Computerondersteunde exploratie van architecturale ontwerpruimtes: een digitaal schetsboek

Computer-Aided Exploration of Architectural Design Spaces: a Digital Sketchbook

Tiemen Strobbe

Promotoren: prof. dr. R. De Meyer, em. prof. dr. ir. J. Van Campenhout
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Architectuur

Vakgroep Architectuur en Stedenbouw
Voorzitter: prof. dr. ir. -architect A. Janssens
Faculteit Ingenieurswetenschappen en Architectuur

Academiejaar 2015 - 2016

UNIVERSITEIT
GENT

ISBN 978-90-8578-850-8
NUR 956, 965
Wettelijk depot: D/2015/10.500/94

Voorwoord

Onderzoek is een dialoog; het ontstaat en krijgt enkel betekenis door verder te bouwen op de kennis van anderen. Een lijst maken van iedereen die hierbij betrokken was, is overbodig en altijd te kort, maar een aantal van hen verdienen een speciale vermelding voor hun bijdrage aan dit onderzoek. Niet in het minst wil ik mijn beide promotoren bedanken die, elk op hun eigen manier, een zeer belangrijke invloed hebben gehad op dit werk. Prof. Ronny De Meyer gaf mij enkele jaren geleden de kans om een doctoraatsonderzoek op te starten en heeft een belangrijk aandeel in het resultaat vanuit zijn interesse in een digitale ontwerpomgeving op maat van de architect. Prof. Jan Van Campenhout introduceerde mij in het onderzoeksdomein van de informatietechnologie, waardoor het resultaat van dit onderzoek ook functioneel is uitgewerkt; *the proof of the pudding is in the eating*. Tijdens mijn onderzoek werd ik goed omringd door behulpzame collega's. De gesprekken op kantoor met mijn dichtste collega's, Pieter en Ruben, hebben mijn traject sterk bepaald, en onze pauzes op het terras van de Vooruit waren een welkome afwisseling tussen het werken door. Ook de goede samenwerkingen met Francis, Marc, en Sara voor diverse onderzoeksprojecten waren een bijzondere verrijking. Bij uitbreiding bedank ik graag alle collega's en medewerkers van de vakgroepen VAS en ELIS, en ver daarbuiten, voor de fijne en interessante discussies. Ik bedank ook graag de leden van examencommissie: prof. Rik Van de Walle, prof. Patrick De Baets, prof. Sara Eloy, prof. Stefan Boeykens, prof. Francis wyffels, dr. Pieter Pauwels, en dr. Ruben Verstraeten. Geen onderzoek zonder middelen; daarom ook dank aan het agentschap voor Innovatie door Wetenschap en Technologie (IWT) om mij een onderzoeksbeurs voor vier jaar te verlenen. Natuurlijk wil ik ook graag al mijn vrienden en familie bedanken; dit proefschrift is opgevat als een boek met een verhaal om het toegankelijker te maken, en jullie steun en interesse in het wat, hoe en waarom van mijn onderzoek heeft hierin een belangrijke rol gespeeld.

Tiemen Strobbe

Table of Contents

List of Figures

List of Tables

List of Acronyms

A

AEC	Architecture, Engineering and Construction
AGG	Attributed Graph Grammar
AI	Artificial Intelligence

B

BFS	Breadth-first search
BIM	Building Information Modelling

C

CA	Cellular automaton
CA(A)D	Computer-Aided (Architectural) Design

D

DFS	Depth-first search

F

FLW	Frank Lloyd Wright

G

GUI	Graphical User Interface

H

HCI	Human-Computer Interaction

I

IFC	Industry Foundation Classes

R

RdB	Rabo-de-Bacalhau
RdB-tf	Rabo-de-Bacalhau transformation grammar

S

SVM	Support Vector Machine

Nederlandse samenvatting

Talloze informatiesystemen werden de laatste jaren ontwikkeld om diverse aspecten van de creatieve ontwerppraktijk te ondersteunen. Hedendaagse informatiesystemen blijken succesvol in het verbeteren van de kwaliteit en nauwkeurigheid van ontwerpdocumenten, in het verkorten van de tijdsduur om informatie uit te wisselen tussen de betrokken actoren en zelfs in het verbeteren van ontwerpen door ondersteuning te bieden bij het maken van simulaties en berekeningen voor structurele, thermische, akoestische en andere ontwerpaspecten. Toch is de impact van dergelijke informatiesystemen in de praktijk meestal beperkt omdat ze enkel gebruikt worden om een idee uit te werken dat de ontwerper reeds in gedachten heeft, met weinig tot geen interactie tussen computer en ontwerper voor gevolg. In onderhavig proefschrift wordt een andere benadering nagestreefd: hoe kunnen informatiesystemen ingezet worden als *agents*, die zich gedragen als assistent of sidekick van de ontwerper? Als dergelijke functionaliteit voorhanden is, kan een gemengde of zogenaamde *mixed-initiative* onderneming ontstaan waarin zowel de ontwerper als de computer elk op de voor hun meest geschikte wijze bijdragen aan het ontwerpproces.

Terwijl *agents* onder vele vormen bestaan, kunnen informatiesystemen die de exploratie van ontwerpalternatieven ondersteunen of zelf versterken van belang zijn in de architecturale ontwerppraktijk; bijvoorbeeld op het vlak van de meer en meer aan belang winnende prestaties van gebouwen. De metafoor van een *digitaal schetsboek*, waarbij menselijke exploratie wordt versterkt door de (reken)kracht van een computer, is de inzet en het onderzoeksthema van dit proefschrift. Exploratie wordt hierbij opgevat als het zoeken naar eerdere ontwerpen en het genereren van nieuwe ontwerpen in een gestructureerd netwerk, met name de ontwerpruimte. 'Hoe informatiesystemen inzetten op een effectieve manier om de exploratie van architecturale ontwerpruimtes te ondersteunen?', vormt als onderzoeksvraag de kern van dit proefschrift.

Deze onderzoeksvraag wordt in dit proefschrift opgestart aan de hand van een onderzoek naar de centrale principes die aan de basis liggen van hedendaagse computerondersteunde ontwerp systemen (CAD). De oorsprong van het concept van de ontwerpruimte gaat terug op het ontstaan van de discipline van de artificiële intelligentie (AI) tijdens de jaren zestig. Deze discipline initieerde het paradigma van het doorzoeken van probleem- of ontwerpruimtes om tot een oplossing te komen. Bij hoofdrolspelers uit AI en CAD ontstond het gezamenlijke idee een ontwerpvraagstuk te formuleren als een door middel van daarvoor geschikte heuristieken 'op te lossen' zoekprobleem. Het heersende paradigma in de jaren zestig was dat een ontwerptaak kan worden voorgesteld door middel van een ontwerpruimte waarin op zoek kan worden gegaan naar ontwerpen die voldoen aan een aantal vooropgestelde doelcriteria. De sleutel tot intelligentie in dergelijke representaties bestond er volgens de pioniers van de AI in om zo snel mogelijk het pad in de ontwerpruimte te vinden met de minst mogelijke weerstand om van de initiële ontwerpfase tot een eindfase te komen. Vele van de hedendaagse CAD-systemen werken op basis van dezelfde principes van het doorzoeken van een ontwerpruimte. Terwijl het doorzoeken van een ontwerpruimte een belangrijk deel van ontwerpexploratie kan zijn, is dit geen kenmerk dat creativiteit en ontwerp onderscheidt van andere disciplines. In dit proefschrift wordt ontwerpexploratie geïnterpreteerd als een '*wicked*', avontuurlijk, en onbevooroordeeld proces dat gekenmerkt wordt door het ontbreken van een rechtlijnige aanpak. Hedendaagse informatiesystemen zijn niet of slechts in beperkte mate in staat om deze vorm van ontwerpexploratie te ondersteunen, wat resulteert in een ogenschijnlijk dilemma tussen de *wickedness* van creatief ontwerpen en de gestructureerde symbolische eigenschappen van informatiesystemen.

CAD-systemen zouden in staat moeten zijn om de *wickedness* van creatieve ontwerpprocessen te ondersteunen, maar ook om de mogelijkheid tot ontwerpexploratie (zoeken, generatie en navigatie) voor de ontwerper te versterken. Vormgrammatica's (*shape grammars*), zoals ontwikkeld in de jaren zeventig door George Stiny en James Gips, kunnen hier met het ondersteunen van de exploratie van architecturale ontwerpruimtes een belangrijke opstap vormen naar het beoogde digitale schetsboek. Vormgrammatica's geven een niet-traditionele, formele visie op (computer ondersteund) ontwerpen, waarin vormen, in plaats van vooraf gedefinieerde symbolen, een prominente rol spelen. Met deze klemtoon op visuele aspecten leveren vormgrammatica's een belangrijke kritiek op het traditionele CAD-discours. Het is een specifieke klasse van productiesystemen waarin generatieve regels gebruikt worden om een ontwerpruimte te genereren. Deze ontwerpruimte bevat (visuele) ontwerpen die beschreven zijn in een specifieke algebra en die kunnen gegenereerd worden door het herhaaldelijk toepassen van (verschillende) regels in de grammatica. Deze regels coderen, of versleutelen, specifieke ontwerpkennis of ontwerpstappen die onder de vorm van regels kunnen bewaard en gebruikt worden. Bovendien laten vormgrammatica's ook een vorm van visueel denken en ambiguïteit toe die kenmerkend is voor het creatief ontwerpen. Ontwerpen kunnen immers gedefinieerd en geïnterpreteerd worden op een ambigue manier, omdat regels worden toegepast door middel van een partiële ordening. Doordat ontwerpen bij iedere stap opnieuw geïnterpreteerd worden, kunnen *emergente* ontwerpen ontstaan, die niet vooraf gedefinieerd zijn in een grammatica, maar die ontstaan zijn door regels toe te passen. Als gevolg hiervan leveren vormgrammatica's een effectieve manier om een ontwerpruimte te representeren in een computersysteem en tegelijkertijd hangen ze nauwer samen met de *wickedness* van het creatief ontwerpen.

De volgende twee onderzoeksvragen resulteren uit de keuze voor het gebruik van vormgrammatica's als onderliggend raamwerk voor het beoogde digitale schetsboek:

- Welke symbolische representaties van vormen zijn zowel geschikt voor computerimplementatie en behouden ook de essentiële kenmerken eigen aan de ambiguïteit van vormen?
- Hoe kan menselijke exploratie van architecturale ontwerpruimtes ondersteund en zelfs versterkt worden door informatiesystemen?

De eerste onderzoeksvraag behandelt het begrip 'representatie' in CAD. De ontwikkeling van geschikte symbolische representaties voor vormen is een hachelijke onderneming, in het bijzonder om (automatische) detectie van emergente sub-vormen mogelijk te maken en ook om parametrische vormen te herkennen die vergelijkbaar maar niet volledig identiek zijn. In dit proefschrift worden geattribueerde deel-relatie grafen (*attributed part-relation graphs*) geïntroduceerd om grammatica's implementeerbaar te maken op een computer. Doordat dergelijke representatie van grafen toelaat om (parametrische) sub-vormen te herkennen, is het een haalbare en waardevolle manier om grammatica's toe te passen. Bovendien is het een rijkere representatievorm doordat semantische in plaats van louter geometrische objecten kunnen beschreven worden. Een praktische gevalstudie van de implementatie van een bestaande vormgrammatica, die tot nu toe nog niet geïmplementeerd was op een computer toont de haalbaarheid aan van de voorgestelde implementatiemethode en toont bovendien aan hoe een dergelijke geïmplementeerde grammatica een meerwaarde kan vormen voor ontwerpers en architecten.

De tweede onderzoeksvraag behandelt het 'proces' als thema binnen CAD. Terwijl traditionele CAD systemen voornamelijk inzetten op het doorzoeken van vaste ontwerpruimtes en/of optimalisatie van ontwerpen, ligt de focus in dit proefschrift voornamelijk op het veranderen van de wijze van benadering van een ontwerper door middel van exploratie, wat een rijker concept inhoudt dan louter zoeken. Door middel van een literatuuronderzoek worden in dit proefschrift verschillende strategieën voorgesteld die exploratie kunnen versterken: het belang van externe representatie, de codificatie van ontwerpstappen en het belang van implicatie. In de meeste (geïmplementeerde) grammatica's worden deze strategieën ondersteund. Andere strategieën zoals het opslaan (*backup*), terugroepen (*recall*) en hernemen (*replay*) van bestaande ontwerpen en het genereren en visualiseren van nieuwe ontwerpalternatieven worden in veel mindere mate ondersteund. Deze laatste strategieën gaan terug op de representatie van een expliciete ontwerpruimte die de verzameling van ontwerpen omvat die gemaakt zijn door een enkele ontwerper of een ontwerpteam. De expliciete ontwerpruimte werkt als een steeds groeiende bibliotheek van potentieel realiseerbare ontwerpen en reeds verkende ontwerpstappen en kan dus waardevol zijn om ontwerpalternatieven te onthullen en te vergelijken, om eerder werk op te slaan en terug te roepen en om eerder gebruikte ontwerpstappen opnieuw te doorlopen. Dit proefschrift introduceert het concept van boomstructuren om ontwerpen in de expliciete ontwerpruimte te bewaren.

De voorgestelde concepten van de vormgrammatica's, de representatie van vormen als grafen, en de boomstructuren vormen de basis van een nieuw grammatica-gebaseerd CAD-systeem voor de exploratie van ontwerpruimtes — het digitale schetsboek. Het doel van dergelijk CAD-systeem is ontwerpers in staat te stellen de ontwerpruimte van een bepaalde grammatica te exploreren op een visuele en interactieve manier. In dit proefschrift worden de hoofdlijnen van het digitale schetsboek beschreven aangevuld met een aantal visuele voorbeelden en gevalstudies. De drie voornaamste kenmerken hiervan zijn: (1) het representeren en visualiseren van de ontwerpruimte als een geheel, (2) het ondersteunen van een expliciete ontwerpruimte om ontwerpen op te kunnen slaan, terug te roepen, en te hernemen en (3) het ondersteunen van ontwerpers in het creëren en aanpassen van regels op een intuïtieve manier. Een softwareprototype dat werd ontwikkeld en gebruikt om de gevalstudies in dit proefschrift uit te werken, laat tot slot toe de voorgestelde aanpak te valideren.

Dit leidt tot de conclusie van dit proefschrift: het digitale schetsboek kan beschouwd worden als een krachtig hulpmiddel voor de exploratie van ontwerpruimtes en komt hiermee tegemoet aan de centrale onderzoeksvraag. Het resulterende CAD-systeem dient daarbij geïnterpreteerd te worden als een hulpmiddel en net zoals bij ieder hulpmiddel is dit maar zo goed als wie het gebruikt. Met andere woorden, de metafoor van een digitaal schetsboek is een voorbeeld van een *mixed-initiative* onderneming waarin menselijke exploratie wordt versterkt door de (reken)kracht van een computer. Met de vele voorbeelden in dit proefschrift wordt aangetoond dat de resultaten van dergelijke samenwerkingen tussen mens en machine de door de ontwerper of de computer gegenereerde resultaten kunnen overtreffen. De hier beschreven samenwerkingen tussen mens en machine zijn misschien minder ambitieus dan de vroege pogingen van de pioniers van de artificiële intelligentie om een creatief ontwerp 'op te lossen', maar blijken minstens even veelbelovend.

English summary

Over the past years, numerous information systems have been realized to support various aspects of creative design practice; in particular, nowadays information systems prove to be successful in enhancing the quality and accuracy of design documents, shortening the duration time to communicate information among the different actors involved, and even increasing the performance of designs by providing simulation and calculation aids for structural, thermal, acoustical, and other aspects. Yet, the impact of such information systems is often limited, because they are used to execute an idea that designers already had in mind, with little or no interaction between the computer and designer. In this thesis, a different line of thought is followed — namely, how information systems could more closely resemble agents, acting like an assistant or sidekick to the designer. With such agent-like functionality available, it is possible to talk about a mixed-initiative enterprise, in which both the designer and computer contribute to the design task that it does best.

While agent-like design tools may come in many guises, information systems that support — and amplify — exploration of design alternatives may be of particular interest in architectural design; for example, in the context of an increased emphasis on building performance. The metaphor of a *digital sketchbook*, in which human exploration is mixed with computer amplification, is the motivating idea and main research topic of this thesis. The definition of exploration given here involves both searching for previous designs and generating new designs in a structured network called the design space. The central research question of this thesis is — how can information systems effectively support *design space exploration*?

In addressing this research question, this thesis starts by looking into some of the main principles for design space exploration underlying current CAD tools. In fact, the origin of the design space is traced back to the 1960s with the birth of artificial intelligence, which initiated the paradigm of searching a problem space for solutions. Through the shared protagonists of the fields of AI and computer-aided (architectural) design, the act of design was formulated as a form of searching — which can be 'solved'

through appropriate heuristics. In particular, the prevailing paradigm in the 1960s was that a design task could be represented by a design space, in which goal designs could be found through searching this design space. The key to intelligence in this kind of representation, according to the early pioneers, is the ability to quickly find the path with the lowest cost that leads from an initial design to a goal design. To some extent, many of nowadays CAD tools operate under the same principles of searching a design space. However, while searching might be an important part of design space exploration, it is not what characterizes design as a distinct kind of behavior. Instead, design space exploration is often characterized as a 'wicked', adventurous, open-minded, and at times weakly guided activity. Nowadays information systems are not, or only so to a limited extent, able to accommodate this kind of design space exploration, resulting in an apparent dilemma between the wickedness of creative design and the structured, symbolical nature of information systems.

So it turns out that CAD tools for design space exploration should be able to support the wickedness of creative design processes, while also amplifying the designer's capabilities in exploring the design space through search, generation, and navigation. In this regard, the theory of shape grammars, developed by George Stiny and James Gips in the 1970s, might offer an important step towards the envisioned digital sketchbook for supporting design space exploration. The theory of shape grammars stands as a critique of the traditional CAD discourse, and provides a non-traditional, formalized view on design and computation, in which shapes — rather than predefined symbols — play the leading role. In particular, shape grammars are a class of production systems, in which generative rules are used to generate a broad range of designs, which can retroactively be called a design space. This design space contains (visual) designs specified in particular algebras that can be accessed by applying rules. These rules encode specific design knowledge or design moves that can then be stored and used for computation. Moreover, it also incorporates a kind of visual thinking and ambiguity that is characteristic for creative design. Through the part and embedding relations under which rules apply, designs can be defined and interpreted in ambiguous ways. As designs can be (re)-interpreted and decomposed freely, this might result in *emergent* designs, which are designs that that are not predefined in a grammar, but arise from the shapes generated by rule applications. As a result, the theory of shape grammars provides a concise and computable framework to represent a design space, by encoding design moves in the form of rules, and at the same time, it more closely coheres with the wicked nature of design space exploration.

Using shape grammars as the underlying framework for the envisioned digital sketchbook, this leads to the two following additional research questions:

- Which symbolic representations for shapes are suitable for computer implementation, on the one hand, and maintain the essential features resulting from the ambiguous nature of shapes, on the other hand?
- How can human design space exploration be supported, and even amplified, by information systems?

The first research question concerns the issue of 'representation' in computer-aided design. The development of appropriate symbolic representations for shapes is challenging, in particular to enable (automatic) detection of emergent subshapes, and also to handle parametric shapes that may be similar though not completely identical. In this thesis, attributed part-relations graph are introduced and are shown to be a feasible and valuable choice to implement grammars — not only because they enable (parametric) subshape detection, but they also extend shape grammars by also including semantic, instead of purely geometric, objects. A practical case study of implementing an existing shape grammar, originally developed on paper, demonstrates the feasibility of this proposed graph-theoretic implementation approach, and more importantly, shows how designers may benefit from such computer implemented grammars.

The second research question concerns the issue of 'process' in computer-aided design. While traditional CAD tools mainly focus on search and/or optimization, the focus in this thesis is mainly on changing the designer's way of thinking through exploration, which is a much richer concept than merely search. Based on a literature review and analysis, several exploration amplification strategies are pointed out, including external representation, codification of design moves, and implication. While these strategies are commonly found in shape grammar theory and its computer implementations, the strategies of replay, recall, backup, and alternatives — all involving the representation of an explicit design space — are available to a far less extent. The representation of the explicit design space might be valuable to reveal and compare alternatives, to backup and recall prior work, and to replay paths previously discovered in a design space. In this thesis, the concept of tree structures is introduced to keep track of the explicit design space.

The theory of shape grammars, the graph-theoretic representation of shapes and designs, and the tree structure together form the keystones of a new kind of grammar-based tool — the digital sketchbook. The focus of such design tool is on supporting designers to explore the language of a grammar in a visual and interactive way. The outline of such a digital sketchbook, together with several visual examples and case studies, are described in this thesis. The three main aspects are (1) representing and visualizing the design space as a whole, (2) enabling backup and recall strategies, and (3) enabling designers to create, change, or delete rules in an intuitive manner. Finally, the idea of a digital sketchbook has been validated through the development of a software prototype for design space exploration, which has been used to develop and explore the case studies in this thesis.

This leads to the final conclusion of this thesis. The resulting digital sketchbook can be considered as a powerful tool for design space exploration, thereby addressing the central research question of how information systems can effectively support design space exploration. In any case, information systems are to be considered as tools for exploration, and much like any other design tool, they are only as good as the person who is using them. In other words, the digital sketchbook metaphor is to be considered as a mixed-initiative enterprise of human creativity and computer amplification. The results of such a mixed-initiative enterprise might surpass the results generated by either the designer or the computer alone — which is demonstrated in the examples throughout this thesis. Human–machine collaborations, such as the one proposed in this thesis, might perhaps be less ambitious than the early attempts of AI pioneers to 'solve' creative design, but they are certainly as promising.

1 Introduction

1.1 On oracles, draughtsmen and agents

In a Lincoln Laboratory documentary [Morash, 1964], a person draws a line on a computer screen using *Sketchpad* and a light-pen. For many people, this was the very first time they had seen someone interacting with a computer, albeit in a very rudimentary way (Figure 1.1). Sketchpad, developed by Ivan Sutherland [1963], is commonly regarded as the ancestor of modern Computer-Aided Design (CAD) systems, and its conception is considered to be the starting point of CAD research. In the early days of CAD research, there was much optimism about the potential benefits of using computers to support design practice. Architect and technologist Nicholas Negroponte [1970] envisioned a dynamic between human and machine that "*would bring about ideas unrealizable by either conversant alone*". More than 45 years later, the progress made in supporting various aspects of creative design practice is remarkable, to say the least. Nowadays, information systems have proven to be successful in enhancing the quality and accuracy of design documents, shortening the duration time to communicate information among the different players involved, and even in increasing the performance of designs by providing calculation aids for structural, thermal, and other aspects. Indeed, such information systems have matured substantially, gaining acceptance by users throughout the whole Architecture, Engineering and Construction (AEC) community.

Figure 1.1: Drawing a line using Sketchpad and a light-pen. Image taken from the Lincoln Laboratory documentary [Morash, 1964].

On the other hand, none of the current information systems resemble the optimistic human–machine interaction that was envisioned in the 1960s. Lawson [2005] describes the current role of computers in creative design practice as *draughtsmen* — advanced drawing tools for generating production and presentation drawings. In this sense, information systems enhance the quality and accuracy of design documents, and they even enable designers to develop complex ideas that would have been impossible to resolve otherwise. However, such information systems are often limited to executing an idea that the designers already had in mind. There is little or no interaction between the computer and the designer, because the idea is already fixed in the mind of the designer before computers become involved. Therefore, the actual impact of such information systems on creative design practice is limited, and current information systems seem to have never evolved beyond this traditional role of draughtsman.

A far more ambitious role for computers is that of an *oracle* — in which an information system proposes the designs that are optimal for a given design task [Lawson, 2005]. In this case, the role of the designer is merely supportive and limited to revising, developing, or rejecting the idea proposed by the computer. This approach has been popular mainly in academic circles during the first decades of CAD research (1960–1980). Researchers first used computers to create mathematical models for solving space planning and circulation problems, and this focus later shifted to environmental design. For example, Whitehead and Eldars [1964] describe a program to design building layouts by minimizing the walking distance between the different rooms. Examples of computer models for optimizing designs towards environmental aspects are described in the extensive body of work of Radford and Gero [1980]. The approach of using computers as oracles has now fallen out of favor, mainly because the resulting solutions are often too limited in scope to be of practical use, and they have not been found to lead to 'better' designs, as some of the pioneers had predicted.

With the impact of draughtsmen and oracles being too limited, how can computers effectively support creative design practice? Lawson [2005] argues that information systems should be conceived as *agents* — autonomous systems that are able to observe and act upon a given (design) situation. An agent-like system is more like an assistant or sidekick, which provides an alternative point of view to the designer or has (computation) skills that the designer does not have. Neither the role of the computer nor the designer is exclusive in such a *mixed-initiative* interaction, but rather it mixes human intervention with machine generation [Allen et al., 1999; Woodbury and Burrow, 2006]. The term 'mixed-initiative' refers to an interaction strategy, where each agent contributes to the task that it does best — "*At any time, one agent might have the initiative — controlling the interaction — while the other works to assist it, contributing to the interaction as required. At other times, the roles are reversed, and at other times again the agents might be working independently, assisting each other only when specifically asked.*" [Allen et al., 1999]. Such characterization corresponds to the agent-like designer–computer interaction envisioned by Lawson — "*It will need to be able to converse with designers in a way that they find helpful.*" [Lawson, 2005]. The quest for such agents is an ongoing undertaking, approached from a variety of angles, and with no definite answer yet available.

1.2 The need for alternatives?

In his paper "*Creating creativity*", Shneiderman [2000] argues that information systems are capable of supporting creative design. The extensive literature available on creativity offers diverse perspectives on how creative behavior can be enhanced — ranging from structuralist theories using problem-solving methods, to situationist theories emphasizing social context as a key part of creativity, and other theories promoting the playful nature of creativity. Based on a combination of these perspectives, Shneiderman [2000] points out several human–computer interaction strategies for information systems to support creative design:

- Searching and browsing digital libraries;
- Consulting with peers and mentors;

- Visualizing data and processes;
- Thinking by free associations;
- Exploring solutions (what–if tools);
- Composing artifacts and performances;
- Reviewing and replaying session histories; and
- Disseminating results.

For some of these strategies, examples within current information systems can easily be found. For example, rendering tools and more advanced game engines enable designers to visualize their ideas, while calculation and simulation tools allow them to consider performance aspects of their designs, such as structure, energy, or acoustics. On the other hand, other strategies such as searching libraries for previous designs, replaying histories, and exploring novel design alternatives, are more difficult to resolve. In a more recent issue of *Architectural Design* on design computation, Woodbury [2013] points out that the ability to explore both previous and novel design alternatives is limited in current information systems:

> "*Strangely, until very recently, computer interfaces in all fields have provided direct interaction with one model at a time. As they always do when provided with poor tools, people have responded with workarounds. Multiple layers, version of files, scripts with elaborate if–then–else constructs, manually programmed tables of alternatives: all of these attest to an unmet need for better support for alternatives. Perhaps the best demonstration of need is any good designer's sketchbook. Typically these can be read as a story of exploration, of a path through a space of possibilities.*" [Woodbury, 2013].

Indeed, a designer's sketchbook can be read as a series of explorations or history of design moves that were undertaken during a design process [Goldschmidt, 2003]. In other words, a sketchbook provides a visual record of previously explored design ideas or concepts, which may help designers to search, browse, or filter these ideas. Also, the ability to review and replay these ideas and histories of design moves previously created (and perhaps abandoned) may become relevant later in another design context. Sketches contain a certain amount of ambiguity, so that designers are free to make their own interpretations of the forms drawn. On the other hand, the sketchbook is an effective tool for generating new ideas by making rapid sketches. Moreover, it supports experimentation in the sense that sketches can easily be refined, adapted, or even permanently erased should an idea be proven futile. These are the kinds of properties that make sketching such a flexible and attractive tool for designers [Goldschmidt, 2003]. As Woodbury [2013] points out, these kinds of interactions are available to a far less extent in current CAD tools, and are often dealt with using workarounds. The use of multiple versions of files or external spreadsheets are typical examples of such workarounds found in architectural design practice.

1.3 Research question and contributions

In this thesis, we investigate the potential of computers to support — and amplify — the exploration of designs. The metaphor of a *digital sketchbook*, in which human exploration is mixed with computer amplification is the motivating idea and central research topic of this thesis. The definition of exploration given here involves both searching for previous designs and generating new designs in a structured network called the *design space*. In other words, we investigate the feasibility of enabling designers to move within a design space, instead of following a linear sequence of decisions — which is more typical for current CAD software. While this concept of *design space exploration* is less supported by current CAD software, as pointed out by Woodbury [2013], it has gained renewed interest in the current context of increased emphasis on building performance. Building designs now need to comply with multiple standards and regulations (structure, energy, acoustics, and so forth), and also several qualitative aspects. In order to meet specific performance requirements, designers have to create and evaluate design alternatives and their performance. For this reason, the central research question of this thesis is how information systems can effectively support design space exploration, thereby more closely resembling the functionality of an agent.

Throughout this thesis, we question some of the principles underlying current CAD tools and we examine the suitability of alternative models for representing and exploring a design space. A good starting point can be found in the domain of Artificial Intelligence (AI), which is concerned with the study of computer models to represent different kinds of information and knowledge. The concept of the design space can be traced back to the AI paradigm of searching a problem space for solutions. In particular, we investigate more closely a specific rule-based generative model, called *shape grammars* [Stiny, 2006; Stiny and Gips, 1972], developed in the CAD research community in the 1970s. The theory of shape grammars is clearly influenced by concepts drawn from the field of AI; however, it also incorporates a kind of visual thinking and ambiguity that is characteristic of creative design. Using shape grammars as the main research methodology, we investigate two important aspects of CAD — *representation* and *process*. The former involves the apparent dilemma between the structured nature of computer representations and the kind of freedom that is involved with creative design. In particular, a closer look is taken at developing appropriate representations for making shape grammars amenable to computer implementation. The latter involves how the design space represented can be explored, and how human design space exploration capabilities can be amplified in a mixed-initiative interaction. In particular, a new kind of grammar-based tool for design space exploration is outlined, in which the designer can interact with the design space through search, generation, and navigation. As a result, the contribution of this thesis is twofold — it advances the state-of-the-art in the computer implementation of shape grammars, and it contributes to the ongoing discussion on the potential of computers in supporting design space exploration.

1.4 Outline of the thesis

The thesis is divided into 6 chapters, discussing the following topics:

- **Chapter 2 — Design and Computation** traces back the origin of the design space to the 1960s with the birth of AI, which can be considered to be one of the first concerted efforts to explore a problem domain (which can retroactively be called a design space). We give a brief historical overview of how architects and CAD researchers adopted various concepts from AI in an attempt to represent and explore a design space using computers. Throughout this chapter, we point out an apparent dilemma in accomplishing design space exploration between the structured nature of computer representations and the kind of freedom that is involved with creative design.

- **Chapter 3 — Shape Grammars** investigates how shape grammars, a rule-based formalism for generating spatial designs, can be used for design space exploration. The theory of shape grammars is proven to provide a concise framework to represent a design space, while maintaining sufficient freedom to allow for visual thinking and ambiguity which are characteristic for creative design. In particular, we point out how shape grammars can be used to encode design moves in the form of rules to perform exploration in a design space that is implicitly represented by the grammar.

- **Chapter 4 — From Shapes to Graphs** addresses the computer implementation of shape grammars using a graph-theoretic approach. While shape grammars seem to be an obvious candidate for computer implementation, there is a tension between the visual nature of shape grammars and the symbolic nature of computation. We refer to concepts from the field of graph theory and graph transformation to propose a graph-theoretic representation of shape grammars that is computable and effective, without losing the essential features of shape grammars that make them such a powerful tool for design space exploration.

- **Chapter 5 — Design Space Exploration** discusses several amplification strategies for design space exploration in CAD tools. In particular, we describe how these strategies can be achieved in the case of grammar-based CAD tools. Special attention is given here to demonstrate how it is possible to keep track of an explicit design space of previously generated designs, which is like a library of recoverable work. Tree structures are shown to be suitable data structures for enabling navigation in the design space, and for recall and replay of previous designs.

- **Chapter 6 — Digital Sketchbook** describes the prototype software tool for design space exploration, which results from the concepts that are introduced in the previous chapters (shape grammars, graphs, and tree structures). We describe the main functionality and user interface of this prototype, after which we demonstrate, through a number of visual examples, how the prototype can be used in (architectural) design practice. These examples range from analytic to original grammars, and from simple to more complex grammars.

- **Chapter 7 — Conclusions** describes the findings and results of the research performed. First, we point out how the proposed implementation approach and the proposed strategies for amplification advance the state-of-the-art situation in shape grammar research. Second, the theory of shape grammars is also used to better understand the more general concept of the design space, including the complex constraints that often define this design space. As a result, the overall findings are not only situated in the domain of shape grammars, but also might help us forwards on the path to CAD software that operates more like an agent.

1.5 Publications

- T. Strobbe, S. Eloy, P. Pauwels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. A graph-theoretic implementation of the Rabo-de-Bacalhau transformation grammar. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 2015 (in press).

- T. Strobbe, P. Pauwels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. Towards a visual approach in the exploration of shape grammars. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 2015 (in press).

- T. Strobbe, F. wyffels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. Automatic architectural style classification using one-class support vector machines and graph kernels. *Automation in Construction*, 2015 (in press).

- T. Strobbe, R. Verstraeten, M. Delghust, J. Laverge, R. De Meyer, and A. Janssens. Using a building information modeling approach for teaching about residential energy use and official energy performance. In *14th International Conference of the International Building Performance Simulation Association*, 2015.

- T. Strobbe, R. De Meyer, and J. Van Campenhout. A semi-automatic approach for the definition of shape grammar rules. In *Real time - Proceedings of the 33rd eCAADe Conference*, 2015.

- P. Pauwels, T. Strobbe, and R. De Meyer. Analyzing how constraints impact architectural decision-making. In *International Journal of Design Sciences and Technology*, 21(1), pages 93-111, 2015.

- P. Pauwels, T. Strobbe, S. Eloy, and R. De Meyer. Shape grammars for architectural design: The need for reframing. In *The Next City - New Technologies and the Future of the Built Environment*, pages 507-526, 2015.

