

Bachelor Thesis
Computing Science



Radboud University

A Specification Language and Semantics for Architectural Design

An exploration of formalising the logic behind
architectural design

Author:
Julian Besems
s4783751

First supervisor:
Dr. Freek Wiedijk
freek@cs.ru.nl

Second supervisor:
Tom Holberton
t.holberton@ucl.ac.uk

21 August 2020

Cover image - A collection of plans of multiple instances of a design for art galleries in Venice. Each gallery follows the same design rationale, but the outcome varies based on the contextual setting, thus producing a wide variety of design proposals. Created as part of a design project completed as part of a Masters in Architecture. [Besems, 2020a]

Abstract

Since the 1960's computers have played an increasing role in architectural design. At first this was only as a drawing and modelling tool, but lately computation is also used to drive design decisions through parametric modelling methods and generative design. The current limitation of these methods is that the input and output is heavily limited to geometric information, with no universal way of connecting the geometric proposition of a design to the underlying design principles. If computers are to play a larger role in the architectural design process, it is important that there is also a way in which they can start to inform the functional aspects of a building, in addition to its shape. For this it is essential that there is a formal way of describing these aspects. This thesis explores a way of doing this through the formulation of a syntax and semantics, by which a set of geometries can be evaluated to determine whether this is a valid instance of a specific project definition. A primary source for the concept of an architectural language is *Christopher Alexander's* Pattern Language, which in turn has had a significant influence on the field of computer science due to it being the primary inspiration for software design patterns.

Contents

1. Introduction	4
1.1 A Pattern Language and OO Design Patterns	5
1.2 Formal Definition of Architecture	5
2. Related Work	8
2.1 Current computation within architecture	8
2.1.1 Space syntax	8
2.1.2 Machine learning	8
2.2 Previous explorations	10
2.2.1 Design project 1: University libraries	10
2.2.2 Design project 2: Art galleries in Venice	13
2.2.3 Thesis: Towards a new architectural entity	15
2.2.4 Observed limitations	16
3. Syntax	17
3.1 Categories	17
3.2 Grammar	19
4. Semantics	28
4.1 Auxiliary Functions	28
4.2 Semantic Functions	30
4.3 Rules	31
4.4 Instance Statements	34
4.5 Building Element Statements	36
4.6 Empty Statements	37
5. Examples	38
5.1 Connected Cubes	38
5.2 C. Alexander pattern 101: Building Thoroughfare	40
5.3 Generated Art Gallery St. Mark's Square	41
5.4 Courtyard	45
6. Implementation	52
6.1 Decidability	52
6.2 Integration in CAD	52
6.2.1 Derivation of spaces	52
6.2.2 Derivation of elements	53
6.2.3 Mutual influence of design and definition	54
6.3 Use for generative architecture	54
7. Conclusion	57
7.1 Discussion	58
List of Illustrations	59
Bibliography	62
Appendices	63
Appendix A	63
Appendix B	64

1.

Introduction

Le Corbusier, one of the most influential architects of the 20th century, made a statement in 1925 about the *Roneo* filing cabinet: “This new system of filing which clarifies our needs, has an effect on the lay-out of rooms, and of buildings.” [Burke & Tierney, 2007]. This statement indicates his interest in classification systems and their possible value within architecture. Le Corbusier expanded on this by presenting three categories that form the principal elements of architectural buildings: The Mass, The Skin and The Plan. This definition was paired with a description of how each element relates to the other [Burke & Tierney, 2007].

In the 1970’s the architect and mathematician Christopher Alexander expanded this idea of generally describing fundamental architectural design patterns. He abstracted the way people relate to different architectural surroundings in 253 patterns, which are described in his book *A Pattern Language* [1977]. Each pattern is defined in a standard structure. Together this forms a framework of interconnected patterns that are to be combined to aid in a design process.

With the introduction of the IBM Drafting System [IBM, 1988] in the 1960’s, architects have started to use computers as part of their design process, although at that time and for a long time after this was limited to drawing purposes. For the majority of architects this hasn’t changed much over the last 50 years, the drawing programs have become three-dimensional modelling programs, and other than technical drawings, computers are now also heavily used for artistic representational purposes, but in the end the computer is still mostly a tool used to create an output. This did start to change with the rise of parametric design, where forms of a building are derived by putting other form based parameters through a set of algorithms, thus starting to use the computer not only as a tool for representation but also to drive design decisions. The limitations of these parametric design methods are however that the algorithms only calculate forms based on other three-dimensional inputs provided by the architect, but not based on what the function of a building element is, or how one space relates to another in terms of the inhabitants’ behaviour.

If computers are to play a larger role in the architectural design process, there should also be a way in which they can be used to aid in design decisions regarding the functional aspects of a building, in addition to its shape. For this it is important that there is a formal way of describing these aspects. The ideas of Le Corbusier and Alexander form a suitable starting point for this.

This thesis aims to describe a syntax and semantics for the combination of building patterns within architectural design. This could be a starting point for the development of a formal framework of building design, enabling computers to more fundamentally aid architectural design methods.

1.1 A Pattern Language and OO Design Patterns

The most well known example of an attempt to express architectural design concepts in some form of a language format is *A Pattern language* by Christopher Alexander [1977]. Alexander argues that users are more sensitive to their needs than architects could be [Alexander, 1975]. Following this principle he set out to describe the essential different patterns that occur within architectural design, so that each individual could design their own built environment. Each pattern first starts with a description of which other patterns or actions are required before this pattern can be put in place. Then the motivation for this pattern is given in one sentence. This is then followed by the discussion of the functional problem that has to be solved whilst implementing this pattern. After this the instruction is given on how to implement this pattern in order to satisfy the conditions posed by the problem. This instruction is generally 3-4 lines, in combination with a sketched diagram. Finally the pattern description is concluded with the reference to other patterns relating to this pattern.

Since each pattern refers to other patterns as possible pre- and post-conditions, a progression through the design problem at hand is defined, through which a loose semantic framework is created for architectural design. This language however lacks a formal definition, making it problematic to implement in a computational context, which is understandable due to the extremely limited way in which computers were used in architectural practice at the time. On the other hand it is worthwhile to note how this concept of architectural design patterns has, unintentionally, created a theoretical connection between the disciplines of computing science and architecture.

The development of software design patterns in Object Oriented programming (OO) was heavily influenced by Alexander's work. This followed from the OOPSLA session in 1987, where Ward Cunningham outlined the way in which his team had started to construct software design patterns as an adaptation of Alexander's architectural patterns. The motivation for this was to have "a radical shift in the burden of design and implementation"[Sowizral, 1987], through defining pieces of reusable code that make it easier to define solutions for the issue addressed by the design pattern in question.

1.2 Formal Definition of Architecture

The difference between the approach of Alexander and the domain specific language (DSL) developed in this thesis, is that Alexander, much like design patterns in OO, focussed on the description of frequently occurring design problems, with a proposed abstracted pattern of finding a solution to this, all set within a larger framework of how these are related. It does however not offer a way in which architectural design proposals can be defined and interpreted in addition to the physical instance, so to say, it is the concept of OO design patterns, but without a programming language within which these can be defined.

In current architectural practice the majority of design is heavily reliant on CAD modelling, but a way to formally define architectural design beyond the geometries that make up the physical form is not available. A programming analogy would be: the output of an algorithm can be written down

110 Main Entrance

Pre-conditions:

- Site Repair (104)
- South Facing Outdoors (105)
- Wings of Light (107)
- Circulation Realms (98)
- Family of Entrances (102)

Motivation:

Placing the main entrance (or main entrances) is perhaps the single most important step you take during the evolution of a building plan.

Functional problem:

The entrance must be placed in such a way that people who approach the building see the entrance or some hint of where the entrance is, as soon as they see the building itself.

Instruction:

Place the main entrance of the building at a point where it can be seen immediately from the main avenues of approach and give it a bold, visible shape which stands out in front of the building

References:

- Family of entrances (102)
- Entrance room (130)
- Entrance transition (112)
- Shielded parking (97)
- Car Connection (113)

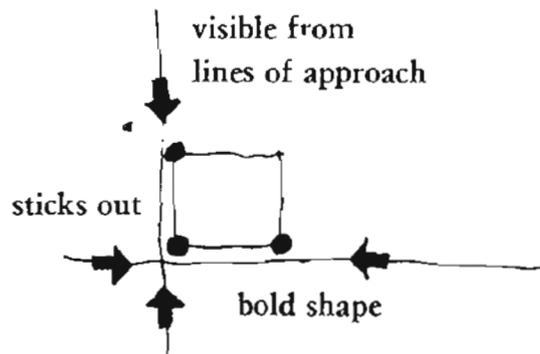


Figure 1. Pattern 110 - Main entrance by Christopher Alexander

Table 1: Summarised version of the main entrance pattern as described by Christopher Alexander [1977, p541]

in a text file, but there is no means of defining the algorithm itself, let alone compile it to automatically generate the output. Andrew Witt [2016] addresses this issue by his statement that architecture lacks a taxonomy, which in turn leads to a limitation in terms of quantifiability of its manifestations. The development of a formal language enabling architects to define a project through abstracted design decisions could offer a way in which architectural projects are not necessarily bound to one particular instance, but rather a range of different realisations of one proposal.

Aspiring to explore ways in which this can be achieved, the research question for this thesis is:

To what extent can the axiomatic components of architectural design and their functional role within the project be defined in a formal language?

In order to address answer this the following sub questions will be addressed:

- To what extent can the ambiguity of design be incorporated within the formulation of such a language, as to not limit the architectural design process?
- What criteria can be set to measure completeness in a formal language describing building patterns?
- How relevant is a potential formal framework of building patterns for the development of generative architecture?

2.

Related Work



Currently the ways in which computational methods are used to inform design usually involve the application of existing computational strategies to an architectural context, but fall short in terms of integrating a computational method as part of the architectural design process. Even where generative design procedures are used to arrive at a design form, it can be argued that these are mere ways of using algorithms to draw in more elaborate ways, thus using computation as a tool rather than that the computation is ingrained within the architectural concept of spatial configurations, since the generative component is only concerned with the form, not with the logic of the building itself.

Prominent architectural theorists of the same timeframe, such as Corbusier, Fuller and Tange, showed interest in mathematical and computational theories such as network theory in order to find ways to model architectural concepts, in this case as part of the multidisciplinary CIAM and Delos meetings [Wigley, 2001]. Since it has seemed the mutual interest of the two disciplines has faded, and the architectural industry has failed to integrate computational methods within the design methodology.

2.1 Current computation within architecture

2.1.1 Space syntax

A research topic within architecture named space syntax has studied the nature of space since the 1970s, based on human interaction with the built environment. Whilst the name suggests a formal language, this discipline primarily investigates geometric zoning algorithms that define and analyse relationships of a spatial setting, in regards to wider social, economic and environmental aspects [Space syntax, 2020]. It is not concerned with formally defining the elements that make up the built environment and through how a spatial configuration is a result of their assembly, and thus does not directly address the missing link from an architectural CAD model to its design rationale.

2.1.2 Machine learning

With the rise of attention to AI and machine learning in general, the interest in this field of computation seems to increase within the field of architecture as well, indicated by various recent papers and articles on the topic. Current implementations of ML within architecture often utilise existing models and frameworks that have not been specifically built with architectural design in mind. A result is that these have to rely on visual media, as seen in the recent

thesis by Stanislas Chaillou - *AI & Architecture* [2019]. As such the difference in implementation is marginal from ways in which AI is implemented within art projects such as the *Quasimodo* project by Mario Klingemann [2019].

With this approach the previously discussed limitations of current computation within architecture, where the link between the derived geometry and the functional meaning of that geometry is absent in the algorithmic approach, is more visible than ever. Since the model is not trained based on what the elements in the drawings mean, the result is a set of images that are the result of training a generative adversarial network (GAN)¹ model to generate something that looks convincing as an architectural drawing, but are devoid of any three dimensional consideration, let alone bear in mind a design logic required to fulfil the requirements of an architectural project. This critique can be expanded further, since the training data, and the resulting output do not even follow the conventional vector based format of an architectural CAD drawing or model, but in order to be compatible with the existing format of the image based GAN the drawings are compressed to pixel based images, resulting in the loss of scale and the distinction between separate, yet overlapping objects.

Arguably a step back would be in order, where the full generation of architectural drawings or even buildings should not be the first milestone to



Figure 2. GAN generated floor plans by Stanislas Chaillou

¹A machine learning technology consisting of two models, a generator and discriminator. The two neural network models train simultaneously, where the generator essentially tries to reproduce the training data, whilst this output is scored by the discriminator.

aim for in terms of ML integration within architecture. A more modest first step where a purpose built model could start to aid to explore a variety to a specific design issue would be a more feasible, and possible also more useful approach.

2.2 Previous explorations

During two recent projects and a thesis completed as part of the MArch Architecture programme at the Bartlett School of Architecture, UCL, I have made efforts to algorithmically define an abstract design of a building. These projects both took an object at the core of the functionality of the building programme, leading to a varying set of proposals, depending on the organisation of the objects.

This method is related to that of L-systems, and shape grammar [Gips & Stiny, 1971]. The aspect where these projects tried to break with those approaches is that the parameters of the generative rules were informed by contextual data, thus trying to achieve a more responsive generative approach, primarily focussed on spatial organisation instead of parametric form finding.

2.2.1 Design project 1: University libraries

The first project aimed to design a set of dynamic university libraries, where the spatial configuration of the library is informed by the way in which the collection of books is organised. In turn this organisation is dependent on the way in which the students and staff interact with the collection. From the way in which the books are sorted, and their internal relationship to each other a meandering shelf organisation was derived, from which walkways and stairs were derived. Depending on the existing structures within the library, this path was reconfigured to fit within those constraints. The result is a design concept that always follows the same principles, and should thus be regarded as a singular design solution, but the instance of the solution depends on the way in which the users interact with the building.

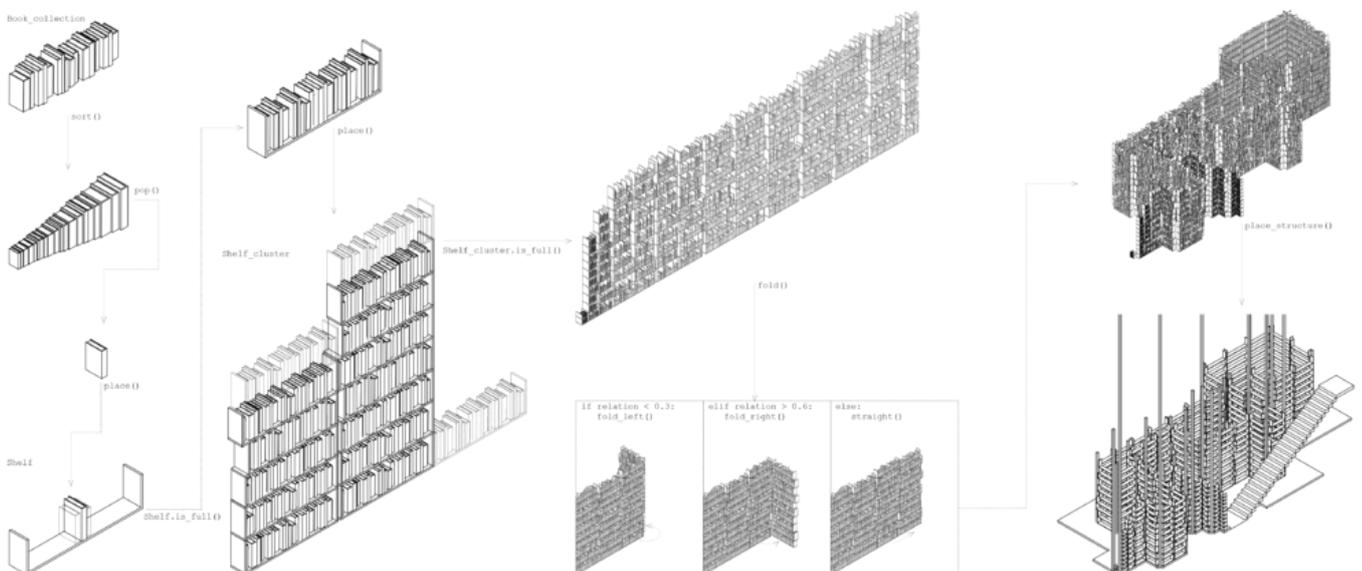


Figure 3. Generative process behind library bookshelves and walkways

Figure 4. Various library outputs depending on different sorting criteria and existing context

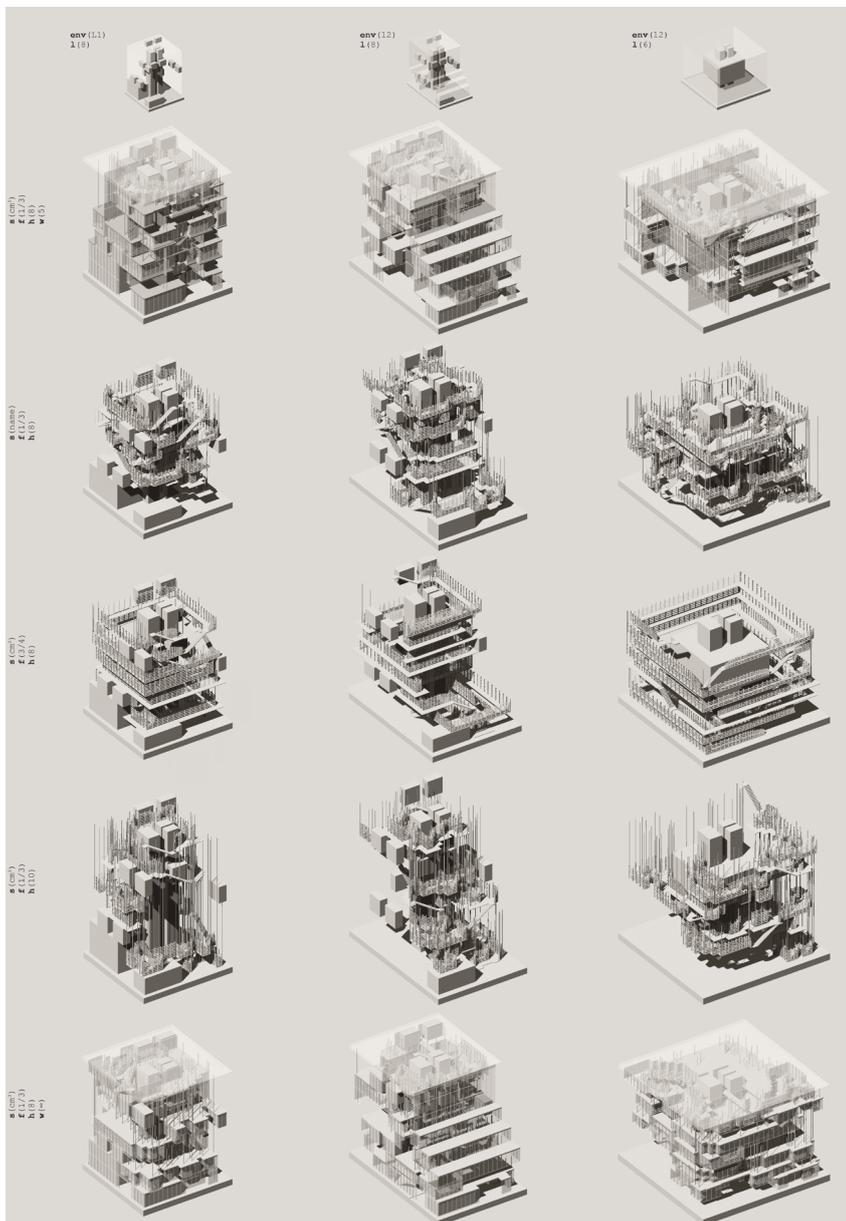
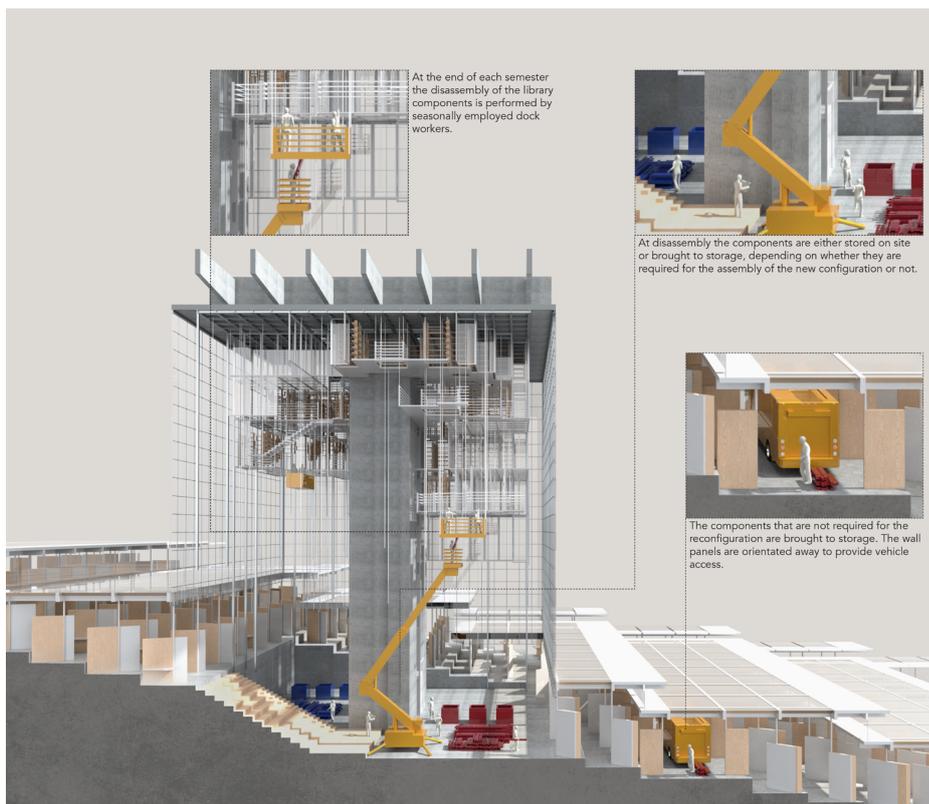


