).* Heraklion, Crete, Greece, pp. 637-644.

Eastman, C., Lee, J, Jeong, Y., and Lee, J. (2009). Automatic rule-based checking of building designs. *Automation in Construction*. 18(8), pp. 1011-1033.

Gibson, J. J. (1954). The visual perception of objective motion and subjective movement. *Psychological Review*, 61(5).

Ginnerup, S. (2009). Achieving full participation through Universal Design. Council of Europe.

Kondyli, V., Schultz, C., and Bhatt, M. (2017). Evidence-based parametric design: Computationally generated spatial morphologies satisfying behavioural-based design constraints. *13th International Conference on Spatial Information Theory (COSIT 2017)*, pp. 11:1-11:14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Kondyli, V., Bhatt, M., and Hartmann, T. (2018). Precedent based design foundations for parametric design: The case of navigation and wayfinding. *Advances in Computational Design*, 3(4), pp. 339-366.

Meacham, R., Bowen, R., Traw, J., and Moore, A. (2005). Performance-based building regulation: current situation and future needs. Performance-based building regulation: Current situation and future needs. *Building Research and Information*, 33(2), pp. 91-106.

Luo, F., Zhong, E., Cheng, J. and Huang, Y. (2011). VGIS-COLLIDE: an effective collision detection algorithm for multiple objects in virtual geographic information system. *Int. J. Digital Earth*, 4, pp. 65-77.

Neufert, E., and Neufert, P. (2000). Architect's Data. Blackwell Science, Malden.

Pauwels, P., Deursen, D., Verstraeten, R., De Roo, J., Meyer, R., Van de Walle, R., and Campenhout, J. (2011). A semantic rule checking environment for building performance checking. *Automation in Construction*, 20, pp. 506-518.

Preidel, C., and Borrmann, A. (2016). Towards Code Compliance Checking on the basis of a Visual Programming Language. *ITcon*, 21, pp. 402-421.

Sacks, R., Ling, M., Yosef, R., Borrmann, A., Daum, S., and Kattel, U. (2017). Semantic Enrichment for Building Information Modeling: Procedure for Compiling Inference Rules and Operators for Complex Geometry. *Journal of Computing in Civil Engineering*, 31(6).

Schultz, C., and Bhatt, M. (2013). InSpace3D: A middleware for built environment data access and analytics. *Procedia Computer Science*, 18, pp. 80-89.

Schultz, C., Bhatt, M., Suchan, J., and Wałęga, P. A. (2018). Answer Set Programming Modulo 'Space-Time'. *International Joint Conference on Rules and Reasoning*, pp. 318-326. Springer, Cham.

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

Wałęga, P. A., Schultz, C., and Bhatt, M. (2017). Non-monotonic spatial reasoning with answer set programming modulo theories. *Theory and Practice of Logic Programming*, 17(2), pp. 205-225.

Wałęga, P. A., Bhatt, M., and Schultz, C. (2015). ASPMT (QS): non-monotonic spatial reasoning with answer set programming modulo theories. *13th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2015)*, pp. 488-501. Springer, Cham.

Winter, S. Hybrid quadtree. (1999). Technical Report, Institute for Geoinformation, Technical University Vienna, Austria.

Yang, Q.Z., and Xu, X. (2004). Design knowledge modeling and software implementation for building code compliance checking. *Building and Environment*, 39(6), pp. 689 - 698.

Zhang, S., Teizer, J., Lee, J., Eastman, C., and Venugopal, M. (2013). Building Information Modeling (BIM) and Safety: Automatic Safety Checking of Construction Models and Schedules. *Automation in Construction*, 29, pp. 183–195.