- M. Delghust, T. Strobbe, R. De Meyer, and A. Janssens. Using BIM-based parametric typologies to supplement single-zone calculations for official performance assessment with multi-zone calculations for predictions on real energy use. In *14th International Conference of the International Building Performance Simulation Association*, 2015.

- T. Strobbe, P. Pauwels, R. De Meyer, and J. Van Campenhout. Design space exploration using a shape grammar implementation. In *Sixth*

International Conference on Design Computing and Cognition, pages 79-80, 2014.

- P. Pauwels, T. Strobbe, J. Derboven, and R. De Meyer. The role of conversation and critique within the architectural design process. In *Sixth International Conference on Design Computing and Cognition*, pages 141-176, 2014.

- P. Pauwels, T. Strobbe, J. Derboven, and R. De Meyer. Analyzing the impact of constraints on decision-making by architectural designers. In K. Zreik, editor, *Architecture, City & Information Design*, pages 97-111, 2014.

- W. Bekers, R. De Meyer, and T. Strobbe. World War I naval camouflage : an evaluation through image analysis. In *Intellectuals and the Great War*, 2014.

- T. Strobbe, R. De Meyer, and J. Van Campenhout. A generative approach towards performance-based design: using a shape grammar implementation. In R. Stouffs and S. Sariyildiz, editors, *Computation and Performance - Proceedings of the 31st eCAADe Conference*, volume 2, pages 627-633. Delft University of Technology, 2013.

- T. Strobbe and R. De Meyer. Generative systems in architectural design. In *FEA PhD Symposium*, 2013.

- V. Mueller and T. Strobbe. Cloud-based design analysis and optimization framework. In R. Stouffs and S. Sariyildiz, editors, *Computation and Performance - Proceedings of the 31st eCAADe Conference*, volume 2, pages 185-194. Delft University of Technology, 2013.

- T. Strobbe, P. Pauwels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. Optimization in compliance checking using heuristics: Flemish energy performance regulations (EPR). In G. Gundason and R. Scherer, editors, *Ework And Ebusiness in Architecture, Engineering and Construction*, pages 477-482. CRC Press/Balkema, 2012.

- P. Pauwels, P. Present, and T. Strobbe. A pragmatic approach towards software usage in construction projects: the port house in Antwerp, Belgium. In G. Gundason and R. Scherer, editors, *Ework And Ebusiness in Architecture, Engineering and Construction*, pages 509-512. CRC Press/Balkema, 2012.

- T. Strobbe, P. Pauwels, R. Verstraeten, and R. De Meyer. Metaheuristics in architecture: using genetic algorithms for constraint solving and evaluation. In P. Leclercq, A. Heylighen, and G. Martin, editors, *Proceedings of the 14th International Conference on Computer Aided Architectural Design (CAADFutures)*, pages 866-867, 2011.

- T. Strobbe, P. Pauwels, R. Verstraeten, and R. De Meyer. Metaheuristics in architecture. In J. Van Wittenberghe, editor, *Sustainable Construction and Design*, volume 2, pages 190-196. Ghent University, 2011.

2 Design and Computation

In this chapter, we trace back the origin of the design space to the birth of artificial intelligence in the 1960s (Section 2.1). The design as a search paradigm quickly fell out of favor, making room for new insights into the wicked nature of creative design (Section 2.2). In this context, we introduce the concept of design space exploration — where exploration is a much richer concept than purely searching. We point out how current CAD tools are unable to support this kind of exploration, resulting in an apparent dilemma between the wicked nature of creative design and the structured nature of computation (Section 2.3).

2.1 The influence of artificial intelligence

The origin of the design space can be traced back to the 1960s with the birth of *Artificial Intelligence* (AI), which initiated the paradigm of searching a problem space for solutions. For researchers in CAD, the AI field has been — and continues to be — a fertile area from which to draw inspiration. The AI field is concerned with the study of representing knowledge in such a form that information systems can be used to perform (complex) tasks with this knowledge. A common definition of AI is the study and design of *rational agents*, which are systems that perceive the environment and act upon that environment to maximize their chance of success [Russel and Norvig, 2010]. The history of AI has known several approaches to achieve this goal — simulating human problem-solving skills, using formal logic, describing knowledge in the form of rules, or using statistical methods. Surprisingly, many of these approaches and AI techniques have influenced (architectural) designers and researchers in CAD, as demonstrated in specialized scientific journals, such as *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* and *Artificial Intelligence in Engineering*.

Of course, artificial intelligence is not the first approach concerned with building intelligent entities. Throughout human history, people have been fascinated with building machines that could support them in some kind of intelligent way. However, it was not until the introduction of the digital computer in the 1950s that the idea of artificial intelligence gained momentum. The foundations for theories about computing and computers were laid in 1936 with the development of the Turing machine — a hypothetical device for symbol manipulation. From this point on, the computer became the artifact of choice for implementing models of intelligence. Researchers were then able to write programs that could actually learn to play checkers [Samuel, 1959] or prove mathematical theorems [Newell et al., 1957]. The term 'artificial intelligence' was first coined at the Dartmouth summer workshop of 1956 [McCarthy et al., 1956], which is considered to be the starting point of AI research. The participants — among whom John McCarthy, Marvin Minsky, Herbert Simon, and Allen Newell — laid the basis for a *problem-solving* paradigm that would dominate AI research for the next few decades.

2.1.1 Problem solving using search and heuristics

At the Dartmouth summer workshop of 1956, Newell and Simon presented the Logic Theorist — one of the first AI programs developed to solve specific problems. In particular, the program was able to prove several mathematical theorems and, in some cases, find shorter proofs than the ones that were available at the time. The early success of the Logic Theorist was quickly followed by the General Problem Solver [Newell et al., 1959] which was developed as a universal problem solver. Unlike the Logic Theorist, this program was designed to mimic human problem-solving skills by creating and solving simpler sub-goals first. The Logic Theorist and the General Problem Solver led to the establishment of problem solving as the canonical task during early AI research [Russel and Norvig, 2010]. This AI paradigm — retroactively called good old-fashioned AI (GOFAI) — was particularly successful at simulating high-level intelligence in small showcase programs [Haugeland, 1985].

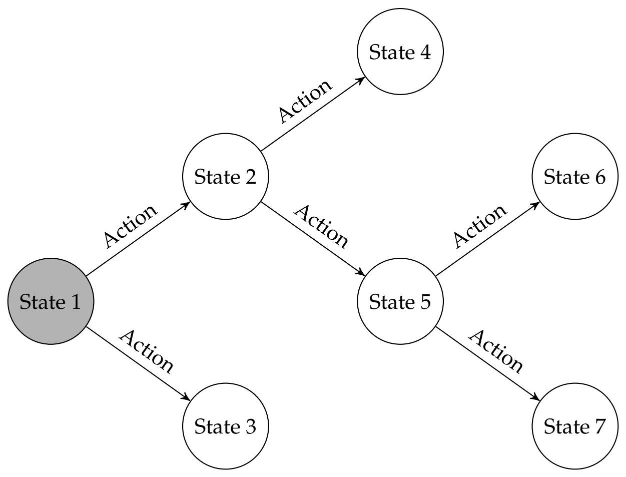


Figure 2.1: An example of a state space consisting of different states which can be reached by performing specific actions. The initial state is indicated in gray.

In order to solve a given problem, such as winning a game of checkers or proving a theorem, many AI programs follow the same principles that were first used by Newell and Simon. In particular, the problem is represented as a *state space* in which a specific goal state is reached from an initial state by performing actions step by step. The problem is defined formally by five components [Russel and Norvig, 2010] — an initial state, a set of available actions, a transition model, a goal test, and a path cost function. The initial state describes an initial hypothesis or situation that forms the starting point of the search process. The set of available actions defines which actions can be performed in a particular state. The transition model returns the state that results from performing an action in a particular state. The initial state, actions, and transition model implicitly define the state space — a directed graph in which the nodes are states and the links between nodes are actions (Figure 2.1).

Every sequence of states connected by actions is a path in the state space. The task of problem solving can now be considered as finding a path that leads from the initial state to a goal state. The goal test determines whether a given state is a goal state, and the path cost function assigns a cost to each path. Every path that leads from the initial state to a goal state is a possible solution to the problem; however, the path with the lowest cost is the optimal solution. For example, in the Logic Theorist program, the initial state is a hypothesis, the goal state is the theorem intended to be proven, and each action corresponds to applying a specific rule of logic.

Every path that leads from the initial hypothesis to the goal theorem constitutes a possible proof of this theorem. As a result, the task of proving a theorem is conceived as finding the shortest path in a state space. In a similar way, playing chess, natural language processing, or solving a puzzle can be conceived as searching a state space to find the shortest path.

Each problem gives rise to a state space that a problem solver — both human and computer — can traverse. The key to intelligence in this kind of representation, according to Newell and Simon, is the ability to quickly find the path with the lowest cost that leads from the initial state to a goal state. The power of information systems is that they can search large state spaces quickly by considering various possible action sequences. This is done by following a specific path step by step, while retaining other options in case the path followed leads to a dead end. Several options on searching a state space are available, of which two common search strategies are depth-first (DFS) and breadth-first search (BFS). The former strategy involves expanding the deepest state first each time, while in the latter strategy every state on a level is traversed before going to a deeper level. In the case of Figure 2.1, DFS would result in the sequence $\{1, 2, 4, 5, 6, 7, 3\}$ and BFS would result in the sequence $\{1, 2, 3, 4, 5, 6, 7\}$. Another interesting way to search state spaces is by using a branch and bound (BB) algorithm, in which states are enumerated systematically by checking first whether a given path can produce a better solution than the best solution found so far. If this is not the case, then the current path is discarded and another path is expanded. Other kinds of (uninformed) search strategies include uniform-cost search, depth-limited search, and iterative deepening DFS [Russel and Norvig, 2010].

Newell and Simon also realized that the state space of real-world problems is often of such order of magnitude that the number of possible action sequences becomes intractable [Newell et al., 1958]. In technical terms, many problems to be solved are part of the complexity class *NP*, which is the class of problems for which a solution can quickly be verified (in polynomial time), but for which no efficient algorithm is known to find the solution. On the other hand, the complexity class *P* contains the problems that are tractable, which means that they can be solved efficiently. In order to solve real-world problems (which are mostly in *NP*), Newell and Simon used *heuristic functions* to trim paths that would be unlikely to lead to a goal state [Polya, 1945]. A heuristic function calculates an estimated distance to a goal state and, therefore, guides the search process by eliminating the paths with the highest estimated path cost. Figure 2.2 demonstrates how some heuristic function might inform search in the state space. For example, Shannon [1950] devised a heuristic function to evaluate moves

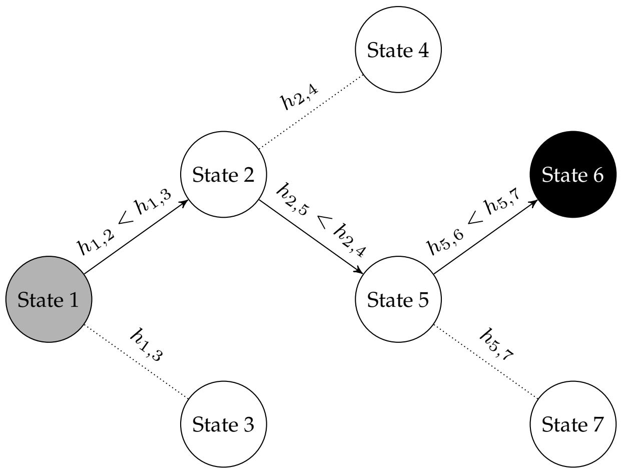


Figure 2.2: Searching a state space using a heuristic function (h) to estimate the path cost to the goal state (black). At each node in the state space, the path with lowest estimated path cost is followed.

in chess by assigning relative values to each piece (queen, rook, bishop, knight and pawn). The benefit of using heuristic search over uninformed search is that this uses problem-specific knowledge to find solutions more efficiently. The flexibility and efficiency of heuristic search comes at a price; at best, it results in a near-optimal solution, whilst a full optimum cannot be guaranteed, because such algorithms often get stuck in local optima. Search algorithms lie at the core of so-called *problem-solving agents*.

2.1.2 Design as a form of searching

At the same time of the birth of artificial intelligence, several influential design schools, journals, and practices started to question the position of the designer as an 'artist', in favor of more rational and objective design methods. This has led to a long-running debate on the relationship between creative design disciplines and science and mathematics. For example, several proponents of the scientific approach, among whom Christopher Alexander, Lionel March, and Leslie Martin worked together at the University of Cambridge [Keller, 2005]. They were among the first researchers and architects who attempted to establish architectural design as a scientific field in post-war Britain. The need to revise the position of the designer arose from a sense of war-time inferiority — "*The extremities of war had forced to the surface many doubts about architecture as a significant profession.*" [Keller, 2006]. Moreover, this apparent need to legitimize the architectural design profession was strengthened by several post-war technological developments [Bayazit, 2004]. Architects then had to deal with new user needs and (automated) production systems, requiring a more 'scientific' approach that allowed decision-making based on objective and quantifiable criteria.

An answer to the need for scientific design approaches was provided in the work of Herbert Simon — one of the early AI pioneers who chose design as the area to demonstrate his science of the artificial. In his book *The Sciences of the Artificial*, Simon [1969] argues that creative design can be considered as a specific kind of problem solving. According to Simon, the functionality of the human mind is similar to the functionality of a computer — "*The evidence is overwhelming that the [human information-processing] system is basically serial in its operation: that it can process only a few symbols at a time and that the symbols being processed must be held in special, limited memory structures whose content can be changed rapidly.*" [Simon, 1969]. In other words, Simon assumes that creative design is open for explicit formalization and can be treated in a similar way as other problem-solving tasks, such as playing checkers, proving a theorem, or solving a puzzle. In particular, he considers creative design to be an ill-structured problem, as opposed to well-structured problems — "*It will generally be agreed that the work of an architect presents tasks that lie well towards the ill-structured end of the problem continuum. Of course this is only true if the architect is trying to be creative.*" [Simon, 1973]. Nevertheless, Simon demonstrates that ill-structured and well-structured problems can both be solved using systematic and rational problem-solving techniques, such as the ones underlying the Logic Theorist and the General Problem Solver.

Such a formulation of creative design as a specific kind of problem solving laid the foundation for AI in design as a form of searching. For creative design specifically, the term 'design space' has been adapted from the field of AI. In the view of Simon, the activity of design can be formulated as the act of searching a design space in order to find designs that satisfy specified criteria expressed in the form of constraints or objectives. Using an appropriate search strategy, a designer would then be able to find a near-optimal solution to the design problem posed. In order to overcome a combinatorial explosion of possibilities, heuristic search can be employed to control the search process or even generate near-optimal designs, in a reasonable amount of time. A key aspect in Simon's theory on design is that a real-world design situation should be formulated as a well-structured design space in order to apply problem-solving techniques, such as heuristic search (Figure 2.3). This process is called the disambiguation or abstraction of the (design) situation as a well-structured problem.

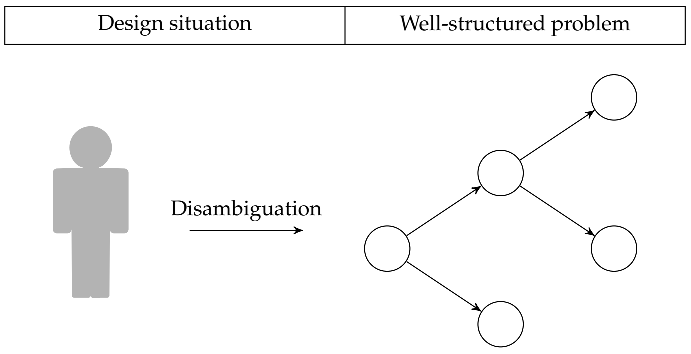


Figure 2.3: The formulation of a design task as a well-structured design space, in which rational problem-solving techniques, such as heuristic search, can be used.

While AI indeed provided a fertile area to draw inspiration from, the paradigm of design as search had already come to the surface in earlier design research on operation research and (numerical) optimization. The field of operations research, which arose during World War II, investigated mathematical models to search for near-optimal solutions in complex organization systems [Churchman et al., 1957]. These mathematical models were picked up by the design methods movement, which was one of the first concerted efforts to explore design alternatives in a design space [Jones and Thornley, 1962]. The contribution made by AI to the paradigm of design as a search was to expand the domain of search computation to a wider range of symbolic representations. Through the use of symbolic representation, search was not only applicable to numerical aspects of design, but the same problem-solving techniques could be used to address other formulations of design. A notable example was expert systems, which became the focus of mainstream AI research in the 1980s. An expert system is built from logical if–then rules that are derived from expert knowledge. Using these rules, expert systems were able to solve problems in a specific knowledge domain. The underlying principles of search remained unchanged, but rules allowed for a wider range of representations. These new developments inspired the design research community, for example in the conception of shape grammars [Stiny and Gips, 1972] — a specific class of rule-based systems for geometric shapes.

2.1.3 AI in architectural design

Christopher Alexander was one of the first architects in the early 1960s who applied the AI paradigm of design as search to architectural design practice. Much like Simon's theory on design, Alexander conceived (architectural) design as a problem that can be solved with heuristic problem-solving techniques — using the computer to implement and solve these problems. His ideas were strongly influenced by the fields of cognitive science, cybernetics, and artificial intelligence — to which he was introduced during his studies at Harvard University [Steenson, 2014]. The published version of his dissertation *Notes on the Synthesis of Form* [Alexander, 1964] quickly became one of the founding works of the design methods movement and has been, and continues to be, very popular in academic circles. However, Alexander's theory is often overlooked by architectural theorists and historians alike due to the underlying anti-architect theme that is found in many of his writings [Steenson, 2014].

In the introduction of *Notes on the Synthesis of Form*, Alexander dislikes the idea of a designer relying "*on his position as an artist, on catchwords, personal idiom, and intuition*" [Alexander, 1964], and proposes a more rational problem-solving approach. In particular, he defines the process of design as being "*an effort to achieve fitness between two entities: the form in question and its context.*" [Alexander, 1964]. In Alexander's words, the context defines the problem, the form is the solution to the problem, and a good fit is the desired property of the solution. In his characterization of design, Alexander defines fitness in its negative form — a list of requirements that should be neutralized in order to achieve a good fit between the context and the form. These requirements cannot be satisfied separately due to the potential interaction between some requirements. By achieving fitness for one particular requirement, this might influence several other requirements, either in a positive or negative way. Figure 2.4 (left) graphically shows an example of design requirements (points) and some existing interactions (links). In order to achieve fitness for as many requirements as possible (which is the goal of the design task), Alexander proposes to break down the requirements into a number of smaller subsets that are, at the lowest level, fairly autonomous.

According to Alexander, the breaking down, or decomposition, of the requirements into a number of small and independent subsets is an important phase of the design process. Alexander argues that the appropriate structure for such decomposition is a tree. For every design problem, a large number of possible decompositions or trees can be devised, though some decompositions make more sense than others. As Alexander points out — "*For every problem, there is one decomposition which is especially proper*

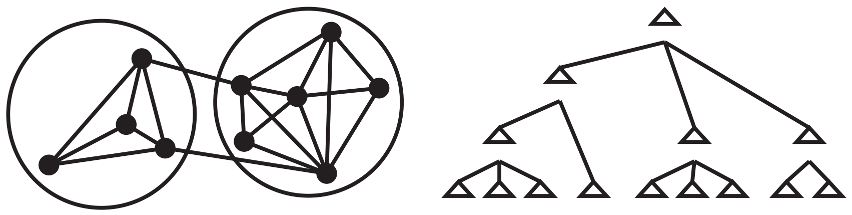


Figure 2.4: *The figure on the left shows an example of 10 design requirements (points) and some existing interactions (links). This system can be decomposed into two subsets (circles) that operate rather autonomously, though a large number of other decompositions can be devised. The figure on the right demonstrates how a decomposition is represented as a tree structure. Reproduced from the original images appearing in Alexander [1964].*

to it" [Alexander, 1964]. In order to find the most suitable decomposition, Alexander conceives the task of finding it as a state space of possible decompositions in which search techniques can be applied. The HIDECS 2 program, designed by Alexander and Manheim [1962] for solving set theory problems, was able to find an appropriate decomposition for problems (with approximately 150 requirements) by using a heuristic search algorithm called hill climbing. The essence of the HIDECS 2 search algorithm is to find graph decompositions with the least number of interactions between them. The result of the search process is a decomposition or tree that consists of fairly autonomous subsets (Figure 2.4 right).

With an appropriate decomposition available, a design can be solved on the lowest level, for each autonomous subset of requirements individually. A solution to a particular subset of requirements is called a diagram, which in the later work of Alexander [1977] evolved into the concept of (design) patterns. A pattern describes a solution to a small system of interacting requirements that is independent of all other requirements. The concept of patterns that could be reused was a highly influential idea in various disciplines — more so than the original application of patterns in the field of architecture. When each autonomous subset is solved using a particular design pattern, these can then be recombined into compound patterns on a higher level of the tree structure, which Alexander called the synthesis of form. As a result, the theory of Alexander conceives the act of design as searching for autonomous subsets of requirements that can be solved and recombined into the larger whole. Alexander demonstrates the application of the method to the design of an Indian village with 141 design requirements — a number that is too large to be solved manually, but small enough to be solved with the computation power then available.

Alexander is one of the first architects who used AI techniques to solve a given design task by searching and solving simpler problems first. Obviously, the AI paradigm of designing as a search has inspired many other designers, architects, and researchers. The impact on architectural design practice during the period 1960–1980 is investigated in the work of Steenson [2014] on the architects Christopher Alexander, Cedric Prince, Nicholas Negroponte, and the Architecture Machine Group at MIT. Also, designing as a search was the predominant AI paradigm in the research on CAD, reflected in the papers of the time. Many of these research efforts can be found in specialized scientific journals, such as *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* and *Artificial Intelligence in Engineering*, which later continued as *Advanced Engineering Informatics*.

2.2 Second thoughts

During the mid 1970s, both the field of AI and the design methods movement experienced setbacks. Just like many technologies that progress on the hype cycle [Fenn and Raskino, 2013], the field of AI could not live up to the bold claims that were made by the early pioneers. When the ambitious expectations of AI were not met, researchers in AI experienced severe cutbacks in funding, which in turn led to a so called *AI winter* — a period in which little research progress was made in this field. The 1970s also witnessed the breakdown of the design methods movement, at least in the field of architectural design. Some of the pioneers disassociated themselves from the movement; for example, Alexander said in an interview that "*There is so little in what is called design methods that has anything useful to say about how to design buildings that I never even read the literature any more. I would say forget it, forget the whole thing.*" [Alexander, 1971]. Even Jones, one of the leaders in the early research on design methods, reacted against the foundations on which design methods were established [Jones, 1977].

2.2.1 Design as a unique discipline

There was a developing concern that design as a search paradigm did not sufficiently capture the kind of thinking involved in creative design. Instead of trying to fix design into the straitjacket of rationality and objectivity, researchers started to acknowledge the distinguishing characteristics of design. For example, Rittel and Webber [1973] described several obstacles in problem-solving approaches in the field of social planning, which are also valid in other fields of creative design [Conklin, 2006]. "*We shall want to suggest that the social professions were misled somewhere along the line into assuming they could be applied scientists — that they could solve problems* *in the way scientists can solve their sorts of problems. The error has been a serious one.*" [Rittel and Webber, 1973]. The tasks dealt with in creative design practice are inherently different from problems that scientists deal with and, therefore, they are classified as *wicked problems* by Rittel and Webber — in contrast to *tame problems* in science. Wicked problems are, among others, characterized by the lack of a definite problem formulation, by the absence of an immediate and ultimate test for the solution (they are not right or wrong), by having an innumerable set of possible solutions, and every wicked problem is essentially novel and unique.

For wicked problems, knowledge about the problem itself is achievable only by iterating subsequent problem formulations and solutions. Maher and Poon [1996] coined the term 'co-evolution' to denote this process, in which both the problem and the solution are evolving simultaneously. Iterative (re-)formulation of the problem is an important step, which is often overlooked in AI-based design approaches. Indeed, an important critique on the design-as-search paradigm is its reliance on a fixed problem formulation — which is often not part of the search process itself, but is a result of designer judgment that takes place before a search is involved. For example, in the work of Alexander, the determination of which requirements and interactions should be taken into account is done before a search can be applied. This choice corresponds to problem formulation, and is always the result of designer judgment — "*We shall say that two variables interact if and only if the designer can find some reason (or conceptual model) which makes sense to him and tells him why they should do so.*" [Alexander, 1964]. In other words, a large (and perhaps the most important) part is completed by applying human judgment before any computers are even involved. Due to the 'wickedness' of design, the set of requirements and interactions cannot be determined definitely but is subject to changes as the design process proceeds. Similarly, in response to the General Problem Solver program, AI critic Hubert Dreyfus pointed out that "*since insight is necessary in solving complex problems ... we should not be surprised to find that in the work of Newell and Simon this insightful restructuring of the problem is surreptitiously introduced by the programmers themselves.*" [Dreyfus, 1972]. These examples signify the problematic nature of trying to capture design situations in explicit and unambiguous representations. Instead, design (like other hard AI problems) is a process that cannot be constrained to the bounds of the initial problem formulation, but may require changes in the problem statement itself. Such absence of a definite problem formulation is one of the key characteristics of the wickedness of creative design, and one of the main reasons why the design-as-search paradigm fails.

2.2.2 The design space revisited

The renewed insights into the nature of creative design did not lead to the rejection of the design-as-search paradigm, but led to a reformulation and expansion of the design space concept. Results from empirical and cognitive studies of designers indicated that the early AI models of searching a fixed symbolic state space could not adequately capture the wickedness of creative design. As Logan and Smithers [1992] noticed in an article on design as exploration — *"In fact one might almost say that the defining characteristics of design problems is that they are not amenable to purely search-based problem solving techniques. This is not to deny that problem solving forms part of the design process. However, it is not what characterizes design as a distinct kind of intelligent behavior."*. Logan and Smithers, both researchers of the *Artificial Intelligence in Design* research program at the University of Edinburgh, proposed an exploration-based model of design [Smithers et al., 1990], in which knowledge about the structure of the design space has to be obtained before goals can be adequately formulated. This determination of the structure of the design space is an act of *exploration* — an adventurous, open-minded, and at times only weakly-guided activity prior to search.

The model of design as exploration as proposed by Logan and Smithers is visually demonstrated in Figure 2.5. Using this model, a design task starts with the construction of an initial design requirement description (R_i) that is incomplete and inconsistent and, therefore, cannot define a goal state in design space. The actual design process (E_d) is the exploration of the design space — *"a collection of concurrent and serial searches each interconnected by intuitive leaps, analytical assessments, syntheses, simulations, pro-*

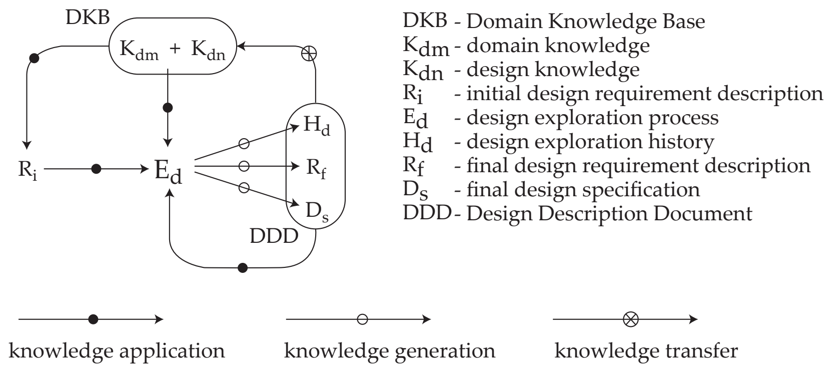

DKB
K_{dm} + K_{dn}
R_i
E_d
H_d
R_f
D_s
DDD
DKB - Domain Knowledge Base
K_{dm} - domain knowledge
K_{dn} - design knowledge
R_i - initial design requirement description
E_d - design exploration process
H_d - design exploration history
R_f - final design requirement description
D_s - final design specification
DDD - Design Description Document
knowledge application
knowledge generation
knowledge transfer

Figure 2.5: A model of design as exploration. Reproduced from the original image appearing in Smithers et al. [1990].

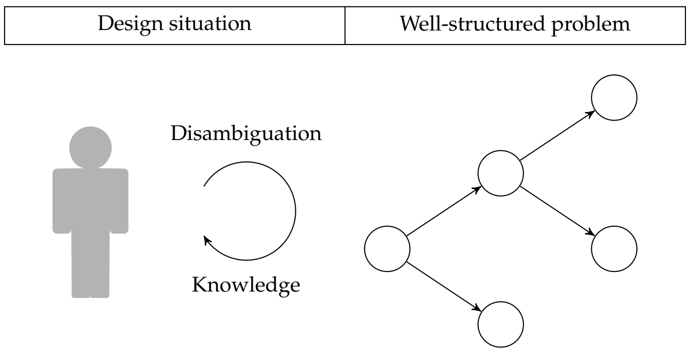


Figure 2.6: A single cycle in the iterative process of formulating and searching a design space, by which new design knowledge is accumulated.

totypes, decisions, choices, skilled and experiential judgments, etc. In other words, an adventure." [Smithers et al., 1990]. This results in a design requirement description (R_f) and design specification (D_s) that are complete and consistent. The former describes what kind of behavior is required and the latter describes how the specified design should work. Together with the record of what parts of the design space were explored (H_d), a design description document (DDD) is defined that represents the knowledge generated and is used for a specific design task.

By means of exploring several design space formulations, a feedback loop is established by which new design and domain knowledge (K_{dn} and K_{dm}, respectively) is accumulated. Instead of using a fixed representation of the design situation at hand, the design process is characterized by continuously (re-)formulating and searching a design space (Figure 2.6). As Logan and Smithers point out — "*The formulation of the problem at any stage is not final: rather it reflects the designer's current understanding of the problem. As the design progresses, the designer learns more about possible problem and solution structures as new aspects of the situation become apparent and the inconsistencies inherent in the formulation of the problem are revealed. As a result, designers gain new insights into the problem (and the solution) which ultimately result in the formulation of a new view: the problem and the solution are redefined."* [Logan and Smithers, 1992]. The process of exploring a design space continues until the incremental gain in knowledge has become insignificant, too costly, or until the available resources (mainly time) have become exhausted.

This process of iterative refinement of both the design problem and the solutions (co-evolution) is a well-studied phenomenon in the field of design thinking and theory. For example, Cross [1997] describes design thinking as an oscillation between problems and solutions — "*During the design process, partial models of the problem and of its solution are constructed side-by-side.*" [Cross, 1997]. In his theory of the designer as a 'reflective practitioner', Schön [1983] describes the importance of 'reframing', which refers to the habit of designers of continuously making representations of the design situation at hand, thereby framing the design situation into a new perspective. The act of framing is not as clear as problem formulation, because the design task can only be fully understood through several attempts to change it. Schön [1988] also points out that "*the work of framing is seldom done in one burst at the beginning of a design process*", but rather it is a continuous process. These kinds of findings are also supported by empirical research work; for example, Goel and Pirolli [1992] conclude, based on protocol studies, that problem formulation reoccurs at regular intervals throughout the design process.

2.2.3 Design space exploration

One of the early definitions and usages of the exploration model in the field of CAD can be found in the work of Gero [1994; 1996]. In this definition, the act of exploration is represented by a change in the design space that either extends or substitutes (a part of) the original design space. More recently, Woodbury and Burrow describe design as exploration in CAD as the idea that "*computers can usefully depict design as the act of exploring alternatives. This involves representing designs in a network structure termed the design space, and exploring this space by traversing paths in the network to visit both previously represented designs and to find sites for new insertions in the network.*" [Woodbury and Burrow, 2006]. In other words, Woodbury and Burrow assume that designers address a design task, as understood in the interpretation of a wicked problem, through *design space exploration* — they continuously represent and search an evolving design space with the aim of finding an adequate design solution to an evolving design problem. In the broadest sense, design space exploration involves several cycles of:

- formulating the structure of the design space;
- searching new designs by traversing paths in the design space;
- navigating between existing designs;
- evaluating designs against goal states.

According to Woodbury and Burrow [2006], design space exploration rests on several premises. Not only it is an effective and feasible basis for computer support, it is also a compelling model from a cognitive point of view. Goldschmidt [2006], for example, defines design space exploration on a more cognition-oriented basis, and argues that exploration is an important part of an inquiry or experiment that designers undertake. These inquiries or experiments are to be understood in the context of Schön's [1983] theory on the designer as a reflective practitioner. Schön describes design as a combination of different kinds of experiments: exploratory experiments, move-testing experiments and hypothesis testing. The inquiry starts by formulating or 'framing' the design situation, after which designers can perform exploratory experiments to either find new design solutions (move testing) or alter their exploration strategy (hypothesis testing). Under the effect of these inquiries or experiments, both the design problem and corresponding solutions co-evolve within the design space.

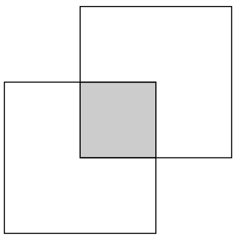
2.3 A digital sketchbook?

In the last couple of years, research on design space exploration has gained renewed interest due to an increasing emphasis on building performance in AEC. The architectural design and renovation process of contemporary buildings is subject to numerous building requirements that are specified in various building codes and standards. Such requirements — including thermal and structural specifications, safety and accessibility regulations, acoustic standards, and others — have a significant impact on the design process. In order to meet these building requirements, designers need to create and evaluate several design alternatives. Information systems might play an important role to support designers in accomplishing these building requirements by guiding them through a design space of possible alternatives. With building performance back on the architectural agenda, the AI models used in the 1960s serve as a warning for the possible pitfalls of using the AI paradigm in design practice.

In current CAD research, the unbridled optimism and bold promises of early pioneers have given way to more cautious and less ambitious attempts. Instead of attempting to support all aspects within one, the quest for agent-like information systems is now being approached from a more pragmatic angle. To name just a few recent examples, the work of Aksoy et al. [2015] describes a heuristic search method to provide decision support for the site layout design of social housing, and Borges and Fakury [2015] describe a method for the generation of complex geometric structures. The "*all-singing, all-dancing, fully integrated, multi-disciplinary design* *decision support system*", as criticized by Maver [1995], seems to have faded away. This has resulted in the incorporation of many novel technologies in CAD tools — all developed with a specific task in mind, and often operating from the background. Mitchell et al. [2003] nicely point out how CAD tools, by definition, are supposed to be supportive, but like any other tool (sketches, drawings, and scale models), they cannot solve all problems — "*Because creativity is associated with novelty, comprehensive tools for creative work will be neither possible nor necessary to develop, any more than it is necessary for a pencil to include all functions for drawing.*"[Mitchell et al., 2003]. In this regard, the digital sketchbook envisioned in this thesis is conceived as tools — one of many — for exploring design alternatives.