Figure 5. Assembly at the end of a semester after which the walkways and bookshelves are adjusted to the behaviour of the inhabitants



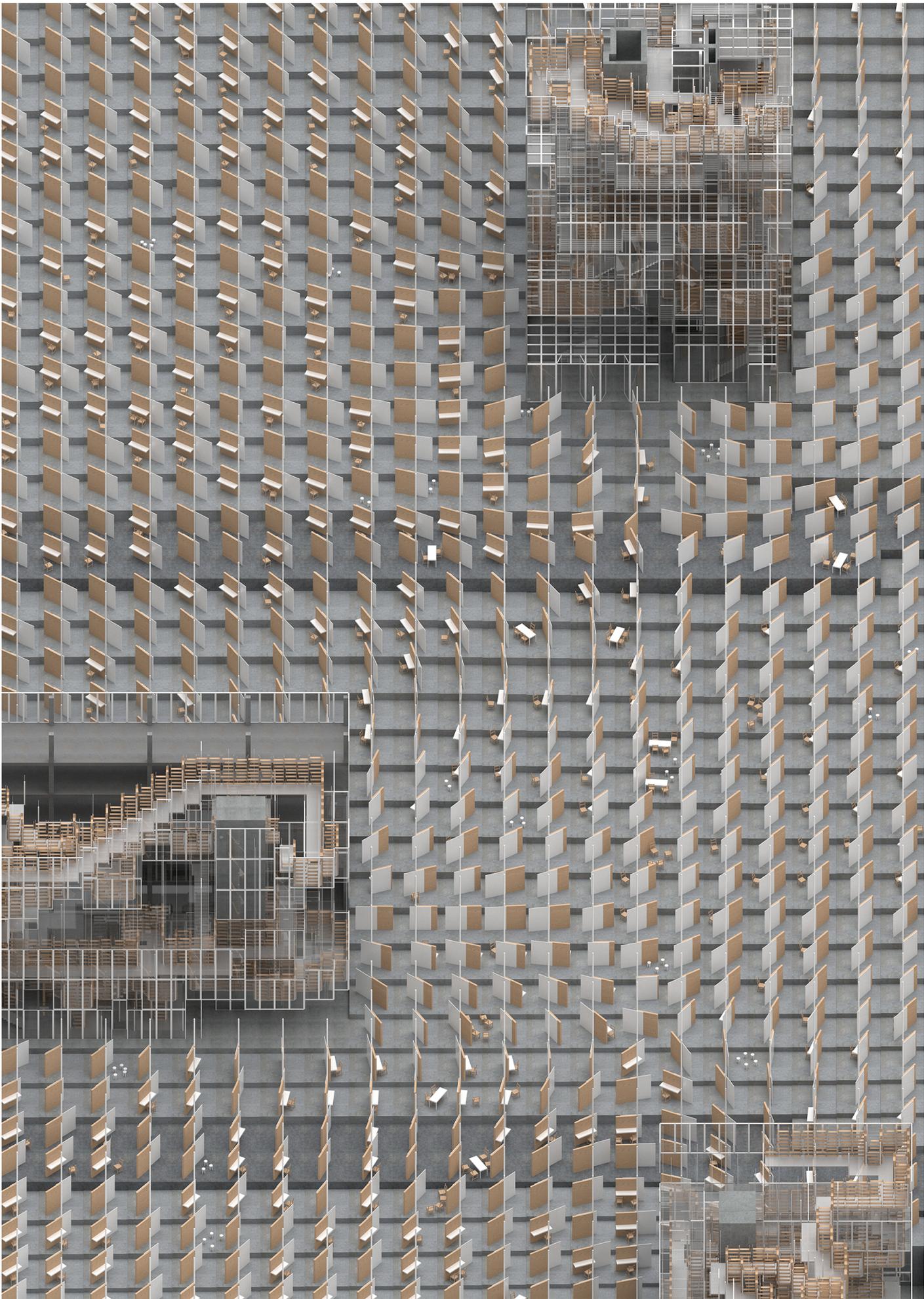


Figure 6. A collection of libraries are situated within a larger dynamic study landscape which follows a vector field.

2.2.2 Design project 2: Art galleries in Venice

The second project revolves around the definition of an art gallery dependent on the collection it houses. The art collection is selected based on the visual qualities of the potential site of the gallery. A similarity graph of the collection is established through image classification models determining the n -nearest neighbours for each art piece within the collection. From this the minimum spanning tree is taken which informs a division of the art into a set of spaces. These spaces are then configured according to the structure of the MST. After this the circulation that is required to make all spaces accessible is generated through the use of visibility graphs. Finally the walls and windows are placed following a ruleset related to views outwards of the building to the relevant parts of the surrounding context.

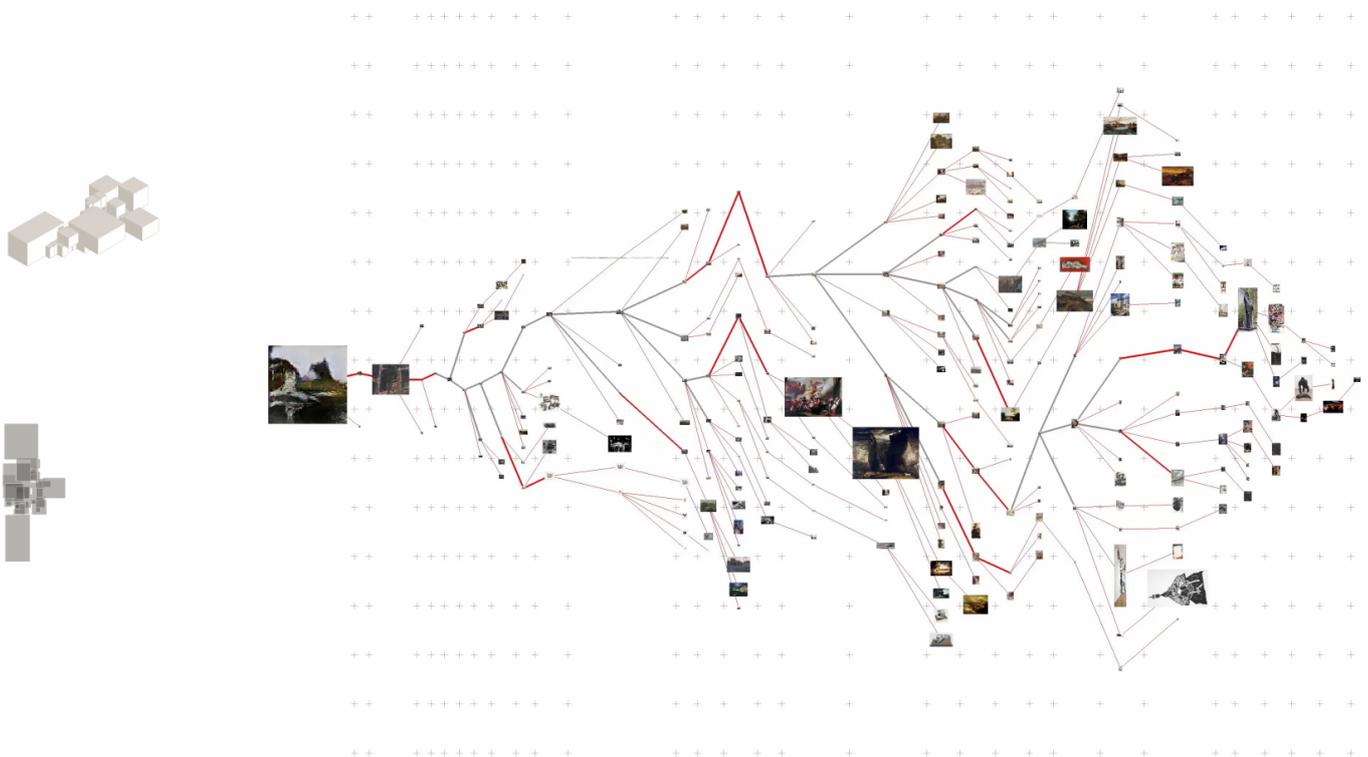


Figure 7. MST of an art gallery collection based on image similarity. The resulting massing of the subdivision of spaces based on main branches is seen on the left.

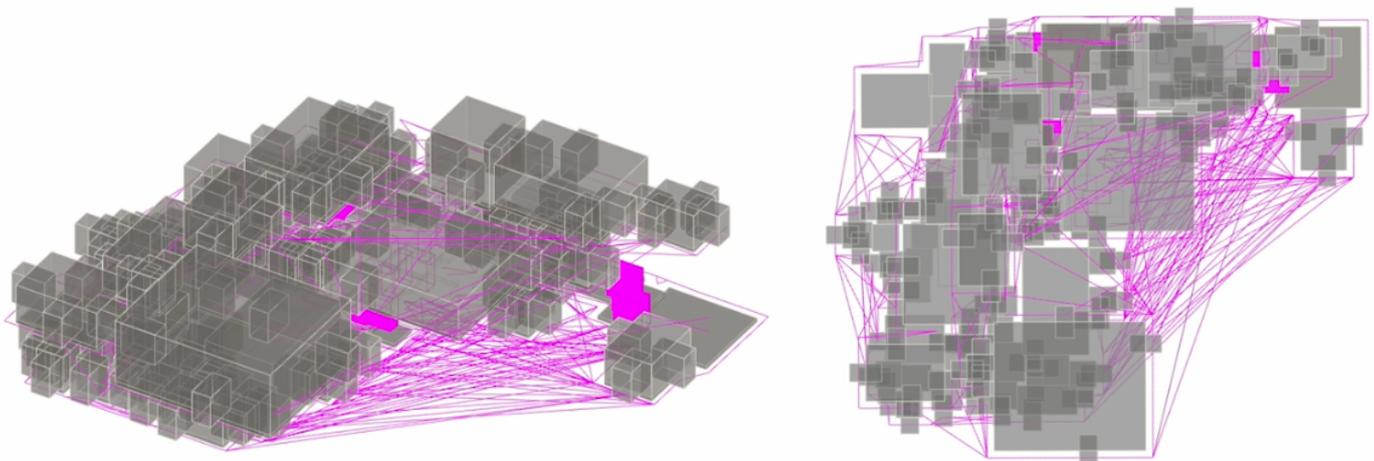


Figure 8. Visibility graph in order to find a valid circulation through the gallery.

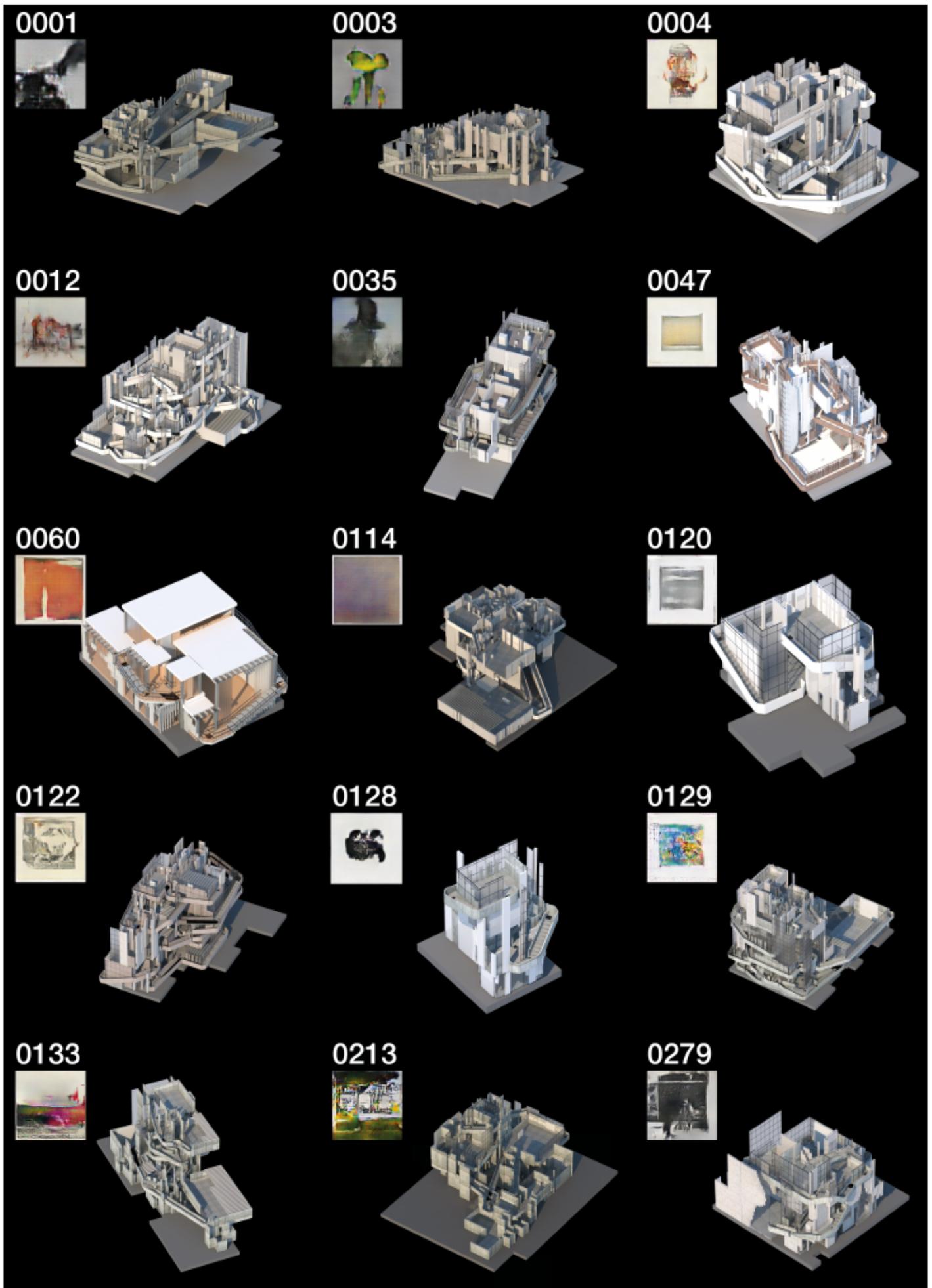


Figure 9. Variety of generated gallery proposals.



Figure 10. Collection of galleries in context.

2.2.3 Thesis: Towards a new architectural entity

In my architecture thesis I explored the way in which the technology of recommender systems could lead to a more responsive way of architectural design. The overall argument is based on how recommender systems implementations in music streaming platforms have contributed to the transition from the album to the playlist as the dominant listening format. From a design standpoint this is interesting since the old entity - the album, composed by the artist themselves, is broken down to its axiomatic components - songs, and reassembled to form a personalised new entity - the playlist, optimised to a very specific situation: the listening behaviour of the user. In an architectural context something similar to that is only possible if the combinatory possibilities of axiomatic architectural components are defined first. For songs these are fairly straightforward, since a playlist is in essence just a list, but an architectural project is inherently three dimensional, and its elements do not belong to one category either.

2.2.4 Observed limitations

The issue with these attempts at integrating computational methods within an architectural design process was that the algorithmic approach remained sequential in nature, since every next step within the generative process relied on the one before. This was inevitable due to the way in which the process was defined and the lack of overview of the building requirements as a whole. This meant that finding a valid circulation for a set of spaces was sometimes impossible due to the configuration of the spaces, leading to an invalid output. This is due to the structure of the generative programme, where the algorithm of the circulation did not contain a feedback loop to that of the configuration of spaces. This is something that could be introduced, but that would mean that the finding of solutions is always bound by a loop between distinct entities, rather than an approach where the different elements of the problem can be evaluated simultaneously to arrive to a solution that takes those factors into account in unison. For this purpose a way to connect the geometric definitions to the functional aspects of the project would offer a means of making the generative process less limited, and more resembling to a regular design approach.

Additionally this type of approach is limited due to its specificity. For each project the relevant data, relations, and restrictive elements have to be redefined and formatted to fit that particular project. This makes it a very time consuming way of designing. To this end a universal way of defining the relation between architectural concepts to geometrical definitions is proposed in the next two chapters in the form of a syntax and semantics. Through this a more abstracted way of exploring multiple instances of one design could be established, not unlike the way in which software design patterns offer a way to formulate abstracted solutions to object oriented programming challenges.

3.

Syntax

Here a basic Syntax for relational spatial configurations within architectural design and the combination of physical elements that synthesise these configurations is given. This is intended as a starting point for the development of a formal framework of building design.

3.1 Categories

First the different categories will be listed, with their respective meta variables, used to refer to elements of the respective categories in the following grammar rules.

The first part of the syntax is primarily concerned with spatial configurations and relations.

- P will range over projects **Proj**
- M will range over models **Mod**
- S will range over sites **Site**
- Φ will range over sets of project function declarations **Funcs**
- ϕ will range over the project function declarations **Func**
- π will range over statements **Stat**
- A will range over lists of arguments **Args**
- α will range over arguments **Arg**
- B will range over sets of buildings **Blds**
- β will range over buildings **Bld**
- Σ will range over sets of spaces **Spacs**
- σ will range over spaces **Spac**
- Δ will range over sets of domains **Doms**
- δ will range over domains **Dom**
- F will range over sets of function invocations **Fns**
- f will range over function invocations **Fn**

The second part of the syntax defines the categories required to define concrete spatial information. As well as more general utility categories such as variables and rational numbers.