To some extent, common CAD modeling environments (AutoCAD[1], Microstation[2], BricsCAD[3], etc.) can be used as tools for design space exploration. Such tools allow designers to represent their design, after which they can evaluate different viewpoints, material choices, etc. Based on the feedback received from the model, the designer might choose to further elaborate, modify, or entirely discard his or her model. This act of subsequently adapting the model corresponds to searching a design space that is implicitly represented by the modeling environment. In this perspective, CAD modeling tools operate on the same principles of searching a state space. The problem here is that this design space is heavily bound by the information structure of the particular modeling environment. This information structure determines how a design model should be constructed; for example, three-dimensional CAD tools enable the designer to use particular operations on three-dimensional geometric objects (including boxes, spheres, and surfaces), and building information modeling (BIM) tools enable the user to model a design using semantic elements (including walls, windows, and columns). When a design model is built according to a particular logic, the model cannot accommodate changes that were not foreseen initially. For example, if two overlapping squares are drawn, a third square 'emerges' from the two overlapping squares (Figure 2.7 top). On the other hand, Figure 2.7 (bottom) shows how only the two original squares — and not the emergent square — are represented in DWG (a common file format for storing geometric data). This example demonstrates how CAD tools restrict designers into seeing designs from a new perspective (reframing). For each geometric structure built up, the model becomes more and more rigid — leaving the designer no choice but to completely rebuild the model if some major changes are necessary.

[1]`http://www.autodesk.com/products/autocad/`
[2]`http://www.bentley.com/products/microstation/`
[3]`http://www.bricsys.com/bricscad/`



```
⟨ AcDbPolyline ⟩        ⟨ AcDbPolyline ⟩
num points   4          num points   4
point2d      (0,0)      point2d      (1,1)
point2d      (2,0)      point2d      (3,1)
point2d      (2,2)      point2d      (3,3)
point2d      (0,2)      point2d      (1,3)
```

Figure 2.7: Two overlapping squares result in a third emergent square (indicated in gray). When this shape is constructed by drawing two squares, this emergent square is not represented in the DWG file format, and thus is unavailable for further computation.

2.3.1 Generative design tools

The metaphor of the design space and design space exploration is made explicit in CAD tools that enable designers to build *generative design* models. In generative design, either a set of parameters or rules is used to generate alternative solutions based on predefined goals and constraints. By this means, the generative design model represents a design space that can be explored; for example by selecting different parameter values or applying different rules. In both academic circles and creative design practice, generative design is becoming increasingly popular. The availability of new (visual) programming environments or scripting capabilities incorporated in traditional CAD software packages make it easy for designers with little programming knowledge to build such generative design models. For example, Processing[4] is a programming environment that serves as a software exploration sketchbook for designers and visual artists. Grasshopper[5] is a visual programming language that enables designers to build parametric models and explore geometric variations of this parametric model. Also, generative design is taught at many schools of architecture, and is gaining ground in architectural and design practice. For example, the architectural firm Foster + Partners has an in-house specialist modeling group to develop and implement generative design models for particular architectural projects [De Kestelier, 2013].

[4]`http://www.processing.org/`
[5]`http://www.grasshopper3d.com/`

2.3.2 Parametric models

Most generative design is based on parametric modeling — the process of representing a design through parameterized components and relationships. Examples of parameterized components include dimensions, formulas, geometric objects, and others. The relationships between the components are structured in a hierarchical chain of dependencies defined by the designer. Based on this hierarchy, some parameterized components act like inputs to the model, while other components are dependently variable. In this way, a parametric model has the structure of a directed graph in which the input values are propagated through the graph. When the input parameters are modified, the model updates itself to reflect the modification, while the graph structure of the model remains consistent. Variations of the input parameters result in different geometric variations of the model. More details about parametric modeling can be found in the work of Aish and Woodbury [2005].

Where traditional CAD tools aim to represent a single design, parametric modeling allows for the divergence of a design space in order to generate geometric variants of the model. The act of searching the design space of the parametric model corresponds to changing the parameter values. Several advantages of parametric modeling are described by Aish and Woodbury — "*Positively, parameterization can enhance the search for designs better adapted to context, can facilitate the discovery of new forms and kinds of form-making, can reduce the time and effort required for change and reuse, and can yield better understandings of the conceptual structure of the artifact being designed.*" [Aish and Woodbury, 2005]. Indeed, the designer might spend less time on producing different design representations, because a parametric model accounts for a design space of alternatives that can easily be generated by changing the input parameters of the model. An example of a typical parametric model in Grasshopper is shown in Figure 2.8. The input parameters — rotation, number of vertical elements, height, a loft curve, and a center point — are visualized on the left, the dependent components and the relationships between them are visualized in the center, and the resulting output is visualized on the right. When the input parameters are modified, either by adjusting a slider value or choosing a geometric object, these changes are propagated through the graph, resulting in different geometric realizations of the model.

Despite the advantages of parametric modeling in searching a design space, this design space is again bounded by the information structure of the model built up. Only the design alternatives within the boundaries of the represented design space can easily be generated (by changing the input parameters), while others require more profound changes in the para-

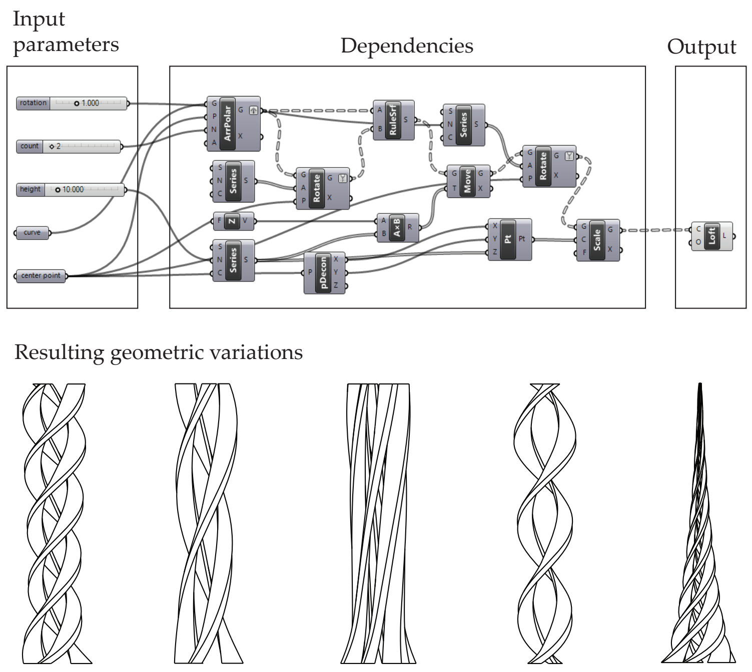


Figure 2.8: The top figure shows a parametric model, built in Grasshopper, driven by five input parameters — rotation, number of vertical elements, height, a loft curve, and a center point. The bottom figure shows some geometric variations of the parametric model.

metric model itself — such as changing the components or dependencies between them. In order to really support exploration, in its broadest sense, it should be easier to accommodate such profound changes. However, there is growing evidence that exploration is quite limited in parametric models used in practice — “*While it is alluring to think of a design representation flexible enough to reduce time spent remodeling, the reality — a reality commonly not addressed — is that parametric models are often quite brittle. Frequently I find my models have grown so tangled they can no longer accommodate even the most trivial change. Often I just start again.*” [Davis, 2013]. Nevertheless, parametric modeling is a promising approach in finding better tools for design space exploration, and several improvements are still being developed, such as the use of re-usable patterns for parametric design [Woodbury et al., 2007], the work of Shireen et al. [2012] on parallel generation and editing of design alternatives, and the work of Tidafi et al. [2011] on using backtracking with parametric models.

2.3.3 Rule-based models

Another kind of generative design tools is based on rule-based modeling — the process of representing a design space through generative rules. The mathematical foundations of rule-based systems were laid in the work of Emil Leon Post on *production systems*. Production systems, in the definition of Post [1943], consist of a set of initial strings and a set of productions (rules) that can be applied in order to generate new strings. Post also pointed out the generative power of productions, and demonstrated that complex systems can be generated with productions of a much simpler form. Production systems were later used to provide some form of artificial intelligence by representing expert knowledge from a certain application domain in the form of rules. Typically, productions are defined as if–then statements in which the conditions (if-part) of a production must match a given state in order to execute the action (then-part) of this production.

A special case of production systems are generative grammars found in the field of linguistics, which are originally described by Noam Chomsky in his book *Syntactic Structures* [Chomsky, 1957]. This type of grammar is defined as a 4-tuple $\langle V, \Sigma, P, S\rangle$, where V is a finite set of symbols called the vocabulary, Σ is a set of terminal symbols, S is a designated symbol called the start symbol, and P is a finite set of rules of the form $(\alpha \rightarrow \beta)$. For example, the grammar

$$\begin{array}{l}
V = \langle S, NP, VP, N, V, ADV\rangle \\
\Sigma = \langle colorless, green, ideas, sleep, furiously\rangle \\
S = \langle S\rangle \\
\begin{array}{l|l}
S \rightarrow NP\ VP & NP \rightarrow A\ NP \\
NP \rightarrow A\ N & VP \rightarrow V\ ADV \\
N \rightarrow ideas & A \rightarrow \langle colorless, green\rangle \\
V \rightarrow sleep & ADV \rightarrow furiously
\end{array}
\end{array}$$

generates the sentence "*Colorless green ideas sleep furiously*" by iteratively applying rules to the start symbol. Using such grammar, all grammatical sentences of a language can be generated, however, these do not necessarily have understandable meaning — as is the case for the sentence given.

The theory of generative grammars influenced CAD research that took production systems or linguistics as the main motivation underlying their formulations. For example, Stiny and Gips [1972] developed their theory of shape grammars in close alignment with such rule-based systems. In the theory of shape grammars, rules are used to implicitly represent a de-

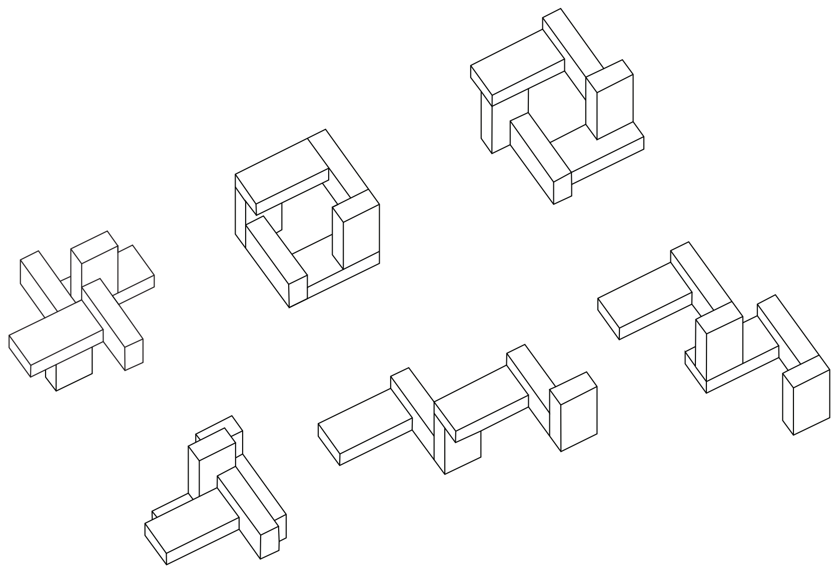


Figure 2.9: Some variations of one spatial relation between volumetric building blocks of the Froebelean type. Image produced using GRAPE, a web-based shape grammar modeling tool [Grasl, 2013].

sign space that can be explored by applying different rules of the grammar. In contrast to parametric models, shape grammars can generate design alternatives that supersede purely parametric variations, including topological configurations and semantic aspects. As a result, shape grammars provide a concise and computable framework to represent a large and complex design space with a fairly small number of shape rules. For example, Figure 2.9 shows some results generated with a single rule that describes a particular spatial relation between three-dimensional building blocks of the Froebelean type. These variations are produced using GRAPE[6], which is a web-based modeling environment based on the shape grammar paradigm. Another notable example is CityEngine[7], a grammar-based modeling approach to generate large-scale architectural models (Figure 2.10. The rules of the grammar are defined in the form $id : predecessor : cond \rightarrow successor : prob$, where each rule has a unique identifier id and is selected with a probability $prob$, $predecessor$ is a shape that is to be replaced with $successor$, and $cond$ is a conditional statement in order for the rule to be applied [Müller et al., 2006].

[6]`http://grape.swap-zt.com/`
[7]`http://www.esri.com/software/cityengine/`

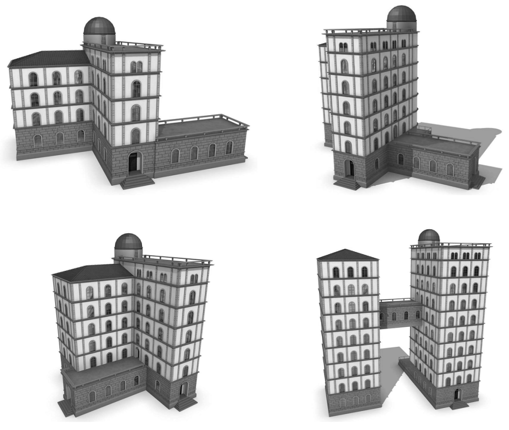


Figure 2.10: Some geometric variations of a grammar-based model, presented in the work of Müller et al. [2006]. The grammar generates variations of a building mass model, after which façade details are created. Reproduced from the original image appearing in Müller et al. [2006].

2.3.4 The dilemma of design and computation

Although information systems for generative design, provided in the form of parametric or rule-based modeling, allow for the divergence of a design space of alternatives, the functionality of such information systems resembles more the act of searching a design space, rather than exploration. Indeed, exploration in the broadest sense of reformulating the design space requires a substantial remodeling effort that can often be accommodated less easily. In particular, the design space is heavily bound by the information structure of a particular modeling environment (by using predefined types) — making it hard for a designer to see designs from a new perspective. For example, reformulating the design space in parametric models requires profound changes to the dependencies of the parametric components and, similarly, changes to rule-based models become more difficult to manage for more complex rule sets. The continuous reformulation of the design space is, however, an important step to support creative design processes, which are inherently wicked.

As a result, there might be an apparent dilemma between the wickedness of creative design and the structured nature of information systems. Creative design is often associated with metaphors like ‘thinking outside the box’, or ‘getting rid of preconceptions’, in order to stress the importance of reformulation. When searching for creative solutions, people prefer not to ’follow the herd’, but rather ‘explore new avenues’ to come up with new ideas. On the other hand, information systems derive their usefulness by relying on explicit problem formulations and a bounded range of solutions. This dilemma was already pointed out by Sketchpad-inventor Sutherland in 1975:

> *"To a large extent it has turned out that the usefulness of computer drawings is precisely their structured nature and that this structured nature is precisely the difficulty in making them. An ordinary draftsman is unconcerned with the structure of his drawing material. Pen and ink have no inherent structure. They only make dirty marks on paper. The behavior of the computer-produced drawing, on the other hand, is critically dependent upon the topological and geometric structure built up."* [Sutherland, 1975].

In order to enhance generative design systems for supporting design space exploration, and thereby achieving a more agent-like role for the computer in a design process, there are several key aspects that should be taken into account. First and foremost, generative design systems should allow an iterative (re)-formulation of the design space, corresponding to the wickedness of creative design. Furthermore, the constructed design space to be explored should be sufficiently large and diverse in order to provide the designer with adequate alternatives. Finally, appropriate functionalities should be available that augment the ability of the designer to interact with the design space in different ways, including visualizing the design space, browsing previously generated designs, comparing design alternatives, and others. In the context of these requirements, the theory of *shape grammars* [Stiny, 2006; Stiny and Gips, 1972] offers a well-studied approach that might bridge the gap between design and computation. Indeed, shape grammars stand as a critique of the traditional CAD discourse, and provide a non-traditional, formalized view on design and computation, in which shapes — rather than predefined symbols or types — play the leading role. The theory of shape grammars might offer an important step towards the envisioned digital sketchbook for supporting design space exploration, which is the main topic discussed in Chapter 3.

3
Shape Grammars

In this chapter, we introduce the concept of shape grammars and discuss some of its key characteristics (Section 3.1). Several examples of shape grammars for analysis and original design are also provided (Section 3.2). Furthermore, we point out how shape grammars provide a concise framework to represent a design space by encoding design moves in the form of shape rules (Section 3.3).

3.1 Definitions

The foundations of the theory of shape grammars were laid in a seminal article *Shape Grammars and the Generative Specification of Painting and Sculpture* by George Stiny and James Gips [1972]. Shape grammars are a specific class of production systems that operate on shapes, using generative rules to describe a family of designs. In the field of creative design, shape grammars are the most commonly used formalism for analyzing and generating (families of) designs in the form of shape rules. They were introduced by Stiny and Gips [1972] as a way of analyzing and synthesizing paintings, but soon adopted for other purposes as well. For example, the Palladian grammar by Stiny and Mitchell [1978a] initiated work on other shape grammars for specifying architectural design corpora, including shape grammars for the architectural corpus of Frank Lloyd Wright [Koning and Eizenberg, 1981], Glenn Murcutt [Hanson and Radford, 1986]

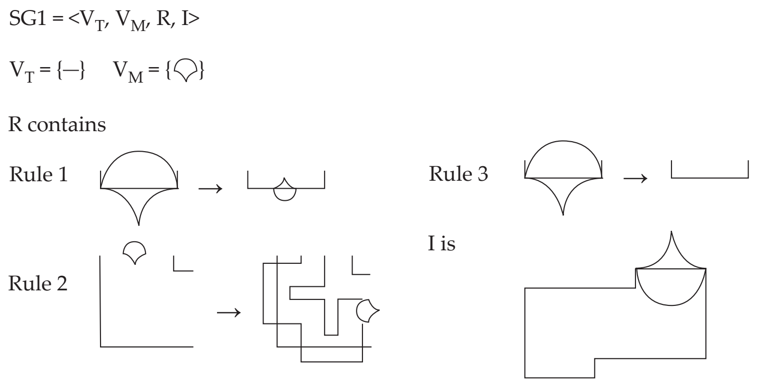


Figure 3.1: A shape grammar ($SG1$) for the generation of a specific class of paintings. Reproduced from the original image appearing in Stiny and Gips [1972].

and Álvaro Siza [Duarte, 2005a], and for the vernacular styles of Queen Anne houses [Flemming, 1987], traditional Turkish houses [Çagdas, 1996] and traditional Portuguese houses [Eloy, 2012]. Examples of other shape grammar application domains include engineering [Geyer, 2008; Schaefer and Rudolph, 2005; Shea and Cagan, 1999], decorative arts [Knight, 1980; Stiny, 1977], and product design [Agarwal and Cagan, 1996; e Costa and Duarte, 2014].

Stiny and Gips based their initial definition of shape grammars on linguistic analogies: "*Where phrase structure grammars are defined over an alphabet of symbols and generate one-dimensional strings of symbols, shape grammars are defined over an alphabet of shapes and generate n-dimensional shapes*" [Stiny and Gips, 1972]. Phrase structure grammars are a specific kind of generative grammars to specify the syntax of languages [Chomsky, 1957]. Indeed, Stiny was influenced by this formalized theory of linguistic structure during his study period at the Massachusetts Institute of Technology. In particular, Stiny and Gips [1972] defined shape grammars in close alignment to generative grammars — a shape grammar (SG) is a 4-tuple $\langle V_T, V_M, R, I \rangle$ where:

- V_T is a finite set of terminal shapes;
- V_M is a finite set of non-terminal shapes or marker shapes;
- R is a finite set of shape rules of the form $u \to v$;
- I is an initial shape that is part of $V_T \cup V_M$.

Figure 3.1 shows a shape grammar ($SG1$) for the generation of paintings, developed by Stiny and Gips. The shapes in the set $V_T \cup V_M$ are the basic elements for the definition of shape rules in the set R and the initial shape I. New shapes are generated from a shape grammar by iteratively applying shape rules to the initial shape. A shape rule is described as an 'if–then' statement $u \to v$, and can be applied if a shape is detected that is geometrically similar to the pattern shape u (if-part) under a certain geometric transformation (translation, rotation, reflection, scaling, or other transformations). The application of a rule involves replacing the transformed shape with the replacement shape v (then-part). Each rule application results in a new shape upon which new rules can potentially be applied. The generation process is terminated when no further rule can be applied. Figure 3.2 shows an example derivation of $SG1$, starting from the initial shape I. The resulting shape from this derivation cannot be transformed any further, because the marker shape V_M is deleted in the last step. Other derivations than the one shown in Figure 3.2 can be performed, each resulting in a new shape that is part of the grammar's language.

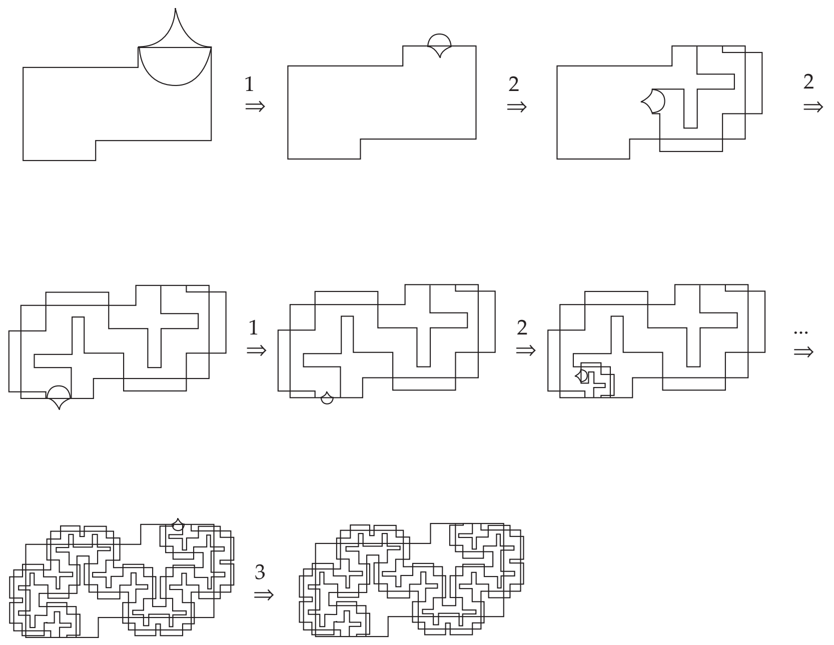


Figure 3.2: An example derivation of $SG1$, starting from the initial shape I. The resulting shape is part of the language of $SG1$. Reproduced from the original image appearing in Stiny and Gips [1972].

Since the seminal work on shape grammars by Stiny and Gips [1972], several other definitions have appeared in the literature — each having a different focus and reflecting a particular understanding of the shape grammar formalism. A chronological survey of the development of shape grammar definitions is presented by Yue and Krishnamurti [2013]. In general, the subsequent definitions evolved from a rather rigid and formal definition to less rigid formalizations. Noticeably, while shape grammars were initially defined in close resemblance to phrase structure grammars, in the later work of Stiny [2006], the formal definition of a shape grammar as a 4-tuple is nowhere to be found. The author questions the analogy between sentences generated by phrase structure grammars and designs generated by shape grammars — "*When I started thinking about grammar and language in design, I had no idea that it was a bad mistake*". According to Stiny, the main issue with the linguistic analogy is the lack of an equivalent for 'words' in shape grammars. He argues that there is no need for a predefined vocabulary of parts in design, but parts may change freely any time rules are applied. Therefore, designs are no longer defined in terms of components that are given in advance ($V_T \cup V_M$), but they are interpreted on the fly. This characteristic — the absence of a fixed vocabulary— is what distinguishes shape grammars from their symbolic counterparts.

3.1.1 Shapes: basic elements and algebras

Shapes play an essential role in the renewed formulation of shape grammars [Stiny, 2006]. A shape is composed of *basic elements* — points, lines, planes, and solids — though this can be extended to curves, surfaces or other geometric elements. The properties of basic elements and how they can be arranged are summarized in *algebras of shapes*. In the general case, algebra involves the study of (mathematical) symbols and how they can be manipulated. A shape algebra U_{ij} consists of basic elements defined in dimension i = 0, 1, 2 or 3, which are points, lines, planes, and solids, respectively. The basic elements can be arranged in an Euclidean space with dimension $j \geq i$. For example, shapes in the algebra U_{12} consist of lines arranged in a two-dimensional space, and shapes in the algebra U_{23} are made up of planes arranged in a three-dimensional space. The shapes in Figure 3.3 are defined in the algebras U_{02}, U_{12}, and U_{22}, and can be built solely from points, lines, and planes, respectively. Each shape reveals the same arrangement of two overlapping squares, but using different basic elements.

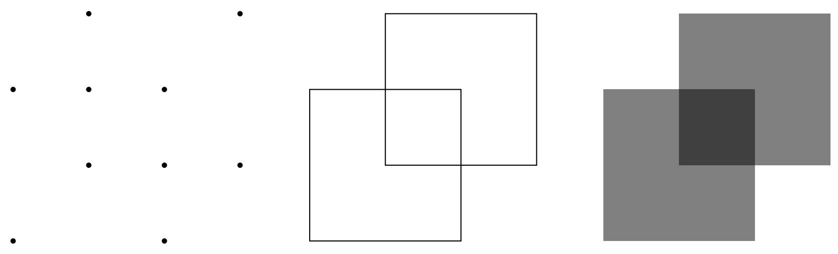


Figure 3.3: Three shapes defined in the algebras U_{02}, U_{12}, and U_{22}, respectively.

The shapes contained in a particular algebra U_{ij} share some common properties. First, the dimension of the basic elements (index i) specifies how shapes of different algebras are related to each other via a *boundary operator*. The boundary of a shape in an algebra with high-dimensional basic elements ($i > 0$) is another shape in an algebra with index $i - 1$. In other words, the boundary of a shape made up of lines, planes, or solids is a shape containing points, lines, or planes, respectively (see Table 3.1). Second, the dimension of the basic elements specifies how shapes in specific algebras can be decomposed into smaller elements (the topology of shapes). Shapes in the algebras with high-dimensional basic elements ($i > 0$) consist of an uncountably infinite set of basic elements. In other words, shapes that consist of lines, planes, or solids can be composed from an infinite number of smaller lines, planes, or solids, respectively (see Table 3.1). For example, the middle shape in Figure 3.3 can be built from at least eight lines; however, it can also be composed using smaller line segments. On the other hand, the left shape in Figure 3.3 is built from eight points, and contains only a single topology. Shapes with high-dimensional basic elements are commonly represented as a set of *maximal elements*. This set of maximal elements contains the smallest number of the largest possible basic elements that are needed to compose the shape. Maximal elements cannot be contained in, overlapping with, or adjacent to other maximal elements.

Algebra	Basic elements	Boundary shapes	Number of parts
U_{0j}	points	none	finite
U_{1j}	lines	U_{0j}	uncountably infinite
U_{2j}	planes	U_{1j}	uncountably infinite
U_{3j}	solids	U_{2j}	uncountably infinite

Table 3.1: Some properties of shapes in different algebras.

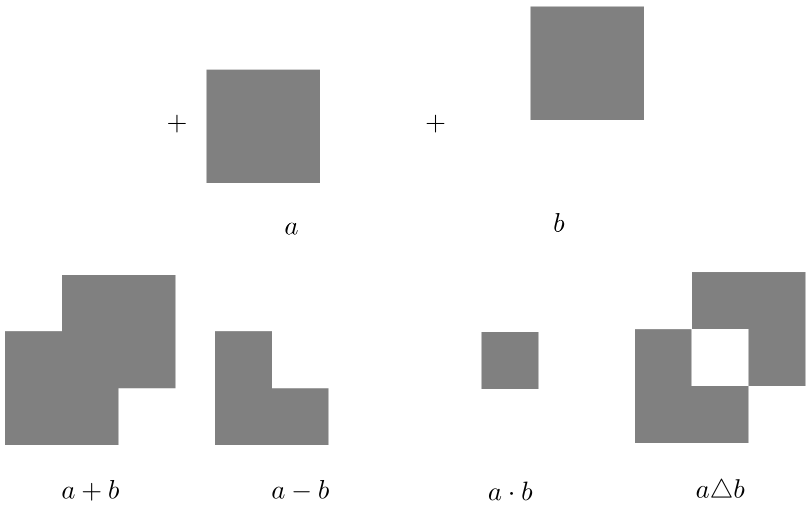


Figure 3.4: The results of sum, difference, product, and product difference operations on two identical square shapes. A registration mark (+) fixes the spatial relation.

A shape algebra U_{ij} specifies how shapes can be manipulated using different kinds of Boolean operations, including sum, difference, product, and symmetric difference. For shapes in the algebras with zero-dimensional basic elements ($i = 0$), the definition of sum and difference is straightforward. When shapes in the high-dimensional algebras ($i > 0$) are added, however, the maximal elements of these shape may fuse if they (partly) overlap. As a result, the maximal elements of shapes may not be preserved if they are added together. For example, the sum of two identical shapes results in an identical shape, because the maximal elements overlap completely. The same holds true for the other operations, such as difference, product, and symmetric difference. Given two identical square shapes a and b, the results of the four Boolean operations (sum, difference, product, and product difference) are shown in Figure 3.4.

3.1.2 The embedding and part relations

The *embedding relation* describes how the basic elements of shapes interact with each other. A basic element is embedded in another basic element, if the first completely overlaps the second. Therefore, every basic element of dimension that is greater than zero ($i > 0$) has infinitely many basic elements of the same kind embedded in it. Mathematically, the embedding relation is defined as a partial order satisfying three conditions:

- reflexivity — every basic element is embedded in itself;
- antisymmetry — if two basic elements are embedded in each other, they are identical;
- transitivity — for three basic elements where each is embedded in the next, the first basic element is also embedded in the last.

Similarly, the *part relation* describes how shapes interact with each other. A shape is part of another shape if every maximal element of the first is embedded in a maximal element of the second. In other words, a shape that is part of another shape is called a *sub-shape* of this other shape. Shapes in the algebras U_{ij} where i is larger than zero can be decomposed into infinitely many parts or sub-shapes. As a result of such partial ordering, a shape is considered to be a discrete topology or hierarchical composition of smaller parts or sub-shapes. As Stiny [1994, 2006] has shown, these topologies should not be fixed, but they can change freely anytime needed. As an example, Figure 3.5 shows a shape defined in the algebra U_{12} and some of its underlying topologies. In particular, this shape can be seen as a composition of two squares (first row), four triangles (second row), two K-shapes (third row), or infinitely many other sub-shapes. In contrast, shapes containing points (U_{0j}) have a finite number of sub-shapes, because points cannot be divided into smaller elements.

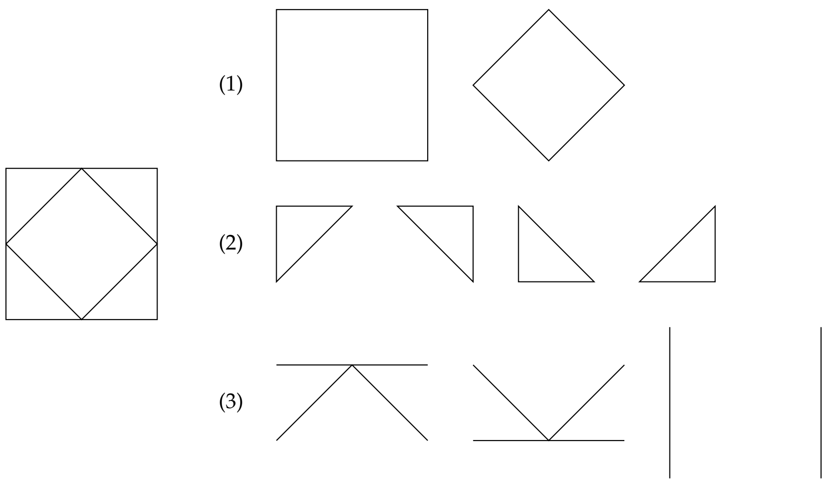


Figure 3.5: A shape defined in the algebra U_{12} (left) and some of its underlying topologies (right).

Although shapes are identified uniquely by maximal elements, the embedding and part relations allow shapes to be decomposed into different sub-shapes. In other words, shapes are not defined in terms of fixed components that would limit the number of parts embedded. This is a key difference with earlier definitions of shape grammars by Stiny [1980], in which shapes are formed by predefined components of the set $V_T \cup V_M$ (the so-called vocabulary). In the current formulation, any sub-shape embedded in the shape is accessible, independent of how the shape is created. As a result, shapes can be (re)-interpreted in a large number of ways, depending on what is needed at the time. Each interpretation reveals different parts of a shape to which shape rules can be applied, which is the real power behind the shape grammar theory.

3.1.3 Shape rules

In the theory of shape grammars, any action on shapes is encoded in the form of a *shape rule*. Shape rules are mechanisms for operating with shapes, independent of the algebra they belong to. In particular, they describe how a specific shape (the *pattern shape*) can be transformed into another shape (the *replacement shape*). Any pair of specific shapes, shown one after the other, determines a shape rule. The pattern and replacement shape are conventionally separated using an arrow ($\rightarrow$) to indicate the action between the two shapes. As a result, shape rules follow the most general rule schema $a \rightarrow b$, in which a and b designate the pattern shape and replacement shape, respectively [Stiny, 2011]. The most common actions described in shape rules involve the Boolean operations sum (for adding shapes) and difference (for deleting shapes). Additionally, registration marks ($+$) are used to specify the spatial relation between the two shapes. An example of a shape rule is shown in Figure 3.6. The rule specifies that, if a square is found in a given shape, a copy of this square will be translated along its diagonal axis.

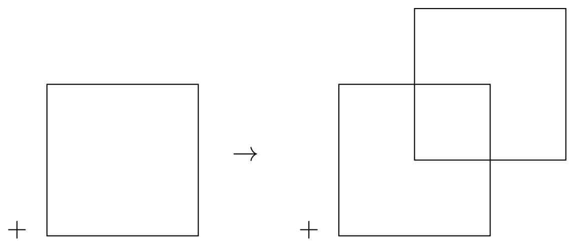


Figure 3.6: A shape rule to translate a copy of a square along its diagonal axis.

Shape rules are applied using the embedding and part relations. In other words, a rule $a \rightarrow b$ can be applied to a given shape c if the pattern shape a of the rule is a sub-shape (or part) of the given shape. If this is the case, the maximal elements of the (sub-)shape and the pattern shape are embedded in each other, and the two shapes are said to be equivalent. Moreover, this equivalence relation can be determined using different *transformations*, including translation, rotation, reflection, scaling, and other linear or non-linear transformations. Wortmann [2013] distinguishes four different sets of transformations that are commonly used for shape rules — identity, isometry, similarity, and topology. For identity, two shapes have to be exactly the same (including position), so no transformations are allowed. Isometry includes the set of Euclidean transformation (translation, rotation, and reflection), thus preserving the size and proportions of shapes. Similarity also consists of scaling transformations, thus retaining only the size of shapes. Other kinds of transformation, such as affine and projective transformations, do not preserve the size or the proportion of shapes. Such transformations are often called parametric transformations, because they only preserve the topology of the shape instead of the geometric characteristics. The division between isometry and topology transformations reflects the usual separation between shape grammars and *parametric shape grammars*, originally defined by Stiny [1980]. In this case, the shapes a and b contain parameters (such as coordinate geometry) to which different values can be assigned. Table 3.2 shows some equivalent shapes to a given shape using different sets of transformations, from identity to topology — with identity the most and topology the least strict.

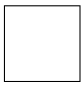
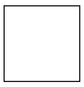
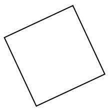
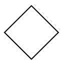
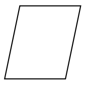
Given shape	Transformations				
	Identity				
	Isometry	″			
	Similarity	″	″		
	Topology	″	″	″	

Table 3.2: A given shape and some equivalent shapes using different sets of transformations (identity, isometry, similarity, and topology).

Shape rules are applied recursively using a particular set of transformations, the part and embedding relations, and the sum and difference operations. In particular, a shape rule can be applied if there is a match between the pattern shape of this rule and a part of a given shape. Given an initial shape and a set of shape rules, the application of a shape rule involves the following three steps:

- Find a transformation (t) such that the pattern shape a is equivalent to (a part of) the initial shape c;
- Define a new shape by subtracting the transformed pattern shape from the initial shape $c - t(a)$; and
- Obtain the final shape by adding the transformed replacement shape $c + t(b)$.

As a result, a new shape $c' = c - t(a) + t(b)$ is generated to which rules can be applied. An example derivation of the translation rule (see Figure 3.6) is shown in Figure 3.7. Where rules are indicated using a single arrow ($\rightarrow$), a double arrow ($\Rightarrow$) is used to indicate a derivation step. For this particular example, similarity transformations (translation, rotation, reflection, and scaling) are used to match the shape rule to the given shape. Each application of this particular shape rule generates additional (sub)-shapes, to which rules can keep being applied. Without an additional rule to terminate the derivation, an infinite number of shapes can be generated.

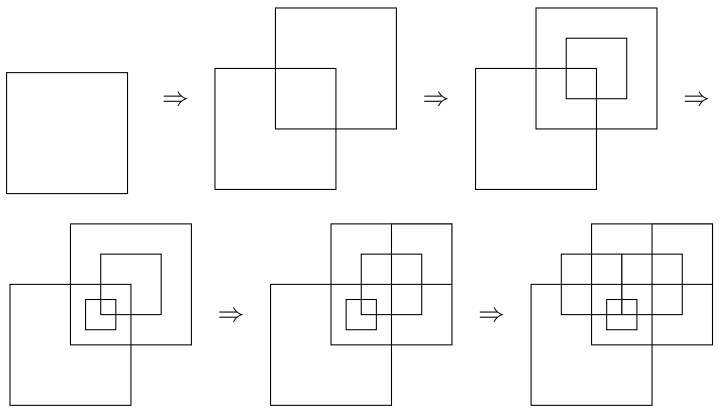


Figure 3.7: A possible derivation of the shape rule in Figure 3.6. A derivation step is indicated with a double arrow (⇒).

3.1.4 Some extensions

So far, the examples given in this chapter have only dealt with shapes, specified in the algebras U_{ij}. While shapes indeed play an important role in describing form and spatial relations in the context of creative design, designs cannot be reduced to shapes alone. In fact, designs are often associated with other kinds of representations that are non-visual, such as functional, social, or aesthetic descriptions. These aspects influence design in different ways and are often expressed verbally. In the view of design associated with the theory of shape grammars, shapes and descriptions go hand in hand — "*Design is drawing — true enough, it is calculating with shapes and rules. Yet most of the time words are involved, too, to say what designs are for and to connect them to other things.*" [Stiny, 2006]. In this sense, it might be preferable to talk about 'design grammars', because they involve more than just shapes, but the term shape grammar is more commonly used.

The algebras U_{ij} can be extended with label and weight algebras V_{ij} and W_{ij}, respectively. Labels may be used to associate semantic meaning to shapes, and weights may be used to incorporate shape properties, such as color, area (for points), thickness (for lines), and texture (for planes). Moreover, composite shape algebras can be formed by combining different algebras. In this case, basic elements of different kinds are arranged in multiple dimensions with different layers. For example, the composite algebra $U_{02} \cdot V_{12}$ contains both points and labeled lines in a two-dimensional space. As a result, algebras and their combinations provide a broad range of tools for specifying shapes and, by extension, designs.

Finally, shape grammars are not limited to two-dimensional shapes, but can also include three-dimensional shapes. For example, Heisserman [1991] describes a shape grammar, or more correctly a solid grammar, that generates a "*three dimensional analogue to Koch snowflakes*". The Koch snowflake is one of the earliest fractal curves described. A fractal is a set

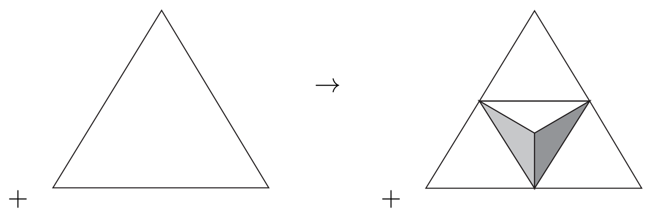


Figure 3.8: A shape rule for the generation of three-dimensional Koch snowflakes. Reproduced from the original image appearing in Heisserman [1991].

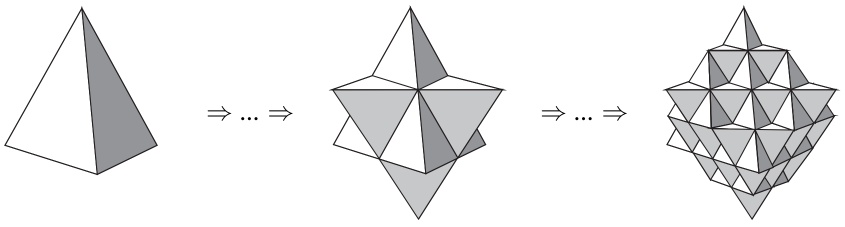


Figure 3.9: An initial shape and a possible derivation using the shape rule in Figure 3.8. Reproduced from the original image appearing in Heisserman [1991].

of mathematical equations intended to generate a geometric pattern that is repeated at every scale. In the mathematical sense, fractals are continuous curves that are not differentiable and which have an infinite surface area. Interestingly, some natural phenomena display or approximate fractal forms (such as trees, crystals, and geographic patterns). A shape rule from this Koch grammar, which adds a three-dimensional pyramid shape to a triangular plane of a given solid, is shown in Figure 3.8. Starting from an initial pyramid shape, this rule can be applied an infinite number of times, because each rule application creates new (smaller) planes to which rules can be applied. A possible derivation is shown in Figure 3.9.

3.2 More ambitious grammars

The shape grammar examples shown so far illustrate some of the most important aspects of the shape grammar formalism — algebras and maximal elements, the embedding and part relations, and shape rules. In recent years, many more specialized shape grammars have been developed for both analysis and original design purposes. Shape grammars of the former kind attempt to analyze an existing corpus of designs and capture the *design rationale* of this particular corpus in the form of rules. Such grammars provide an explanatory description of the corpus in the form of shape rules. Shape grammars of the latter kind focus on the generation and exploration of original designs. Such grammars are used to study the variations of original designs, while maintaining a certain coherence. In shape grammars for analysis, rules are used mainly for their descriptive power, while in shape grammars for original design, rules are used as generative devices. As a result, shape grammars offer a framework that unifies the storage and analysis of existing designs with the creation of new ones. The numerous formal studies available demonstrate the analytic and generative power of shape grammars.

3.2.1 Shape grammars for analysis

The earliest shape grammar applications focus almost exclusively on the analysis of the existing corpora of designs. This analysis includes both the classification of designs and the prediction of design characteristics. A design is classified either as part of a corpus or not, depending on whether it can be generated with the rules that describe this corpus. On the other hand, an analytic shape grammar can be used to predict design characteristics that follow from this grammar. The first application for analysis purposes is a shape grammar for Chinese ice-ray designs described by Stiny [1977]. This grammar demonstrates the descriptive power of shape grammars by using just a few rules to capture the compositional logic of ice-ray designs. This approach was quickly adopted in other application domains, including architectural design, engineering and product design.

The first analytic architectural shape grammar is the Palladian grammar by Stiny and Mitchell [1978a]. This grammar is an analytic shape grammar in the sense that the authors attempt to capture, in the form of rules, the underlying design principles of floor plans of villas appearing in Palladio's *Four Books of Architecture*. This grammar also generates novel designs that are not in the original corpus, however. The rules of this grammar are structured in distinct stages — grid definition, exterior wall definition, room layout, interior-wall realignment, principal entrances, exterior ornamentation, windows and doors, as well as termination. The first set of rules describes the construction of a rectangular grid with bilateral symmetry relative to a north–south axis. Once the grid is generated, it is circumscribed by a rectangle to define the exterior walls of the floor plan. An example of a three-by-three grid is shown in Figure 3.10.

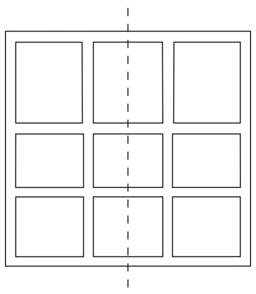


Figure 3.10: An example of a three-by-three grid generated using the Palladian grammar by Stiny and Mitchell [1978a]. An axis (dashed line) denotes the bilateral symmetry of the floor plan.

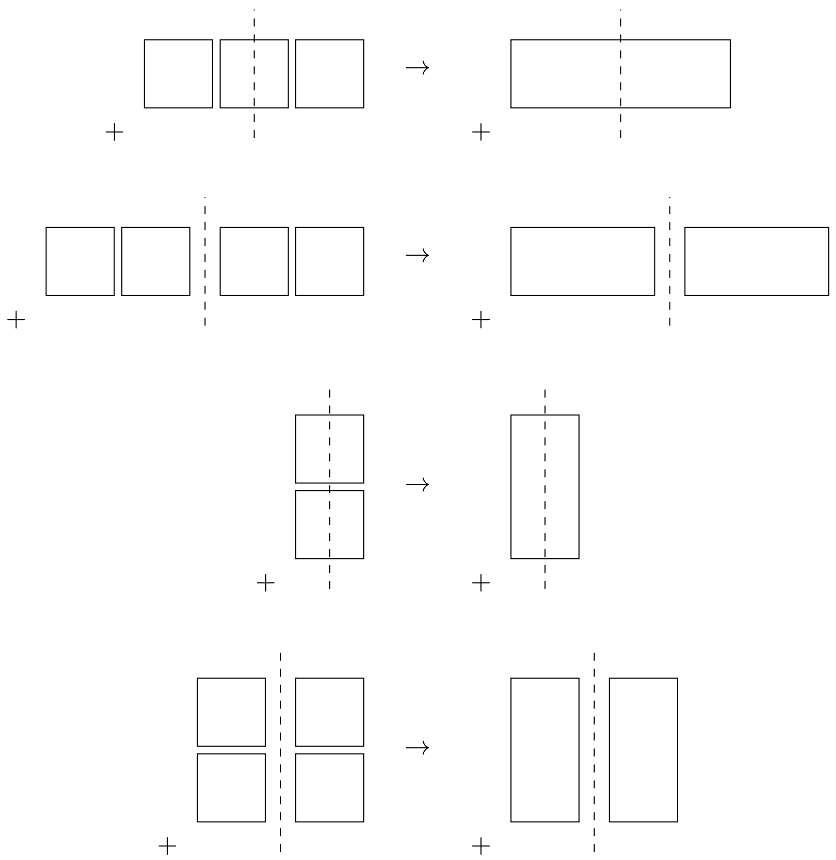


Figure 3.11: Four shape rules from the Palladian grammar to concatenate spaces, while preserving the symmetry of the plan — in particular, rules 12–15 in Stiny and Mitchell [1978a].

The next step in the derivation is the concatenation of spaces in the floor plan to create larger spaces. In doing so, the symmetry of the floor plan must always be preserved, which is reflected in the four concatenation rules shown in Figure 3.11. Indeed, due to the axis (dashed line) in the pattern shape of the rules, concatenation is only possible for symmetric spaces. The actual Palladian grammar also includes rules to generate cross-shaped or T-shaped spaces by concatenation. Another set of rules describes the placement and ornamentation of the principal entrance on the north–south axis. An example of a floor plan layout generated from the initial three-by-three grid, including the placement of the principal entrance, is shown in Figure 3.12 (left).

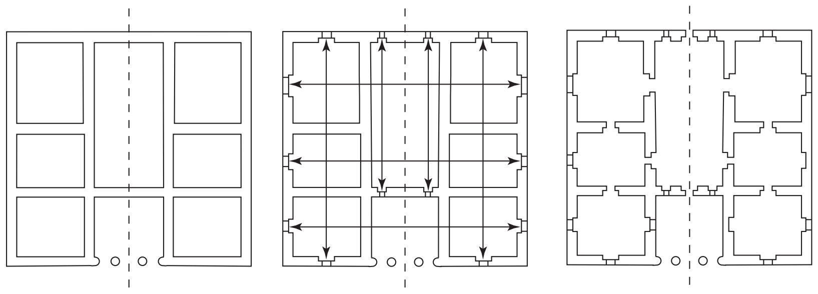


Figure 3.12: Floor plan layout definition (left), window placement (middle), and door placement (right) of villa Angarano using the Palladian grammar by Stiny and Mitchell [1978a].

A final set of rules describes how windows and doors are located along a horizontal or vertical axis in the floor plan. An example of such a rule from the Palladian grammar, shown in Figure 3.13 (top), defines the placement of two windows along a horizontal axis. This rule locates windows in two opposing wall segments and, additionally, adds a new horizontal axis (↔) between these two windows. This additional axis is needed to locate interior doors on the intersections of the axis and interior walls, using a second rule shown in Figure 3.13 (bottom). For example, Figure 3.12 shows the placement of windows and their corresponding axes (middle), after which doors are placed on the intersections to finalize the floor plan (right). The resulting floor plan corresponds to the villa Angarano that appears in Palladio's *Four Books of Architecture*.

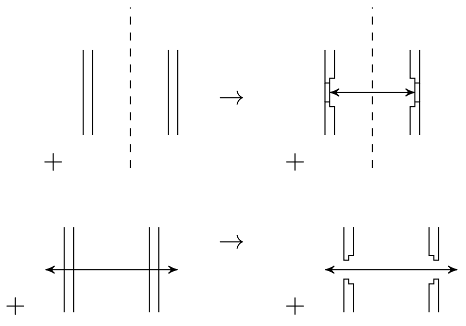


Figure 3.13: Two shape rules from the Palladian grammar for placing windows (top) and doors (bottom) along a horizontal axis — in particular, rules 59 and 60 in Stiny and Mitchell [1978a].

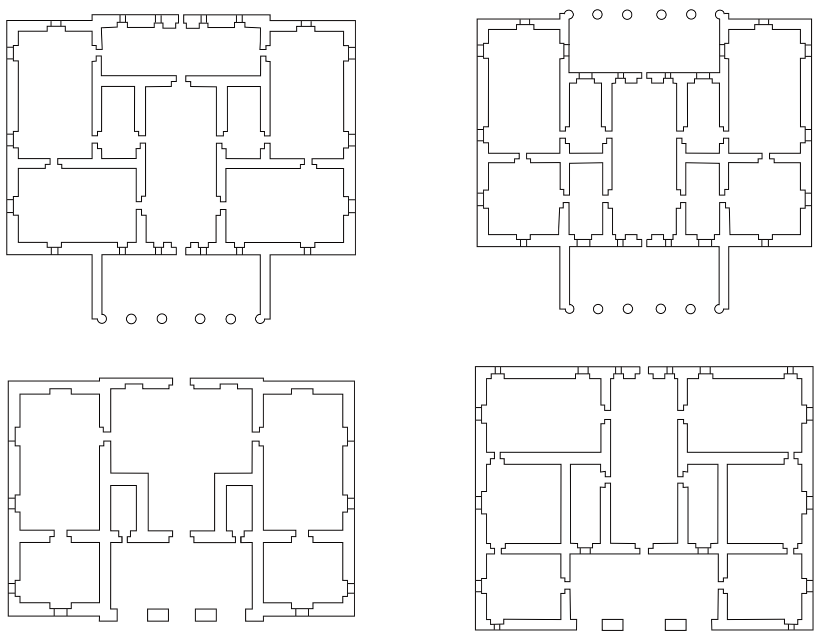


Figure 3.14: Four possible derivations of the Palladian grammar, one of which is a new design. Reproduced from the original image appearing in Stiny and Mitchell [1978b].

In the paper *Counting Palladian plans*, Stiny and Mitchell [1978b] demonstrate both the analytic and generative power of this shape grammar by generating several existing and original floor plans, though at a certain level of abstraction. Figure 3.14 shows four resulting floor plans generated using the Palladian grammar — one of which is a new design and does not appear in the *Four Books of Architecture* (which one?). As a result, the grammar described by Stiny and Mitchell is shown (1) to capture the design principles underlying the floor plans of Palladian villas and (2) to generate new designs that go beyond the existing corpus of designs. However, this shape grammar is neither to be understood as a historically accurate description, nor as a claim for a specific design method. As Stiny [2006] points out — "*What I am doing is not a commentary on contemporary architecture and how it is practiced, or an argument for classical principles of building or for the plan as a method of designing. Nor is my interest in Palladio historical.*". Historical accuracy is seldom the main goal of analytic shape grammars. Instead, they may reveal design principles, such as compositional, functional, and technical aspects.

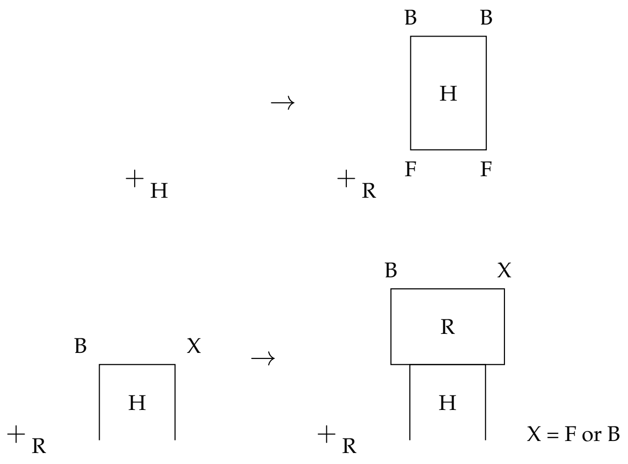


Figure 3.15: Two shape rules of the Queen Anne shape grammar. Reproduced from the original image appearing in Flemming [1987].

This is nicely demonstrated in the shape grammar of Queen Anne houses, developed by Flemming [1987]. This shape grammar generates, step-by-step, three-dimensional designs, where *"each step is based on an identifiable technical or compositional logic, very much in the way architects use sequential diagrams to explain the logic behind a certain composition"* [Flemming, 1987]. In particular, this grammar describes the arrangement of rooms around a core entrance hall in order to satisfy certain structural and spatial requirements. The top shape rule in Figure 3.15 is the initial rule, which places the entrance hall using a labeled shape (H). The labels F and B are used to identify the front and back of the floor plan. The bottom shape rule in Figure 3.15 adds a new room (R) by extending a floor plan towards the side or back. In total, 17 rules describe the spatial organization and 32 rules describe the exterior articulation of Queen Anne houses. In this way, analytic shape grammars provide a comprehensive way of describing and explaining specific design principles.

3.2.2 Shape grammars for analysis and original design

Another kind of shape grammar application focuses on the generation, and by extension exploration, of original designs. For such shape grammars, the emphasis is on the generative aspect of rules, rather than the analytic or descriptive value of shape rules. The theoretical distinction between analytic and original shape grammars made here is often less apparent in practice. Many shape grammars developed by researchers in the architec-

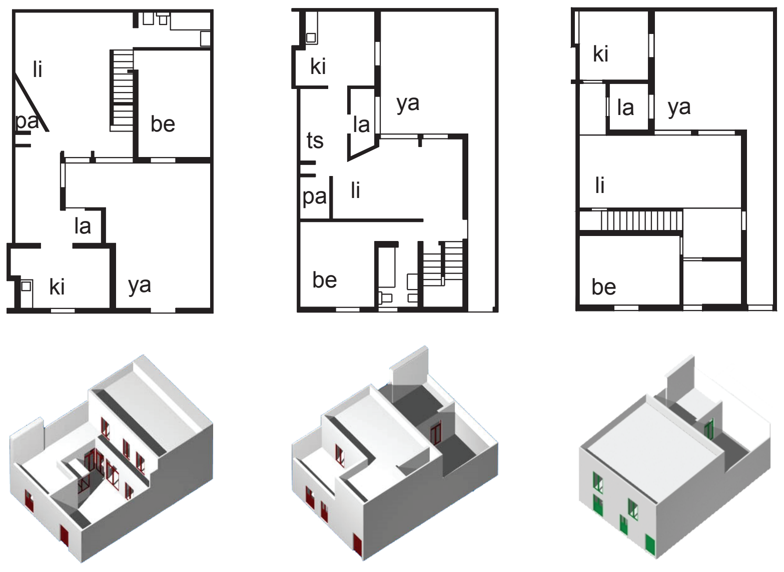


Figure 3.16: Floor plan and three-dimensional view of some possible derivation results of the Malagueira shape grammar. Only the first two results are existing designs in the corpus. The labels are living room (li), bedroom (be), patio (pa), kitchen (ki), yard (ya), laundry (la), and service (ts). Reproduced from the original image appearing in Duarte [2005b].

tural design domain are hybrid analysis/original shape grammars. For example, Duarte [2005a] describes a shape grammar for the Malagueira housing system designed by the Portuguese architect Álvaro Siza Vieira. While this grammar encapsulates the design principles underlying the specific corpus of Malagueira designs, the grammar is also used to generate new variations in the context of mass-customization of housing designs. The proposed design system generates both known designs from the Malagueira corpus and new customized designs that use the same compositional principles underlying the Malagueira corpus. Figure 3.16 shows the floor plan and the three-dimensional view of three designs generated using the grammar. The first two results are existing houses in the Malagueira corpus, designed in 1978. The third result is a new design for a five-bedroom backyard house that is customized for a specific client. A thorough evaluation of the descriptive and generative power of this grammar can be found in the work of Duarte [2005b]. Noticeably, "*when the new design was shown to Siza amidst other Malagueira designs, he did not notice that it was not his own design.*" [Duarte, 2005b].

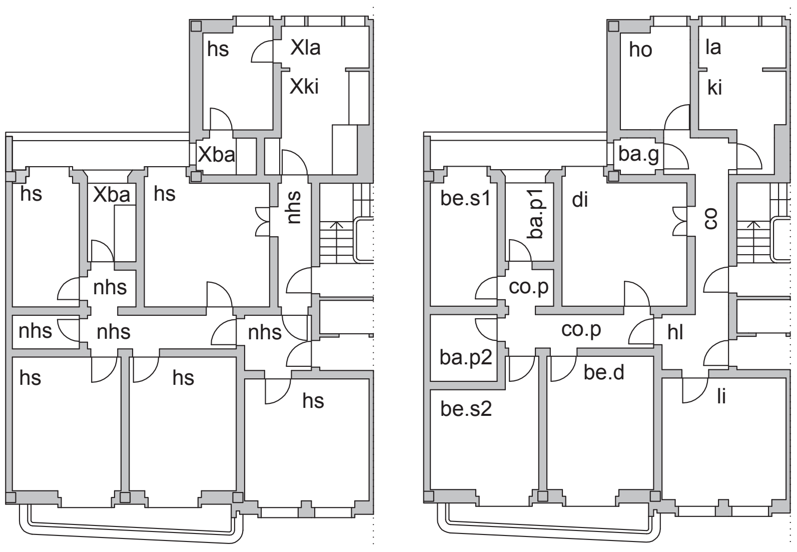


Figure 3.17: The floor plan of an existing house (left) and an adapted floor plan generated using the Rabo-de-Bacalhau grammar (right). The labels are explained in Table 4.3 (p. 95). Reproduced from the original image appearing in Eloy [2012].

Eloy [2012] describes a shape grammar for the renovation of traditional Portuguese houses. The grammar attempts to provide an answer to the need for mass rehabilitation of the existing housing stock in Portugal. In order to develop a more general methodology, Eloy has first developed a transformation grammar for the specific case study of traditional Portuguese *Rabo-de-Bacalhau* (or 'cod-tail') houses. This shape grammar encapsulates various customized transformation strategies for adapting existing houses to the current standards, depending on specific client needs and cost requirements. Unlike analytic grammars, the rules are inferred from other relevant experience of rehabilitation work and expert knowledge [Eloy and Duarte, 2014]. In other words, the rules are not derived from an existing corpus of design, but they encode design knowledge that is acquired directly from architects. For this grammar, the floor plan of any existing Rabo-de-Bacalhau house can be used as the starting point of the grammar. By applying different rule sequences (the so-called transformation strategies), multiple transformed floor plans can be generated, each adapted to the comfort and accessibility standards. For example, Figure 3.17 shows the resulting floor plan of a particular transformation strategy that involves re-assigning functions, changing room dimensions, and making rooms more accessible by widening connections.

3.2.3 Some further remarks

The theory of shape grammars has been in existence for a few decades now. Several definitions, improvements, and other kinds of contributions have followed, thereby adding new insights from different perspectives (for example, mathematical foundations, new application domains, or computer programs for the development and application of shape grammars). To date, shape grammars are an established field of study both in computer-aided (architectural) design and design theory. Further refinements of the theory are still being made [Economou and Kotsopoulos, 2014; Stouffs, 2014], as well as methods for developing and assessing shape grammars [Grasl and Economou, 2014; Königseder and Shea, 2014] and new application domains [Coutinho et al., 2014; e Costa and Duarte, 2014; Krstic, 2014]. The broad range of this work demonstrates the appeal of shape grammars as a tool for analyzing and generating languages of designs, but also as a general theory for studying creative design.

On the other hand, the theory of shape grammars is, at least, controversial, and subject to a long-standing debate about their usefulness in creative design practice. A common criticism of shape grammars is that designs are supposedly to be treated in terms of linguistic concepts [Fleisher, 1992; Gerzso, 2003], including the analogy between sentences generated by phrase structure grammars and designs generated by shape grammars. Such a criticism presumably arises from the term 'grammar' that conveys a linguistic association to Chomsky's phrase structure grammars. Fleisher [1992] points out, rightly so, that linguistic associations between language and architecture are problematic; however, such alleged associations are probably the result of misinterpretations. Flemming [1994] argues that "*the 'grammar' part in shape grammars is to be understood in a purely technical sense, no analogies, legitimate or not, are implied.*". A similar argument is made by Stiny [2006] — "*Sometimes, analogies imply a lot more than they should ... I did not think that designs were sentences, but instead that grammars could generate both sentences and designs.*".

Another criticism is that grammar-based CAD tools, and CAD tools in general, cannot support every aspect of designs (such as subjectivity, aesthetic qualities, or cultural aspects). For this reason, CAD tools are only able to support some reductionist definition of design (design as search, design as problem-solving, design as ...). In this sense, CAD tools could not be used in a meaningful way to support a creative design practice [Flemming, 1994]. Indeed, the theory of shape grammars is no exception, because it, by no means, provides a complete or all-knowing model of design. However, this criticism vanishes if CAD tools are considered, not as models of design, but merely as tools to explain or support some particular aspects of design. In this sense, shape grammars are conceived as (generative) tools that might be useful in supporting analysis and generation of designs and for working out the implications of particular design theories. As tools, shape grammars are particularly appealing due to their flexibility in reinterpreting shapes, and also due to their expressive power in describing and generating designs using a unified formalism of shape rules.

In this context of using shape grammars as tools for analysis and generation, another recurring criticism is that there is little evidence of the practical usability of such grammar-based design tools. Indeed, while shape grammars are an established field of study within the field of CAD research, they are far less known outside the academic world. Unlike other technologies (such as parametric modeling), shape grammars have not yet found widespread adoption — neither in creative design practice, nor in computer-aided design tools. In order to expand the impact of shape grammars, research on computer implementations is gaining renewed interest; for example, the work of McKay et al. [2012] on grammar-based software approaches, or the work of both Grasl [2013] and Wortmann [2013] on computer representations of shapes and shape grammars. Of particular interest, is to investigate whether shape grammars can be used as tools for design space exploration.

3.3 Design space exploration with shape grammars

Shape grammars might provide a concise and computable way to represent and explore a design space. A design space is a special kind of state space, in AI terminology, and typically consists of an initial design, a set of available actions, a transition model, a goal test, and a path cost function. Shape grammars, on the other hand, define a language of designs through generative rules. An analogy between such a formulation of a design space and shape grammars is first described in the early work of Gero and Kazakov [1996]. In particular, an initial shape is to be understood as the initial design of the design space. The set of shape rules in shape grammars corresponds to the set of applicable actions in the design space. Each shape rule is defined as a pair of shapes, shown one after the other, and separated using an arrow ($\rightarrow$) to indicate the action between the two shapes. A such, a shape rule describes a transition between a given (sub-)shape and a resulting shape, and involves navigating from one shape in the design space to another. A sequence of shapes connected by a sequence of rule applications designates a path in the design space. The shapes generated with a specific shape grammar are part of the language of this grammar,

State space definition [Russel and Norvig, 2010]	Shape grammar definition [Stiny, 2006]
Initial state	Initial shape
Set of available actions Transition model	Set of shape rules
Goal test Path cost	Goal test and path cost, only if the design task is formulated as an optimization problem
State space	Language of the grammar
Path	Sequence of shapes

Table 3.3: Analogy between the definition of a state space and shape grammars.

which is the set of shapes that can be generated by the grammar. This language implicitly defines a design space in a way that is both concise and computable, because it is represented through a finite set of rules and because these rules provide the main mechanism for searching the design space. The analogies drawn between state spaces (as defined by Russel and Norvig [2010]) and shape grammars are shown in Table 3.3.

In the context of problem solving, a goal test and path cost should be included in the definition of the state space, enabling the search for the path with the lowest cost to a goal state (which is the solution to the problem). However, a goal test and path cost are typically not included in the definition of shape grammars, since they are not so much conceived as tools for problem-solving, but rather as tools for the analysis and generation of designs. In the context of creative design, goals are not fixed and their formulation is part of the design process itself (they are wicked problems). In some cases, particular aspects of the design task at hand involve an explicit goal test — for example, the optimization of structures, costs, and project scheduling. In these cases, heuristic search methods can be used to find near-optimal designs in a reasonable amount of time. For example, in the work of Cagan and Mitchell [1993], a heuristic search method called 'simulated annealing' is combined with shape grammars to "*control choice among alternative rule applications as shapes are derived. The result is an optimally directed design solution.*" [Cagan and Mitchell, 1993]. In the work of Shea and Cagan [1999], this method has been applied for the design of truss structures. More recently, Grasl and Economou [2014] describes different strategies for automatic rule selection in shape grammars, ranging from random selection (which may be useful to spark new ideas) to genetic algorithms [Holland, 1992] that mimic the process of natural selection.

Unlike the symbolic kind of state space defined by Russel and Norvig [2010], a design space represented by shape grammars consists of visual designs defined in a specific algebra. In other words, a design space is a structured network of shapes (U_{ij}) possibly associated with other kinds of representation that are non-visual (V_{ij} or W_{ij}). The structure of the design space is defined by the shape grammar rules that make designs accessible through rule application. Therefore, the resulting design space is a directed graph in which the nodes are shapes and the links between nodes are shape rule applications. Figure 3.18 shows a visualization of (a small part of) the design space that is represented by the Palladian grammar. In particular, only the possible actions of the four concatenation rules (Figure 3.11) are shown, starting from a three-by-three grid.

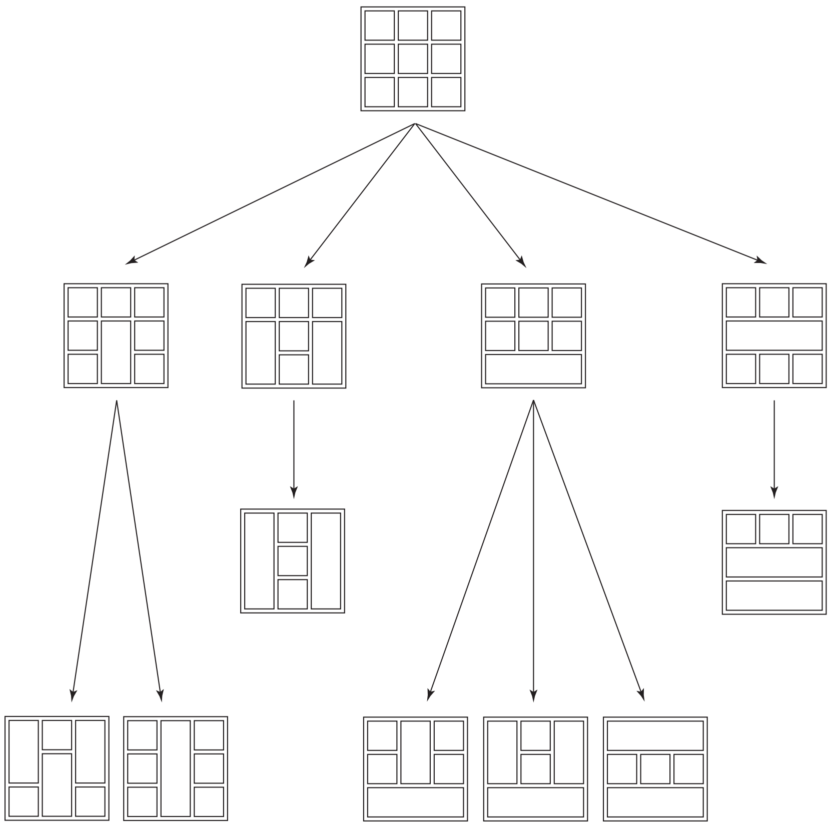


Figure 3.18: A part of the design space represented by the Palladian grammar [Stiny and Mitchell, 1978a]. Rule applications ($\rightarrow$) relate one design to another.