- O will range over sets of object declarations **Objs**
- o will range over objects **Obj**
- G will range over sets of geometries **Geoms**
- g will range over geometries **Geom**
- V will range over voids **Void**
- X will range over lists of variables **Vars**
- x will range over variables **Var**
- n will range over extended rational numbers **Num**

The smallest variables correspond to the building elements listed by Rem Koolhaas in *Elements of Architecture* [2014]. In this work reduces architectural components to 15 axiomatic organisational elements. These are separately analysed in great detail, however only in terms of individual development, shapes and characteristics. The combination of the elements is left out, thus providing a study of each architectural element devoid of the architectural composition.

These elements are chosen to be the smallest resolution defined within the syntax since the interest is in the composition of a building, not necessarily its physical construction, which is why material elements such as columns, bricks, etc. are not taken into consideration. The physical qualities of these elements are represented by the abstract Geometry object, providing the possibility to assign coordinates to element instances.

- H will range over the set of building elements **Elems**
- η will range over building elements **Elem**
- λ will range over floors **Flr**
- ς will range over stairs **Str**
- w will range over walls **Wall**
- c will range over facades **Fac**
- r will range over roofs **Rfs**
- ω will range over windows **Wns**
- d will range over doors **Drs**
- e will range over elevators **Elv**
- ρ will range over ramps **Rmp**

In all cases where a category is specified as a set the elements are un-ordered. In the case of X and A the order is relevant, hence specified as a list.

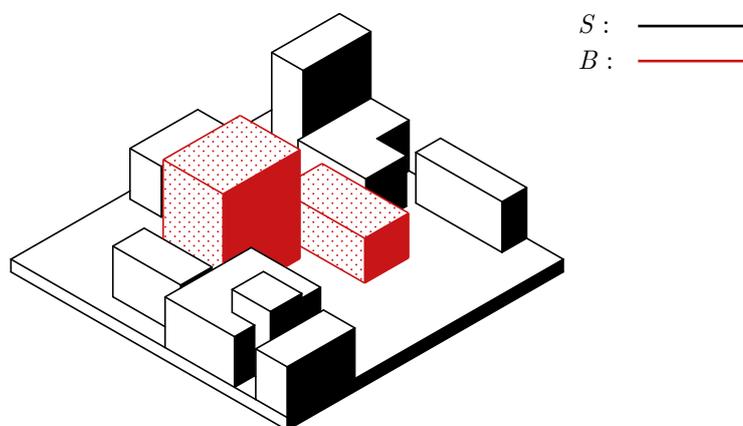
3.2 Grammar

In this section the grammar of the syntax is defined through a set of rules, one for each of the categories listed previously. The grammar follows a top down approach, where the larger categories are broken down into their subsequent parts.

PROJECT

The highest level variable is the architectural project as a whole. Within this syntax a project is defined by a site, a set of function declarations, a set of buildings and a set of function invocations that apply to the overall project. The syntax of **Proj**:

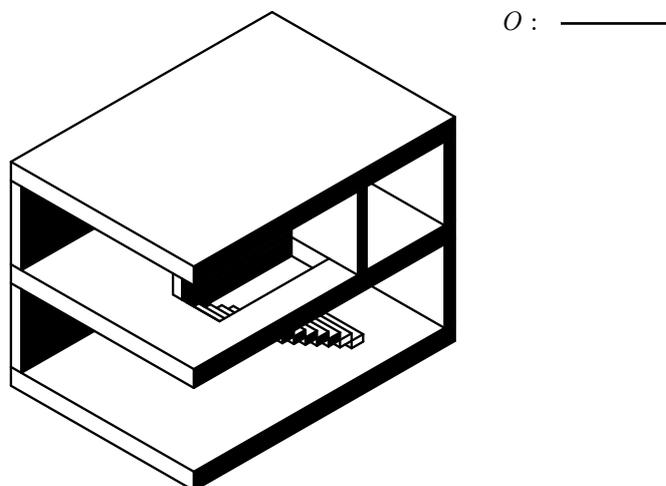
$$P ::= S \Phi B F$$



MODEL

The fundamental variable that can be evaluated is the model. This represents a raw CAD model only containing the definition of combinations of geometric objects without a notion of what these strive to represent in an architectural context. The syntax of **Mod**:

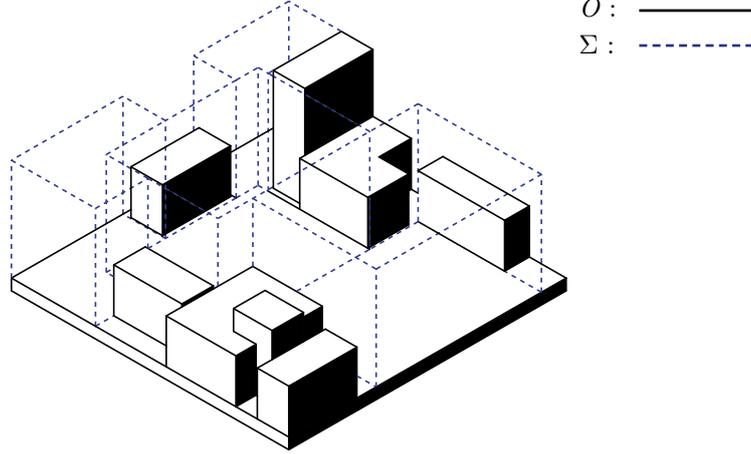
$$M ::= O$$



SITE

The site within the project is declared by a name and a set of objects and spaces. These spaces have to have a specific geometry as a domain, as the site is treated as a constant entity within the evaluation of the model. The syntax of the **Site**:

$$S ::= \varepsilon | \text{site } x(\text{objs}(O), \text{spacs}(\Sigma))$$



SET OF FUNCTION DECLARATIONS

Syntax for a set of function declarations **Funcs**:

$$\Phi ::= \varepsilon | \phi, \Phi$$

FUNCTION DECLARATION

A function declaration consists of a name, followed by a list of variables that act as abstract arguments. The body of the function consists of a statement over the set of previously defined variables. Syntax for a project function declaration **Func**:

$$\phi ::= \text{func } x(X)\{\pi\}$$

STATEMENT

The statements facilitate the formulation of the desired functional aspects of the project. The syntax for predicate statements **Stat**:

$$\begin{aligned} \pi ::= & \text{true} | \text{false} | x | f | \pi \wedge \pi' | \pi \vee \pi' | \pi \rightarrow \pi' | \pi \leftrightarrow \pi' | \neg \pi | x \in D | \\ & x = x | x || x' | x \# x' | x _ x' | x \bar{x}' | x \sim x' | x \approx x' | \exists x \in D[\pi] | \forall x \in D[\pi] \\ & \mathcal{R}[x] | \mathcal{G}[x] \end{aligned}$$

Where D is used to describe a domain, given by a list of buildings, spaces, elements, geometries, objects, a mixed set of those, \mathbb{R} , \mathbb{R}^2 or \mathbb{R}^3 . \mathcal{R} and \mathcal{G} are semantic functions retrieving the real number domain of a geometry and a geometry of an argument respectively, formally specified in the semantics section of this thesis.

Additional to the regular predicate logic symbols, additional symbols are used concerning spatial relationships:

- $||$: adjacency - the geometric domains of two arguments share a vertical border
- $\#$: intersects - the geometric domains of two arguments overlap more than just their borders
- $_$: supported - the geometric domain of the first argument is supported by that of the other
- $\overline{_}$: covered - the geometric domain of the first argument is covered by that of the other
- \sim : connected - the geometric domains of two arguments share a border
- \approx : chain connected - within the larger set of arguments there is an unbroken set of connected geometric domains from one argument to the other

LIST OF ARGUMENTS

Syntax for a list of arguments **Args**:

$$A ::= \varepsilon \mid \alpha, A$$

Technically this rule will result in an extra comma at the end of each list, but since this is an abstract syntax and not meant to be interpreted literally in terms of ASCII character it is chosen to leave it like this.

ARGUMENT

Syntax for argument **Arg**:

$$\alpha ::= \beta \mid \eta \mid \sigma \mid g \mid o$$

SET OF BUILDINGS

Syntax for a set of buildings **Blds**:

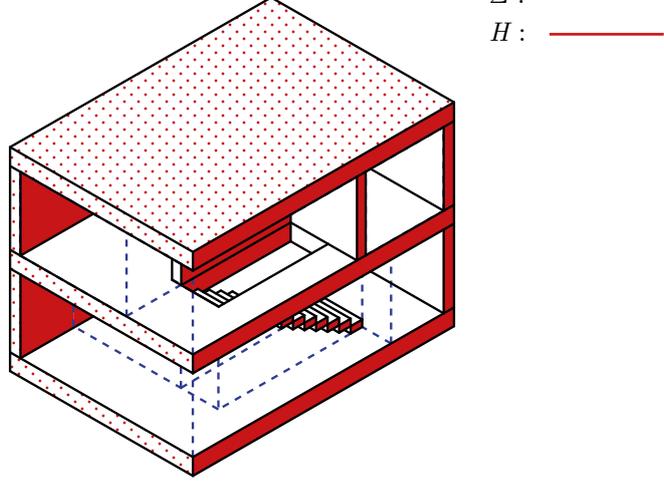
$$B ::= \varepsilon \mid \beta, B$$

BUILDING

A building consists of a variable for its name, set of spaces, a set of building

elements and a set of functions that apply to the internal organisation of the building. Syntax of **Bld**:

$$\beta ::= \text{bld } x(\text{spacs}\{\Sigma\}, \text{elems}\{H\}, \text{fns}\{F\})$$



SET OF SPACES

Syntax for a set of spaces **Spacs**:

$$\Sigma ::= \varepsilon \mid \sigma, \Sigma$$

SPACE

A space is declared by a name, and a domain restriction definition for its dimensions. Syntax for **Spac**:

$$\sigma ::= \text{spac } x(\delta)$$

SET OF DOMAINS

Syntax for a set of domains **Doms**:

$$\Delta ::= \varepsilon \mid \delta, \Delta$$

DOMAIN

An object domain is defined by an empty domain, posing no restriction, an area domain only binding the area, a volume domain, a three dimensional domain definition only specifying width, depth and height, all determined by an interval of real numbers. Alternatively a domain can be a set of specific geometries. Syntax for **Dom**:

$$\delta ::= \varepsilon \mid \text{m}2(n, n) \mid \text{m}3(n, n) \mid \text{w}(n, n), \text{d}(n, n), \text{h}(n, n) \mid G$$

 SET OF FUNCTION INVOCATIONS

Syntax for a set of function invocations **Fns**:

$$F ::= \varepsilon \mid f, F$$

 FUNCTION INVOCATION

A function invocation contains the name of a function declaration, followed by a list of arguments to which the predicate logic statement has to apply to. Syntax for the function calls **Fn**:

$$f ::= x(A)$$

 SET OF OBJECTS

Syntax for a set of objects **Objs**:

$$O ::= \varepsilon \mid o, O$$

 OBJECT

An object is a named set of geometries. Syntax for objects **Objs**:

$$o ::= \text{obj } x(G)$$

 VOID

Syntax for void **Void**:

$$V ::= G$$

 SET OF GEOMETRIES

Syntax for a set of geometries **Geoms**:

$$G ::= \varepsilon \mid g, G$$

 GEOMETRY

A geometry is constructed through a variety of geometric object declarations. For simplicity only rectilinear geometries are included, with the addition of a mesh, a collection of triangles, which can be used to approximate non defined geometries. In order to facilitate all geometries that can be defined in a CAD

software this would have to be expanded with curves, circles, spheres and nurbs.
 Syntax of **Geom**:

$$G ::= \varepsilon \mid \text{point}(x, y, z) \mid \text{line}(p_1, p_2) \mid \text{rect}(p_1, p_2, p_3) \mid \\ \text{box}(r, (x, y, z)) \mid \text{tri}(p_1, p_2, p_3) \mid \text{mesh}(t_1, \dots, t_n)$$

where x, y, z are **Num**, p is a **point**, r is a **rect** and t is a **tri**.

VARIABLE

The variables can be any letter and number sequence, here illustrated by a regular expression. Syntax for variables **Var**:

$$x ::= (a\dots z \mid A\dots Z \mid 0\dots 9)^*$$

REAL NUMBERS

Syntax for extended real numbers **Num**:

$$n ::= \mathbb{R} \cup \{-\infty, +\infty\}$$

SET OF BUILDING ELEMENTS

Syntax for a set of building elements **Elems**:

$$H ::= \varepsilon \mid \eta, H$$

BUILDING ELEMENT

Syntax for building elements **Elem**:

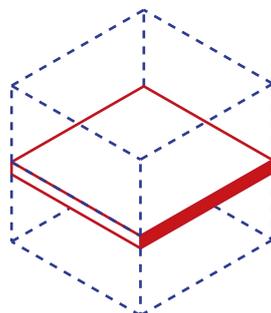
$$\eta ::= \lambda \mid \varsigma \mid w \mid c \mid r \mid \omega \mid d \mid e \mid \rho$$

All building elements consist of both a relational, and a geometrical aspect. The relational aspect is made up of a selection of spaces to which the element relates. The geometrical aspect is a domain. If the element is a general definition, the geometrical domain is merely an indication of its size, and the spaces that are referred to determine the role the element has within the building specification. If the domain and the spaces are specific instances bound by a instances of geometries instead of size intervals the element definition should conform to the resulting geometrical restrictions to be valid. This is formally defined in the semantics section of this thesis.

FLOOR

A floor is defined by the spaces beneath it and the ones above it, and geometrically bound by a domain. Syntax for floors **Flr**:

$$\lambda ::= \text{flr}(\Sigma, \Sigma')\{\delta\}$$

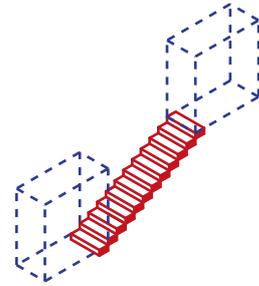


Σ : - - - - -
 λ : _ _ _ _ _

STAIRS

Stairs are defined by the lower space and upper space it connects, and geometrically bound by a domain. Syntax for stairs **Str**:

$$\varsigma ::= \text{str}(\sigma, \sigma')\{\delta\}$$

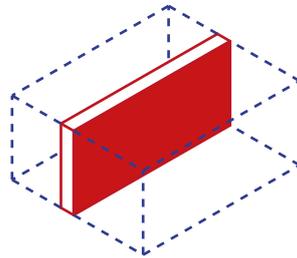


$$\begin{array}{l} \Sigma : \text{---} \\ \varsigma : \text{---} \end{array}$$

WALL

Walls are defined by the spaces it is adjacent to either side, and geometrically bound by a domain. Syntax for walls **Wall**:

$$w ::= \text{wall}(\Sigma, \Sigma')\{\delta\}$$

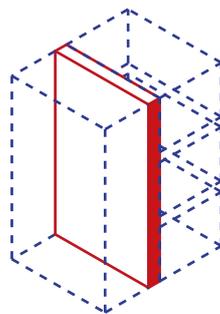


$$\begin{array}{l} \Sigma : \text{---} \\ w : \text{---} \end{array}$$

FACADE

Facades are defined by the indoor it is adjacent to the inner side, and geometrically bound by a domain. Syntax for facades **Fac**:

$$c ::= \text{fac}(\Sigma)\{\delta\}$$

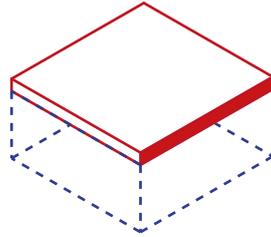


$$\begin{array}{l} \Sigma : \text{---} \\ c : \text{---} \end{array}$$

ROOF

Roofs are defined by the spaces below it, and geometrically bound by a domain. Syntax for roofs **Rfs**:

$$r ::= \text{rf}(\Sigma)\{\delta\}$$

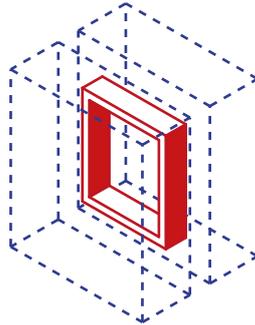


Σ : - - - - -
 r : —————

WINDOW

Windows are defined by the indoor space it is located in, and geometrically bound by a domain. Syntax for windows **Wns**:

$$\omega ::= \text{wn}(\sigma)\{\delta\}$$

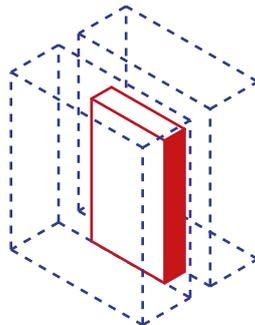


Σ : - - - - -
 ω : —————

DOOR

Doors are defined by the two spaces it connects, and geometrically bound by a domain. Syntax for doors **Drs**:

$$d ::= \text{dr}(\sigma, \sigma')\{\delta\}$$

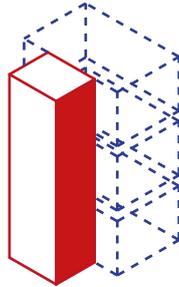


Σ : - - - - -
 d : —————

ELEVATOR

Elevators are defined by the spaces it connects, and geometrically bound by a domain. Syntax for elevators **Elv**:

$$e ::= \text{elv}(\Sigma)\{\delta\}$$

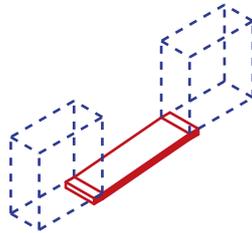


$$\begin{array}{l} \Sigma : \text{---} \\ e : \text{---} \end{array}$$

RAMPS

Ramps are defined by the lower space and upper space it connects, and geometrically bound by a domain. Syntax for ramps **Rmp**:

$$r ::= \text{rmp}(\sigma, \sigma')\{\delta\}$$



$$\begin{array}{l} \Sigma : \text{---} \\ r : \text{---} \end{array}$$

4.

Semantics

4.1 Auxiliary Functions

A set of functions is defined for retrieving specific components of different categories.