As shape rules are applied using the embedding and part relations, the design space is structured according to a partial ordering (which means that not every pair of designs is related). In some cases, different rule sequences may result in the same design. As a result, the design space might contain designs that look the same, and only differ in the rule sequence that was used to generate the particular designs (the creation history). For illustrative purposes, the design space shown in Figure 3.18 only contains the twelve distinct possibilities of floor plan layouts that can be generated — each following a single derivation path. The actual resulting design space would look more like a network with equivalent shapes and multiple links between them. Also, the actual design space contains equivalent designs under particular transformations, such as isometric shapes, which are identical under translation, rotation, and reflection transformation. These equivalent shapes are not shown in Figure 3.18.

3.3.1 Design move codification

The most significant ability of shape grammars to represent a design space lies in the encoding of *design moves* in the form of shape rules. The term 'design move' originates from the design thinking research field, where it is defined as "*a step, an act, an operation, that transforms the design situation somewhat relative to the state it was in before that move.*" [Goldschmidt, 2014]. Whereas in a problem-solving context, such as playing chess or proving a theorem, the notion of moves is clearly defined as an action that leads from one state to another, the notion of design moves is often more equivocal. A design move varies in temporal extent from a few seconds to more profound changes in the development of designs. For most shape grammar applications, the rules describe design moves that are quite short, though in some cases they cover more profound interventions; for example, Knight [1981] describes how new designs can be created by an informed and deliberate manipulation of known or given information. In particular, Knight uses shape equivalence rules to encode how an existing design is to be transformed to new designs. In a more general way, shape grammars explicitly encode different kinds of design moves in the form of shape rules.

By encoding design moves as shape rules, designers might be able to foresee the possible effects of taking a particular design move, or to store successful design moves, their own or others, for later design projects. As a result, shape grammars represent a design space in which both navigation to existing designs and generation of original designs can take place. Several implementations of shape grammars on computer systems have made this codification explicit; for example the early work of Heisserman [1994] and the more recent shape grammar implementations described in the work of McKay et al. [2012]. The design space represented by shape grammars is sufficiently large and diverse, as demonstrated by shape grammars representing an entire corpus of designs such as the Palladian grammar [Stiny and Mitchell, 1978a], and others. On the other hand, the design space represented by shape grammars is limited in its breadth and maintains a certain amount of coherence. By representing such a deliberately limited, yet broad enough design space, exploration occurs in a design space in which the goal is refinement of a coherent and well-developed design idea. The advantage of considering a limited number of coherent design alternatives, rather than many widely different alternatives, has been indicated in several (empirical) research studies, such as the work of Goldschmidt and Tatsa [2005] and Goldschmidt [2006].

At first sight, the act of testing design moves resembles the act of searching a design space. Shape rules enable a designer to consider designs in a design space that is already defined implicitly when the grammar was constructed. Indeed, using a predefined shape grammar, it is possible to foresee the possible effects of designs within the implicit design that is represented; however, designs beyond the scope of this implicit design space seem to be out of reach. In other words, the design space represents only those designs that are in the language of the shape grammar. So, while shape grammars adequately account for the act of searching a design space, the question remains — how can shape grammars support design space exploration in the broadest sense, including the (re)-formulation of this design space? Two possibilities are *reformulation* of the shape grammar by changing the rules, and *reinterpretation* by seeing designs differently.

3.3.2 Shape grammar reformulation

An important aspect of design space exploration is the (re)-formulation of the design space, which involves setting the boundaries in which the search can occur. This act of (re)-formulation might completely precede the process of searching the design space; for example, for shape grammars that have already been defined before they are used. On the other hand, (re)-formulation might also occur in sequential stages, similar to the process of oscillation [Cross, 1997] or co-evolution [Schön, 1983] of problems and solutions. Reformulation involves the process of creating a new design space or modifying an existing design space by means of changing the underlying mechanism of this design space. As shape rules are the main structuring mechanism of the design space represented by a shape grammar, an obvious way to enable design space reformulation is by making changes to the rules themselves. For every change in the shape grammar, such as adding new rules, deleting rules, or modifying rules, this results

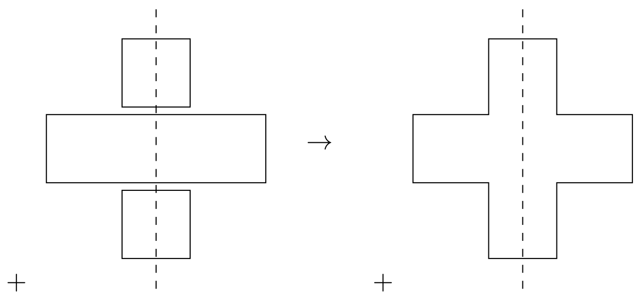


Figure 3.19: Additional shape rule (rule 18 in Stiny and Mitchell [1978a]) from the Palladian grammar to concatenate spaces, whilst preserving the symmetry of the plan.

in a reformulation of the design space represented. By adding additional rules, new paths for exploration are made available that might (or might not) lead to interesting areas in the design space. In a similar way, deleting or modifying shape rules redefines the scope of the design space represented. For example, adding a shape rule for concatenating spaces to a cross-shaped space (Figure 3.19) to the Palladian grammar results in the possibility to generate more floor plan layouts.

The iterative reformulation of a grammar and its application to search alternative designs should be a key aspect of grammar-based information systems, as pointed out in a workshop on the computer implementation of shape grammars [McKay et al., 2010, 2012] — "*We anticipate the role of designers and engineers changing to include the development of grammars from which shapes could be computed in generate–test cycles. System users would design, develop, and use their own grammars to generate designs in, for example, a given style or to suit the capabilities of a particular fabrication process with associated constraints.*" [McKay et al., 2012]. In this context, one of the key features of grammar-based design tools should be to provide designers with the possibility to make changes to the grammar in an interactive and intuitive manner. By doing so, the design space might become more dynamic, in the sense that it does not remain static during the exploration process. This could in turn lead to a more agile exploration of a design space through generate–test cycles. Chakrabarti et al. [2011] point out several strategies to support designers in the iterative development of a shape grammar, such as building specialized, yet intuitive, shape grammar editors or building self-learning shape grammar implementations.

While the possibility to make changes to the grammar during exploration could indeed prove to be a valuable approach, the possibility to manually intervene by modifying designs with no regard for rules could prove equally valuable. In doing so, designs could be generated that are outside the language of the grammar. In the terminology of the design space, this would enable the designer to make shortcuts — not following any design path — thereby allowing 'on-the-fly' experimentation.

3.3.3 Seeing shapes differently

Another way to support design space exploration is a result of the embedding and part relations that allow shapes to be decomposed into different sub-shapes; in other words, by using different underlying topologies. Shapes can freely be (re)-interpreted and decomposed, thereby revealing different parts to which rules can be applied. In this way, a new understanding or alternative perspective on the design space is obtained that might lead to new designs — ranging from new but anticipated designs to unconventional designs, as pointed out by Knight [2003]. The Malagueira housing grammar [Duarte, 2005a], which generates new customized designs that are similar to the existing ones, is an example of the former approach. On the other hand, the grammar for the design of truss structures [Shea and Cagan, 1999] generates new and structurally sound designs that are difficult to predict beforehand. A few possible derivations of this grammar are shown in Figure 3.20.

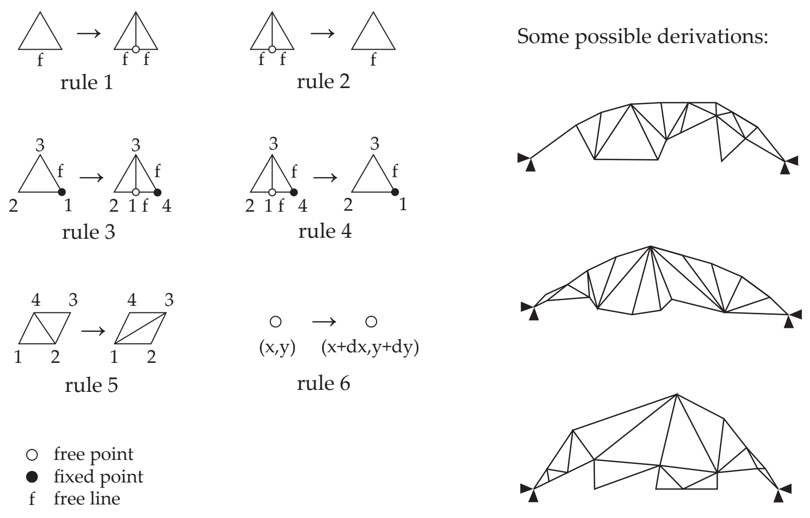


Figure 3.20: Shape grammar for the design of truss structures (left) and some possible derivations. Reproduced from the original image appearing in Shea and Cagan [1999].

The embedding and part relations might also result in *emergent shapes*. Emergence is a concept rooted in 19th century philosophy, and became widely popular in the field of computer science. In the book *Emergence: From Chaos to Order*, Holland [1999] points out that "*emergence, in the sense used here, occurs only when the activities of the parts do not simply sum to give activity of the whole. For emergence, the whole is indeed more than the sum of its parts.*". The concept of emergence is often associated with terms such as novelty and surprise [Knight, 2003], and is therefore, the subject of many computational models that attempt to generate and understand emergence. Among the most well-known computation models of emergence are *cellular automata* (CA) [Wolfram, 2002]. In their most simple form, CAs are two-dimensional configurations of cells that are in an active or non-active state (Figure 3.21). Starting from an initial configuration with one active cell, rules can be applied to change the states of particular cells in the configuration. These rules are defined in terms of the current state of a particular cell and its immediate neighbors. The top row of the rules, shown in Figure 3.21, contains all the possible combinations of states for a cell and its immediate neighbors. The bottom row then specifies what state the center cell should be after the rule is applied. The recursive application of such rules may give rise to more complex emergent patterns. For the two examples given in Figure 3.21, the emergent patterns are a nesting of triangles (left) and a random configuration of local structures (right). In other words, while low-level cells of the automaton are changed through local rules, they result in high-level patterns that are not described in the rules.

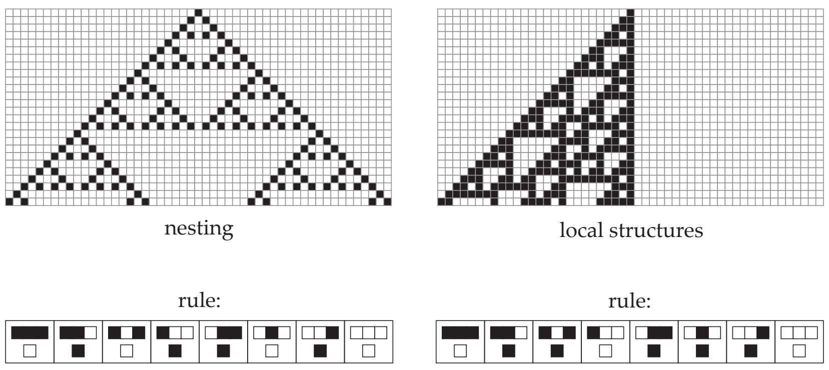


Figure 3.21: Two example cellular automata (top) and the corresponding rules (bottom). The recursive application of the rules results in emergent patterns (nesting and local structures) [Wolfram, 2002].

The kind of emergence in cellular automata can also be found in other kinds of computational rule-based systems, such as shape grammars [Stiny, 1994]. In the context of shape grammars, emergence refers to the ability to recognize and, more importantly, to operate on shapes that are not predefined in a grammar, but arise, or are formed from the shapes generated by rule applications [Knight, 2003]. In particular, an emergent shape is a shape that is not directly added by a previous rule application, but is the result of fusing basic elements. As an example, Figure 3.22 shows one possible derivation of two shape rules that shift a shape along a diagonal axis. The bold lines indicate how a rule is applied in each derivation step, and an asterisk indicates whether an emergent shape is used. The resulting shape of this derivation might appear somewhat surprising, especially considering the geometric simplicity of the initial shape and rules.

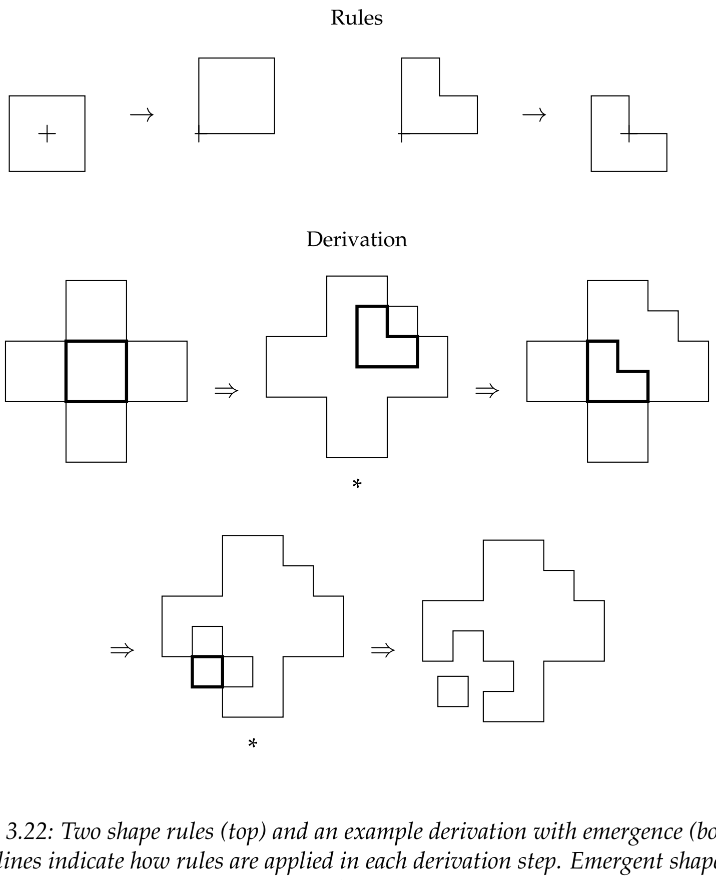


Figure 3.22: Two shape rules (top) and an example derivation with emergence (bottom). Bold lines indicate how rules are applied in each derivation step. Emergent shapes are indicated with an asterisk. Reproduced from the original image appearing in Knight [2003].

At each step of the shape grammar derivation, new opportunities for exploration occur by seeing designs in a different way or by means of emergent shapes. Rules can then be applied in ways other than they were initially created for — possibly resulting in unexpected, or perhaps even surprising, designs. According to Stiny, this is what distinguishes design from search — "*But design is not search. It is far more than sifting through combinations of predetermined parts that are the set results of prior analysis ... I don't have to know what shapes are, or to describe them with definite units, for them to work for me as I design.*" [Stiny, 2006]. In other words, shapes are inherently ambiguous, and this ambiguity is "*a limitless source of novelty*" [Stiny, 2006]. The key importance of ambiguity and emergent behavior has been discussed at length in the work of Stiny. In particular, he describes an iterative process of *seeing* and *doing*, where the designer applies a rule based on what he sees. The act of seeing is a continuous process that precedes every move that designers undertake, as understood by Stiny. As a result, emergence can be considered to be an act of exploration, because it changes the design space and, in particular, it changes the way the design space is understood by the designer. Through emergence and reinterpretation, a new understanding or alternative perspective on the design space is obtained that might lead to serendipitous developments, thereby taking designs in new directions.

3.4 Bridging the gap?

As shape grammars embed some elements of design space exploration — in reformulation, reinterpretation, or through emergent designs — they enable a certain kind of freedom that is characteristic for creative design. In particular, shapes are defined and interpreted in ambiguous ways (through the part and embedding relations) and as a designer, you can "*use any rule(s) you want, whenever you want to*"[Stiny, 2011]. In other words, the view of designing associated with the theory of shape grammars does not relieve designers from their creative task of coming up with new rules and being able to deviate from preconceptions. Simultaneously, it involves storage, repetition, and copying of existing rules in new design situations via reinterpretation — "*The idea is to let embedding work for you — to see things in new ways, and to have the means to do something about this without having to invent [the rules]*"[Stiny, 2011].

On the other hand, the theory of shape grammars also describes particular aspects of creative design as a specific kind of visual calculation with rules. This formalized understanding of exploration enables the codification of design knowledge or design moves into rules that are more amenable to information systems because they can be stored and be used for computation. As Tapia points out — "*Shape grammars naturally lend themselves to computer implementations: the computer handles the book keeping tasks (the representation and computation of shapes, rules and grammars and the presentation of correct design alternatives), while the designer specifies, explores, develops design languages, and selects alternatives.*"[Tapia, 1999]. The computer implementation of shape grammars makes the codification of design moves explicit and amenable to the computational power of information systems — not so much to automate the design process, but rather to support the designer in storing and applying rules. Such characterization more closely resembles a mixed-initiative interaction strategy [Allen et al., 1999], in which each agent contributes to the task that it does best.

The theory of grammars is clearly influenced by concepts drawn from AI, such as the formulation of a rule production system. However, this also deviates from the traditional approaches to AI by avoiding any form of explicit or fixed representation. As a result, shape grammars offer a means of bridging the gap between the wickedness of creative design and the structured nature of information systems [Wortmann, 2013]. This is achieved by avoiding predefinitions of shapes, on the one hand, while enabling explicit formalization, on the other. In doing so, shape grammars are free from restrictions inherent to other generative design tools, such as parametric modeling or BIM. Such traditional tools do not allow for ambiguity or emergence, because shapes are constructed using some set of predefined primitives and "*the behavior of the computer-produced drawing ... is critically dependent upon the topological and geometric structure built up in the computer memory as a result of drawing operations.*" [Sutherland, 1975]. While traditional generative tools merely focus on representation and/or optimization, shape grammars can support design space exploration not limited by a bounded range of solutions.

Given their formulation as rule production systems, shape grammars seem to be an obvious candidate for computer implementation. Such computer implementations promise more designerly exploration design tools, compared to their traditional counterparts. However, the implementation of shape grammars on to computer systems is not as straightforward as it might seem, because by directly representing shapes as symbols, the essential features of ambiguity and emergence of shapes would be lost. The tension between the visual nature of shape grammars and symbolic computation is well pointed out by Gips — "*The tension in computer implementation of shape grammars is the tension between the visual nature of shape grammars and the people who want to use them and the inherently symbolic nature of the underlying computer representations and processing.*"[Gips, 1999].

4 From Shapes to Graphs

In this chapter, we look at the topic of implementing shape grammars so that they can be used on computer systems. An overview of previous approaches to implementation is provided (Section 4.1), and particular attention is given to graph-theoretic implementation approaches (Section 4.2). Furthermore, we describe a step-by-step approach to implementing a shape grammar using a graph-theoretic representation (Section 4.3). This approach is evaluated by implementing an existing shape grammar, originally developed on paper, on to a computer system (Section 4.4).

4.1 Computer implementations of shape grammars

The computer implementation of shape grammars has been the subject of many research efforts since their original conception in the 1970s. While the theory of shape grammars has produced a long series of formal studies, the corresponding efforts for the computer implementation of these grammars have not met with the same success [Grasl and Economou, 2013]. A number of authors [Chau et al., 2004; Gips, 1999; McKay et al., 2012] have identified several issues and challenges for computer implementations of shape grammars. Characteristic requirements that crop up regularly for an ideal implementation are: support for sub-shape detection, parametric shape grammars, three-dimensional or curved shapes, and providing an intuitive user interface. Most of these challenges remain valid to date.

Among the key challenges is finding suitable symbolic representations for shape grammars that are both amenable to computer implementation and which support the emergent nature of shapes. In other words, predefinitions of shapes should be avoided, but instead, the chosen representation should preserve the embedding and part relations. As a result, a computer-implemented shape grammar should be able to detect (emergent) sub-shapes to which shape rules can be applied — which is commonly referred to as solving the *sub-shape detection problem*. The problem of sub-shape detection is one of the main difficulties in computer implementations of shape grammars — "*The hardest and certainly most crucial step in the application of a shape rule to a labeled shape is in actually determining whether or not the shape rule applies to the labeled shape. In general, there may be several sub-shapes in a given labeled shape to which a given shape rule may be applied.*"[Krishnamurti, 1981]. Moreover, Yue and Krishnamurti [2013] demonstrate that no general algorithm is known to efficiently solve the sub-shape detection problem.

A second key challenge results from the transformations under which the pattern shape of shape rules are matched to a given shape. For isometry and similarity transformations of shapes, it is possible to detect sub-shapes based on specific geometric characteristics, such as size and proportion (for isometry) or only proportion (for similarity). The seminal approach for sub-shape detection, described by Krishnamurti [1981], uses such coordinate geometry from three distinct points to detect similar shapes. However, for *parametric shape rules* operating under transformations that only preserve the topology of the shape, thereby omitting geometric characteristics, the problem of sub-shape detection cannot be solved in such way. Indeed, geometric realizations of the same parametric shape can be visually different, making it difficult to detect equivalence between these shapes. Many existing shape grammars, developed only on paper, are parametric shape grammars — such as the Palladian grammar [Stiny and Mitchell, 1978a] and the Queen Anne grammar [Flemming, 1987].

The question of how to enable sub-shape detection for parametric shape grammars remains an open research discussion to date. Other important challenges include the support for three-dimensional shapes and curved shapes. Such extensions make the problem of finding appropriate symbolic representations of shapes more difficult, especially for general shape grammar implementations. Another challenge is the development of intuitive interfaces and user interaction mechanisms — on the one hand, to allow designers to develop, implement, and explore their grammars, but also to deal with the combinatorial explosion and infinite number of emergent possibilities to which shape grammars are subject.

4.1.1 Why computer implementations?

While the computer implementation of shape grammars presents a number of difficulties and challenges, such implementations have merit in several specific situations. For example, Gips [1999] identifies a number of possible tasks that shape grammar implementations could support:

- Generation — generating designs in the language of a given shape grammar;
- Parsing — determining whether a given shape is in the language generated by a shape grammar;
- Inference — generating a shape grammar from a set of given shapes or designs;
- Development — assisting the user in creating a shape grammar by providing suitable (editor) tools.

In the context of supporting design space exploration, the first task (generation) and latter task (development) are of special interest. By supporting these tasks, shape grammar implementations might provide the basis for a new generation of design tools for design space exploration. On the one hand, generation tools for shape grammars support the designer in storing shape grammars and to generate or analyze designs in the language of the grammar — which is particularly useful for grammars that are too extensive to explore manually. On the other hand, development tools for shape grammars allow designers to design, develop, and use their own grammars, thereby supporting iterative generate–test cycles [McKay et al., 2012]. An example of parsing shape grammars can be found in the work of Teboul et al. [2011], and for an example of inferring visual grammars from a set of given designs, see the work of Talton et al. [2012].

Many of the challenges involved with shape grammar implementations result from supporting the embedding and part relations in automatic sub-shape detection. As a result, some authors question whether it is necessary to solve the general sub-shape detection problem in every implementation [Gips, 1999]. Indeed, in some cases of very specific and specialized grammar implementations, it might be beneficial to let the user decide where to apply the rules, instead of enabling automatic sub-shape detection. In other cases, the embedding relation and parametric functionalities might be omitted, thereby reducing the complexity of rule applications and causing the implemented shape grammar to be more manageable but also limited in its generative power. Wortmann [2013] gives an overview of several implementations without embedding and parametric shapes, but he also correctly remarks that because of their limited functionality, such implementations are relatively uninteresting as tools for exploratory design. The most useful implementations, in the context of design space exploration, are those that fully support (parametric) sub-shape detection, because they enable recognition of emergent shapes and they relieve the designer from having to manually represent and compute shapes and rule applications.

4.1.2 Overview of previous approaches

Several overviews of the computer implementations of shape grammars are available; for example, in the work of Gips [1999], Chau et al. [2004], Yue [2009], and McKay et al. [2012]. At a 2010 workshop, the current state-of-the-art, limitations, and key challenges in computer implementations of shape grammars were discussed [McKay et al., 2010]. In particular, an evaluation and comparison was performed in terms of modeling capabilities, semantics (ranging from general geometric elements to domain specific elements), interface definition (ranging from sketching to scripting), and generative capabilities (automatic sub-shape detection or rule application) of the main representative shape grammar implementations — which are the ones discussed in Correia et al. [2010]; Ertelt and Shea [2010]; Hoisl and Shea [2011]; Jowers et al. [2010]; Jowers and Earl [2011]; Li et al. [2009]; Trescak et al. [2012]. Also, McKay et al. [2012] analyze these systems with regard to four broad aspects — the representation and algorithms for geometry and semantic objects, user interaction and user interfaces, support for specific design tasks, and integration of the system into design and product development processes.

A milestone of primary importance in the development of shape grammar implementations, was the work of Krishnamurti [1981] — this was the first to feature a maximal representation of (straight) lines, thereby enabling sub-shape detection and shape emergence. However, the applicability of the underlying sub-shape detection algorithm is limited, because only shapes with at least three non-collinear points are processed; the notion of rational shapes is introduced to overcome the problem of non-exact arithmetic; and the proposed algorithm is computationally expensive. Many research efforts that followed this, built further on the initial work set out by Krishnamurti; for example, the work of Trescak et al. [2012] is notable for improving the sub-shape detection algorithm to the point that real-time calculation of sub-shape detection is achievable. While the earliest research efforts mainly focused on general shape grammar implementations or solving the sub-shape detection problem, later research efforts demonstrate a broader scope, including work on implementations for specific design problems and implementations that do not support emergence [Chase, 2010]. In particular, new approaches have been developed that focus on the interface and user interaction [Chase, 2002; Tapia, 1999], curved elements [Jowers and Earl, 2011], and three-dimensional extensions [Hoisl and Shea, 2011; Krishnamurti and Earl, 1992].

Most approaches rely on the geometric representations of shapes (often using maximal geometric elements) to detect and compute shape rule applications. Such approaches take advantage of specific geometric characteristics of shapes to enable sub-shape detection. Two notable approaches that do not rely on such geometric representations include computer vision approaches [Jowers et al., 2010] and the use of graph-theoretic representations of shapes [Grasl, 2013; Keles et al., 2010; Wortmann, 2013]. The former approach involves the use of object recognition, a specific research domain of computer vision, to match shapes by detecting similarities between them. Among the main benefits of this approach, is the ability to work with complex curved shapes and hand-drawn sketches; however, this approach has proven to be less successful in supporting parametric shape grammars [Jowers et al., 2010]. The latter approach involves the representation of shapes as graphs that describe the topological structure of shapes. These graphs can be constrained according to various properties, such as size, proportion, angles, and position — thereby enabling different non-parametric (isometry and similarity) and parametric transformations. Also, sub-shape detection can then be achieved by searching for a specific sub-graph in graphs, which is known as the *sub-graph isomorphism problem* — a well-studied research topic in the field of graph theory.

If shapes are represented as graphs, then graph rewriting or *graph transformation systems* can be used to create new graphs out of an original graph, similarly to how it occurs for shape grammars. Among the first attempts to describe the language of spatial objects (three-dimensional solids), using a graph-theoretic representation is the approach of Fitzhorn [1990]. In this approach, a graph transformation system is defined to generate valid three-dimensional solids. This approach is adopted and extended in the work of Heisserman [1994], in which a so-called boundary solid grammar was developed for the generation of three-dimensional solids. More recently, Grasl [2013]; Wortmann [2013] have emphasized the importance of graph-theoretic representations of shapes, by demonstrating how sub-shape detection for parametric shapes can be achieved. Grasl and Economou [2013] translate shapes to graphs, after which a graph transformation system is used to search for isomorphic sub-graphs to which rules can be applied. These isomorphic sub-graphs are then constrained (size and proportion) to limit the candidate parametric sub-shapes to specific geometric realizations.

4.2 Graph-theoretic representation of shapes

Considering the open research challenges, it is clear that a definite answer for the computer implementation of shape grammars — the "*Big Enchilada*" as described by Gips [1999] — is not available to date. The technique of using a graph-theoretic representation of shapes, together with a corresponding graph transformation system, might provide one of the stepping stones towards grammar-based design tools for design space exploration, by making possible several essential features of shape grammars that were significantly more difficult to achieve before, such as (parametric) sub-shape detection and supporting the ambiguous nature of shapes.

4.2.1 Definitions

Graphs are mathematical structures to model relations between objects. They are used in many fields because of their intuitive way of representing relations; for example, to model road networks or electronic circuits. Formally, a graph $G = (V, E)$ consists of a set of vertices or *nodes* V, together with a set E of node pairs or *edges* [Skiena, 2009]. Graphs exist in different kinds — undirected, directed, unlabeled, labeled, and multigraphs. A graph is called a *directed* graph, if it only contains ordered pairs of nodes

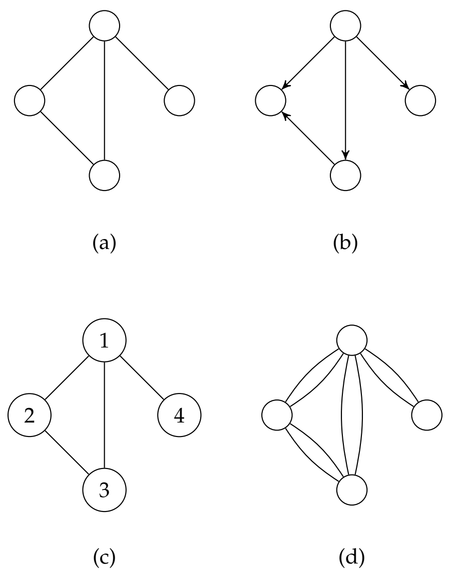


Figure 4.1: Different types of graphs — (a) an undirected and unlabeled graph with four nodes and four edges, (b) a directed graph, (c) a labeled graph, and (d) a multigraph.

or, in other words, if every edge (x, y) is directed from node x to node y. For a *labeled* graph, each node or edge is associated with a unique name or identifier to distinguish it from all other nodes and edges. A graph is called a *multigraph* if it contains edges that occur more than once in the graph. Examples of different graph types are shown in Figure 4.1.

Graphs can easily be represented on a computer system; for example, in the form of a two-dimensional adjacency matrix. An adjacency matrix $\mathbf{A}$ of the graph G with n nodes has the form $\mathbb{N}^{n\times n}$ in which the non-diagonal entries A_{ij} are the number of edges from node i to node j. For example, the adjacency matrix of the graph shown in Figure 4.1 (a) is

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

If two graphs are identical they are said to be *isomorphic*. The graph isomorphism problem consists of finding a mapping f from the nodes of one graph to another in such a way that the two graphs are identical [Skiena, 2009]. Given two graphs $G = (V_g, E_g)$ and $H = (V_h, E_h)$, they are isomorphic if for every edge (x, y) of G there exists one and only one edge $(f(x), f(y))$ that is part of H. A special case of the graph isomorphism problem consists of finding a subset of edges and nodes of a graph that is isomorphic to a smaller graph, which is called the sub-graph isomorphism problem. In general, the sub-graph isomorphism problem is proven to be in the complexity class NP-complete, which means that should an efficient (polynomial time) solution be known, then all problems in NP could be solved in polynomial time. In other words, the problem cannot be solved efficiently using any currently known algorithm, but nevertheless, practical (heuristic) solutions for handling sub-graph isomorphism are available in graph transformation systems [Geiß et al., 2006; Taentzer, 2004]

Graph transformation involves the rule-based manipulation of graphs, where each graph rule application leads to a graph transformation step. The process of generating new graphs from an original graph is similar to how it occurs for shape grammars — for example, Krishnamurti and Stouffs [1993] point out a unified description of different kinds of grammars. In particular, applying a graph rule $L \rightarrow R$ involves (1) searching an occurrence of the pattern graph L in a given graph G by detecting isomorphic sub-graphs of L, and (2) replacing the chosen occurrence of L by the replacement graph R [Ehrig et al., 2006]. In other words, if shapes and shape rules are represented as graphs and graph rules, respectively, graph transformation systems can be used to represent and explore a shape grammar on a computer system.

4.2.2 Overview of graph-theoretic representations

Mapping between shapes and graphs is needed if a graph transformation system is to be used for the implementation of shape grammars. Various approaches to represent shapes as graphs have been proposed before, each having specific benefits and drawbacks. Wortmann [2013] proposes a number of criteria to which a graph-theoretic representation should comply. First and foremost, the representation should capture not only the defining properties of shapes, which are maximal elements, intersections, but also the embedding and part relations. Second, the representation should be compact (as few nodes and edges as possible) to reduce the computational complexity. Third, the representation should be complete and sufficiently transparent to make the mapping between shapes and graphs intuitive. Table 4.1 points out the different approaches found in the literature and summarizes some of the main properties of these approaches. With the exception of the first approach, which is added because of its simplicity, maximal elements are maintained in every approach. The main difference between the remaining approaches is the kind of information represented either in the graph nodes or in the graph edges. Depending on the nature of the shape to be represented (for example, the number of intersecting lines or the algebra in which it is defined), one approach is preferable to the other.

	Maximal elements?	Nodes represent:	Edges represent:	Example references
Direct graph or plan graph		Intersections or endpoints	Lines	[Yue and Krishnamurti, 2014]
Overcom-plete graph	✓	Intersections or endpoints	Maximal lines	[Keles et al., 2010]
Hypergraph	✓	Intersections or endpoints	Maximal lines	No reference known
Inverted graph or edge graph	✓	Maximal lines	Intersections or endpoints	[Wortmann, 2013]
Elaborated or part-relation graph	✓	(Geometric) elements	Relations	[Grasl, 2013; Heisserman, 1991]

Table 4.1: Different kinds of graph-theoretic representations of shapes and some of their properties.