A function is defined to retrieve the geometry component bound to a specific variable. The same function can operate on different categories, with a universal output of a set geometries. Semantics of \mathcal{G} :

For a space, the function \mathcal{G} can only be applied if the domain is a set of specific geometries.

$$\mathcal{G}[\text{spac } x(G)] = G$$

$$\mathcal{G}[\sigma, \Sigma] = \mathcal{G}[\sigma] \cup \mathcal{G}[\Sigma]$$

$$\mathcal{G}[\text{obj } x(G)] = G$$

$$\mathcal{G}[o, O] = \mathcal{G}[o] \cup \mathcal{G}[O]$$

Similarly to spaces, the function can only be applied to a building element if the domain is a set of specific geometries. In that case it returns this set of geometries. Since all elements have a domain this is here not written out in full. The signature:

$$\mathcal{G}[\eta] = G$$

Following this pattern:

$$\mathcal{G}[\text{elemType}(args)\{G\}] = G$$

$$\mathcal{G}[\eta, H] = \mathcal{G}[\eta] \cup \mathcal{G}[H]$$

$$\mathcal{G}[\text{bld } x(\text{spacs}\{\Sigma\}, \text{elems}\{H\}, \text{fns}\{F\})] = \mathcal{G}[\Sigma] \cup \mathcal{G}[H]$$

Another function is defined that breaks down a geometry further to \mathbb{R}^3 domains:

$$\mathcal{R}[\text{point}(n_1, n_2, n_3)] = [n_1, n_2, n_3]$$

$$\mathcal{R}[\text{line}(\text{point}(n_1, n_2, n_3), \text{point}(n'_1, n'_2, n'_3))] =$$

$$\{[n_1, n_2, n_3] + t[n'_1 - n_1, n'_2 - n_2, n'_3 - n_3] \mid 0 \leq t \leq 1\}$$

$$\begin{aligned} \mathcal{R}[\text{rect}(p_1, p_2, p_3)] &= \\ \{v + (t(\mathcal{R}[p_3] - \mathcal{R}[p_1])) \mid v \in \mathcal{R}[\text{line}(p_1, p_2)] \mid 0 \leq t \leq 1\} \end{aligned}$$

$$\begin{aligned} \mathcal{R}[\text{box}(r, (n_1, n_2, n_3))] &= \{v + t[n_1, n_2, n_3] \mid v \in \mathcal{R}[r] \mid 0 \leq t \leq 1\} \\ \mathcal{R}[\text{tri}(p_1, p_2, p_3)] &= \\ \{t_1 \mathcal{R}[p_1] + t_2 \mathcal{R}[p_2] + t_3 \mathcal{R}[p_3] \mid 0 \leq t_1, t_2, t_3 \mid t_1 + t_2 + t_3 = 1\} \end{aligned}$$

$$\mathcal{R}[\text{mesh}(t_1, \dots, t_n)]^* = \bigcup_{i=1}^n \mathcal{R}[t_i]$$

* This definition is only accurate if it is an open mesh, in order to be able to handle closed meshes this definition would need to be expanded to include all internal points contained within the mesh.

Sometimes it is necessary to obtain an attribute of an instance of a certain category. For this a suffix function is defined followed by the signature of that attribute contained in the category instance. A few examples will be written out in full, but this principle can be applied to any category if it contains other categories within its definition:

For the building category there are three ways of applying this function:

$$\text{bld } x(\text{spacs}\{\Sigma\}, \text{elems}\{H\}, \text{fns}\{F\}).\text{spacs} = \Sigma$$

$$\text{bld } x(\text{spacs}\{\Sigma\}, \text{elems}\{H\}, \text{fns}\{F\}).\text{elems} = H$$

$$\text{bld } x(\text{spacs}\{\Sigma\}, \text{elems}\{H\}, \text{fns}\{F\}).\text{funcs} = F$$

Additional there is the special function `.name`:

$$\text{bld } x(\text{spacs}\{\Sigma\}, \text{elems}\{H\}, \text{fns}\{F\}).\text{name} = x$$

A set of elements can contain a mix of all element types, as defined by the grammar rule:

$$\eta ::= \lambda \mid \varsigma \mid w \mid c \mid r \mid \omega \mid d \mid e \mid \rho$$

Because of this another set of functions is declared that yield a subset of those elements of one specific type. For example in the case for **Flr**:

$$\frac{H' \text{ is the set of all floors contained in the set of mixed elements in } H}{H.\text{flr} = H'}$$

4.2 Semantic Functions

Additional to conventional predicate logic five symbols were added to express spatial configurations: adjacency, intersect, supported, covered, contact and chained contact. These are defined for any variable or object that has a geometry component, which corresponds to the permitted arguments for project functions.

For these it is necessary to define the sets of vertical and horizontal planes as well as the complete set of lines and planes in \mathbb{R}^3 :

Vertical planes:

$$vPlanes = \{\{r = p + s[0, 0, 1] + t[x, y, 0] \mid s, t \in \mathbb{R} \mid x, y \in \mathbb{R} \mid p \in \mathbb{R}^3\}$$

Horizontal planes:

$$hPlanes = \{\{p + s[0, 1, 0] + t[1, 0, 0] \mid s, t \in \mathbb{R} \mid p \in \mathbb{R}^3\}$$

Lines:

$$lines = \{\{[n_1, n_2, n_3] + t[n'_1 - n_1, n'_2 - n_2, n'_3 - n_3] \mid 0 \leq t \leq 1\} \mid n_1, n_2, n_3, n'_1, n'_2, n'_3 \in \mathbb{R}^3\}$$

Planes:

$$planes = \{r = p + sv_1 + tv_2 \mid s, t \in \mathbb{R} \mid p, v_1, v_2 \in \mathbb{R}^3 \mid v_1, v_2 \text{ are linearly independent}\}$$

An abbreviation is defined for the following pair of predicates over a set $N \subset \mathbb{R}^3$ since these have to hold for all spatial relations:

$$valid(N) \iff N \neq \emptyset \wedge \neg N \in lines$$

Adjacency $||$:

$$[adj.] \frac{\alpha, \alpha' \in A \quad \mathcal{R}[\mathcal{G}[\alpha]] \cap \mathcal{R}[\mathcal{G}[\alpha']] = N \quad valid(N) \quad \exists P \in vPlanes[N \in P]}{A \vDash \alpha || \alpha'}$$

Intersect $\#$:

$$[int.] \frac{\alpha, \alpha' \in A \quad \mathcal{R}[\mathcal{G}[\alpha]] \cap \mathcal{R}[\mathcal{G}[\alpha']] = N \quad valid(N) \quad \nexists P \in planes[N = P]}{A \vDash \alpha \# \alpha'}$$

Supported $_:$

$$[sup.] \frac{\alpha, \alpha' \in A \quad \mathcal{R}[\mathcal{G}[\alpha]] \cap \mathcal{R}[\mathcal{G}[\alpha']] = N \quad valid(N) \quad \exists P \subset hPlanes[N \in P] \quad \forall [n_1, n_2, n_3] \in \mathcal{R}[\mathcal{G}[\alpha]] \quad \forall [n_4, n_5, n_6] \in \mathcal{R}[\mathcal{G}[\alpha']] \quad [n_6 \leq n_3]}{A \vDash \alpha _ \alpha'}$$

Covered $\bar{_}$:

$$[cov.] \frac{\alpha, \alpha' \in A \quad \mathcal{R}[\mathcal{G}[\alpha]] \cap \mathcal{R}[\mathcal{G}[\alpha']] = N \quad valid(N) \quad \exists P \subset hPlanes[N \in P] \quad \forall [n_1, n_2, n_3] \in \mathcal{R}[\mathcal{G}[\alpha]] \quad \forall [n_4, n_5, n_6] \in \mathcal{R}[\mathcal{G}[\alpha']] \quad [n_3 \leq n_6]}{A \vDash \alpha \bar{_} \alpha'}$$

Contact \sim :

$$[\text{con. a}] \quad \frac{\alpha, \alpha' \in A \quad \alpha || \alpha'}{A \models \alpha \sim \alpha'}$$

$$[\text{con. s}] \quad \frac{\alpha, \alpha' \in A \quad \alpha _ \alpha'}{A \models \alpha \sim \alpha'}$$

$$[\text{con. c}] \quad \frac{\alpha, \alpha' \in A \quad \alpha \bar{_} \alpha'}{A \models \alpha \sim \alpha'}$$

Chained contact \approx :

$$[\text{c. con. a}] \quad \frac{\alpha, \alpha' \in A \quad \alpha \sim \alpha'}{A \models \alpha \approx \alpha'}$$

$$[\text{c. con. b}] \quad \frac{\alpha, \alpha' \in A \quad \exists \alpha'' \in A [\alpha \sim \alpha'' \wedge \alpha'' \approx \alpha']}{A \models \alpha \approx \alpha'}$$

All other symbols used in the predicate statements are evaluated as expected, and will not be written out in full.

4.3 Rules

This section addresses the semantic rules on how one categories can be derived from other categories.

The judgements for these rules are as follows:

- $M \models P$ for a model **Mod** to a project **Proj**
- $O \models S\Phi BF$ for a set of objects **Objs** to a site **Site**, set of function declarations **Funcs**, buildings **Blds** and function invocations **Fns**
- $\Phi SO \models B$ for a set of function declarations **Funcs**, a Site **Site**, and a set of objects **Objs** to a set of buildings **Blds**
- $\Phi SO \models \beta$ for a set of function declarations **Funcs**, a Site **Site**, and a set of objects **Objs** to a building **Bld**
- $SO \models \Sigma H$ for a site **Site** and a set of objects **Objs** to a set of spaces **Spacs** and a set of elements **Elms**
- $SO \models \Sigma$ for a site **Site** and a set of objects **Objs** to a set of spaces **Spacs**
- $SO \models V$ for a site **Site** and a set of objects **Objs** to a void **Void**
- $SO \models g$ for a site **Site** and a set of objects **Objs** to a geometry **Geom**
- $V \models \sigma$ for a void **Void** to a space **Spac**
- $SO\Sigma \models H$ for a site **Site** and a set of objects **Objs** to a set of elements **Elms**
- $SO\Sigma \models \eta$ for a site **Site** and a set of objects **Objs** to an element **Elem** *

- $\Phi A \vDash F$ for a set of function declarations **Funcs** and arguments **Args** to a set of function invocations **Fns**
- $\phi A \vDash f$ for a set of function declarations **Funcs** and arguments **Args** to a function invocation **Fn**

* The rules for element derivation are gathered in section 4.5, since these all follow the same general pattern, but have different rules depending on the specific type of element.

The rules for the judgements listed above are defined using proof trees. This means that the judgement at the bottom of the proof tree is a valid only if all judgements and statements above are valid as well.

The bottom of the proof tree is the derivation of a project declaration from a model.

$$[\text{model}] \quad \frac{P = S\Phi BF \quad M = O \quad O \vDash S\Phi BF}{M \vDash P}$$

In order to verify the validity of the model in regards to the project definition objects of the model have to infer the buildings and functions that act on the project, within the given site. This means that the site and objects together with the function declarations of the project can be evaluated to a set of buildings. This set of buildings has to contain the requirements as defined in the project description. In turn this set of buildings in the site context has to meet the functions that are defined in regards to the project as a whole.

$$[\text{objects}] \quad \frac{\Phi SO \vDash B' \quad B \vDash B' \quad \Phi S \cup B' \vDash F}{O \vDash S\Phi BF}$$

Based on the function declarations of the project definition and the model the buildings are derived.

$$[\text{der. blds}] \quad \frac{\Phi SO' \vDash \beta \quad \Phi SO'' \vDash B' \quad O' \cup O'' = O \quad O' \cap O'' = \emptyset \quad \{\beta\} \cup B' = B}{\Phi SO \vDash B}$$

A singular building is derived through the evaluation of the model to spaces and building elements. After which those are evaluated to test if the function requirements are met.

$$[\text{const. bld}] \quad \frac{SO \vDash \Sigma H \quad \Phi \Sigma \cup H \vDash F}{\Phi SO \vDash \text{bld } x(\text{spacs}\{\Sigma\}, \text{elems}\{H\}, \text{fns}\{F\})}$$

Site and objects evaluate to spaces and building elements by separately evaluating the objects to elements and in turn the elements together with the site to spaces.

$$[\text{der. spac, elem}] \quad \frac{SO \vDash \Sigma \quad SO \Sigma \vDash H}{SO \vDash \Sigma H}$$

Construct spaces from site and set objects.

$$[\text{constr. spaces}] \quad \frac{SO \models V \quad V \models \sigma \quad SO \models \Sigma' \quad \{\sigma\} \cup \Sigma' = \Sigma}{SO \models \Sigma}$$

Construct Voids from site and set of objects.

$$[\text{constr. voids}] \quad \frac{SO \models g \quad SO \models V' \quad \{v\} \cup V' = V}{SO \models V}$$

Construct Void from site and set of objects.

$$[\text{constr. void}] \quad \frac{\forall g' \in \mathcal{G}[[S.\text{objs}]][\neg g' \# g] \quad \forall o \in \mathcal{G}[[O]][\neg o \# g] \quad \exists O' \subset O[g_O']}{SO \models g}$$

Construct space from voids.

$$[\text{constr. space}] \quad \frac{V = \delta}{V \models \text{spac } x(\delta)}$$

Derive elements from objects.

$$[\text{der. elems}] \quad \frac{SO'\Sigma \models \eta \quad SO''\Sigma \models H' \quad \{\eta\} \cup H' = H \quad O' \cup O'' = O \quad O' \cap O'' = \emptyset}{SO\Sigma \models H}$$

Verify functions of provided arguments and function declarations.

$$[\text{ver. funcs}] \quad \frac{\phi A \models f' \quad \Phi A \models F'' \quad F \models F' \quad \{f'\} \cup F'' = F' \quad \phi \in \Phi}{\Phi A \models F}$$

Verify function is met by provided arguments and function declaration.

$$[\text{ver. func}] \quad \frac{\phi A' \models f \quad A' \subset A}{\phi A \models f}$$

Evaluate function invocation.

$$[\text{eval. func}] \quad \frac{\models \pi[X := A]}{\text{func } x(X)\{\pi\} \quad A \models x(A)}$$

Where $\pi[X := A]$ means substituting the variables X in the predicate equation of the function declaration by the arguments A

4.4 Instance Statements

The following set of statements define how one variable of a category can be derived to a more specific instance of that same category:

First the different instance rules for domains **Dom**:
Instance of an empty domain can be any domain.

$$[\text{inst. dom. empty.}] \quad \frac{}{\varepsilon \vDash \delta}$$

Instances of m2 domains.

$$[\text{inst. dom. m2,m2}] \quad \frac{n_1 \leq n_3 \leq n_2 \quad n_1 \leq n_4 \leq n_2}{\text{m2}(n_1, n_2) \vDash \text{m2}(n_3, n_4)}$$

$$[\text{inst. dom. m2,xyz}] \quad \frac{n_1 \leq n_3 \times n_5 \leq n_2 \quad n_1 \leq n_4 \times n_6 \leq n_2}{\text{m2}(n_1, n_2) \vDash \text{x}(n_3, n_4), \text{y}(n_5, n_6), \text{z}(n_7, n_8)}$$

$$[\text{inst. dom. m2,G}] \quad \frac{\textit{The surface area of the groundplanes of } G \textit{ is within the domain of } \{n_1, n_2\}}{\text{m2}(n_1, n_2) \vDash G}$$

Instances of m3 domains.

$$[\text{inst. dom. m3,m3}] \quad \frac{n_1 \leq n_3 \leq n_2 \quad n_1 \leq n_4 \leq n_2}{\text{m3}(n_1, n_2) \vDash \text{m3}(n_3, n_4)}$$

$$[\text{inst. dom. m3,xyz}] \quad \frac{n_1 \leq n_3 \times n_5 \times n_7 \leq n_2 \quad n_1 \leq n_4 \times n_6 \times n_8 \leq n_2}{\text{m3}(n_1, n_2) \vDash \text{x}(n_3, n_4), \text{y}(n_5, n_6), \text{z}(n_7, n_8)}$$

$$[\text{inst. dom. m3,G}] \quad \frac{\textit{The cubic area of } G \textit{ is within the domain of } \{n_1, n_2\}}{\text{m3}(n_1, n_2) \vDash G}$$

Instances of wdh domains.

$$[\text{inst. dom. wdh}] \quad \frac{n_1 \leq n_7 \leq n_2 \quad n_1 \leq n_8 \leq n_2 \quad n_3 \leq n_9 \leq n_4 \quad n_3 \leq n_{10} \leq n_4 \quad n_5 \leq n_{11} \leq n_6 \quad n_5 \leq n_{12} \leq n_6}{\text{w}(n_1, n_2), \text{d}(n_3, n_4), \text{h}(n_5, n_6) \vDash \text{w}(n_7, n_8), \text{d}(n_9, n_{10}), \text{h}(n_{11}, n_{12})}$$

$$[\text{inst. dom. wdh, G}] \quad \frac{\textit{The width, depth, height of } G \textit{ fall within } \{n_1, n_2\}, \{n_3, n_4\}, \{n_5, n_6\} \textit{ respectively}}{\text{w}(n_1, n_2), \text{d}(n_3, n_4), \text{h}(n_5, n_6) \vDash G}$$

Instance of a space:

$$[\text{inst. spac.}] \quad \frac{\delta \vDash \delta'}{\text{spac } x(\delta) \vDash \text{spac } x(\delta')}$$

Instance of an element (not separately written out in full for all elements, but defined following the universal pattern of the element definition):

$$\text{[inst. elem.]} \quad \frac{\text{elemType1} = \text{elemType2} \quad \text{args} \vDash \text{args}' \quad \delta \vDash \delta'}{\text{elemType1}(\text{args})\{\delta\} \vDash \text{elemType2}(\text{args}')\{\delta'\}}$$

Instance of set of spaces:

$$\text{[inst. spacs.]} \quad \frac{\sigma \vDash \sigma' \quad \Sigma'' \vDash \Sigma''' \quad \{\sigma\} \cup \Sigma'' = \Sigma \quad \{\sigma'\} \cup \Sigma''' = \Sigma'}{\Sigma \vDash \Sigma'}$$

Instance of set of empty spaces:

$$\text{[inst. spacs.e]} \quad \frac{\Sigma = \emptyset}{\Sigma \vDash \Sigma'}$$

Instance of set of elements:

$$\text{[inst. elems.]} \quad \frac{\eta \vDash \eta' \quad H'' \vDash H''' \quad \{\eta\} \cup H'' = H \quad \{\eta'\} \cup H''' = H'}{H \vDash H'}$$

Instance of set of empty elements:

$$\text{[inst. elems.]} \quad \frac{H = \emptyset}{H \vDash H'}$$

Instance of list of arguments:

$$\text{[inst. args.]} \quad \frac{\alpha \vDash \alpha' \quad A \vDash A'}{\alpha, A \vDash \alpha, A'}$$

Instance of a building:

$$\text{[inst. bld.]} \quad \frac{\Sigma'' \subset \Sigma' \quad H'' \subset H' \quad \Sigma \vDash \Sigma'' \quad H \vDash H''}{\text{bld } x(\text{spacs}\{\Sigma\}, \text{elems}\{H\}, \text{fns}\{F\}) \vDash \text{bld } x(\text{spacs}\{\Sigma'\}, \text{elems}\{H'\}, \text{fns}\{F\})}$$

Instance of set of buildings:

$$\text{[inst. blds.]} \quad \frac{\beta \vDash \beta' \quad B'' \vDash B''' \quad \{\beta\} \cup B'' = B \quad \{\beta'\} \cup B''' = B'}{B \vDash B'}$$

$$\text{[inst. fn.]} \quad \frac{A \vDash A'}{x(A) \vDash x(A')}$$

$$\text{[inst. fns.]} \quad \frac{f \vDash f' \quad \{f\} \cup F'' = F \quad \{f'\} \cup F''' = F'}{F \vDash F'}$$

For any attribute of a category, it is true that the attribute is an instance of another object's attribute if the same is true for the parent object. As an example the rules for spaces and elements of a building are written out:

$$\text{[inst. sBlds.]} \quad \frac{B \vDash B'}{B.\text{spacs} \vDash B'.\text{spacs}}$$

$$\text{[inst. eBlds.]} \quad \frac{B \vDash B'}{B.\text{elems} \vDash B'.\text{elems}}$$

4.5 Building Element Statements

Construct floor from site, objects and spaces:

$$\text{[constr. flr]} \quad \frac{\delta = \mathcal{G}[[O]] \quad \Sigma' \subset \Sigma \quad \Sigma'' \subset \Sigma \quad \forall o \in O[\exists g \in \mathcal{G}[[\Sigma']][g_o]] \quad \forall o \in O[\exists g \in \mathcal{G}[[\Sigma'']][g^-o]]}{SO\Sigma \vDash \text{flr}(\Sigma', \Sigma'')\{\delta\}}$$