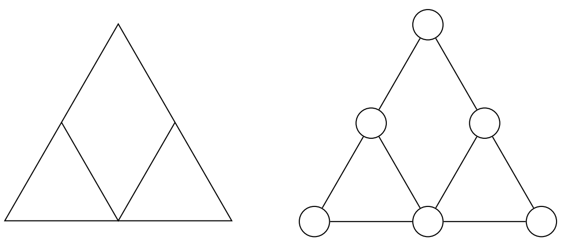


Figure 4.2: Shape defined in the algebra U_{12} (left) and the corresponding direct graph representation (right). Graph nodes are commonly drawn as circles and graph edges are drawn as lines between the nodes.

The most straightforward approach is to construct a *direct graph* [Wortmann, 2013] or *plan graph* [Grasl and Economou, 2013], in which intersections or endpoints are represented as graph nodes and line segments are represented as graph edges. The terms 'direct graph' and 'plan graph' refer to the same approach, which was initiated in the early work of Steadman [1976]. Figure 4.2 shows an example of a shape defined in the algebra U_{12} and the corresponding direct graph representation. The direct graph approach is an intuitive way of representing shapes because, for shapes in the algebra U_{12}, the shape and its graph representation are visually similar. However, this representation is incomplete in the sense that maximal elements are not preserved, but instead they are decomposed into smaller line segments. As a result, it is not always possible to detect similar shapes using a single pattern graph. For the shape in Figure 4.3(a) the large outer triangle is similar to the small triangles inside, but the graph representation of the large triangle (b) is different from the graph representation of the small triangles (c). As a result of this decomposition, two different pattern graphs, defined in two different rules, would be needed to detect the similar triangles in the shape. An example of this direct graph approach can be found in the work of Yue and Krishnamurti [2014].

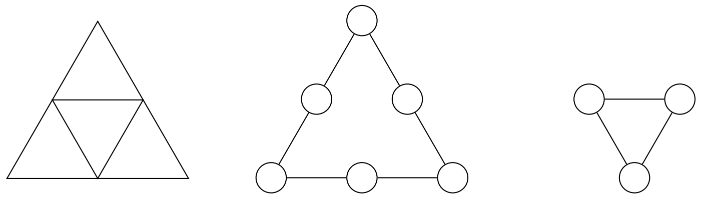


Figure 4.3: While the large triangle is similar to the small triangles in shape (a), the direct graph representations of the large triangle (b) and the small triangles (c) are different.

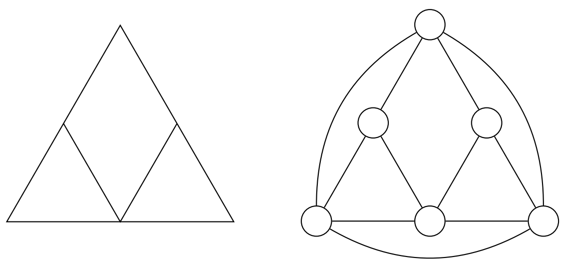


Figure 4.4: Shape defined in the algebra U_{12} (left) and the corresponding overcomplete graph representation (right).

A second approach involves the construction of an *overcomplete graph* — as proposed in the work of Keles et al. [2010]. The overcomplete graph representation is similar to the direct graph representation, except that for every line segment between each pair of points coincident with the same straight line, an additional edge is added to the graph. Figure 4.4 shows an example of a shape defined in the algebra U_{12} and the corresponding overcomplete graph representation. In this case, a single pattern graph is sufficient to return both the large triangle and the small triangles. The overcomplete graph representation preserves maximal elements and their decompositions, though at the expense of compactness and intuitiveness.

A third approach, somewhat similar to overcomplete graphs, is to construct a *hypergraph*. A hypergraph is a generalization of a graph in which an edge can connect an arbitrary set of nodes (instead of just a pair of nodes). Instead of adding an edge between each pair of points coincident with the same straight line, all these points are connected with a single hyperedge. Figure 4.5 shows the hypergraph representation of an example of a shape. The hyperedges are shown as sets (gray fill) that can connect any number

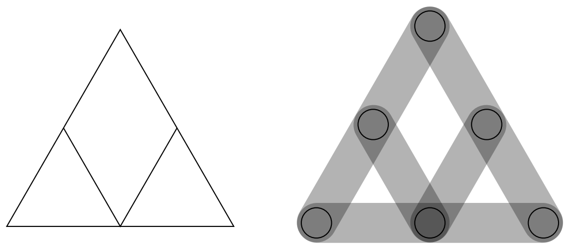


Figure 4.5: Shape defined in the algebra U_{12} (left) and the corresponding hypergraph representation (right).

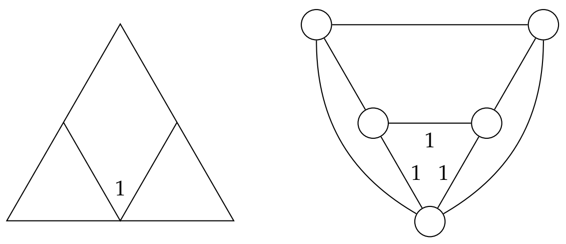


Figure 4.6: Shape defined in the algebra U_{12} (left) and the corresponding inverted graph representation (right).

of nodes. The hypergraph representation preserves maximal elements, and the compactness is preferable to overcomplete graphs because less graph edges are needed to represent lines with multiple intersections. Currently, there are no precedents to this approach in the context of implementing shapes and shape grammars, presumably due to the limited availability of practical hypergraph grammar libraries [Grasl and Economou, 2013].

A fourth approach involves the construction of an *inverted graph* [Wortmann, 2013] or *edge graph* [Grasl and Economou, 2013], in which maximal lines are represented as nodes and their intersections as edges. In other words, this approach is the inverse of the direct graph approach. Figure 4.6 shows the hypergraph representation of an example of a shape. As edges can only represent intersections between two maximal lines, at most, additional information is needed to represent intersections with more than two lines intersecting at the same point. In the approach of Wortmann [2013], such intersections are handled by labeling the edges that represent intersections with more than two lines (see label '1' in Figure 4.6). For each pair of elements that intersects at the same point, the corresponding edge is given an identical label. The inverted graph approach represents maximal elements (as nodes), while maintaining sufficient compactness and allowing similar shapes to be detected using a single pattern graph. On the other hand, the representation of lines as nodes and intersections as edges might seem counterintuitive. The work of Wortmann [2013] describes the algorithms needed for constructing graphs for shapes in the algebra U_{12}, based on the inverted graph approach. Also, this work contains four heuristics to represent shapes with different kinds of intersections of maximal lines (full intersections, partial intersections, and empty intersections). For example, empty intersections occur for parallel lines, and partial intersections appear for non-parallel lines without a physical intersection. In these cases, the shapes to be represented are not fully connected, and selecting the intersection points depends on the perception of the user.

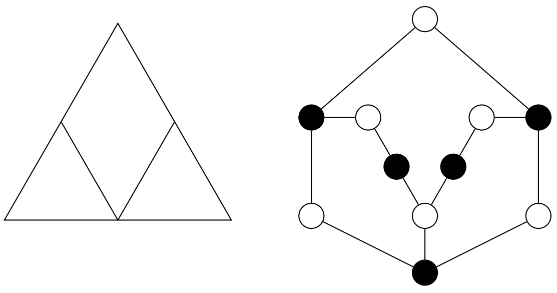


Figure 4.7: Shape defined in the algebra U_{12} (left) and the corresponding overcomplete graph representation (right). Two different node types are used to represent points (white) and maximal lines (black).

A final approach is to construct an *elaborated graph* [Wortmann, 2013] or *part-relation graph* [Grasl and Economou, 2013], in which all geometric elements (maximal lines, intersections, and endpoints) are represented as nodes and the relations between these elements as edges. Figure 4.7 shows the part-relation graph representation of an example of a shape, using two different node types to represent points (white) and maximal lines (black). Maximal elements are represented as nodes and similar shapes can be detected using a single pattern graph. The origin of the part-relation approach is traced back to the work of Heisserman [1991] that describes a graph-theoretic boundary representation of three-dimensional solids. In particular, a part-relation graph is constructed including geometric elements such as *vertex*, *edge-half*, *loop*, *face*, *shell* or *solid*, and different kinds

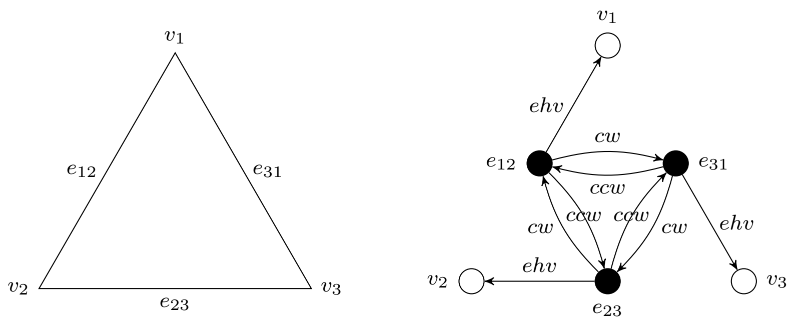


Figure 4.8: Part-relation graph representation proposed in the work of Heisserman [1991]. The edge types include next clockwise edge-half (cw), next counterclockwise edge-half (ccw), and vertex of edge-half (ehv).

of relations (Figure 4.8). In later work of Grasl and Economou [2013], the construction of a part-relation graph is focused on representing maximal elements and their intersections, and using a single kind of relation. The former approach of Heisserman is more aimed towards efficiently accessing information about shapes — for example, what is the next adjacent edge? — while the latter approach is aimed at compactness and representing maximal elements.

4.2.3 Discussion

Several graph-theoretic representations of shapes are available — including direct, overcomplete, hyper- , inverted, and part-relation graphs. While the direct graphs approach is the most intuitive, this representation is incomplete because it does not preserve maximal elements. The other approaches maintain the defining properties of shapes (maximal elements and the part and embedding relations). This means that shapes represented by such graphs behave in a similar way as the shapes themselves. In other words, shapes represented by graphs can be decomposed into various parts (similarly to the part and embedding relations) — thus new shapes might emerge from applying rules to them. Figure 4.9 shows how an example of a shape and its corresponding part-relation graph can be decomposed in a similar way. Moreover, graphs can easily be implemented on a computer system (due to their symbolic nature) — making graphs a highly suitable formalism for the computer implementation of grammars.

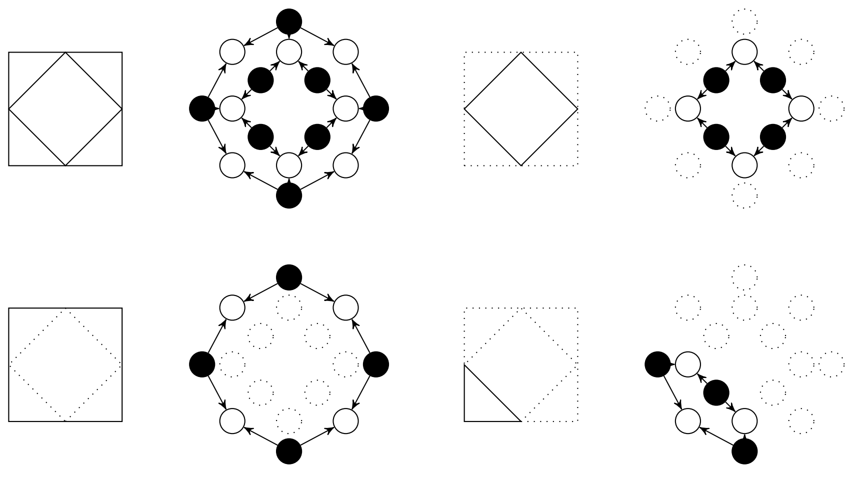


Figure 4.9: Graphs can be decomposed in to various sub-graphs, similarly to how this occurs for shapes. In this examples, shapes are represented by part-relation graphs.

Using graphs to represent shapes, it is also possible to support subshape detection for similar shapes using a single search pattern, and to enable both non-parametric and parametric transformations. Such transformations are enabled because graphs only represent the topological structure of the shape, and therefore, each graph representation corresponds to a large number of geometric shapes. Additional constraints can be imposed on the graph to limit the number of shapes to specific geometric realizations (Table 4.2). As a result, the graph-theoretic representation enables non-parametric (isometry and similarity) and parametric transformations.

The selection of the most suitable approach among the available graph-theoretic representations depends on the design task at hand. For example, the compactness of the different representations depends on the number of (coincident) intersections; for shapes with only a few coincident intersections, the inverted graph representation is the most compact. As each graph-theoretic representation has slight drawbacks and benefits compared to the others, the selection of a particular approach is often a

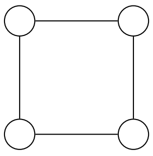
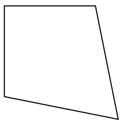
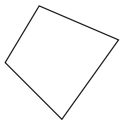
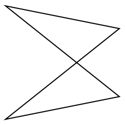
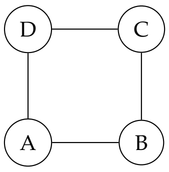
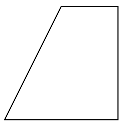
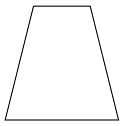
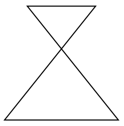
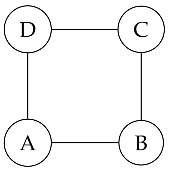
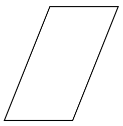
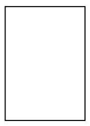
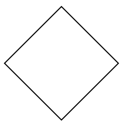
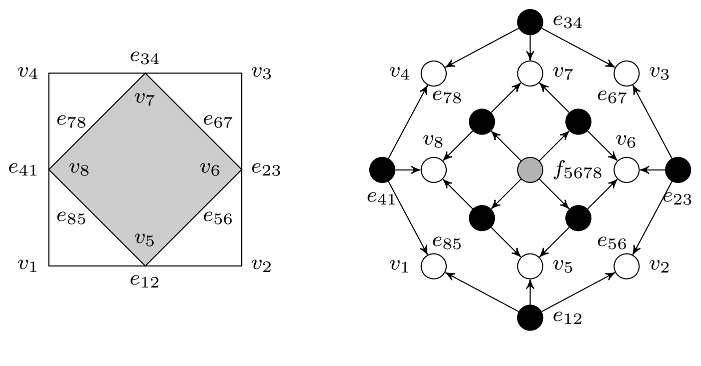
Graph	Possible shapes		
			
 $AB \parallel CD$			
 $AB \parallel CD$ $AD \parallel BC$			

Table 4.2: Some examples of direct graph representations and their geometric realizations. For each constraint imposed on the graph representation, the number of possible shapes is reduced. The unconstrained graph (top) represents all quadrilateral shapes.

practical one. For example, the presence or absence of geometric elements other than lines and points and the limited availability of practical hypergraph grammar libraries are two feasible arguments for choosing the part-relation graphs approach.

The approach of representing shapes as part-relation graphs is complete and the resulting graphs are rather transparent and easy to interpret (though some authors disagree — *"[the part-relation graph] is potentially confusing in its employment of nodes for both elements and intersections.* [Wortmann, 2013]). In order to realize a complete mapping between shapes and graphs, additional coordinate geometry should be associated with the graph nodes, because graphs only represent the topology of shapes. An example of representing a shape as a part-relation graph, which includes three node types and coordinate geometry, is shown in Figure 4.10. The part-relation graph has the benefit of being flexible in representing node types other than points and lines. Some additional node types that might be of interest include faces and labels, but also non-geometric or semantic concepts such as space, wall, window, door, etc. In this regard, such graph grammars deviate from traditional shape grammar theory and previous implementation approaches. Indeed, once semantic node types are included in the graph representation of a shape, this 'disambiguates' the shape, meaning that the emergent nature of shapes is no longer supported. Nevertheless, grammars that include semantic node types, might be as useful as purely shape grammars; for example, in later stages of the design process, thus beyond the sketch stage, where emergence is of lesser importance. As a result, part-relation graphs support both shape grammars and more semantic grammars, depending on which node types are chosen.



$v_1 : (0,0)$
$v_2 : (2,0)$
$v_3 : (2,2)$
$v_4 : (0,2)$
$v_5 : (1,0)$
$v_6 : (2,1)$
$v_7 : (1,2)$
$v_8 : (0,1)$

Figure 4.10: Part-relation graph representation of an example of a shape. The node types used include points (white), maximal lines (black), and faces (gray). The mapping from graph nodes to coordinate geometry is shown on the right of the figure.

4.3 A new approach for grammar implementation

So far, graphs have been used only to implement (parametric) shapes but, as shown in the remainder of this chapter, they may also be used to implement richer (semantic) kinds of grammars. The computer implementation of a grammar, using a graph-theoretic representation, involves three steps. The first step is to define a suitable graph-theoretic representation, by specifying which node and edge types should be considered. This can be done by creating a *type graph*, which specifies which node and edge types are included in the graph. The second step involves the actual construction of the graph-theoretic representation of the shape and the mapping of graph nodes to the coordinate geometry. This is needed in order to constrain the topology to a specific geometric shape — either fully constrained or allowing several (non-)parametric transformations. The third step is the specification of the graph rules by adding application conditions to guide and control rule application. The theory of graph transformation provides several approaches to specify such application conditions. This approach proposed here differs from previous approaches, because both shape grammars and more semantic grammars can be implemented, depending on the node and edge types chosen.

4.3.1 Step 1: Defining a type graph

In accordance with the part-relation graph definition, geometric and semantic elements are represented as nodes, and the relations between these elements as edges. The question remains which elements and relations between these elements should be maintained in the graph-theoretic representation. Several possibilities exist, and choosing how to represent geometric elements and their relations not only influences the compactness of the graph-theoretic representation, but also the time needed to access information about the design or shape. For example, the representation used in the work of Heisserman [1991] is aimed at enabling efficient reasoning about shapes. In particular, multiple edge types (next clockwise edge, next counterclockwise edge, and vertex of edge) are maintained (Figure 4.8). As a result, it is possible to efficiently perform adjacency queries on a (three-dimensional) shape — though at the cost of losing compactness. In the context of shape grammar implementations, however, compactness is a critical issue because of the NP-completeness of the sub-graph isomorphism problem. In other words, the time required to detect sub-graph isomorphism increases exponentially as the size of the graph grows. This would mean that automatic rule application becomes infeasible for large shapes or complex designs.

In order to specify type descriptions of graph nodes and edges, this can be done using *type graphs*. A type graph is a distinguished graph $TG = (V_{TG}, E_{TG})$, in which V_{TG} and E_{TG} are called the node and the edge type sets, respectively [Ehrig et al., 2006]. In general, the nodes in TG represent the collection of node types in a graph G, and the edges in TG represent the collection of edge types in a graph G. The benefit of using a type graph, instead of a type set for nodes and edges, is that a type graph also specifies the number of nodes that may be connected to edges of the given edge type. This number of nodes that can be connected, either at the source or the target end of an edge, is called the multiplicity of the given edge. Additionally, type graphs are specified with node type inheritance; in other words, each node type can have one or more parent node type(s) from which it inherits the multiplicity and edges.

As an example, a type graph with six node types (face, line, point, and three subtypes of the point element) and two edge types (line-point and face-line) is shown in Figure 4.11. The three different kinds of points are used to distinguish between endpoints, intersections, and projected intersections of maximal line elements. An abstract node type (point) is defined as the parent node type, from which these three kinds of points inherit the multiplicity and relations. Figure 4.11 shows this type graph, including five node types, one abstract node type, and two edge types. The multiplicity of the edge types is indicated at the source and the target end of the edges, where an asterisk (*) indicates that an indefinite number of connections is allowed.

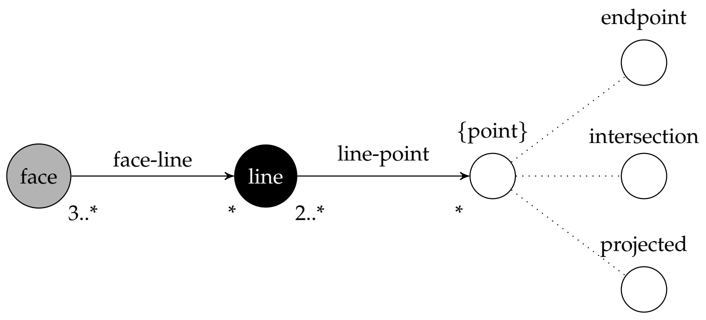


Figure 4.11: Example of a type graph for the construction of part-relation graphs, including five node types (face, line, endpoint, intersection, and projected intersection), one abstract node type (point), and two edge types (line-point and face-line).

While the type graph defined in Figure 4.11 is sufficient to represent shapes in the algebra U_{12} (see Figure 4.10), other useful type graph definitions can also be devised. Indeed, the definition of a type graph depends on the characteristics of the shape grammar to be implemented. For the computer implementation of the Palladian grammar [Stiny and Mitchell, 1978a], an approach with only three node types (orientation, room, and portico) and two edge types (east–west and north–south) has proven to be sufficient to represent the floor plans [Grasl, 2012]. Although these types of generated graphs do not contain a lot of information, they can be mapped to shapes due to implicit knowledge about the floor plans — including "*the orthogonal nature of the designs, the reduced formal vocabulary, the dominance of the underlying grid, and the uniform placement of doors and windows along an axis.*" [Grasl, 2012]. In the approach of Grasl [2012], several semantic concepts are represented as node types, such as 'room' and 'orientation' (Figure 4.12). This is an important difference to the theory of shape grammars, where all kinds of information are strictly described in terms of visual shapes in an algebra U_{ij} and labels in V_{ij}.

Humans are very good at interpreting shapes and readily make meaning from visual patterns. For example, a room in a two-dimensional floor plan is generally perceived as a void enclosed by walls, and can easily be detected by just 'seeing' the floor plan. On the other hand, if the same approach — representing a floor plan in terms of strictly geometric elements (maximal lines and points) — would have been followed for the computer-implemented grammar, this would result in an overly complex

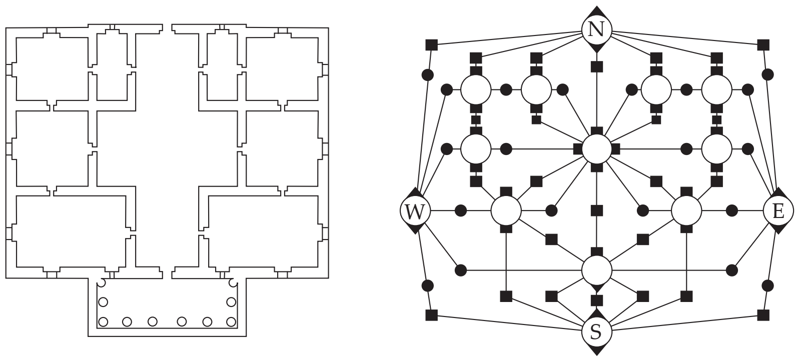


Figure 4.12: Part-relation graph representation of a floor plan, including three node types (orientation, room, and portico) and two edge types (east–west ● and north–south ■). The different node types are indicated using markers. Reproduced from the original image appearing in Grasl [2012].

graph with a large number of nodes and edges. Also, the search pattern of rules would be highly complicated, even for simple rules such as detecting a specific room in a floor plan. In some cases, the introduction of semantic node types might simplify the graph-theoretic representation, thereby enhancing the compactness of the graph. Other semantic concepts that can be represented as node types include architectural elements such as 'wall', 'column', 'window', and 'door', but also labels and metadata.

In AI terminology, determining which elements and relations are to be maintained in the graph-theoretic representation is called the process of *knowledge engineering* or *ontological engineering* [Russel and Norvig, 2010]. An ontology — meaning 'the study of the things that be' —- describes a set of types, properties, and relations between types. In other words, the ontology determines which information can be represented, and how this information is 'interpreted' by an information system. In general, the knowledge engineering step has to be performed for each task individually, so there exists a broad range of different type graphs. First, a type graph containing only point and line types is more suitable for early design stages, in which the designer might benefit from the typical emergence characteristics of shapes. As shown earlier in this thesis, part-relation graphs that contain only points and lines behave in a similar way as shapes. Second, by introducing semantic node types, the graphs become disambiguated, thereby omitting the emergent nature of shapes and shape grammars. Such type graphs can be more useful in later stages of the design process, because emergence might then be of lesser importance, and because designs can then be represented in a more compact way, thereby facilitating automatic rule application.

4.3.2 Step 2: Constructing attributed graphs

The second step involves the construction of the part-relation graph of shapes, in correspondence with the type graph defined in the first step. At this moment, the resulting graph only represents the topology of the shape or design and is not yet limited to a specific geometric realization. In this sense, the part-relation graph accounts for several parametric or non-parametric transformations of the shape. A mapping of graph nodes to coordinate geometry is needed in order to constrain the topology to a (set of) specific geometric shape(s). This can be achieved by extending the classical notion of a graph $G = (V, E)$ to an attributed graph $G = (V_G, V_D, E_G, E_{NA}, E_{EA})$ [Ehrig et al., 2006], in which:

- V_G and E_G are the common graph nodes and edges, respectively;
- V_D is the set of data nodes (attributes);

- E_{NA}, E_{EA} are the node attribute and edge attribute edges, respectively.

In other words, an attributed graph contains data nodes V_D that can be associated with both graph nodes V_G and edges E_G, using special attribute edges E_{NA} and E_{EA}, respectively. Attribute edges are different from graph edges, because the source of these edges can be either a graph node or an edge node, and the target of these edges is always a data node. Data nodes or attributes are a special kind of graph nodes that contain an attribute type (numeric, string, boolean), a name, and a value. As a result, multiple attributes can be associated with either nodes or edges of graphs. Attribute values are also taken into account in graph transformations, where they can be modified in going from the pattern graph to the replacement graph. In this respect, attributed graphs differ from labeled graphs, because labels of graph objects are preserved in graph transformations.

In general, graph attributes represent different kinds of information that are not topological in nature, for example numerical features or textual descriptions. At the very least, attributes are needed to constrain the topology of graphs to specific geometric shapes. In particular, the point node type should contain numerical attributes 'x', 'y', and 'z' for three-dimensional shapes. Attributes can also be added to semantic node types (space, wall, column, window, door, and label) to describe functional aspects of spaces, to characterize the material properties of walls and columns, to describe the geometric properties of doors and windows, or to assign a value to label nodes. Attributes further specify the graph-theoretic representation of the shape or design in a way that can be implemented on an information system. As a result, such graph-theoretic representation is complete in the sense that topological, geometric, and descriptive information are included in a single representation of graph objects and attributes.

Figure 4.13 shows a single-room floor plan and the corresponding attributed part-relation graph. The node types used in this example are point (white), edge (black), space (light gray), wall (dark gray), door (+), and window (×). The introduction of semantic nodes (such as wall, door, and window) enables the representation of more than purely geometric concepts. Attributes are in fact a special kind of (data) nodes, however, they are commonly not visualized as such, instead they are shown as labels associated with the nodes. In this example, the attributes are coordinate geometry (x and y), function (f), wall thickness (t), and width (w). The resulting graph can directly be mapped to its shape equivalent with no implicit knowledge needed — which is not the case, for example, in the approach of Grasl [2012] (see Figure 4.12).

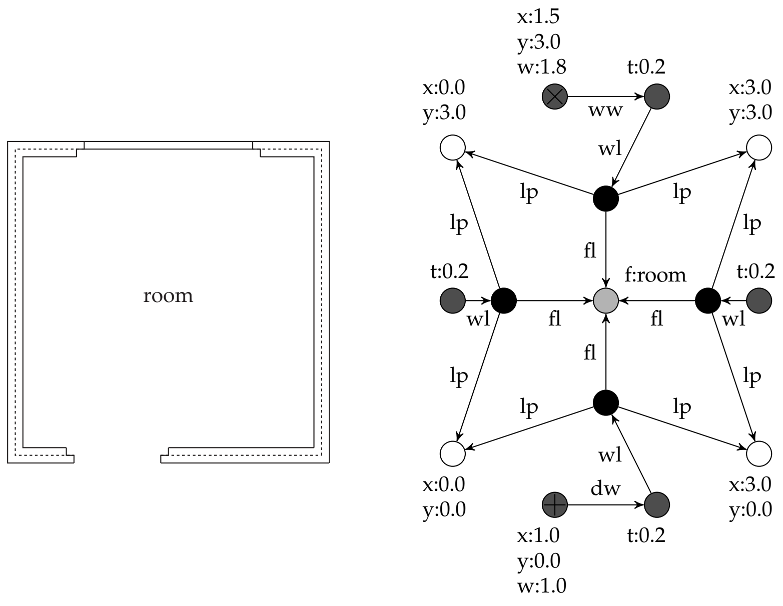


Figure 4.13: Attributed part-relation graph representation of a floor plan. The node types are point (white), edge (black), space (light gray), wall (dark gray), door (+), and window (×). The edge types are face-line (fl), line-point(lp), wall-line (wl), door-wall (dw), and window-wall (ww). The attributes are coordinate geometry (x and y), function (f), wall thickness (t), and width (w).

4.3.3 Step 3: Defining graph rules

The final step involves the definition of graph rules $L \rightarrow R$ that contain a pattern graph (L), a replacement graph (R), and a morphism from L to R. The pattern graph defines the search pattern of the rule, of which an occurrence must be found in a graph G in order to apply the rule $L \rightarrow R$ to G. This can be done automatically by a graph transformation system or manually by the user. The replacement graph describes the action of the rule — either adding graph objects (nodes and edges), deleting graph objects, or modifying graph attributes. Graph objects in L that are not in R are to be deleted, while graph objects in R that are not in L are to be added. Special attention is required for rules that delete graph nodes, because such rule applications may result in dangling edges, which are edges with a missing source or target node. The morphism between L and R specifies which graph objects of L are preserved in R. Figure 4.14 shows a graph rule (top) and a possible application (bottom). The rule morphism is indicated by showing identical numbers for each graph object in L and R.

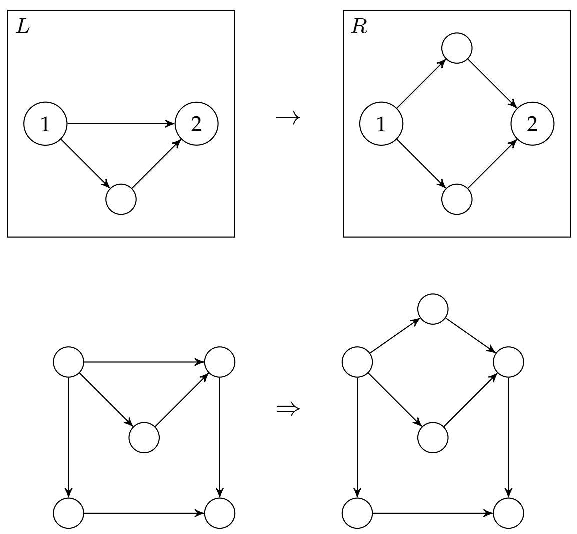


Figure 4.14: A graph rule $L \to R$ (top) and an example of graph transformation (bottom). The two nodes indicated in the pattern graph L are preserved in the replacement graph R, while the third node in L is deleted and two new nodes are added in R.

Graph rules also specify transformations on graph attributes; for example, modifying an attribute value from the pattern graph L to the replacement graph R. The pattern graph of a rule contains *attribute variables* to which a value is assigned once this rule is matched to a given graph. If this is the case, the attribute variables take the value of the matched graph object attributes. The scope of these attribute values is the rule itself. As a result, attribute values defined in L can also be used or modified in R. The replacement graph either contains new attributes values or attribute variables from L. Moreover, the attribute variables can be combined to expressions — including arithmetic expressions, boolean expressions, or conditional expressions. These expressions are evaluated once the rule is applied to a given graph.

A straightforward example of a rule that modifies the attribute values is a rule to scale a quadrilateral shape. In this case, the topology of the shape is preserved, while the coordinate geometry is altered according to the scaling ratio. The graph rule shown in Figure 4.15 illustrates the use of attribute variables (X_1, Y_1, X_2, and Y_2) to scale down a quadrilateral shape. In the pattern graph of the rule, these attribute variables are combined in expressions to modify the coordinate geometry of the point nodes according to a scaling ratio (t).

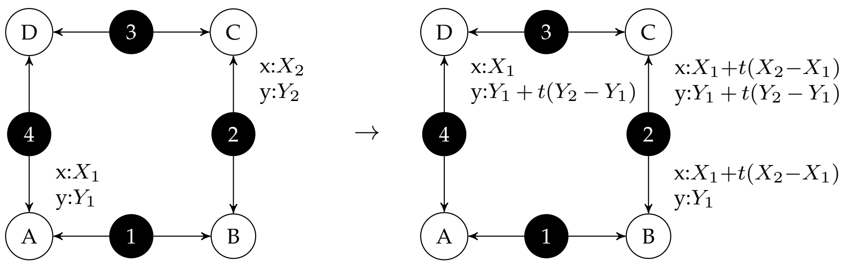


Figure 4.15: A graph rule to scale a quadrilateral shape using attribute variables X_1, Y_1, X_2, and Y_2.

Another aspect of graph rules is that they can be subject to *application conditions* to further specify in which conditions, rules can be applied. In the field of graph transformation theory, such application conditions are defined as *attribute conditions* and *negative application conditions* [Ehrig et al., 2006]. Attribute conditions specify restrictions on the attributes of graph objects in the pattern graph of a rule. Such attribute conditions are defined as conditional expressions using constants and attribute values declared in the pattern graph of the rule. In general, attribute conditions define descriptive requirements — including geometric constraints (length or area) or functional constraints. As a result, they can be used to constrain parametric topologies to specific geometric realizations. For example, in order to constrain the rule in Figure 4.15 to squares, instead of all quadrilateral shapes, attribute conditions should constrain opposing lines to be of equal length ($|AB| = |CD|$ and $|AD| = |BC|$), at least one pair of adjacent lines to be equal length ($|AB|$=$|BC|$), and the diagonals to be of equal length ($|AC|$=$|BD|$).

On the other hand, negative application conditions specify requirements for the non-existence of graph objects. Negative application conditions can be specified for graph nodes, edges, or even a specific sub-graph. In general, they are used to ensure that rules can only be applied if specific graph objects are absent. A common usage of negative application conditions is to avoid the generation of duplicate graph objects by specifying a negative application condition on the replacement graph of the rule. Both the attribute conditions and negative application conditions can be specified for each graph rule individually and, therefore, they provide a convenient way to guide and control rule application.