Construct stair from site, objects and spaces:

$$\text{[constr. str.]} \quad \frac{\delta = \mathcal{G}[[O]] \quad \sigma \in \Sigma \quad \sigma' \in \Sigma' \quad \forall o \in O[\forall o' \in O[O \vDash o \approx o']] \quad \exists o \in O[o^- \mathcal{G}[[\sigma]]] \quad \exists o \in O[o^- \mathcal{G}[[\sigma']]]}{SO\Sigma \vDash \text{str}(\sigma, \sigma')\{\delta\}}$$

Construct wall from site, objects and spaces:

$$\text{[constr. wall]} \quad \frac{\delta = \mathcal{G}[[O]] \quad \Sigma' \subset \Sigma \quad \Sigma'' \subset \Sigma \quad \forall o \in O[\exists g \in \mathcal{G}[[\Sigma']][g||o]] \quad \forall o \in O[\exists g \in \mathcal{G}[[\Sigma'']][g||o]]}{SO\Sigma \vDash \text{wall}(\Sigma', \Sigma'')\{\delta\}}$$

Construct wall from site, objects and spaces:

$$\text{[constr. fac.]} \quad \frac{\delta = \mathcal{G}[[O]] \quad \Sigma' \subset \Sigma \quad \Sigma'' \subset S \quad \forall o \in O[\exists g \in \mathcal{G}[[\Sigma']][g||o]] \quad \forall o \in O[\exists g \in \mathcal{G}[[\Sigma'']][g||o]]}{SO\Sigma \vDash \text{fac}(\Sigma', \Sigma'')\{\delta\}}$$

Construct roof from site, objects and spaces:

$$\text{[constr. rf.]} \quad \frac{\delta = \mathcal{G}[[O]] \quad \Sigma' \subset \Sigma \quad \forall o \in O[\exists g \in \mathcal{G}[[\Sigma']][g^-o]]}{SO\Sigma \vDash \text{rf}(\Sigma')\{\delta\}}$$

Construct window from site, objects and spaces:

$$\text{[constr. wnd.]} \quad \frac{\delta = \mathcal{G}[[O]] \quad \sigma \in \Sigma \quad \sigma' \in \Sigma \cup S \quad \forall o \in O[o||\mathcal{G}[[\sigma]] \wedge o||\mathcal{G}[[\sigma']]]}{SO\Sigma \vDash \text{wnd}(\sigma, \sigma')\{\delta\}}$$

Construct door from site, objects and spaces:

$$\text{[constr. dr.]} \quad \frac{\delta = \mathcal{G}[[O]] \quad \sigma \in \Sigma \quad \sigma' \in \Sigma \cup S \quad \forall o \in O[o || \mathcal{G}[[\sigma]] \wedge o || \mathcal{G}[[\sigma']]]}{SO\Sigma \models \text{dr}(\sigma, \sigma')\{\delta\}}$$

Construct elevator from site, objects and spaces:

$$\text{[constr. elv.]} \quad \frac{\delta = \mathcal{G}[[O]] \quad \Sigma' \subset \Sigma \quad \forall \sigma \in \Sigma' [\exists o \in O[o || \mathcal{G}[[\sigma]]]] \quad \forall o \in O [\forall o' \in O [O \models o \approx o']]}{SO\Sigma \models \text{elv}(\Sigma')\{\delta\}}$$

Construct ramp from site, objects and spaces:

$$\text{[constr. rmp.]} \quad \frac{\delta = \mathcal{G}[[O]] \quad \sigma \in \Sigma \quad \sigma' \in \Sigma' \quad \forall o \in O [\forall o' \in O [O \models o \approx o']] \quad \exists o \in O [o \bar{=} \mathcal{G}[[\sigma]]] \quad \exists o \in O [o \bar{=} \mathcal{G}[[\sigma']]]}{SO\Sigma \models \text{rmp}(\sigma, \sigma')\{\delta\}}$$

4.6 Empty Statements

Finally there is a set of statements for if the right side is an empty set:

$$\text{[empty bls.]} \quad \overline{\Phi S \emptyset \models \emptyset}$$

$$\text{[empty spacs.]} \quad \overline{SO \models \emptyset}$$

$$\text{[empty voids.]} \quad \overline{SO \models \emptyset}$$

$$\text{[empty elems.]} \quad \overline{S \emptyset \Sigma \models \emptyset}$$

$$\text{[empty fns.]} \quad \overline{\Phi A \models \emptyset}$$

5.

Examples

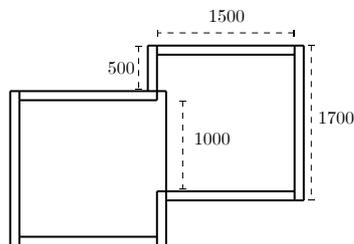
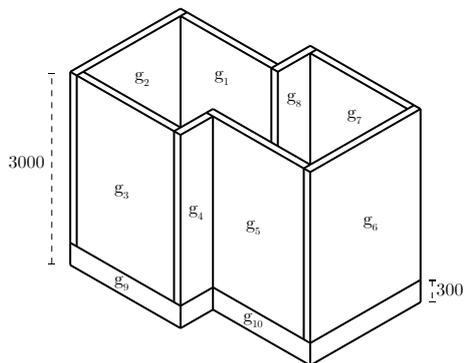
In this chapter 4 examples of project definitions will be given following the syntax and semantics defined in the previous two chapters. The first example is meant as a base case to show the way in which a project can be defined, using a rudimentary example. The second example attempts to capture one of Alexander's patterns. The third example is one of the galleries of the design project described in 2.2.2, following the basic algorithmic principles used for the design. The last example mostly serves as an illustration of the challenges of capturing a not necessarily clear definition of an architectural concept in a formal manner, in this case a courtyard.

5.1 Connected Cubes

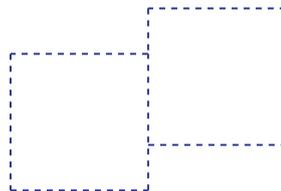
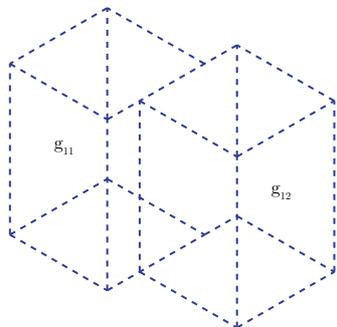
The base case example is that of two connected cubes, each between 1 to 2 metres in floor space. For this example the full proof tree is given in Appendix A, to show how the model given on the next page, can be shown to fit the project definition given below based on the defined syntax and semantics.

```
func noIsolatedSpaces( $B_1$ ){ $\forall s_1 \in B_1$ .spacs[ $\exists s_2 \in B_1$ .spacs[ $s_1 || s_2$ ]]}
bld connectedCubes(spacs{
  spac  $s_1$ (m2(10002, 20002)),
  spac  $s_2$ (m2(10002, 20002))},
  elems{}, funcs{})
noIsolatedSpaces(
  bld connectedCubes(spacs{
    spac  $s_1$ (m2(10002, 20002)),
    spac  $s_2$ (m2(10002, 20002))},
    elems{}, funcs{}))
```

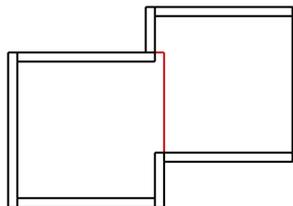
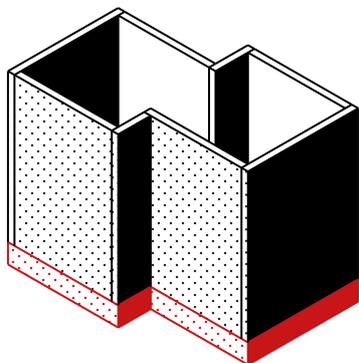
$M =$



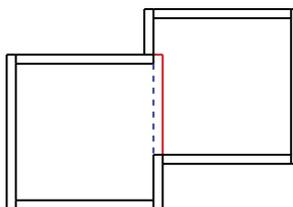
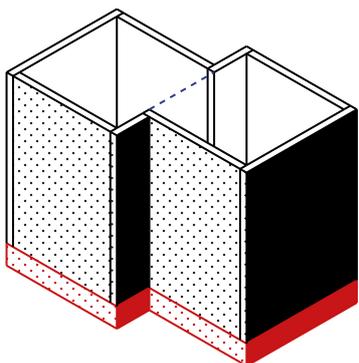
$V =$



$H =$



$P =$



5.2 C. Alexander pattern 101: Building Thoroughfare

The project description below represents a simplified interpretation of the pattern of a thoroughfare as described by Alexander in *A Pattern Language* [1977, p492]. The basic idea is that if there is no way to access the site from a space in the building, then it has to access the thoroughfare, which is connected to two external spaces. The dimensions of this thoroughfare space are bound to the dimensions as defined in the pattern description. The full description by Alexander can be found in Appendix B.

```

site location(objs(), spacs())
func access(B1, s1, s2) {
    s1 || s2 ∨
    ∃ e1 ∈ B1.elems.str ∪ B1.elems.dr ∪ B1.elems.elv ∪
    B1.elems.rmp [s1, s2 ∈ e1.spacs] }
func buildingThoroughfare(B1, S1, st) {
    st ∈ B1 ∧
    ∃ s1, s2 ∈ S1.spacs [¬(s1 = s2) ∧ access(B1, st, s1) ∧ access(B1, st, s2)] ∧
    ∃ e1 ∈ B1.elems [st — e1] ∧
    ∀ s3 ∈ B1.spacs [¬∃ s4 ∈ S1.spacs [access(B1, s3, s4)] → access(B1, st, s3)] }
bld publicBuildingComplex(
    spacs {spac indoorStreet(w(3353, 7000), d(7000, ∞), h(3657, 9000))},
    elems {}, fns {} )
buildingThoroughfare(Bpb, Ssite, Σspaces)

```

Abbreviations:

S_{site} = site location(objs(), spacs())

B_{pb} =

```

bld publicBuildingComplex(
    spacs {spac indoorStreet(w(3353, 7000), d(7000, ∞), h(3657, 9000))},
    elems {}, fns {} )

```

Σ_{spaces} = spac indoorStreet(w(3353, 7000), d(7000, ∞), h(3657, 9000))

5.3 Generated Art Gallery St. Mark's Square

The core concepts of the algorithmic design method for the galleries are the MST determined spaces and connections, together with that all spaces need to be accessible. Additionally views out to the context informed wall placements. These features have been included in the project definition given below.

```

site StMarksSquare(objs( $O_{site}$ ), spacs( $\Sigma_{site}$ ))

func access( $B, s_1, s_2$ ){
   $s_1 || s_2 \vee$ 
 $\exists e_1 \in B.\text{elems.str} \cup B.\text{elems.dr} \cup B_1.\text{elems.elv} \cup B.\text{elems.rmp}[s_1, s_2 \in e_1.\text{spacs}]$ 
}

func accessible( $B, s_1, s_2$ ){
   $\text{access}(s_1, s_2) \vee$ 
 $\exists s_3 \in B.\text{spacs}[\text{access}(s_1, s_3) \wedge \text{accessible}(s_3, s_2)]$ 
}

func fullyAccessible( $B, S$ ){
   $\forall s_1 \in B.\text{spacs}[$ 
 $\exists s_2 \in S.\text{spacs}[\text{accessible}(s_1, s_2)]]$ 
}

func visible( $B, g_1$ ){
   $\exists p_1, p_2 \in \mathbb{R}^3[$ 
 $p_1 \in \mathcal{R}[\mathcal{G}[[B.\text{spacs}]]] \wedge$ 
 $p_2 \in \mathcal{R}[[g_1]] \wedge$ 
 $\neg \exists g_2 \in \mathcal{G}[[B.\text{elems} \setminus B.\text{elems.wn}]][\text{line}(p_1, p_2) \# g]$ 
]
}

bld gallery(spacs { $\Sigma_{gallerySpaces}$ }, elems{ }, fns{ })
fullyAccessible( $B_{gallery}, \text{site } StMarksSquare(\text{objs}(O_{site}), \text{spacs}(\Sigma_{site})))$ 
)

F_{spaceRels}
F_{visibility}

```

Abbreviations:

$B_{gallery} = \text{bld } gallery(\text{spacs } \{\Sigma_{gallerySpaces}\}, \text{elems}\{\}, \text{fns}\{\})$

$\Sigma_{gallerySpaces}$: Here the number of spaces $s_1 \dots s_n$ in the gallery are defined, together with the required cubic area for each subset of art. This is informed by the MST of the art collection.

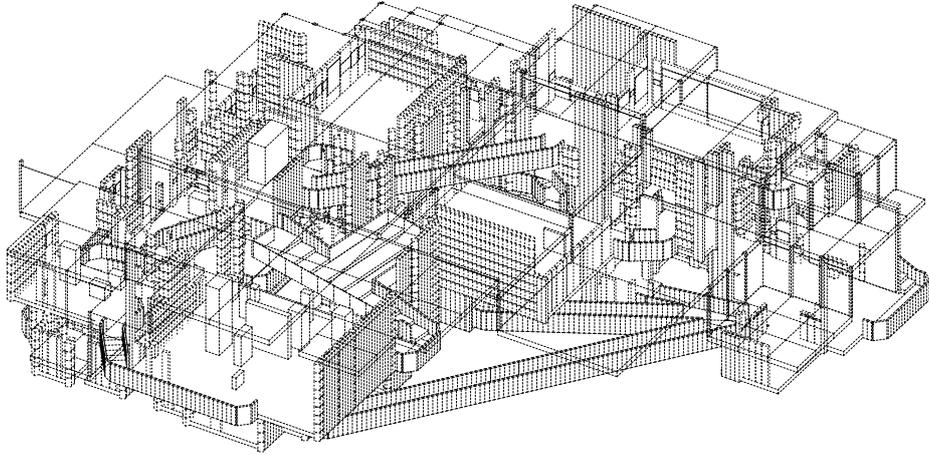
$F_{spaceRels}$: For each sub collection of art housed in space s_i , and all neighbouring sub collections housed in space s_j , the following function call is defined:

$$\text{access}(B_{gallery}, s_i, s_j)$$

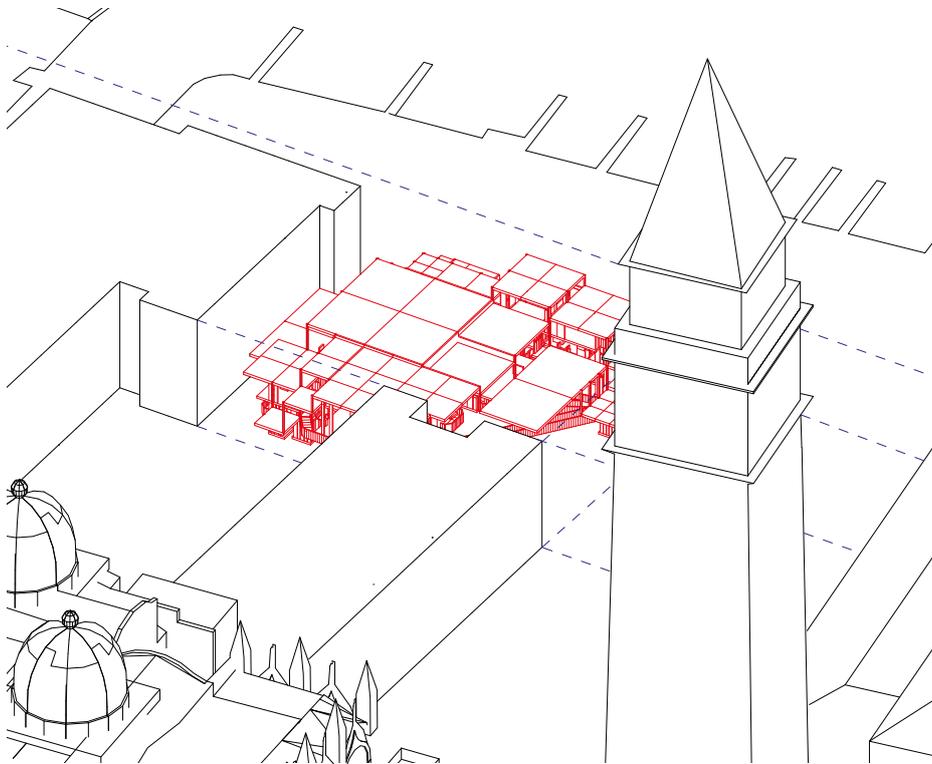
$F_{visibility}$: For each site geometry g_i that is deemed as a relevant view(see image with plus signs), the following function call is defined:

$$\text{visible}(B_{gallery}, g_i)$$

M =



P =

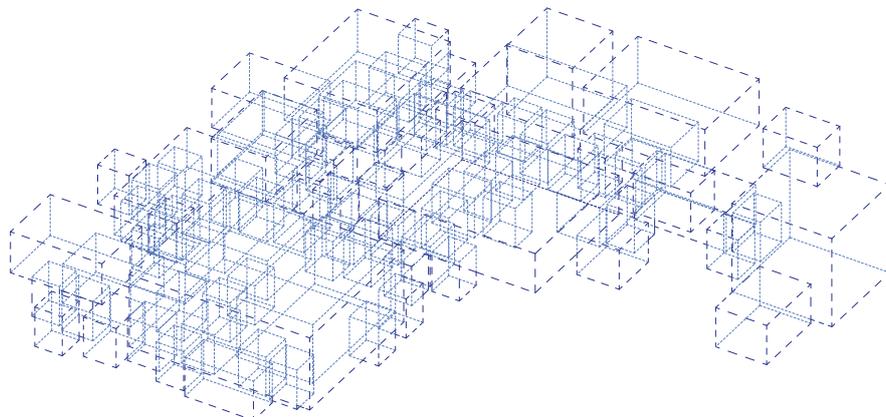


StO = ———

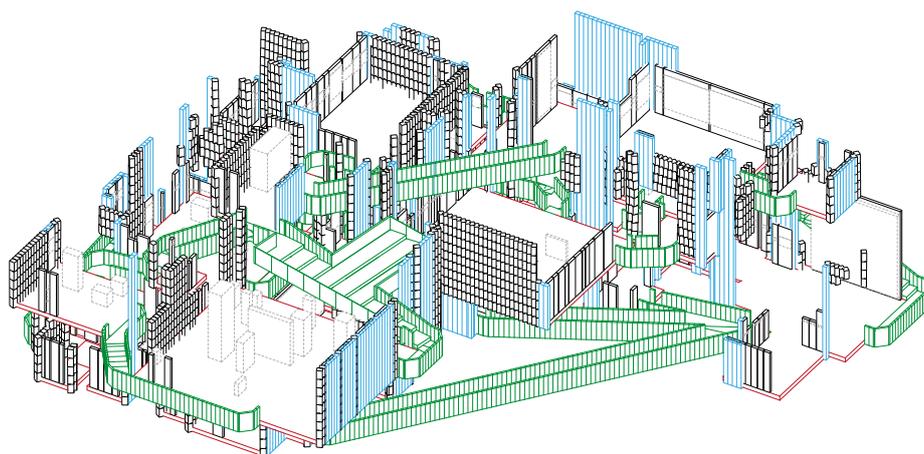
StS = - - - - -

Gallery = ———

Gallery.spacs =



Gallery.elem =

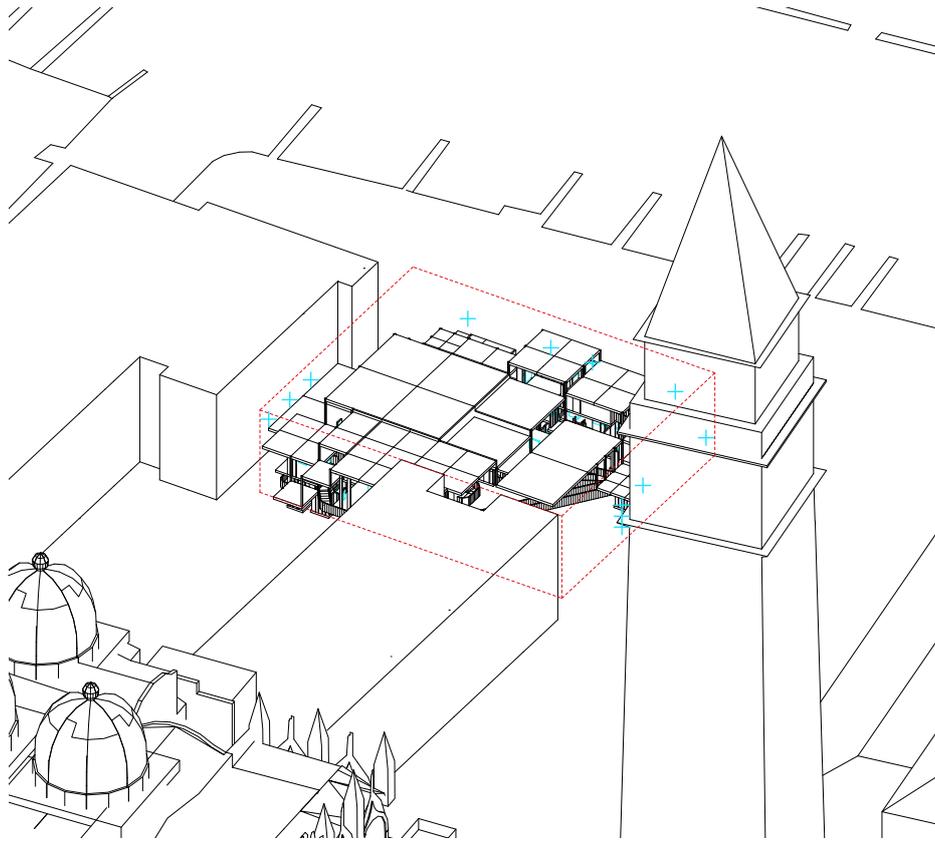


Gallery.fr = 

Gallery.str = 

Gallery.wall = 

Gallery.wn = 



The blue cross signs in this drawing depict the views from the bounding box outwards that were deemed to be relevant to the visual qualities of the collection. The visibility functions of the project declaration would have to be applied to all g in **STO** that are in line of these views.

5.4 Courtyard

For this last example only the function definition is given in full, as this is only meant to convey a certain architectural concept: that of the courtyard. Following the project description below, a set of basic models is given with motivation on why some of these do and others don't follow the project description. After this a set of architectural examples will be discussed in regards to this definition and the basic models.

In words the function for a building with a courtyard is specified as follows:

There is a space in the building for which:

- *there is no roof*
- *all adjacent building spaces do have a roof*
- *all adjacent walls are between this space and a space with a roof*
- *there is no adjacency to the site spaces*

```

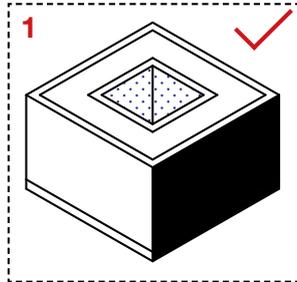
site location(objs(), spacs( $\Sigma_{site}$ )){
func courtyard(B){
   $\exists s_c \in B.spacs[$ 
     $\neg \exists e_r \in B.elems[s_c \text{---} e_r] \wedge$ 
     $\forall s_b \in B.spacs[s_b || s_c \rightarrow \exists e_r \in B.elems[s_b \text{---} e_r]] \wedge$ 
     $\forall e_w \in B.elems.wall[e_w || s_c \rightarrow \exists s_b \in B.spacs[$ 
       $e_w || s_b \wedge \exists e_r \in B.elems[s_b \text{---} e_r]] \wedge$ 
     $\neg \exists s_s \in \Sigma_{site}[s_c \sim s_s]$ 
  ]
bld buildingWithCourtyard(spacs {}, elems {}, fns {})
courtyard(bld buildingWithCourtyard(spacs {}, elems {}, fns {}))

```

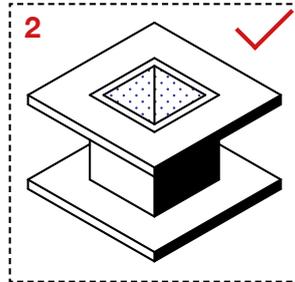
Abbreviations:

Σ_{site} = The external spaces of the surrounding site context.

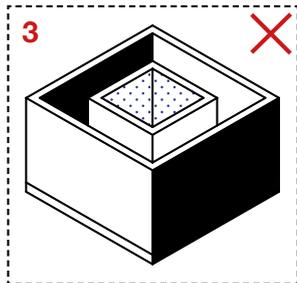
The following images depict a set of elementary models that either do or don't satisfy the project description given for a courtyard. The blue dotted area signifies the space which would be classified as the courtyard: s_c



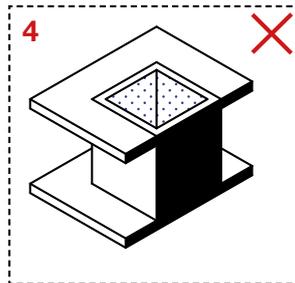
1
The obvious example of an open space fully surrounded by a closed space.



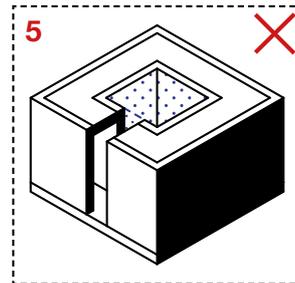
2
The spaces adjacent to the surrounding wall do not need to be closed, only covered, making this a valid model.



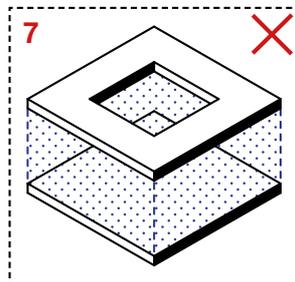
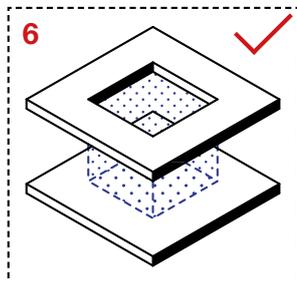
3
The surrounding space is not covered, hence not satisfying the specification.



4
One wall is adjacent to the context space, making it invalid.



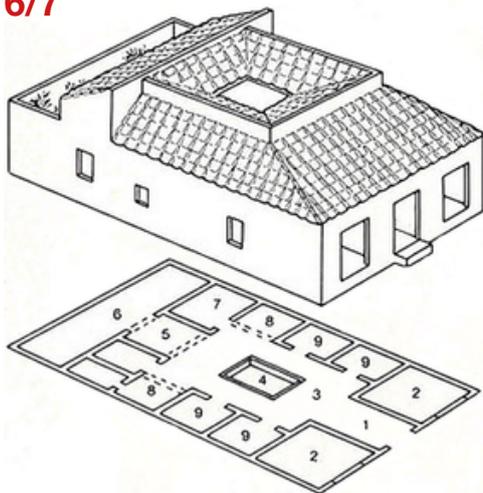
5
The open corridor is not covered, thus resulting in an invalid instance.



6
7
In this case it depends on how the spaces of the building are evaluated, if the space is limited to the centre this is a valid instance, otherwise it isn't since it is adjacent to the context, and there is an element that covers it.

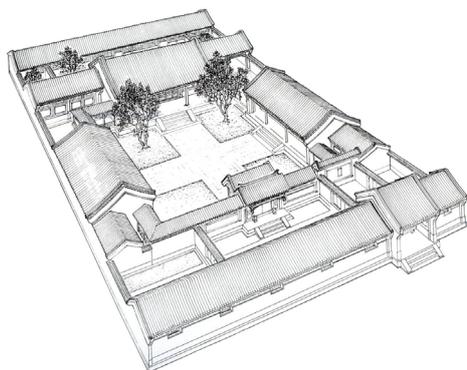
Now follows a selection of architecture projects of various scales, time periods and styles, all with a feature that could be interpreted as a courtyard. It will be discussed whether these satisfy the given definition **P** or not, and how this compares to an architectural interpretation. It will also be indicated which scenario the buildings follow in terms of the examples given on the previous page. (Image labels are given at the end of this section).

6/7



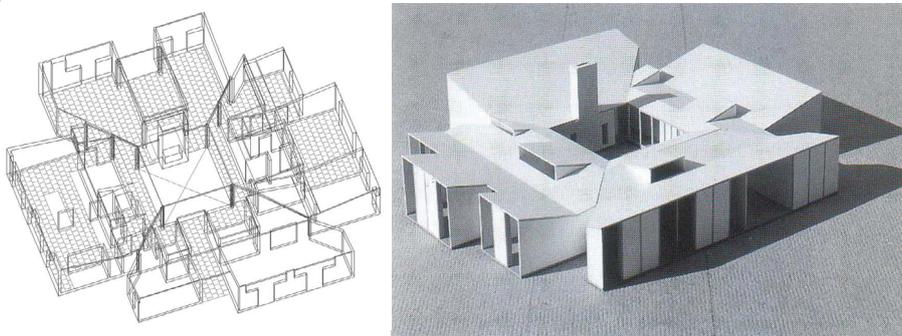
The Roman ‘Domus’ house typically has a specific type of courtyard in the middle with a pool for rainwater underneath. If this pool (nr 4 in the image) would be identified as a separate space it follows the definition **P**. However if the surrounding area including the pool would be interpreted as one singular space it wouldn’t, since part of the proposed courtyard space would be covered. A similar scenario to example 6 and 7.

1



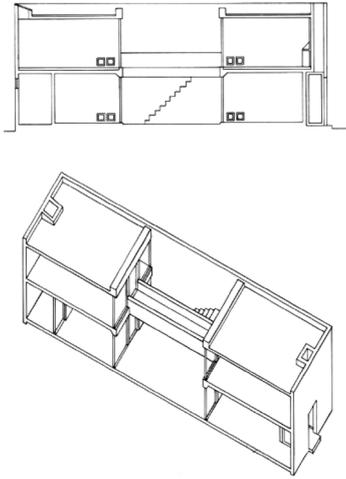
A Chinese ‘Siheyuan’ type residence would unambiguously satisfy **P**. This is due to the complete surrounding of covered spaces around the central courtyard, alike example 1.

1



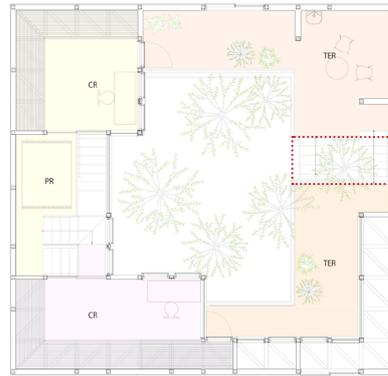
The unbuilt project ‘The Goldenberg House’ by Louis Kahn is also a clear cut example of a building that would fit the definition given in **P**, similar to example 1.

4

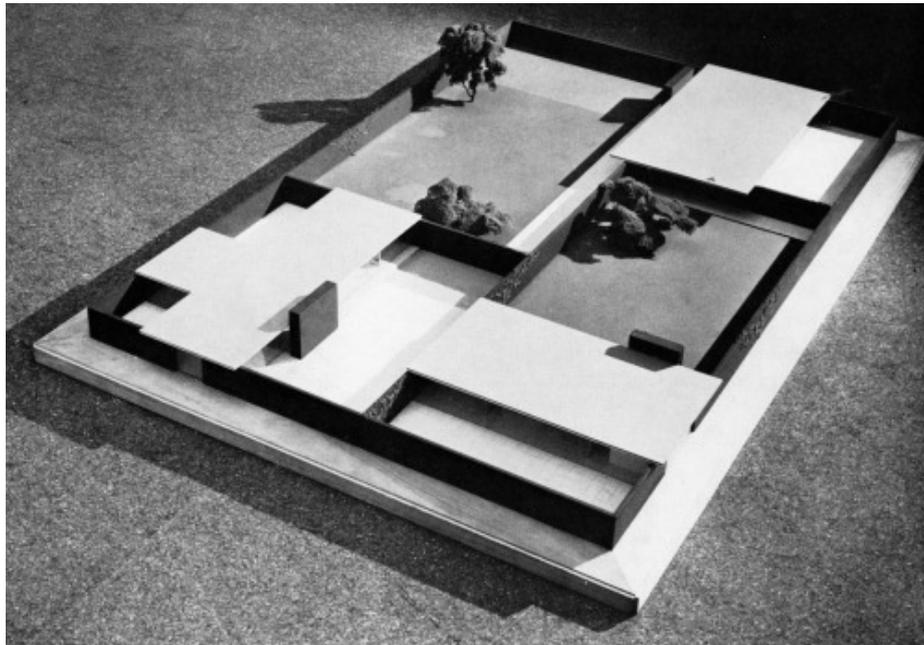


Tadao Ando's famous Azuma House with an open area between the two enclosed parts of the house does not satisfy **P**. It is a similar case to example 4, since there are walls enclosing the open area, but both are adjacent to the external context.

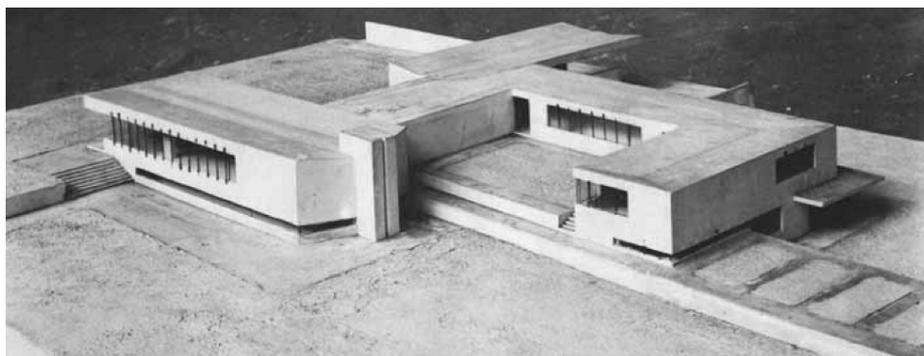
1



A more recent Japanese house, 'loop house' by Tomohiro Hata, does satisfy the given definition, on one condition. If the courtyard is considered as one space, excluding the entire staircase area marked with red in the plan above, this would fit the definition given in **P**. If the ground floor section of the staircase area, marked in the photo on the left would be included in the definition of that space, this would not fit **P**.

3/4

The next two projects are both by Mies van der Rohe. The first is a more difficult example, since it is a set of 3 houses, called 'Group of Three court houses'. As suggested in the name, the architect saw these houses as having a courtyard. However, because all open spaces are adjacent to either the site, or another open space it does not fit the definition of **P**. This follows examples 3 and 4.

5

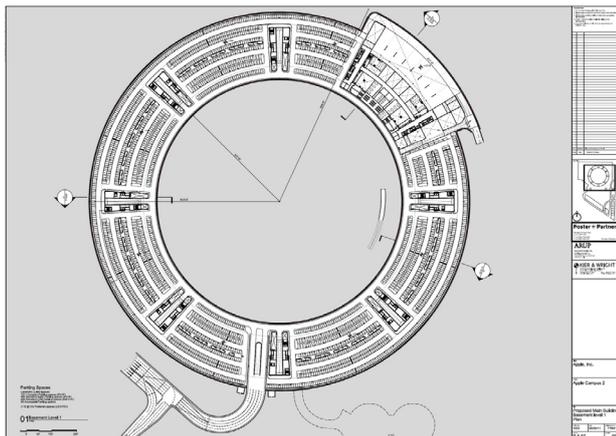
This project by van der Rohe for a concrete country house includes an open area at the centre, which is on one side completely exposed to the remaining context. Essentially an extreme version of example 5, thus not following **P**. However due to the change in levels from the surrounding area architecturally it could be debated that this is a courtyard.

6/7



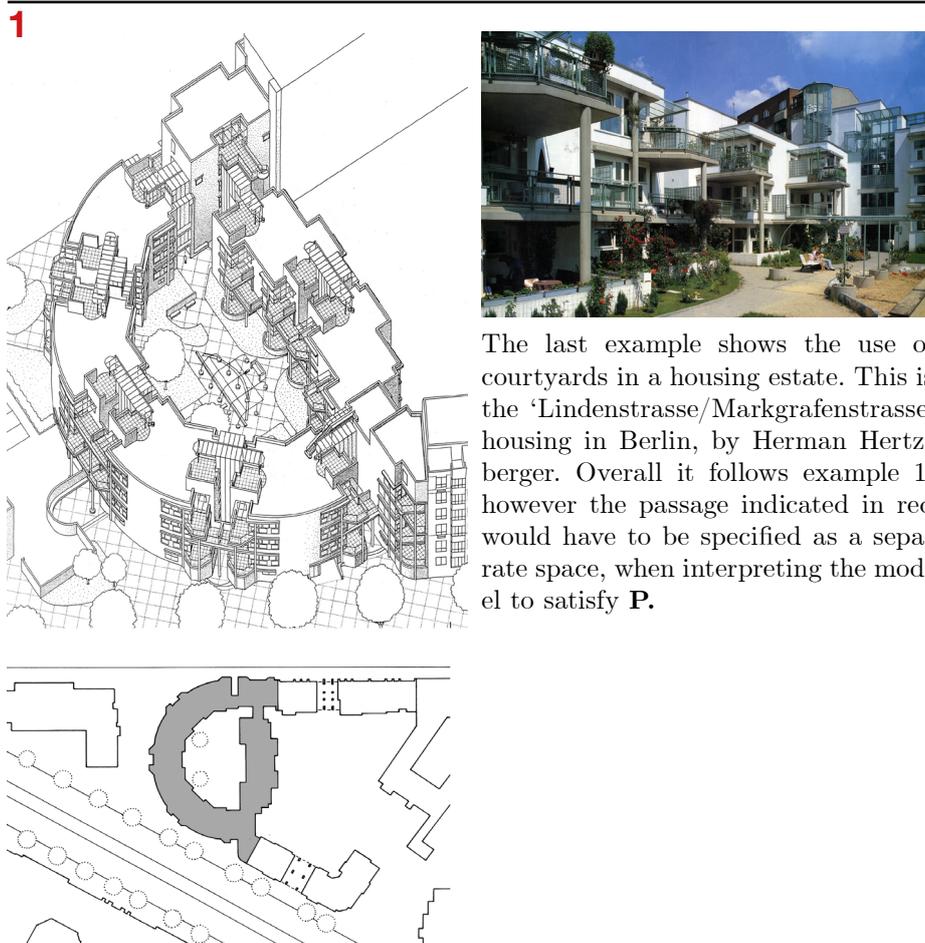
Le Corbusier's La Tourette Monastery. Another case similar to examples 6/7. Architecturally this would certainly qualify as a courtyard, but since there is a direct access to the site context from the courtyard, without the need of entering the building it would not meet **P** if the colonnade would be regarded as one singular space together with the courtyard.

1



By far the largest project discussed in this example: the Apple campus designed by Foster + Partners. Fits example 1 in terms of structure, but due to its size it could be argued from an architectural standpoint that the courtyard does not follow that of the conventional definition.





Labels of figures used in order of appearance in this section

- Figure 11. Domus*
- Figure 12. Siheyuan*
- Figure 13. The Goldenberg House axon*
- Figure 14. The Goldenberg House model*
- Figure 15. Azuma house axon*
- Figure 16. Azuma house facade*
- Figure 17. Azuma house internal*
- Figure 18. Loop house external*
- Figure 19. Loop house plan*
- Figure 20. Loop house internal*
- Figure 21. Court houses*
- Figure 22. Concrete country house*
- Figure 23. La Tourette Monastery external*
- Figure 24. La Tourette Monastery internal*
- Figure 25. Apple campus plan*
- Figure 26. Apple campus render*
- Figure 27. Lindenstrasse axon*
- Figure 28. Lindenstrasse plan*
- Figure 29. Lindenstrasse internal*

6.