4.4 Evaluation of the proposed approach

The examples of graphs and graph rules shown so far illustrate the most important steps for a graph-theoretic representation of shapes, and by extension designs — which are (1) defining type graphs, (2) constructing attributed part-relation graphs, and (3) specifying application conditions. Using the approach proposed in this thesis, grammars can be implemented on a computer system, thereby providing the basis for a new kind of CAD tools for design generation, exploration, and development. In order to evaluate whether such computer implementations are feasible — namely, that designs can be generated in reasonable time — a benchmark for measuring the generation time is devised. A second important research question is whether the proposed approach is effective when it should be generalized from 'showcase' grammars to more complex shape grammars.

4.4.1 Graph grammar benchmarks

The generation time of designs in the language of a grammar is an important measure of the responsiveness and feasibility of the proposed approach. This generation time should be kept below a reasonable threshold for the implementation to be of practical use. Indeed, a low generation time enhances an intuitive way for design space exploration, because new design paths are quickly unveiled, thereby making available new areas in the design space. For common tasks involved with shape grammar implementations, a generation time below a few seconds should be reasonable — though for very complex and specialized designs or rules, a higher threshold can sometimes be justified. In general, the most complex (runtime intensive) step in the generation process is the automatic detection of (parametric) sub-shapes to which rules can be applied. In this case, the sub-shape detection step corresponds to solving a sub-graph isomorphism problem, sometimes called *pattern matching*.

General benchmarks for graph transformation systems [Geiß and Kroll, 2007; Varró et al., 2005] demonstrate how the time needed for pattern matching is influenced by (1) the size of the pattern graph of rules and (2) the density of the graph to which rules are to be applied. In general, matching a large pattern graph to a dense graph will take longer than matching a small pattern graph to a graph with fewer connections. In order to demonstrate what this means for graph-theoretic implementations of shape grammars, two benchmark scenarios are defined — the first to measure the time needed for finding all matches of a rule to a given shape, and the second to calculate only the first matching of a rule. The reported benchmarks are

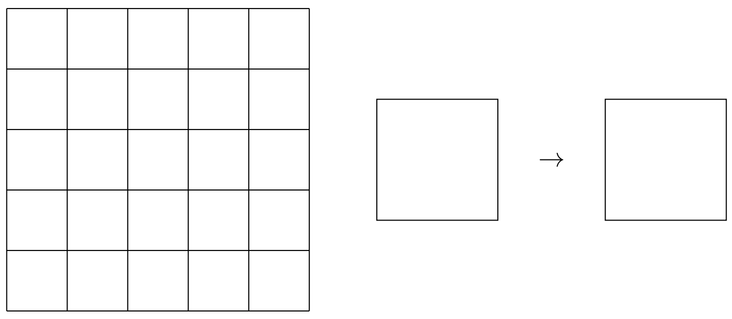


Figure 4.16: Benchmark scenario for finding all rectangles embedded in a two-dimensional (5×5) grid (left), using a parametric shape rule (right).

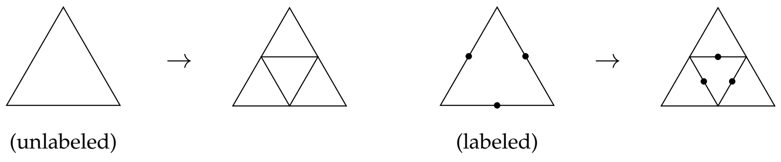
performed using AGG [1], an interpreted graph transformation tool written in Java [Taentzer, 2004], on an Intel Core 2 Quad processor (3.00GHz). A 'warm-up' phase of a few iterations is taken into account and the measurements of the benchmarks are averaged over 30 iterations to avoid noise coming from external influences.

A first benchmark is selected for the scenario where a small part of a design or shape is modified. This scenario is characterized by a large and nearly-static graph, where each rule application modifies only a small part of the graph. This is particularly useful for shape grammars that emphasize enumeration of possible alternatives where rules can be applied. Examples of shape grammars in which enumeration is important include the transformation grammar of Eloy [2012] and the Malagueira grammar [Duarte, 2005a]. In such cases, the time needed for finding all matches exceeds the time needed for executing the actual rules. As an example, consider a two-dimensional (5×5) grid, in which the task is to find all rectangles that are embedded in this grid (Figure 4.16). This task is typically perceived as difficult to solve for humans, because they easily overlook particular rectangles that are 'hidden' in the grid. Also, this task is particularly useful as a benchmark because it involves both sub-shape detection and parametric rules. In order to do this, an identity rule is defined that operates under parametric transformations; in other words, the pattern shape can be matched to all kinds of rectangles in the grid. The graph-theoretic representation of the grid contains 48 graph nodes and 72 graph edges. The time needed to find and store a total number of 225 rectangles using the proposed set-up is measured to be around half a second. In the case of a 6×6 grid, 441 rectangles are found in less than one second.

[1] `http://user.cs.tu-berlin.de/~gragra/agg/`

A second benchmark is selected for the scenario where a rather small design or shape is modified, but this time in a more profound way. In this case, the time needed to perform the entire rule application (both pattern matching and graph transformation) is measured. Only the time needed to calculate the first matching of a rule is measured (instead of the time needed for finding all matches). This scenario corresponds to shape grammars where rules are applied one-by-one and where the selection of alternatives is less important. Examples of such shape grammars often use labels to guide rule application, thereby constraining the number of possible rule applications within reasonable limits [Flemming, 1987; Koning and Eizenberg, 1981].

As an example, the shape rule in Figure 4.17 (left) is applied recursively 50 times, and the time needed for each generation of a new shape is measured. The generation time increases exponentially as the number of graph objects (nodes and edges) becomes larger. For long rule sequences (>50 ap-



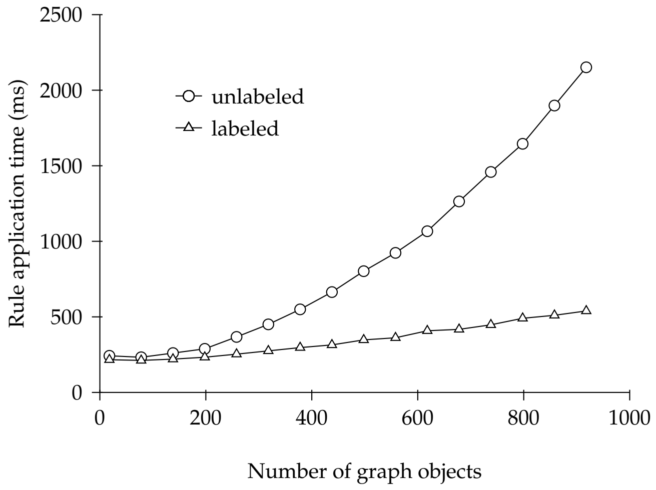


Figure 4.17: The generation time (ms) measured for a sequence of 50 applications of an unlabeled shape rule (left) and a labeled shape rule (right). The generation time increases as the number of graph objects (nodes and edges) becomes larger.

plications) and large graphs ($>$1000 graph objects), the average generation time exceeds 2 seconds. It is possible to reduce the generation time for complex shapes or designs by choosing a more compact graph-theoretic representation or by using more constrained rules. For example, the generation time of the rule in Figure 4.17 (right) is remarkably lower. In this rule, labels are used to limit the number of pattern matches that can be found to one, which is the most inner triangle.

In general, the results of the benchmarks demonstrate that the implementation of shape grammars using a graph-theoretic approach is feasible in the sense that designs can be generated in reasonable time. Moreover, because the measurements of the benchmarks are performed on a general graph transformation system, further improvements in performance can be expected when a more specific transformation system should be used. The implementation of shape grammars as graph grammars is shown to be a valuable approach, at least from a technical point of view. The question remains what can be the added value of using graph grammars over shape grammars? In order to provide an answer, the *Rabo-de-Bacalhau transformation grammar* (RdB-tf grammar), originally developed on paper by Eloy [2012], has been implemented using the proposed approach to compare the characteristics of both grammars.

4.4.2 Implementation of the RdB-tf grammar

In many papers describing shape grammar implementations, the case studies provided with the proposed implementation approach include a few 'showcase' grammars which are often quite straightforward to implement. In practice, the more useful and extensive shape grammars, such as the ones describing the corpus of prairie houses [Koning and Eizenberg, 1981], Malagueira houses [Duarte, 2005a], or the vernacular styles of traditional Portuguese houses [Eloy, 2012], are far more complex to implement. In contrast to showcase grammars, such complex grammars contain a large number of rules, or they are defined in different algebras U_{ij}, V_{ij}, and W_{ij}. These grammars are developed on paper and relatively few have been implemented on a computer system — some exceptions include the work of Grasl [2012] and Granadeiro et al. [2013]. The question whether grammars of such extent can be implemented on a computer system, and what might be the potential benefits of doing so, remains largely unanswered due to the limited availability of such implementation efforts in the literature.

In order to investigate whether the graph-theoretic implementation approach of complex grammars is possible, and more importantly, whether designers benefit from such implementation, the RdB-tf grammar developed by Eloy [2012] has been implemented using the proposed approach.

The original RdB-tf grammar was only available in the form of rules written on paper, and thus could not be explored on a computer system. The RdB-tf grammar contains rules for the refurbishment of housing designs in order to meet the current comfort and functional standards, also depending on specific client needs and cost requirements. Eloy and Duarte [2014] describe the process undertaken to develop the grammar, and discuss how both designer knowledge and knowledge acquired from other experiences are incorporated in the grammar. The RdB-tf grammar provides an interesting case study to evaluate the proposed implementation approach of shape grammars, due to its extensive nature (142 rules), the grammar is defined in different algebras (including weights and labels), the rules contain a lot of dimensional and functional conditions, and also the rules are applied using parametric sub-shape detection. Moreover, the computer implementation can be seen as the next step in the development of a computer-aided methodology to support mass housing rehabilitation.

The full details of the implementation of the original RdB-tf grammar — following the proposed approach of defining type graphs, constructing attributed graphs, and specifying application conditions — are described in Appendix A. An initial RdB housing design has been translated to a part-relation graph (Figure 4.18). This part-relation graph is defined with six node types (point, edge, space, wall, door, and window),

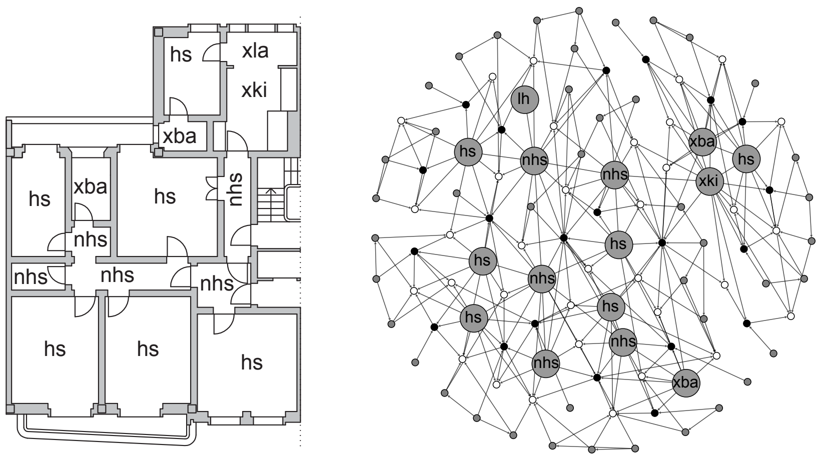


Figure 4.18: Original representation of an existing RdB housing design (left) and the corresponding part-relation graph (right). Point nodes are indicated in white, edge nodes are indicated in black, and the remaining semantic nodes are indicated in gray. The attributes are not shown here, except for the function attribute of space node types.

nhs	Non-habitable space	co	Corridor
hs	Habitable space	co.p	Private corridor
xba	Existing private bathroom	la	Laundry
xki	Existing kitchen	hl	Hall
xla	Existing laundry	ba	Bathroom
be	Bedroom	ba.p	Private bathroom
be.s	Single bedroom	ba.g	Guest bathroom
be.d	Double bedroom	lh	Lift hall
ki	Kitchen	di	Dining room
li	Living room	ho	Home office

Table 4.3: The function attributes used in the Rdb-tf grammar.

seven edge types (space-edge, space-point, edge-point, wall-edge, door-wall, window-wall, and adjacent-space), and six attribute node types (including coordinate geometry, thickness, function, and dimension). For illustrative purposes, the attributes are not shown in the graph representation, except for the function attribute of space node types. An overview of the abbreviated function attributes is shown in Table 4.3. The resulting graph contains 113 nodes, 294 edges, and 126 attributes. This initial housing design serves as a possible starting point for the RdB-tf grammar.

The rules defined in the original RdB-tf grammar encode two transformation strategies — which are (1) moving the kitchen from its original position to strengthen the relationship between the social and service areas, and (2) maintaining the position of the kitchen, and relocating other spaces, thereby keeping construction transformations to a minimum. Part of these shape rules have been translated to a set of graph rules. In particular, the set of rules implemented corresponds to the second transformation strategy encoded in the grammar. Several examples of these rules and their implemented counterparts can be found in Appendix A. One possible derivation of the implemented RdB-tf grammar is shown in Figure 4.19. The visual representation of the grammar derivation is shown for illustrative purposes. Each design state is the result of one or multiple rule applications, which is indicated with a double arrow (⇒). Starting from an initial housing design (state a), the first set of rules locates the kitchen, defines the position of the hall, and locates the private spaces (state b). Next, the room dimension of the bathroom is changed by moving a wall (state c) and the bedroom assignment is permuted due to area criteria (state d). The next set of rules locates the remaining (social) spaces and circulation area (state e). Finally, the connection between the hall and corridor is widened (state f). The derivation was completed in approximately 1.5 seconds, which is in line with the results of the benchmarks reported.

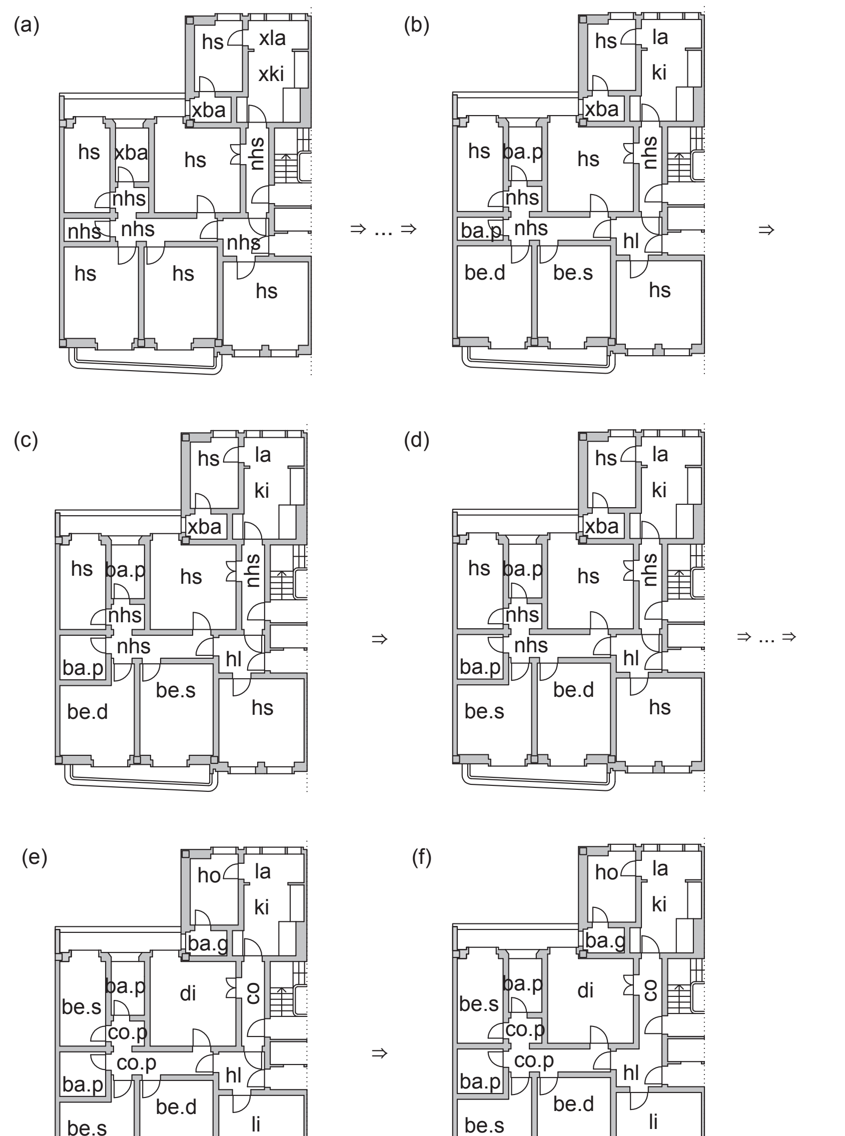


Figure 4.19: Random derivation of the implemented RdB-tf shape grammar. Each design state shown is the result of one or multiple rule applications — (a) existing housing design, (b) assignment of spaces (kitchen, hall, bedrooms, and bathrooms), (c) changing room dimension, (d) permuting bedroom, (e) assignment of remaining spaces, (f) widening the connection between hall and corridor.

Among the key findings and issues encountered during the implementation and comparison of the original and implemented grammar are (1) the difference in representation, (2) the complementariness of shape grammars and graph grammars, and (3) the benefit of graph grammars in revealing new possibilities.

First, the representations used in shape grammars and graph grammars differ in how semantic meaning is treated. On the one hand, shapes and shape grammars nicely cohere with the kind of freedom that is typically associated with creative design. Human designers are able to draw and (re-)interpret shapes and rules during a process of design exploration, and they are extremely good at recognizing visual patterns in shapes. On the other hand, computer systems are only able to 'interpret' the visual information in terms of the ontology used, or in this case, the underlying graph representation. While several architectural or semantic elements, such as spaces, walls, doors, and windows, can indeed be drawn as shapes, they are treated as symbolic entities in the implemented graph grammar. The main benefit of doing so is that the design and the rules can be defined with as few graph objects as possible, thereby avoiding overly large graphs when only geometric node types (point and edge) should be used. Such compact representations positively influence the time needed for rule matching or sub-shape detection, and also make it easier to specify rules. Nevertheless, an intuitive rule editor is of key importance, because the construction of graph rules is error-prone and perhaps less natural than shape rules.

Second, the development of shape grammars and graph grammars has shown to be complementary in the sense that both lead to an alternative understanding of the grammar at hand. In some cases, the direct implementation of the original RdB-tf grammar has resulted in undesired situations; for example, rules being applied in the wrong way or the generation of invalid designs. Such situations are largely due to the original rules being underconstrained or ambiguous. Human designers easily make meaning from visual patterns in the rules and, as a result, such forms of ambiguity often remain unnoticed. This is not the case for computer implemented grammars, in which such forms of ambiguity directly become noticeable. As a result, the effort of computer implementation might lead to a deeper understanding and further development of the grammar. In practice, the implementation effort involves several generate-test iterations (as shown in Appendix A) because shape and graph grammars operate using different principles. This also forces designers to think about several aspects of their grammar in a different way, for example the definition of ontologies and rule application conditions — which demonstrates how the development of graph grammars is complementary to the development of shape grammars.

Third, perhaps the greatest benefit of computer implemented grammars can be gained for extensive and complex grammars that are more difficult to explore manually. For example, the rules of the RdB-tf grammar contain many dimensional and functional conditions that must be satisfied before they can be applied. Using the original grammar, it might be difficult to detect all possibilities where rules can be applied, because of the large amount of conditions that need to be taken into account. On the other hand, with a computer implementation, it is possible to efficiently handle this management of rule applications and to reveal where rules can be applied. In other words, the main benefit of this specific grammar lies in pattern matching and enumerating all the possible design alternatives. This implementation of the RdB-tf grammar might provide the next step in the development of a computer-aided methodology to support mass housing rehabilitation.

To conclude, the implemented RdB-tf grammar demonstrates that the proposed implementation approach remains feasible when it is generalized to a more complex grammar and that designers might benefit from such implementations. The RdB-tf grammar is a highly specialized grammar built for the specific case of transforming existing RdB housing designs. The rules in the grammar are defined entirely before the process of exploring the grammar takes place, as they are inferred from the design process carried out by several architects, who acted as experimental subjects in a procedure to capture the rules [Eloy and Duarte, 2014]. In other words, the design space implicitly represented by the grammar does not change during the process of exploring the grammar. An entirely different situation occurs when rules are created or modified on the fly, thereby changing the structure of the design space. This more closely resembles the act of design space exploration in the broadest sense of interacting with the design space. Supporting or amplifying this act of design space exploration on a computer system requires additional functionalities, such as the possibility to consider multiple design alternatives simultaneously, to backup and recall previous designs, and to navigate in the design space. These topics are the subject of the next chapter.

5
Design Space Exploration

In this chapter, we discuss several amplification strategies for design space exploration in CAD tools, and grammar-based CAD tools, in particular. An overview of those amplification strategies is given in Section 5.1. A literature review of how those strategies have been integrated into CAD tools to date is included in Section 5.2. Furthermore, we consider the concept of tree structures to keep track of an explicit design space of previously generated designs, thereby amplifying several key design space exploration functionalities (Section 5.3).

5.1 Amplification strategies

The theory of shape grammars provides a generative mechanism to represent a design space. As shown in the previous chapter, representing such a design space is proven to be feasible and effective when a graph-theoretic representation of shapes or designs is used, not least when this is generalized to more complex and specialized shape grammars. Clearly, the existence of such a design space representation is a necessary condition for supporting design space exploration, though it may not be a sufficient condition. In order to support design space exploration in the broadest sense, thus involving the exploration of alternatives and reformulating the design space at hand, a more agent-like role for the computer should be pursued in which the limits of human design space exploration capabilities are amplified. In other words, computers should enable *amplification strategies* for design space exploration that reach beyond unaided human capabilities. In doing so, a mixed-initiative interaction [Allen et al., 1999] can be achieved in which each agent (the computer and the designer) contributes to a result that would have been impossible to achieve by either participant alone.

Most previous research efforts towards design space exploration focus on the representation of individual design states or (heuristic) search algorithms, while relatively little work has been done on the design space itself [Woodbury and Burrow, 2006]. As a result, it is difficult to claim, based on empirical evidence, whether design space exploration can be amplified, and if so, how this can be achieved — "*Given the state of design space exploration work today, we have little evidence for amplification of designer action through supporting exploration. We also have little ground for arguing against such amplification. Having almost no systems that engage designers with multiple states, we can make only shallow and tentative inferences about their utility.*" [Woodbury and Burrow, 2006]. In their paper on design space exploration, Woodbury and Burrow [2006] sketch some opportunities for amplification strategies that are vital to the successful exploration of the design space — (1) externalizing design ideas and encoding design moves to help designers foresee the implications of taking particular design moves, and (2) representing an *explicit design space*, which is the record of designs already explored, expanding over time as a library of potentially recoverable work.

5.1.1 Representation, codification, and implication

Much like other kinds of representations used by creative designers (sketches, drawings, and scale models), digital representations (CAD-models, scripts, etc.) provide a means for externalizing mental images or design ideas outside the human mind. For example, in the paper *The Dialectics of Sketching*, Goldschmidt [1991] shows how sketches serve as an extension of imagery. Sketches are a particularly powerful medium for quickly representing mental images or design ideas, after which designers can choose to further develop, modify, or entirely discard their ideas, based on the feedback received from the sketch. The same holds true for other kinds of representations, including digital representations, as they enable designers to externalize design ideas and allow them to act upon those representations. The possibility of being able to generate such representations is an important amplification strategy of CAD tools; for example to generate complex forms that would have been impossible to resolve otherwise. Solid modeling tools enable designers to create partial, three-dimensional models of their design ideas, after which it is possible to evaluate different viewpoints, material choices, etc. The term ‘partial’ refers to the fact that representations describe only those parts and aspects of the design that are of interest at a particular moment in time. BIM models enable designers to model architectural or building elements (such as walls, doors, windows, and building systems), and are rather intended to document and exchange building information. As pointed out in Chapter 2, it is particularly challenging to find suitable representations which are sufficiently meaningful, and also allow enough freedom to associate alternative meanings to the model represented.

In the context of design space exploration, an additional amplification strategy involves the explicit codification of design moves, which are the operations that serve to transform a design relative to the state it was in before that move. In other words, this strategy not only involves the representation of designs, but also the transformations that relate the different states of designs in a structured network, called the design space. The codification of design moves goes hand in hand with the ability of information systems to store these encoded moves and use them for computation. In particular, codification is one of the main amplification strategies behind visual scripting environments and generative CAD systems. Such generative CAD systems enable designers to iteratively encode and evaluate the implications of design moves; for example, in the form of parametric models. As a result, one of the roles of creative designers might be to develop design moves from which designs could be generated in generate–test cycles. Furthermore, the value of codification not only lies in the generation of designs, but the codification of past (successful) design moves also enables designers to recall these moves in future design projects.

An important property of representations, either of designs or a design space, is what can be inferred from them [Woodbury and Burrow, 2006]. Inference or implication is an amplification strategy found in many traditional CAD systems. For example, rendering tools enable designers to explore a model through different viewpoints and under different lighting conditions. In the context of design space exploration, implication is strongly related to codification, and therefore, one of the main amplification strategies underlying generative CAD systems. By explicitly encoding design moves, it is possible to foresee the potential implications of taking a particular design move (testing ‘what–if’ scenarios). In other words, opting for a particular design move grants access to new areas in the design space that can be explored further. The strategy of implication also enables designers to leave certain design decisions open for future consideration. A typical example is parametric modeling, which allows designers to make rapid changes along a limited range of geometric variations. The parametric variables can be given some initial value, which can subsequently be adapted to a specific (geometric) context later in the design process. As a result, parametric models enable designers to test multiple 'what–if' scenarios by trying different parameter values.

External representation, codification of design moves, and implication are three of the main strategies for amplification found in shape grammar theory and its computer implementations. First, designs are represented in different algebras U_{ij}, V_{ij}, and W_{ij} to describe visual information and descriptive information (labels and weights), respectively. In this sense, designs are represented in a partial manner, because only those aspects that are of interest to the design situation at hand are considered. For example, the original RdB-tf grammar, developed by Eloy [2012], is defined in five algebras — including a two-dimensional floor plan with labels and weights, topological relations, and spatial voids. Moreover, due to the part and embedding relations, the given design can be reinterpreted, or decomposed, in multiple ways. Second, shape grammars encode different kinds of design moves in the form of shape rules — each defining a particular transformation between two states of a design. The notion of designing backed by the theory of shape grammars involves both the codification of new rules, and also the storage and retrieval of existing rules in new design situations. In other words, using shape grammars does not relieve designers from their creative task to create new rules or to adapt existing rules in a new context. Third, as shape rules describe an action between two states, their application involves navigating from one state in the design space to another. As a result, it is possible to foresee the implications of making a particular design move. Computer implementations of shape grammars, such as the ones described in the work of McKay et al. [2012], have made the codification of design moves amenable to computer systems. Designs in the language of such a grammar can then be explored on a computer system — either step-by-step or by using some kind of automatic rule selection strategy.

5.1.2 The explicit design space

In their characterization of the design space, Woodbury and Burrow [2006] distinguish between the implicit design space and the explicit design space. The former includes the collection of all design states that can be generated using a specific generative system; for example, the set of geometric variations encoded in a parametric model or the language of a (shape) grammar. The latter includes the collection of design states which designers (either a single designer, a design team, or a group of unrelated designers) have generated over time. The explicit design space is a sub-

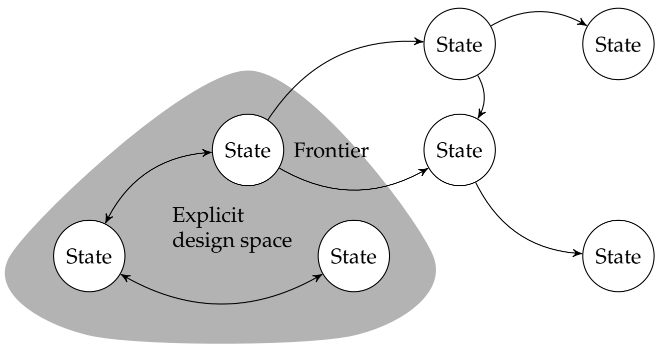


Figure 5.1: An example of an implicit design space and an explicit design space (gray). A design move in the implicit design space involves generation (→) while a design move in the explicit design space involves navigation (↔).

set of the implicit design space (Figure 5.1). The implicit design space is shown as a directed graph of connected design states — in close resemblance to AI state spaces. The explicit design space is indicated as a subset (gray), where the frontier contains all the design states that can be further explored. In other words, the frontier marks the boundary between the explicit and implicit design space. Depending on the position of the navigator (or designer), either at the frontier or elsewhere in the explicit design space, the act of taking a design move has different implications. In the former case, this involves the generation (→) of new design states relative to a particular design state at the frontier of the explicit design space, while in the latter, it is possible to navigate (↔) from a known design state to another along explicit paths that express some measure of relatedness between the two states.

The importance of the explicit design space as an expanding library of potentially recoverable designs or explored design paths is emphasized in the paper of Woodbury and Burrow [2006]. In particular, the importance of representing the explicit design space has multiple facets, namely (1) to provide alternatives, (2) to back up and recall prior work, and (3) to replay paths previously discovered in a design space [Woodbury and Burrow, 2006]. First, the availability of design alternatives is important for reasons of revelation and comparison. Indeed, design alternatives reveal multiple new paths in the design space to be explored, resulting in new, anticipated, or sometimes unexpected, designs [Shea and Cagan, 1999]. Also, design alternatives can be compared against each other along multiple criteria — some of which are quantifiable, while others are not. In this case, comparison should be considered as an act of 'satisficing' [Simon, 1956], rather than an act of optimizing. Satisficing, a combination of 'satisfy' and 'suffice', directs a comparison of the alternatives towards criteria for adequacy rather than fully rational solutions, due to the limited resources (time and information) available. The availability of design alternatives, and more specifically the recording of which design alternatives are explored and which alternatives are ignored, constitutes the history of how a resulting design is achieved. This history might go some way towards explaining the design.

Second, the representation of the explicit design space is important to enable backup and recall of prior work. The benefit of backup strategies in a design process is obvious, as they allow designers to recover earlier design states, thereby avoiding having to start from scratch in case of a mistake. In almost all CAD systems, short-term backup is supported along the current design path through a typical 'undo' command. Longer-term backup requires the use of automatic version control systems or manually performed versioning (by saving different file versions) at regular intervals. In some cases, it might be beneficial to enable designers to recover design states unrelated to the current exploration process. Woodbury and Burrow [2006] define the concept of 'recall' as the metaphor for such distant access. In particular, recall enables designers to recover designs that were generated at different times, in different projects or contexts, or by different designers. Examples of recall in CAD systems include catalogues of drawings, symbol collections, and libraries with specific functionalities.

Third, another amplification strategy is to enable the 'replay' of paths previously discovered in a design space. Replay involves using recalled design states or paths in a new context, and is available in many CAD systems through a typical 'copy and paste' command. Although to some extent, replay is a successful amplification strategy for independent objects, Woodbury and Burrow [2006] correctly point out that replay is also very fragile if object dependencies are involved. For example, many shape grammar rules have labels that make them difficult to apply outside the context of the original grammar in which they were defined. In the case of the RdB-tf grammar [Eloy, 2012], labels contain intentional information about the stage in derivation in which rules can be applied. If these labels are not available in the new grammar, the rules cannot be applied. The same holds true for parametric design patterns [Woodbury, 2010], which are (successful) parts of a parametric model that can be replayed if particular components are available in the new context.

5.2 Shape grammar implementation tools

In general, representation, codification, and implication are functionalities that are available in most (generative) CAD systems. Also, these functionalities are among the main amplification strategies underlying shape grammars and their implementations. Even though the functionalities of replay, recall, backup, and alternatives (all involving the representation of the explicit design space) may not be available at all (or only to a significantly reduced extent) in current CAD systems and shape grammar implementations, they might also prove to be valuable amplification strategies.

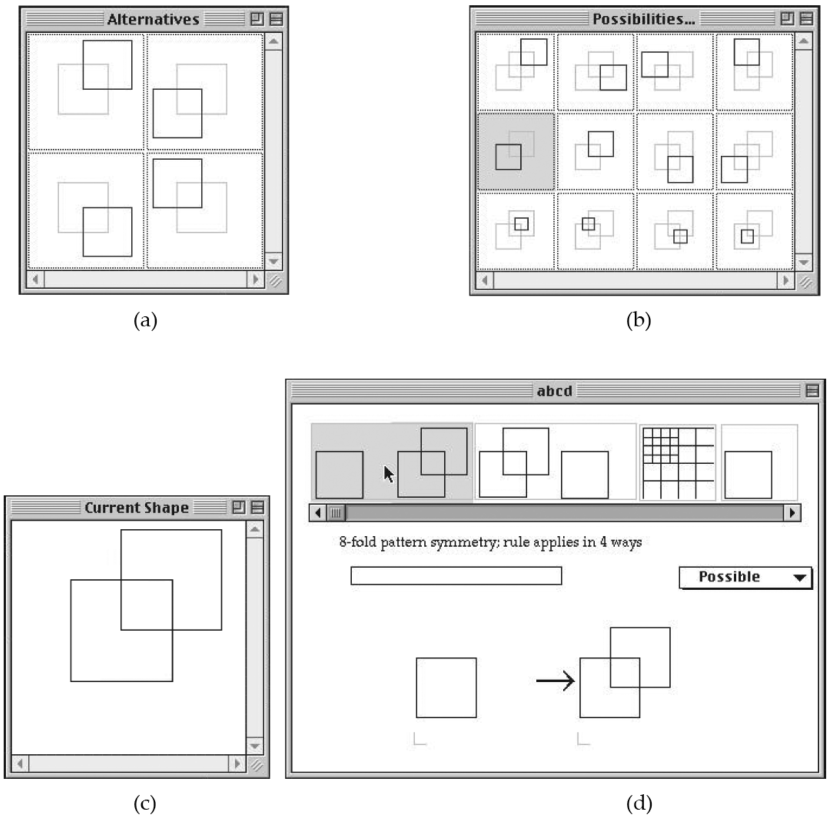
5.2.1 Overview

A literature review of existing tools for shape grammar implementation [Chau et al., 2004; Gips, 1999; McKay et al., 2012; Yue, 2009] reveals that most implementation tools focus on the representation of typical characteristics of shape grammars; such as subshape detection, parametric rules, curvilinear shapes, etc. Researchers have focused far less on representing the design space and amplifying exploration in the interface of the implementation tool. A comparative overview of design space exploration functionalities of four shape grammar implementation tools [Grasl, 2013; Hoisl and Shea, 2011; Tapia, 1999; Trescak et al., 2012] is summarized in Table 5.1. This overview lists the functionalities required to (1) represent a design space, (2) support the generation of new design states, and (3) support the exploration of already visited design states (the explicit design space). The shape grammar implementation tools considered — GEdit [Tapia, 1999], Spapper [Hoisl and Shea, 2011], SGI [Trescak et al., 2012], and GRAPE [Grasl, 2013] — have been selected, because they take a particular approach to supporting design space exploration. To compile this overview, either a working copy of the tool was obtained, or its functionality was determined from a tutorial or published paper.