Implementation

The last three chapters were concerned with the development of a way in which an architectural proposal can be quantified in terms of the functions it is meant to fulfil. This chapter focusses on how this could then be implemented in such a way that it can aid in computational design methods that aim to find an optimal solution to a given project definition.

6.1 Decidability

For possible implementations of the described DSL it is of importance whether the following question is decidable: Given a model **Mod** and a project description **Proj**, does the model satisfy the requirements of the project?

Whether a model satisfies a project according to the defined syntax and semantics, depends on the possibility of deriving a combination of spaces and elements from the model, in such a way that the project requirements are fulfilled. This is the case if those derived spaces and elements are valid instances of the spaces and elements as defined in the building definitions of the project, and filling in those instances in the function statements of the project need to evaluate to true.

Deciding on if a space or element is a valid instance only relies on checking relations between real numbers. The functional statements as defined in the DSL are equivalent to first-order logic statements over the real numbers. Because of this the decidability of a derivation of a model to a project definition can be seen as a satisfiability modulo theories (SMT) problem [Barrett et al., 2008]. The constraints are determined by the model, as this determines what spaces and elements can be derived from it, and the building definitions, limiting the domain of the possible instances. Then within these constraints a valid set of derivations need to be found that satisfy the function statements.

All operations over domains of real numbers in the specification are linear, which would make it possible to implement the DSL in such a way that the problem is reduced to a system of linear inequalities. By applying Fourier-Motzkin elimination [Monniaux, 2010] it would seem that the overall problem of checking the validity of a model in terms of a project description within this DSL would be decidable.

6.2 Integration in CAD

6.2.1 Derivation of spaces

From the courtyard example it becomes clear that the way in which the spaces

of the project are derived from the objects of the model can heavily influence whether a model is recognised as a valid instance of the project definition or not. In many architectural projects the lines between spaces are blurry at best, especially since space is a rather subjective term. The way in which architects refer to space often depends on the scale at which a project is discussed, thus making it a fluid concept within the design process. This is also the reason behind the way spaces are defined within the syntax and semantics, where they are not part of the model itself, but directly derived from it through a collection of voids. After all, in CAD models it is unusual to include a geometry of the void that forms a specific space, both because it is in fact not a physical geometry, and it is often not something that is set in stone within the project. However in the context of a formal expression of architectural design this does present a problem in terms of possible ambiguous interpretations.

One way of solving this would be the introduction of a convention within architectural CAD modelling where the spatial organisation of a building is made more distinct. In that case the model category in the syntax would need to be adjusted to contain one set of physical and one of organisational objects - to be evaluated to spaces. This would make the derivation from model to project significantly less ambiguous.

Another option would be to introduce a way in which the designer has to indicate how certain spaces are interpreted in different scenarios. This would align more with the way in which the concept of space is currently treated in architectural design. That way it might be suitable to adjust the semantics in such a way that multiple instances of spaces can be defined, where for one function an area is regarded as one space, where for another criteria that same area would be broken down in multiple different spaces.

The first option would be more in line with the proposed goal of this thesis where the functional aspects of a project are directly integrated in the CAD modelling process, thus adding a second layer to conventional computational design models. This would add the advantage that changes could directly be evaluated in regards to the project definition.

6.2.2 Derivation of elements

The second aspect within the semantics that allows for varying interpretations is the derivation of building elements from the model. Many creative design solutions stem from the blurred lines between the distinction of elements and their functions: should a door be a distinct object, or is a gap in a wall also seen as a door, where does the transition from wall to ceiling end, is a staircase only a staircase or also part of a floor if it is a gradual transition, etc. The current way in which the syntax and semantic rules are defined would force the designer to make a distinction between where one object ends and the next one starts, in order to be able to evaluate to the respective elements. This was done so that no objects from the model can be omitted, which would create possibilities of false positives, since an obstructing object could be chosen not to be interpreted as an element. A possible adjustment to the syntax to make it possible to interpret one object as multiple different building elements would be one way to address this. A statement would need to be added where one object can be interpreted as two elements simultaneously. After all the syntax and semantics that are introduced do not aim to be a design manual, but rather a means of verifying the validity of a model in terms of its design goals.

¹ <https://github.com/JulianBesems/PlaceBuildingsAI> (I'll probably create a new repo with neater commits)

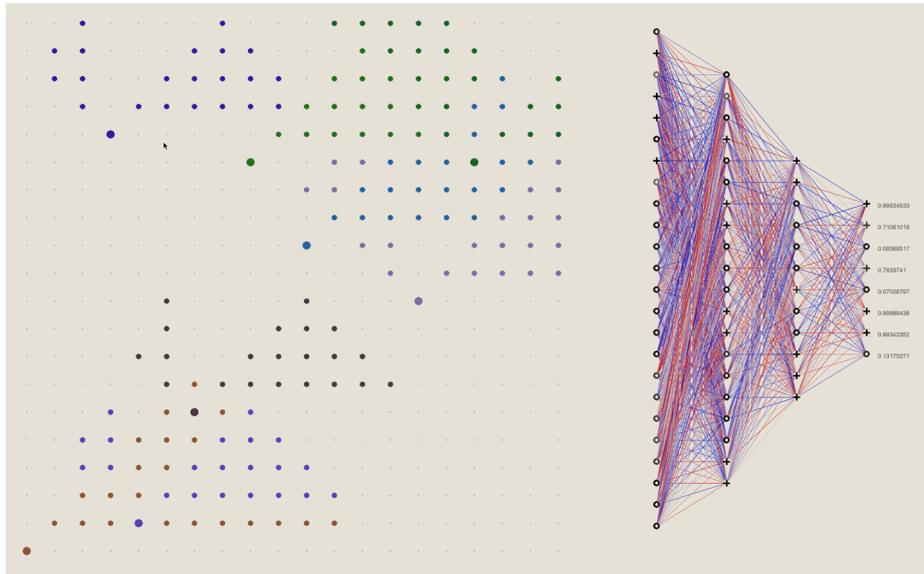


Figure 30. Zoning proposal with the neural network.

6.2.3 Mutual influence of design and definition

In an architectural project, the building programme is often re-evaluated as a direct result of the development of its design. This means that during the process of defining an instance to satisfy the project definition, that very definition could also be adjusted again. Essentially any design process is a continuous feedback loop between requirement and solution. This is also why the option for integration of spatial definition during the design was advocated above manual interpretation in 6.2.1. It is likely that an implementation of a formal framework to connect the geometry to the functional aspects of a building would prompt a continuous dialogue between modelling and adjusting the functional specification. However once one completed design instance would be modelled, and the specification of the functional aspects would be perfected to exactly fit that one instance, it should be possible to then develop a generative algorithm, using the geometric conventions used in the model to explore various instances of the design.

6.3 Use for generative architecture

In *The Creativity Code* by Marcus du Sautoy [2019] highlights how the AI *Alpha Go* achieved a level of playing the game go that exceeded that of any player, something he deemed as a truly creative act. This was seen as truly creative and different from AI's playing other games since *Alpha Go* didn't just master current playing methods, but actually changed the current way of playing the game through breaking with fundamental tactical conventions. One method behind this achievement is the use of a genetic neural network, where the training data isn't an aggregation of known outputs, but where the training process is informed by how well each attempt is according to a fitness function, essentially playing against itself.

This method of machine learning to drive creativity fundamentally differs from the previously discussed implementation of using image based GAN's in 2.1.2, since the neural network is not trained on historical data, or at least not exclusively. Rather the quality of the output is judged in terms of it's goals and validity, by means of a fitness function.

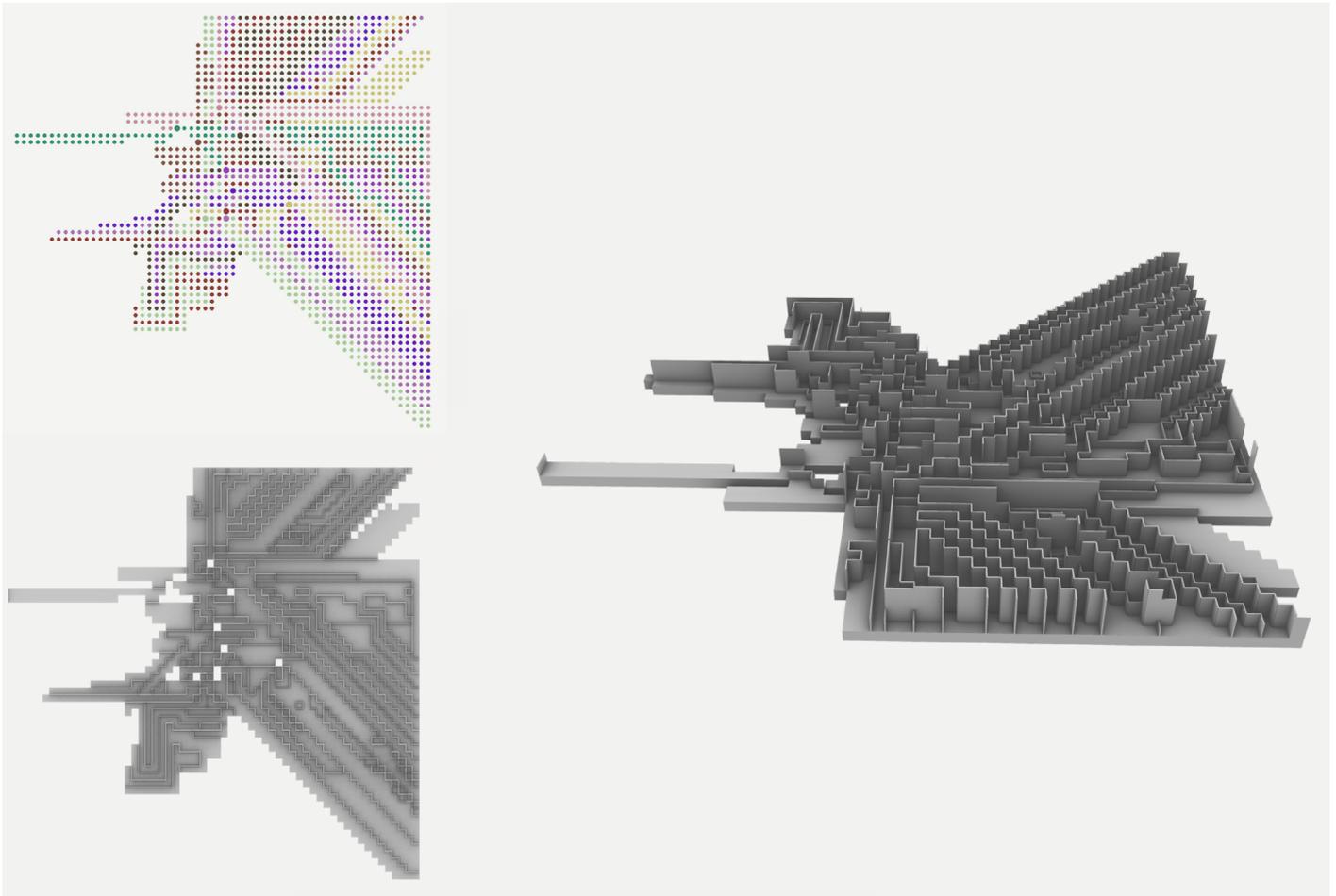
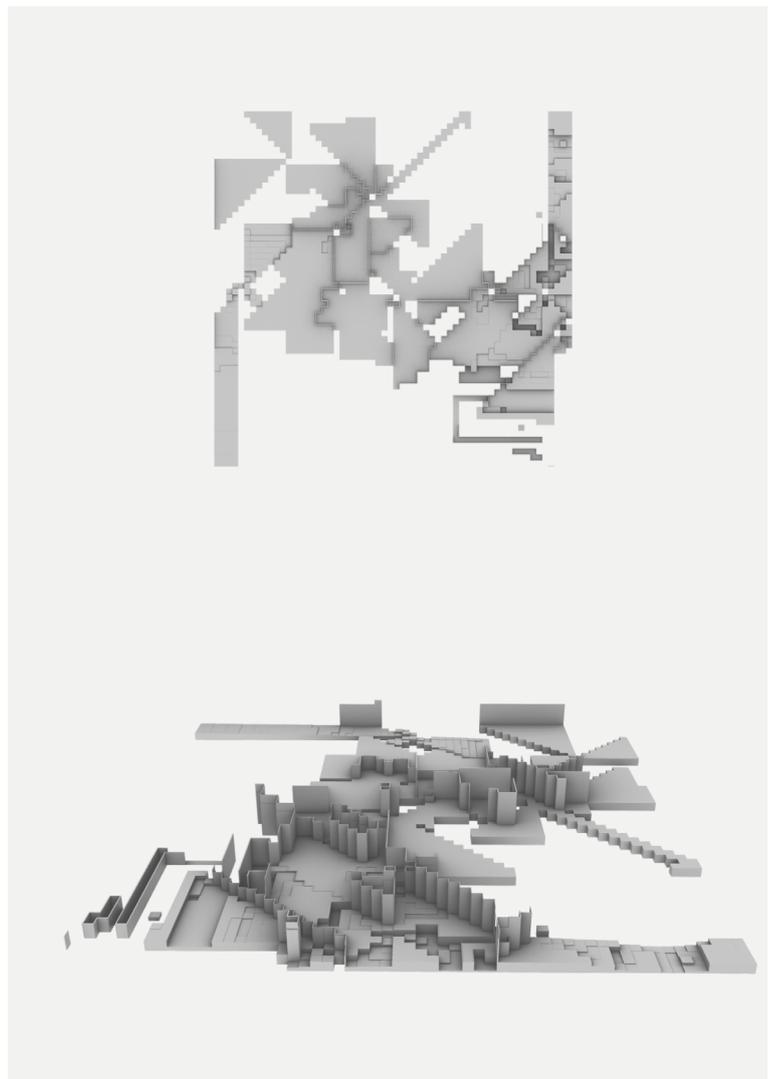


Figure 31. Outcome of a zoning 'game', with 3D interpretation.



Figures 32. Maze like outputs. The influence of the 8 directions by which the game moves were restricted remain extremely visible through the 'wind mill' patterns.

Based on this idea I attempted to implement a genetic neural network in order to generate architectural zoning proposals¹. The basic idea is that given a set of spaces with a set number of tiles each, and a relation score between each space, the neural network is to propose a configuration of the tiles that best reflects the relations between the spaces. The implementation was based on an open source genetic neural network that plays the game snake [Greerviau, 2019]. The algorithm was adjust so that instead of snakes there are multiple nodes, each forming the starting point for a space. Then each space takes turns with placing one of its tiles, until all tiles of all spaces have been placed.

The input of the neural network is based from the centre tile of the space who's turn it is. From that tile it is calculated how many tiles of its own there are, how close tiles of other spaces are, and how many empty spaces there are for 8 directions. The output is the decision for each direction, the highest score is where the tile is placed. At the end of each attempt the score is calculated based on a fitness score looking at adjacency of spaces according to the relational scores.

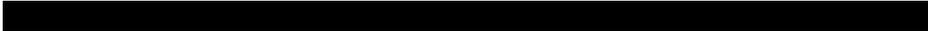
The result is a maze like tiling for the set of given spaces. The difficulty proved to be twofold. First of all it was a challenge to find a way to essentially turn an architectural massing exercise into a game. Second of all it proved extremely challenging to evaluate the outcome in an accurate way through a fitness function. Additionally my knowledge of neural networks and genetic algorithms is rather limited, which restricted me to fairly basic attempts.

I would argue that this approach could possibly be combined with a more straightforward algorithmic approach such as the ones applied in the two design projects described in *2.2.1* and *2.2.2*. However the evaluation issue would remain on how to score the outcome, which brings us back to the proposal of a syntax and semantics for architectural design. This could provide a means of scoring generated outcomes in terms of how well they satisfy the desired project features.

¹Code available on: <https://github.com/JulianBesems/ArchitecturalZoningAI>

7.

Conclusion



The development of a language to formally define functional aspects of architectural design is meant to fill a void in computational technologies used for architectural design. Current computational development within the field of architecture is primarily focussed at ways of form finding, thus limited to the geometrical aspect of design. This can partially be led back to the loss in mutual interest between the fields of computer science and architectural design, since this has led to the more heavy reliance of architects on software not specifically developed for the field of architecture.

In defining a way to express the functional aspects of an architectural proposal in a formal language, the first step was to establish a hierarchy in which an architectural design can be defined. The difficulty in this is that an architectural project is not evaluated in a sequential way, given that at least the final proposal of a design is static in nature. Consequentially the defined language does not formalise the process of reaching a solution, but is focussed at specifying the conditions this solution has to satisfy in order to meet the project requirements. The structure of the syntax is top down, from least specific to most specific. Essentially a hierarchy is declared for an architectural proposal, breaking down the whole to the axiomatic organisational elements that tie the project together. The functional aspect of the specification consists of statements that define conditions over the combination of the aforementioned elements. Within the project description the use of specific geometry is kept to a minimum, focussing on compositional conditions. The semantics then declares the way in which a specific model of a proposed solution can be evaluated to determine if the project specification is satisfied.

The focus on functional aspects inevitably led to a structure within which ambiguity was included, precisely because the current understanding of architectural design is not absolute. The central functional category **Spac** in the grammar represents the architectural concept of space. This concept is at the centre of architectural design processes, but virtually always remains implicit in the design outcome. The way in which spaces are derived from the model is structured so that this ambiguity is maintained. This means that a language is probably not the most suitable to distinguish whether a model is a complete nonsensical building proposal. It is more suited to the functional verification of proposals that meet a certain level of rudimentary correctness. On the other extreme it is also not suitable for the evaluation of aesthetic characteristics, as this heavily relies on the physical aspects of the project proposal, which is not what this methodology focusses on. Completeness is only measured by the correlation between an abstracted CAD model and the extended first order logic statements over the given categories. As such it is not intended to aid in fully automating the design process, but rather to be incorporated in

current design methods as a tool to facilitate design abstraction.

This encourages a way of working where the specification of the project would be developed in correspondence to the development of its solution. In effect this means that throughout the process the designer is forced to reevaluate their own work, and adjust both specification and solution to reach a satisfactory whole. Often a creative architectural solution to a project brief also involves the creative interpretation and reevaluation of that brief. As such this way of working shows resemblance to the way in which Agile project management is used within software development, where the software requirements are reevaluated based on the review.

7.1 Discussion

The evaluation of a model in terms of validity in the current DSL is restricted to an absolute outcome, the model is valid or it isn't. For further implementation this might not be the most appropriate format of evaluation. On a detailed level the propositions used for whether one element is adjacent to another does not take into account how large this overlap is. This means that if the specification says that one space needs to be connected to another, and there is a 1mm gap between separating walls, this would evaluate to true. As such a more fine-tuned way of expressing these relations might be desirable. On a larger scale the limitation of this way of evaluation is that the outcome does not give insight in where the issue lies. Because of this further development might look at ways to numerically score the validity of a model, instead of being limited to a boolean evaluation. This would also address the potential implementation where the evaluation serves as fitness function for a generative approach using a genetic neural network. On the other hand a balance is needed between simplicity and specificity, at the moment the specification of the elements is almost entirely restricted to their relational role, this means that within an instance there exists a freedom in how these are being shaped. The danger of over-specifying the project is that the design solutions will be bound to preconceived limitations, thus standing in the way of solutions that were not considered beforehand. A possible way of addressing this would be to connect the specification to a set of parametric tools, where the requirements can be more subtly adjusted throughout the design process, allowing for more detail to be added once the design direction has become more clear.

Previously it was mentioned that an architectural proposal is static, but the experience of the user for which the design is intended is certainly not. A different approach to the one taken in this thesis would be to focus on the interaction of the dynamic user with the static project definition. Possibly this would then focus on evaluating the experience of a sequence through the building, requiring the specification of those sequences instead of the overall functional aspects in a top down format.

List of Illustrations

Cover Image - *Project Atlas*, Author's own work, 2020. In Besems, J. (2020a). *createGalleries.py*. UCL. Design project. <http://unit-21.com/?p=9454>

Figure 1. - *Pattern 110 - Main entrance* by Christopher Alexander. In Alexander, C. (1977). *A Pattern Language*. New York. Oxford University Press

Figure 2. - *GAN generated floor plans* by Stanislas Chaillou. In Chaillou, S. (2019). *AI + Architecture: Towards a New Approach*. Harvard University.

Figure 3. - *Generative process behind library bookshelves and walkways*, Author's own work, 2019. In Besems, J. (2019). *Books as Bytes*. UCL. Design project. <http://unit-21.com/?p=8106>

Figure 4. - *Various library outputs depending on different sorting criteria and existing context*, Author's own work, 2019. In Besems, J. (2019). *Books as Bytes*. UCL. Design project. <http://unit-21.com/?p=8106>

Figure 5. - *Assembly at the end of a semester after which the walkways and bookshelves are adjusted to the behaviour of the inhabitants*, Author's own work, 2019. In Besems, J. (2019). *Books as Bytes*. UCL. Design project. <http://unit-21.com/?p=8106>

Figure 6. - *A collection of libraries are situated within a larger dynamic study landscape which follows a vector field*, Author's own work, 2019. In Besems, J. (2019). *Books as Bytes*. UCL. Design project. <http://unit-21.com/?p=8106>

Figure 7. - *MST of an art gallery collection based on image similarity. The resulting massing of the subdivision of spaces based on main branches is seen on the left*, Author's own work, 2020. In Besems, J. (2020a). *createGalleries.py*. UCL. Design project. <http://unit-21.com/?p=9454>

Figure 8. - *Visibility graph in order to find a valid circulation through the gallery*, Author's own work, 2020. In Besems, J. (2020a). *createGalleries.py*. UCL. Design project. <http://unit-21.com/?p=9454>

Figure 9. - *Variety of generated gallery proposals*, Author's own work, 2020. In Besems, J. (2020a). *createGalleries.py*. UCL. Design project. <http://unit-21.com/?p=9454>

Figure 10. - *Collection of galleries in context*, Author's own work, 2020. In Besems, J. (2020a). *createGalleries.py*. UCL. Design project. <http://unit-21.com/?p=9454>

Figure 11. - *Domus*, Unknown Author. In Art History 342: Roman art, <https://arh342.weebly.com/rooms-in-the-domus.html>

Figure 12. - *Siheyuan*, Unknown Author. In Apple Eden's Blog, (2010), *Siheyuan, Quadrangles, Four Side Enclosed Courtyard (Reprinted)*, <https://appleeden.wordpress.com/2010/10/21/siheyuanquadranglesfour-side-enclosed-courtyardreprinted/amp/>

Figure 13. - *The Goldenberg House Axon* by Louis Kahn. In Fabrizi, M, (2014), *The Plan is a Society of Rooms*: Goldenberg House by Louis Kahn (1959)*, <http://socks-studio.com/2014/04/08/the-plan-is-a-society-of-rooms-goldenberg-house-by-louis-kahn-1959/>

Figure 14. - *The Goldenberg House Model* by Louis Kahn. In Fabrizi, M, (2014), *The Plan is a Society of Rooms*: Goldenberg House by Louis Kahn (1959)*, <http://socks-studio.com/2014/04/08/the-plan-is-a-society-of-rooms-goldenberg-house-by-louis-kahn-1959/>

Figure 15. - *Azuma House Axon* by Tadao Ando. In Baek, J. (2010). *Climate, Sustainability And The Space Of Ethics*, Architectural Theory Review, 15:3, 377-395, DOI: 10.1080/13264826.2010.497181

Figure 16. - *Azuma House Facade* by Tadao Ando. In Sgustok Design, *Tadao Ando: Azuma House*, <https://sgustokdesign.com/tadao-ando-azuma-house>

Figure 17. - *Azuma House Internal* by Tadao Ando. In Sgustok Design, *Tadao Ando: Azuma House*, <https://sgustokdesign.com/tadao-ando-azuma-house>

Figure 18. - *Loop House External* by Tomohiro Hata. In Astbury, J. (2019), *Tomohiro Hata's Loop House faces inwards onto a central courtyard*, Dezeen. <https://www.dezeen.com/2019/07/14/loop-house-tomohiro-hata-architect-associates-hyogo-japan-courtyard/>

Figure 19. - *Loop House Plan* by Tomohiro Hata. In Astbury, J. (2019), *Tomohiro Hata's Loop House faces inwards onto a central courtyard*, Dezeen. <https://www.dezeen.com/2019/07/14/loop-house-tomohiro-hata-architect-associates-hyogo-japan-courtyard/>

Figure 20. - *Loop House Internal* by Tomohiro Hata. In Astbury, J. (2019), *Tomohiro Hata's Loop House faces inwards onto a central courtyard*, Dezeen. <https://www.dezeen.com/2019/07/14/loop-house-tomohiro-hata-architect-associates-hyogo-japan-courtyard/>

Figure 21. - *Court Houses* by Mies van der Rohe. In Archive of Affinities. (2011), *MIES VAN DER ROHE, GROUP OF THREE COURT HOUSES, PROJECT, 1938*. <https://archiveofaffinities.tumblr.com/post/14571923635/mies-van-der-rohe-group-of-three-court-houses>

Figure 22. - *Concrete country house* by Mies van der Rohe. In Architectural Drawings, Models, Photos, etc.... (2014), *Project for a concrete country house*. <https://davidhannafordmitchell.tumblr.com/post/98384566063/tgphi-pps-ludwig-mies-van-der-rohe>

Figure 23. - *La Tourette Monastery external* by Le Corbusier, photo by Oliver Martin Gambier. In Winston. A. (2016), *Le Corbusier's La Tourette monastery is among his iconic buildings on the World Heritage List*. <https://www.dezeen.com/2016/07/22/le-corbusier-la-tourette-monastery-grass-roof-france-unesco-world-heritage/>

Figure 24. - *La Tourette Monastery internal* by Le Corbusier, photo by Fernando Schapo. In Winston. A. (2016), *Le Corbusier's La Tourette monastery is among his iconic buildings on the World Heritage List*. Dezeen. <https://www.dezeen.com/2016/07/22/le-corbusier-la-tourette-monastery-grass-roof-france-unesco-world-heritage/>

Figure 25. - *Apple campus plan* by Foster + Partners. In Basulto, D. (2011), *More about Foster + Partner's new Apple Campus in Cupertino*. ArchDaily. https://www.archdaily.com/160044/more-about-foster-partners-new-apple-campus-in-cupertino?ad__medium=gallery

Figure 26. - *Apple campus render* by Foster + Partners. In Basulto, D. (2011), *More about Foster + Partner's new Apple Campus in Cupertino*. ArchDaily. https://www.archdaily.com/160044/more-about-foster-partners-new-apple-campus-in-cupertino?ad__medium=gallery

Figure 27. - *Lindenstrasse axon* by Herman Hertzberger. In Herzberger, H. (1984-1986), *LINDENSTRASSE / MARKGRAFENSTRASSE HOUSING, BERLIN, GERMANY*. AHH. <https://www.ahh.nl/index.php/en/projects2/14-woningbouw/78-lindenstrasse-housing-berlin-germany>

Figure 28. - *Lindenstrasse plan* by Herman Hertzberger. In Herzberger, H. (1984-1986), *LINDENSTRASSE / MARKGRAFENSTRASSE HOUSING, BERLIN, GERMANY*. AHH. <https://www.ahh.nl/index.php/en/projects2/14-woningbouw/78-lindenstrasse-housing-berlin-germany>

Figure 29. - *Lindenstrasse internal* by Herman Hertzberger. In Herzberger, H. (1984-1986), *LINDENSTRASSE / MARKGRAFENSTRASSE HOUSING, BERLIN, GERMANY*. AHH. <https://www.ahh.nl/index.php/en/projects2/14-woningbouw/78-lindenstrasse-housing-berlin-germany>

Figure 30. - *Zoning proposal with the neural network*, Author's own work.

Figure 31. - *Outcome of a zoning 'game', with 3D interpretation*, Author's own work.

Figure 32. - *Maze like outputs. The influence of the 8 directions by which the game moves were restricted remain extremely visible through the 'wind mill' patterns*, Author's own work.

Bibliography

Alexander, C. (1975). *The Oregon Experiment*. New York. Oxford University Press

Alexander, C. (1977). *A Pattern Language*. New York. Oxford University Press

Barrett, C. & Sebastiani, R. & Seschia, S. & Tinelli, C. (2008). *Satisfiability Modulo Theories*. In Biere, A. & Heule, M. & Maaren van, H. & Walsch, T. *Handbook of Satisfiability*. IOS Press.

Besems, J. (2019). *Books as Bytes*. UCL. Design project. <http://unit-21.com/?p=8106>

Besems, J. (2020a). *createGalleries.py*. UCL. Design project. <http://unit-21.com/?p=9454>

Besems, J. (2020b). *Towards A New Architectural Entity - How the n-Dimensional could inform the 3-Dimensional*. UCL. Thesis. <http://unit-21.com/wp-content/uploads/2020/07/ThesisJBesems.pdf>

Burke, A. & Tierney, T. (2007). *Network practices: new strategies in architecture and design*. New York, Princeton Architectural Press

Chaillou, S. (2019). *AI + Architecture: Towards a New Approach*. Harvard University.

Greerviau. (2019). *SnakeAI*. <https://github.com/greerviau/SnakeAI>

IBM. (1988). *IBM Architecture And Engineering Series*. Announcement Letter Number 288-215 dated May 3, 1988. IBM. https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?appName=skmww-w&htmlfid=897%2FENUS288-215&infotype=AN&mhq=service%20initializer&mhsrc=ibmsearch__a&subtype=CA

Koolhaas, R. (2014). *Elements of architecture*. Tashen. Cologne.

Lea, D. (1994). *Christopher Alexander: An Introduction for Object-Oriented Designers*. ACM SIGSOFT

Monniaux, D. (2010). *Quantifier elimination by lazy model enumeration*. CAV 2010, Edinburgh, UK. pp. 585-599

Nielson, H. & Nielson, F. (1999) *Semantics With Applications - A Formal Introduction*. John Wiley & Sons

Sowizral, H. (1987). *Design Methodology for Object-Oriented Programming*. OOPSLA'97 Panel Session.

Appendix B

Thoroughfare Pattern 101
As described in A Pattern Language by Christopher Alexander [1977, p492 - 498]

IOI BUILDING THOROUGHFARE



492

. . . if the building complex is built at high density, then at least part of the circulation cannot be made of outdoor PEDESTRIAN STREETS (100) because the buildings cover too much of the land; in this case, the main spines of the CIRCULATION REALMS (98) must take the form of building thoroughfares similar to pedestrian streets, but partly or wholly inside the buildings. Building thoroughfares replace the terrible corridors which destroy so much of modern building, and help to generate the indoor layout of a BUILDING COMPLEX (95).



When a public building complex cannot be completely served by outdoor pedestrian streets, a new form of indoor street, quite different from the conventional corridor, is needed.



An indoor street.

The problem arises under two conditions.

1. *Cold weather.* In very cold climates to have all circulation outdoors inhibits social communication instead of helping it. Of course, a street can be roofed, particularly with a glass roof. But as soon as it becomes enclosed, it has a different social ecology and begins to function differently.

2. *High density.* When a building complex is so tightly packed

BUILDINGS

on the site that there is no reasonable space for outdoor streets because the entire building complex is a continuous two, three, or four story building, it becomes necessary to think of major thoroughfares in different terms.

To solve the problems posed by these conditions, streets must be replaced by indoor thoroughfares or corridors. But the moment we put them indoors and under cover, they begin to suffer from entirely new problems, which are caused by the fact that they get sterilized by their isolation. First, they become removed from the public realm, and are often deserted. People hardly ever feel free to linger in public corridors when they are off the street. And second, the corridors become so unfriendly that nothing ever happens there. They are designed for scuttling people through, but not for staying in.

In order to solve these new problems, created when we try to put a street indoors, the indoor streets—or building thoroughfares—need five specific characteristics.

1. *Shortcut*

Public places are meant to invite free loitering. The public places in community buildings (city halls, community centers, public libraries) especially need this quality, because when people feel free to hang around they will necessarily get acquainted with what goes on in the building and may begin to use it.

But people rarely feel free to stay in these places without an Official Reason. Goffman describes this situation as follows:

Being present in a public place without an orientation to apparent goals outside the situation is sometimes called lolling, when position is fixed, and loitering, when some movement is entailed. Either can be deemed sufficiently improper to merit legal action. On many of our city streets, especially at certain hours, the police will question anyone who appears to be doing nothing and ask him to “move along.” (In London, a recent court ruling established that an individual has a right to walk on the street but no legal right merely to stand on it.) In Chicago, an individual in the uniform of a hobo can loll on “the stem,” but once off this preserve he is required to look as if he were intent on getting to some business destination. Similarly, some mental patients owe their commitment to the fact that the police found them wandering on the streets at off hours without any apparent destination or purpose in mind. (Erving Goffman, *Behavior in Public Places*, New York: Free Press, 1963, p. 56.)

101 BUILDING THOROUGHFARE

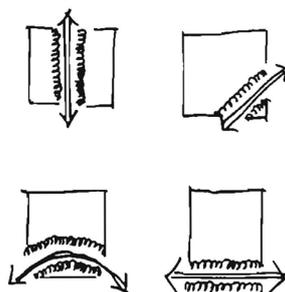
If a public space is to be really useful it must somehow help to counter the anti-loitering tendency in modern society. Specifically, we have observed these problems:

a. A person will not use a public place if he has to make a special motion toward it, a motion which indicates the intention to use the facility "officially."

b. If people are asked to state their reason for being in a place (for example, by a receptionist or clerk) they won't use it freely.

c. Entering a public space through doors, corridors, changes of level, and so on, tends to keep away people who are not entering with a specific goal in mind.

Places which overcome these problems, like the Galleria in Milan, all have a common characteristic: they all have public thoroughfares which slice through them, lined with places to stop and loiter and watch the scene.



Shortcuts.

2. Width

An indoor street needs to be wide enough for people to feel comfortable walking or stopping along the way. Informal experiments help to determine how much space people need when they pass others. Since the likelihood of three people passing three people is not high, we consider as a maximum two people passing two people, or three people passing one person. Each person takes about two feet; there needs to be about one foot between two groups which pass, so that they do not feel crowded; and

BUILDINGS

people usually walk at least one foot away from the wall. The street width, therefore, should be *at least 11 feet*.

Our experiments also indicate that a person seated or standing at the edge of a street feels uncomfortable if anyone passes closer than five feet. Thus, in places in the street where seats, activities, entrances, and counters are placed, the street should widen to about 16 feet (one-sided) or 20 feet (two-sided).

3. Height

Ceiling heights should also feel comfortable for people walking or standing along an indoor street. According to CEILING HEIGHT VARIETY (190), the height of any space should be equal to the appropriate horizontal social distances between people for the given situation—the higher the ceiling, the more distant people seem from each other.

Edward Hall, in *The Hidden Dimension*, suggests that a comfortable distance between strangers is the distance at which you cannot distinguish the details of their facial features. He gives this distance as being between 12 and 16 feet. Thus, the ceiling height in an indoor street should be at least in that range.

Where people sit and stand talking to each other, the appropriate social distance is more intimate. Hall gives it a dimension of four to seven feet. Thus, the ceiling in activity and “edge” places should be seven feet.

This suggests, for a large indoor street, a ceiling that is high in the middle and low at the edges. In the middle, where people are passing through and are more anonymous, the ceiling may be 12 to 20 feet high, or even higher, according to the scale of the passage. Along the edges of the thoroughfare, where people are invited to stop and become slightly more engaged in the life of the building, the ceilings may be lower. Here are three sections through an indoor street which have this property.



Cross sections of an indoor street.

IOI BUILDING THOROUGHFARE

4. *Wide entrance*

As far as possible, the indoor street should be a continuation of the circulation outside the building. To this end, the path into the building should be as continuous as possible, and the entrance quite wide—more a gateway than a door. An entrance that is 15 feet wide begins to have this character.

5. *Involvements along the edge*

To invite the free loitering described above under *Shortcut*, the street needs a continuum of various “involvements” along its edge.

Rooms next to the street should have windows opening onto the street. We know it is unpleasant to walk down a corridor lined with blank walls. Not only do you lose the sense of where you are but you get the feeling that all the life in the building is on the other side of the walls, and you feel cut off from it. We guess that this contact with the public is not objectionable for the workers, so long as it is not too extreme, that is, as long as the workplace is protected either by distance or by a partial wall.

The corridor should be lined with seats and places to stop, such as newspaper, magazine, and candy stands, bulletin boards, exhibits, and displays.

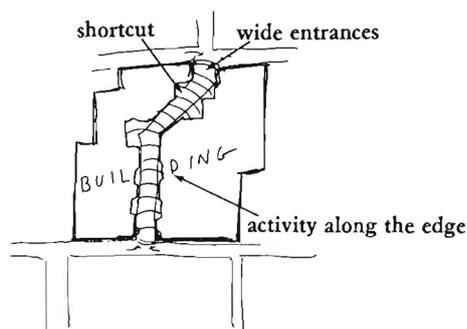
Where there are entrances and counters of offices and services off the corridor, they should project into the corridor. Like activities, entrances and counters create places in the corridor, and should be combined with seats and other places to stop. In most public service buildings these counters and entrances are usually set back from corridors which makes them hard to see, and emphasizes the difference between the corridor as a place for passing through, and the office as a place where things happen. The problems can be solved if the entrances and counters project into the corridor and become part of it.

Therefore:

Wherever density or climate force the main lines of circulation indoors, build them as building thoroughfares. Place each thoroughfare in a position where it functions as a shortcut, as continuous as possible with the public street

BUILDINGS

outside, with wide open entrances. And line its edges with windows, places to sit, counters, and entrances which project into the hall and expose the buildings' main functions to the public. Make it wider than a normal corridor—at least 11 feet wide and more usually, 15 to 20 feet wide; give it a high ceiling, at least 15 feet, with a glazed roof if possible and low places along the edge. If the street is several stories high, then the walkways along the edges, on the different stories, can be used to form the low places.



❖ ❖ ❖

Treat the thoroughfare as much like a PEDESTRIAN STREET (100) as possible, with OPEN STAIRS (158) coming into it from upper stories. Place entrances, reception points, and seats to form the pockets of activity under the lower ceilings at the edges—FAMILY OF ENTRANCES (102), ACTIVITY POCKETS (124), RECEPTION WELCOMES YOU (149), WINDOW PLACE (180), CEILING HEIGHT VARIETY (190), and give these places strong natural light—TAPESTRY OF LIGHT AND DARK (135). Make a connection to adjacent rooms with INTERIOR WINDOWS (194) and SOLID DOORS WITH GLASS (237). To give the building thoroughfare the proper sense of liveliness, calculate its overall size according to PEDESTRIAN DENSITY (123). . . .