GEdit, developed by Tapia [1999], is an early example of a shape grammar implementation tool in which design space exploration is considered, at least to some extent. The graphical user interface of GEdit, shown in Figure 5.2, contains multiple windows showing (a) where rules can be applied, (b) the possible rule applications, (c) the current design, and (d) a visualization of the rules in the grammar. In other words, the design space is represented through a visualization of the current design state and the possible alternatives that are the result of a single rule application. The design alternatives are structured in a two-dimensional array. The possible resulting shapes of a chosen number of rule applications are first generated and shown in this two-dimensional array, before they are definitely ap-

	GEdit [Tapia, 1999]	Spapper [Hoisl and Shea, 2011]	SGI [Trescak et al., 2012]	GRAPE [Grasl, 2013]
(1) Design space visualization:				
Current design state	Visual	Visual	Visual and symbolical	Visual
Rule application results	Array	Individual results	List	Individual results
Derivation history	No	No	Yes, current derivation only	No
(2) Generation of alternatives:				
Rule application	Semiauto-matic	Semiauto-matic or manual	Automatic	Semiauto-matic
Automatic detection of applicable rules	No	No	Yes	No
Manual manipulation of shapes	No	No	No	Yes
(3) Navigation and storage of designs:				
Backtracking	Yes	(?)	No	No
Save and reuse design states	No	Yes, current (.dxf, etc.)	Yes, current (.xml)	Yes, current (.dxf)
Save derivation history	No	No	No	Yes, current derivation (.dxf)
Reuse derivation history	No	No	No	No

Table 5.1: Comparison of design space exploration possibilities in several shape grammar implementations.

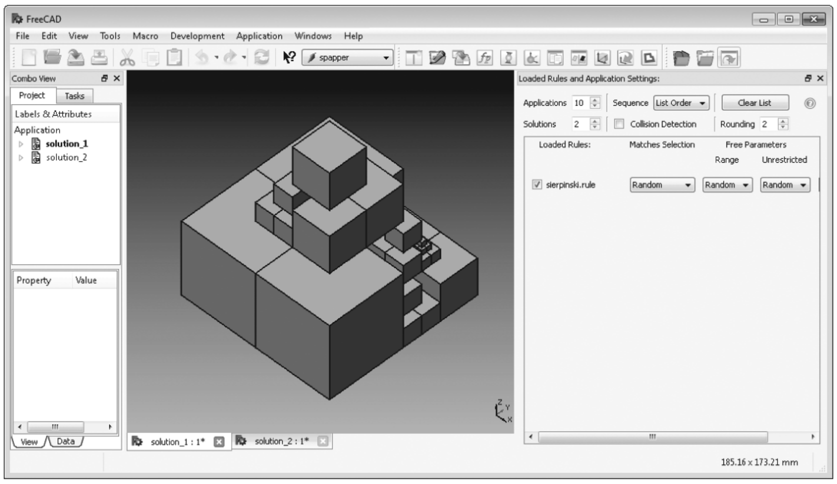


8-fold pattern symmetry; rule applies in 4 ways

Figure 5.2: The graphical user interface of GEdit — (a) subshape detection alternatives, (b) possible rule applications, (c) current design, and (d) rules of the grammar. GEdit is among the first implementations to consider some aspect of design space exploration, such as structuring design alternatives in a two-dimensional array. Reproduced from the original image appearing in Tapia [1999].

plied. New design alternatives are generated in a semi-automatic manner — all possible rule applications are calculated automatically, after which alternatives are selected manually. In general, GEdit is one of the first implementations that recognizes the importance of design space exploration — '*the designer explores the language of designs, generating designs, imposing additional constraints, halting the generation process, backtracking to a previous design, or saving the current state.*' [Tapia, 1999].

Spapper is a more recent shape grammar implementation tool, developed by Hoisl and Shea [2011], which provides a visual way of editing or developing shape grammars. Similarly to GEdit, the interface visualizes a single design that can be altered over time. Only the current design state in the design space is visualized. Designers can choose to explore the language of a grammar using manual rule application (through a predefined sequence or manual selection), semiautomatic rule application, or automatic (random) rule application. However, it is not possible to automatically detect which rules can be applied to the current design. The current design state can be stored using the standard functionality of the underlying CAD system (Figure 5.3). Only the resulting designs of rule applications are stored (in separate files), without the history of rule applications used. As a result, the stored designs are considered as fresh designs (without history) when they are used in a new exploration process.



Applications 10 | Sequence List Order
Solutions 2 | Collision Detection | Rounding 2

Loaded Rules:	Matches Selection	Free Parameters Range	Free Parameters Unrestricted
✓ sierpinski.rule	Random	Random	Random

Figure 5.3: *The graphical user interface of Spapper [Hoisl and Shea, 2011]. Spapper is integrated into a three-dimensional CAD environment, allowing the import and export of three-dimensional shapes to and from conventional CAD-formats.*

SGI, developed by Trescak et al. [2012], includes several features to enable interactive exploration of the language of a shape grammar. First, a render line view shows the current derivation line of the exploration process (Figure 5.4). This provides designers with the possibility of tracing the execution of the shape grammar from the initial design state to the current design state. Second, new designs can be generated using a particular search algorithm, such as depth-first or breadth-first search, or using a sub-shape detection algorithm. The sub-shape detection algorithm is an improved version of the one described in the work of Krishnamurti [1981]. The resulting shapes are stored in a list view in which the user is able to select and delete design alternatives manually. Third, the current design state is displayed in both a visual and a symbolic manner by also showing a list of all the design properties (for example a name, position, etc.). As a result, designs can be compared in terms of both geometric and non-geometric design properties.

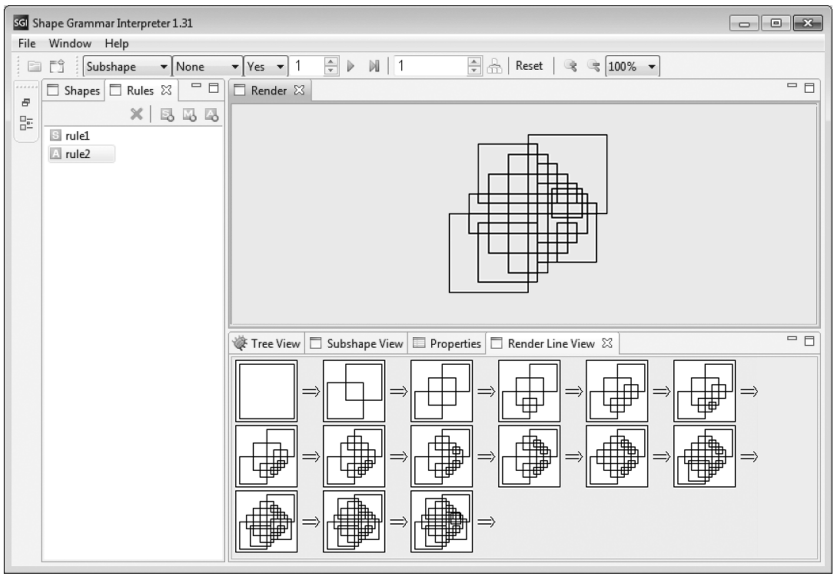
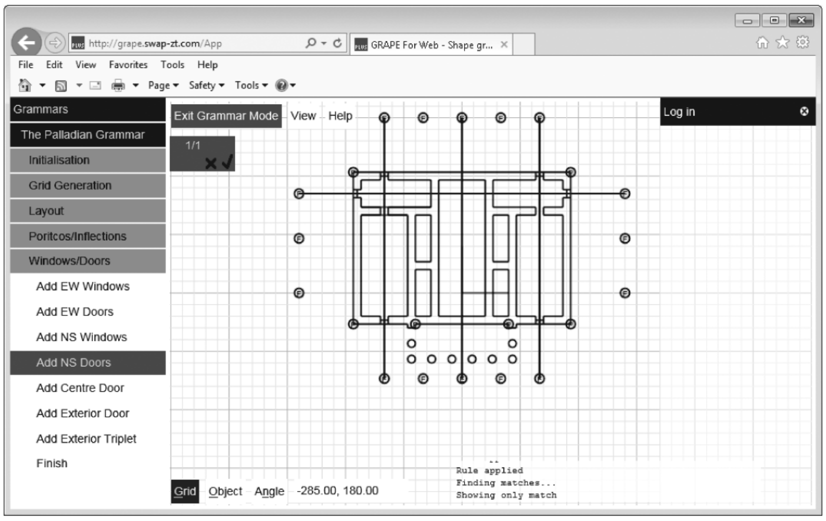


Figure 5.4: The graphical user interface of SGI [Trescak et al., 2012]. The render view (top) shows the current shape. The render line view (bottom) shows the current derivation line of the exploration process. In the example shown, new shapes are generated using the sub-shape detection algorithm.

GRAPE is a graph-based shape grammar library, developed by Grasl [2013]. Several interfaces to the GRAPE library have been developed, either based on commercial CAD packages or as a web application (Figure 5.5). The design space is represented through a single design state view that shows the current design state. Alternatives are generated in a semi-automatic manner, because designers explore and select designs one at a time. Based on the functionality of the underlying CAD package, it is possible to manually intervene in the exploration process by adding, deleting and modifying designs without the use of shape rules. This feature is reflected in the structure of the user interface, which distinguishes between a common CAD mode and a grammar mode. The CAD functionality makes it possible to store the current design state (without history). Also, it is possible to export a snapshot of the current derivation line (from the initial design to the current design). This derivation cannot be re-used in a new exploration process. Several approaches are discussed to automating rule selection in recent work of Grasl and Economou [2014]. These approaches are divided into two sub-approaches, namely, extensive enumeration and goal-directed generation, with a view to reducing the grammar's design space and filtering out only those alternatives that might be of interest.



```
Rule applied
Finding matches...
Showing only match
```

Figure 5.5: The graphical user interface of the GRAPE web application [Grasl, 2013]. The current design (right) can be transformed using shape grammar rules (left) or by switching from the grammar mode to a CAD mode in which designs can be modified without the use of rules.

5.2.2 Keeping track of the explicit design space?

The overview in Table 5.1 demonstrates that design space exploration in current shape grammar implementations is not supported or if it is, then only to a limited extent. The representation of the (explicit) design space is often restricted to a small subset of design states and paths, or even a single design state. Also, navigation in the design space is limited to new alternatives being generated, with limited scope for backtracking or recalling prior design states. According to Woodbury and Burrow [2006], design space exploration for shape grammar implementations is less supported because shape rules do not represent an explicit design space:

> *A flaw in standard rule-based accounts of design space exploration in implicit space is that the usual formulation of rules cast navigation solely in terms of derivation, thus putting the landscape of the explicit space forever beyond the sight of the navigator. Applying rules of the form $\alpha \rightarrow \beta$ involve removing a transformed version of α from the design and substituting for it an identically transformed version of β. This means that the granule of movement specified in a rule can go from one point in a design space to another point irrespective of any underlying design space order.* [Woodbury and Burrow, 2006].

In other words, while shape grammars do represent an implicit design space by structuring a set of rules, they do not provide an explicit representation of the paths already traversed in the design space. This is due to the fact that rules do not keep track of the design state that is to be replaced. For example, a particular rule application might delete everything in a design state, after which it is not possible to return to this design state. In a direct response to Woodbury and Burrow, Krishnamurti [2006] acknowledges this 'flaw', but also adds that it should not be difficult to envisage an implementation of grammars that maintains a history of derivations. Following this line of thought, two possible approaches can be devised. A first approach involves the use of reverse rules to go back in the design space. As Krishnamurti and Stouffs [1993] postulate — a shape rule $a \rightarrow b$ cannot simply be inverted to $b \rightarrow a$, because the resulting shape of the inverse rule may not equal the original shape (Figure 5.6). The reversibility of a rule depends on the shape c to be transformed — in particular, the additive part of the rule $(b - a)$ that is also part of the original shape $(b - a) \cdot c$ is not maintained when the rule is applied. As a result, a rule is only reversible if $(b - a) \cdot c = \varnothing$. If this is not the case, the shape $(b - a) \cdot c$ should be stored at each stage of rule application in order to support backtracking. This can be done easily by associating this shape with each design state generated.

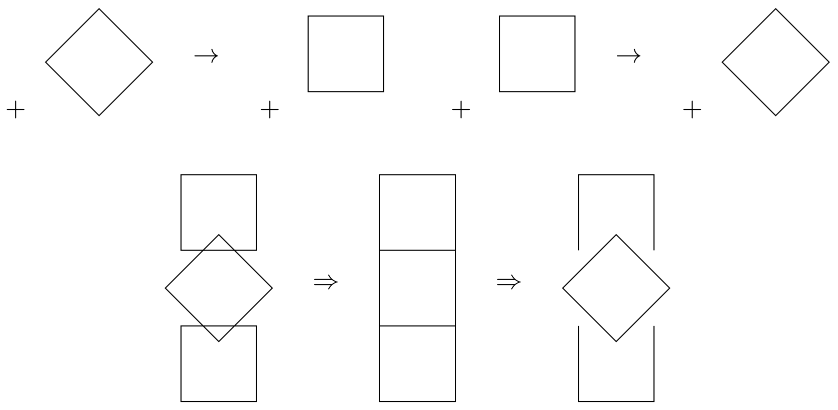


Figure 5.6: An example of an irreversible rule (left). After subsequently applying the rule and its inverse rule (right), the resulting shape is not the same as the original shape (bottom).

5.3 Tree structures

A second approach to keeping track of the explicit design space is to maintain a tree structure with pointers to the parent nodes. In the theory of shape grammars, rules are applied using the part relation, which means that shapes or designs can be decomposed in infinitely many ways. In other words, designs in the language of a grammar are structured according to a partial order, which means that not every pair of designs is related. In set theory, a *tree* is a specific kind of a partially ordered set that has one root element (an initial design). The design space represented by a shape grammar can thus be described as a tree. In computer science, a tree is also a widely used data structure to represent hierarchical structures. In particular, a tree structure contains nodes, along with parent–child edges that represent the hierarchical relations between nodes (Figure 5.7). In other words, tree structures maintain a pointer to the parent of every node in the tree. Each node in the tree structure has exactly one parent node — except for the root node, which has none. As a result, a tree structure is a directed acyclic graph in which any pair of nodes is connected by exactly one path. In the context of representing a design space, a tree structure could be used to represent the different states of a design, starting from an initial design state. In this case, the parent–child edges represent specific rule applications between parent and child design states. As both the design states and the relations are stored in the tree structure, it is possible to keep track of the explicit design space.

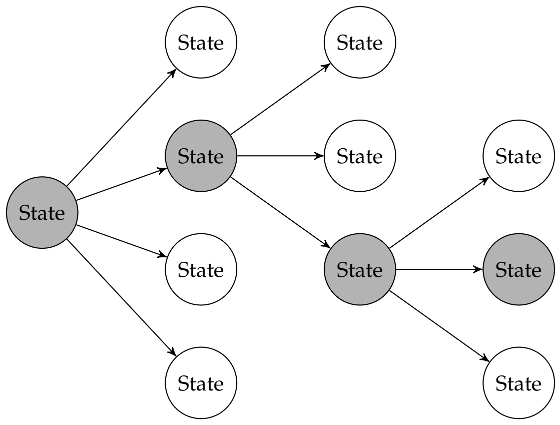


Figure 5.7: Representation of the design space as a tree. The nodes represent different states of a design. The edges between the nodes represent rule applications. The current design path is indicated in gray.

While a tree structure neatly corresponds to the partial order of designs generated by a shape grammar, other structures can be used as well. In some cases, it is possible to generate the same design by applying different rule sequences; in other words, different paths in the design space might lead to the same design state. This is, for example, the case for shape grammar with commutative rules, which means that their order can be changed without affecting the result. If this is the case, a design state has pointers to a non-empty set of parents instead of a single parent. The resulting design space structure is then no longer a tree, but a network. A structure of this kind has the advantage of being more compact (visually equivalent designs are stored only once in the network), but is less effective in maintaining the derivation history of design states due to the multi-valued nature of the parent relation.

The advantage of representing the design space as a tree structure is that several efficient off-the-shelf algorithms [Skiena, 2009] are available for common operations (such as enumerating, searching, adding, and deleting items) and for 'walking' the tree. If the design state associated with a node in the tree structure is yet unexplored, it is possible to generate new child nodes from this unexplored node (Figure 5.8 left). In other words, these child nodes correspond to the possible design alternatives that can be generated from the parent node. If, however, the design state associated with a node in the tree structure is already explored, it is possible to walk from this node to another by means of the connections between parent and child nodes — this is commonly called walking the tree (Figure 5.8 right). The

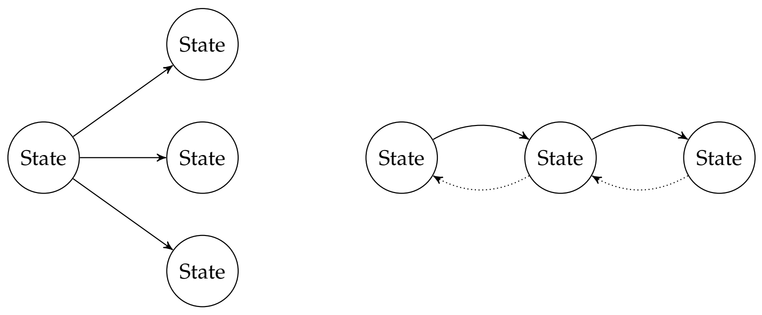


Figure 5.8: Two operations on tree structures; (a) generation of new child nodes and (b) traversing nodes. These operations correspond to generation and navigation in a design space, respectively.

act of walking the tree structure corresponds to navigating in the explicit design space. By enabling both generation and navigation, trees provide a practical and elegant solution to structuring the implicit and explicit design space.

5.3.1 Generation of alternatives

A design in the language of a grammar can be generated in several ways — ranging from a manual approach to an automatic approach. For each generation step, there are several actions that occur [Chase, 2002]:

- Determination of a rule to be applied;
- Determination of a sub-shape to which the rule can be applied; and
- Determination of matching conditions.

A first possible approach is to perform these actions in an entirely manual way. If done manually, new design alternatives can be explored one by one by selecting a rule, detecting a (sub)-shape to which this rule can be applied, calculating the necessary transformations, and finally applying the rule. Of course, performing these actions manually is a labor-intensive task that can easily be automated. In the automatic case, the computer system is used to detect which rules can be utilized and where they can be applied (sub-shape detection). For a graph-theoretic representation of shape grammars, (parametric) sub-shape detection can be detected automatically using subgraph isomorphism detection (see Chapter 4). As a result of this automatic approach, a set of all possible design alternatives is generated, relative to a given design state. The availability of design alternatives is beneficial for reasons of revelation (they reveal new areas in the design space) and also for comparison based on multiple criteria. For grammars with many rule application possibilities, however, such an approach quickly triggers exponential explosion. For this reason, heuristic selection (given some predefined selection criteria) might be necessary to constrain the set of design alternatives.

The next step involves the manual selection (in the set of alternatives) of the design alternative to be further explored. This design can then again be reconsidered for the generation of new alternatives. As a result, a tree structure with multiple levels and branches is constructed during the exploration process. The levels in the tree structure correspond to specific moments in time, while the branches correspond to the designs generated at these specific moments in time. As a result, this semi-automated approach for generating alternatives involves both computer generation and human intervention, which is in line with Tapia's characterization of shape grammar implementations — "*the computer handles the book keeping tasks (the representation and computation of shapes, rules and grammars and the presentation of correct design alternatives) while the designer specifies, explores, develops design languages, and selects alternatives.* [Tapia, 1999].

5.3.2 Design space navigation

In order to walk a tree structure, nodes must be visited by means of the parent-child connections. This can be done in several ways — either level by level, where the root node is visited first, followed by the child nodes, grandchild nodes, and so forth, or in the opposite direction, starting from child nodes and visiting their respective parent nodes (predecessors). These operations correspond to navigating back and forward in the explicit design space. Back and forward navigation is only possible for design states that have already been discovered and explored. If this is not the case, for example in the case of nodes on the frontier of the explicit design space, forward navigation actually involves the creation of new child nodes. The key idea here is to mark the places and paths already traveled, which can be done easily using boolean flags. Other navigation possibilities include 'pruning' a specific branch or even a whole section of the tree structure; for example, when this particular area of the design space is considered to be no longer relevant. Figure 5.9 shows an example of back navigation, where the top design states are generated first, after which new design alternatives are generated starting from a previously generated design state.

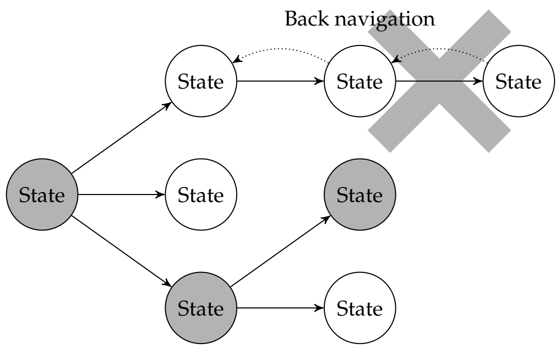


Figure 5.9: An example of back navigation in the explicit design space, after which new alternatives are generated starting from a previously generated design state.

If design states are explicitly stored (in a tree structure), it is possible to recall these design states in two distinct ways. The first approach involves navigation by means of connections between nodes in the tree structure. In this case, recall is achievable by navigating from one design state to another along explicit paths that relate the design states to each other. The second approach involves searching stored design states based on design state properties; for example, a name, a timestamp, a rule, or other kinds of descriptive information associated with each design state. This kind of recall can be done without reference to a given design state. In this case, some kind of persistence system (such as a database) is needed to efficiently store and organize design states, after which they can be retrieved based on particular search queries. Design states that belong in one grammar or design context can then be recalled in contexts other than the one in which they were created.

A similar case can be made for the replay of design paths. As Stiny [1994, 2006] has shown, it is always possible to find a retroactive explanation for any sequence of rule applications in such a way that continuity is maintained. As a result, the replay of shape grammar derivations does not only involve a sequence of changes in design states, it might also lead to restructured descriptions because derivations — just like shapes — can be interpreted in multiple ways due to the part and embedding relations [Krishnamurti, 2006]. The use of designs, derivations, and rules outside their original context might provide an important amplification strategy [Woodbury and Burrow, 2006]. The importance of recall and replay of existing work in a new design can also be found in the large body of (mainly theoretical) work on case-based reasoning systems [Aamodt and Plaza, 1994].

Those systems scan a collection for relevant existing cases by identifying commonalities, after which the retrieved case is generalized and mapped to the new design case. From a technical point of view, recall also has a positive impact on the computational complexity of generating alternatives, because already-visited design states can be recalled, instead of having to be regenerated.

To conclude, tree structures can be used to keep track of the explicit design space, to enable generation of new designs, and to enable navigation in the explicit design space. Unlike previous research efforts in which trees are used [Trescak et al., 2012], they are not used as devices for searching design solutions, but they are used to implement several aspects of design space exploration — which is a far richer concept than merely searching. The tree structure forms one of the keystones of a new kind of grammar-based tool for design space exploration, which is described in the next chapter (Chapter 6).

6 Digital Sketchbook

In this chapter, we look at the prototype software tool for design space exploration that has been developed. The main functionality and user interface of this prototype is described in Section 6.1. Also, we provide three more examples of grammar-based explorations in Section 6.2, thereby demonstrating how the prototype can be used in practice. The first example involves the ice-ray grammar, originally developed by Stiny; the second example demonstrates a graph grammar for exploring spatial configurations; and the third example describes the implementation and visual exploration of the Frank Lloyd Wright grammar.

6.1 A digital sketchbook for design space exploration

In the previous chapters of this thesis, several concepts have been introduced and discussed that might facilitate design space exploration. First, the theory of shape grammars (Chapter 3) provides a concise and computable framework to represent a design space by encoding design moves in the form of shape rules. Second, the graph-theoretic representation of shapes and grammars, as described in Chapter 4, enables the representation of a design space that can be effectively explored on a computer system, using rules as the main device for doing so. Third, the tree structures (Chapter 5) keep track of the explicit design space, making it possible to interact with the design space in a number of ways, including generation and navigation. The theory of shape grammars, the graph-theoretic representation of shapes and designs, and the tree structures together form the keystones of a new kind of grammar-based tool for design space exploration — the *digital sketchbook*. The focus of such a design tool is on supporting designers to explore the language of a grammar in a visual and interactive way. In fact, the envisioned sketchbook for design space exploration provides the following functionality:

- Design space visualization;
    - Representation of the design space as a visual whole, allowing designers to compare alternatives simultaneously.
    - Visualization of the explicit design space and the current design path.
    - Representation of design states in both a visual and descriptive manner.
- Backup of design states and recall of previous design states and design paths;
- Changing the structure of the design space;
    - On-the-fly generation of new rules.
    - Manual intervention without the use of grammar rules.

6.1.1 Design space visualization

An important difference between present and previous shape grammar implementation tools is the visualization of the (explicit) design space as a whole, rather than merely displaying a single design state that can be altered over time. The design states are organized in a two-dimensional grid of $m \times n$ cells. Figure 6.1 shows a schematic visualization of the graphical user interface of the proposed shape grammar implementation tool. Each cell in the grid layout contains a visual representation of the design, and has zoom and pan functionalities — allowing designers to consider rule applications that are more subtle, and therefore harder to identify. Each column stores the n design alternatives at a specific level in the design space. Once a specific design state is selected for further exploration by the designer, a new set is generated in a new column $m + 1$, which is added to the grid. The current design path is indicated in gray color to show the position of the designer in the design space. Also, it is possible to navigate back in the explicit design space by selecting a design state at a previous level or column $x < m$. If this state has not been expanded yet, a new set of alternatives is generated in a new column $x + 1$, which is subsequently added to the grid. If this is the case, the descendants of the selected shape or design state are no longer visible, and they are replaced by a new set of alternatives. As a result, designers are able to return to a prior design state in the design process, thereby making new areas available for future exploration. It is also useful to note how emergent shapes occur in the example shown in Figure 6.1.

The layout of the user interface is an intuitive interpretation of the explicit design space as a tree. The tabular layout enables designers to scroll both in a horizontal and a vertical direction, allowing them to examine multiple alternatives and shape derivations in a limited amount of screen space. In the proposed approach, only the current design path and the sibling design states are shown in the interface. While the visualization of the complete explicit design space can be useful in some cases, it quickly becomes overly complex for larger design spaces. Nevertheless, it is possible to explore the complete explicit design space by selecting different designs in the tree. In addition to the proposed two-dimensional grid layout, other layouts can also be useful in specific cases; for example, a radial layout can be useful to show visually adjacent design states or shapes. The different components of the layout can be changed, or resized, according to user preferences. Currently, new design states are generated one step, or rule

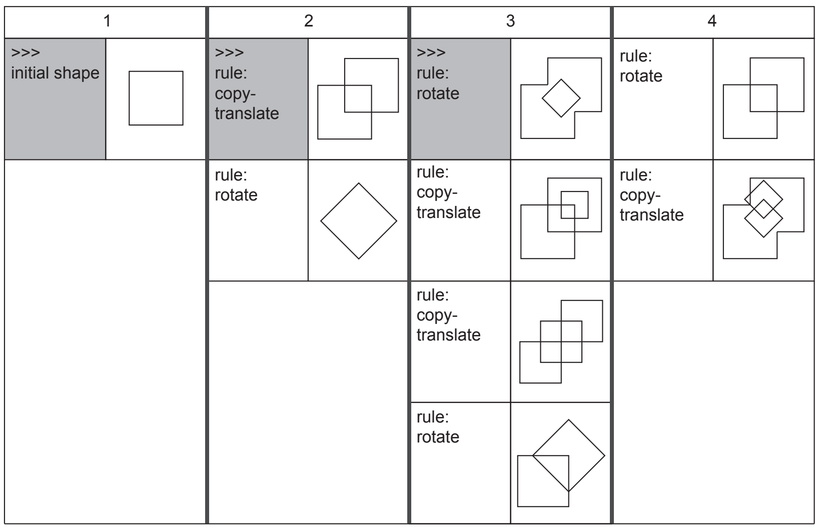


Figure 6.1: Schematic visualization of the graphical user interface of the proposed shape grammar implementation tool. The design states are organized in a two-dimensional grid, in which each column contains the design alternatives at a specific level in the design space. The current design path is indicated in gray color.

application, from the selected shape in the tree. This could be extended using search algorithms for the creation of new designs at a greater depth; for example, breadth-first, depth-first, or even heuristic search algorithms (if some heuristic function is available to guide the search process).

As shown in Chapter 4, graphs provide a natural and efficient way of representing shapes, labels, and other kinds of descriptive information. However, this graph-theoretic representation should only be used for representation and calculation purposes, because designers typically prefer visual over mathematical representations (see the work of Bleil de Souza [2012]). Indeed, the graphs quickly become complex in size and number of node types, and therefore they become difficult to grasp intuitively. For this reason, the designs are shown in a visual manner, together with a textual description, while the graph-theoretic representation is maintained for representation and calculation only. The visual representation facilitates the comparison of design alternatives, while the textual information can be used to compare alternatives towards non-visual design properties, as well as to facilitate design state recall based on searching design properties. This textual information might include the rule that was used to create this design state, the grammar that was applied, the project name, and also other kinds of information that might be relevant. A design process is often associated with a specific program, design intent, or design reasoning, which is expressed verbally and can then be associated with a specific design state in a descriptive manner.

6.1.2 Backup and recall

Another important difference to previous shape grammar implementation tools is the ability to recall design states and derivations stored in a previous project. This is the result of implementing an external persistence system that allows for the efficient storage of design states, after which they can be retrieved based on particular search queries. These search queries may direct to metadata associated with the stored design states, such as a timestamp, the rule used to create the design state, or the project and grammar in which the design state was generated. These search queries, however, might also involve searching for particular properties that are associated with design states. These properties describe some design intent or reasoning (often expressed verbally) that is related to the design state at hand. Figure 6.2 outlines the entities that are stored in a database system (design states, grammars, projects, and properties) and their relations. With these entities and relations stored, it is possible to fetch design states from the database together with their corresponding grammar, or alternatively, an existing design state or grammar can be recalled from the database in a new design project. Also, since the parent–child relations between design states are maintained, it is possible to fetch successors of a design state from the database without having to regenerate them. As a result, it is possible to recall entire derivations, which has a positive impact on the computational complexity of generating alternatives because design states are pulled directly from the database.

With such a persistence system in place, it is possible to recall design states, as long as they were created using the proposed design tool. An additional external recall functionality can be used to import designs that were generated outside the proposed design environment; for example, designs or drawings generated in other CAD environments, or even manually drawn sketches. In the former case, the geometric elements in the original CAD drawing are translated into the corresponding graph objects in an attributed part-relation graph. In the present shape grammar implementation tool, it is possible to import two-dimensional CAD drawings using the Drawing interchange format ($.dxf$) and more complex three-dimensional building information models (BIM) using the Industry Foundation Classes (IFC) data model. For example, the initial floor plans that are the starting point of the RdB-tf grammar [Eloy, 2012] have been imported in this manner. In the latter case, either the drawings have to be converted manually into the appropriate graph-theoretic representation, or this can be done automatically using a tool like the one described in the work of de las Heras et al. [2014]. This tool recognizes floor plans using pattern recognition methods that are inspired by the way designers draw and interpret floor plans, after which they are converted into graphs, so they can be imported in the system.

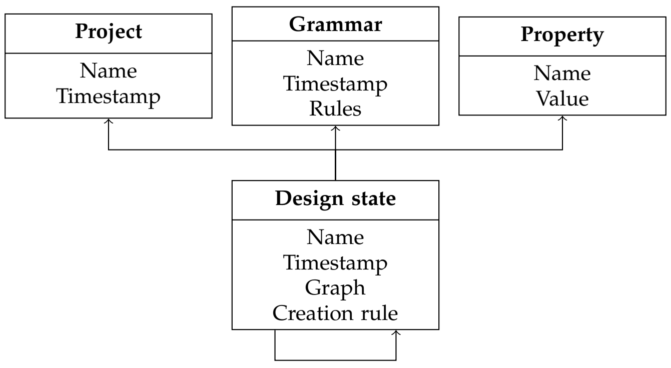


Figure 6.2: Outline of the entities that are stored in the database system (design states, grammars, projects, and properties) and their relations.

6.1.3 Changing the structure of the design space

As described earlier in this thesis, the ambiguous and emergent nature of shapes enables the designer to obtain new understandings or alternative perspectives on the design space. In this way, new opportunities for exploration occur by seeing designs from a different angle, possibly leading to a restructuring of the design space. A more explicit way to design space reformulation involves providing designers with the ability to make changes (such as adding new rules) to the grammar and the design space represented, and to enable them to do this in an interactive and intuitive manner. This might trigger a more agile exploration of the design space, in the sense that it does not remain static during the actual exploration. In most previous grammar implementation tools, the definition and exploration of the shape grammar are two different steps in the process; for example, Chase [2002] describes a model of interaction between designers and a computer system in which the development of the grammar completely precedes the derivation of this grammar. Nevertheless, it may be beneficial to allow designers to create, adapt or delete rules in the grammar during the derivation process, which is something McKay et al. [2012] refer to as exploring the design space through generate–test cycles. In the context of design space exploration, 'on-the-fly' generation or modification of rules results in an altered design space, possibly leading to design alternatives that were unreachable until then. Some visual examples of this are described in the following section.

In the present approach, new rules can be specified at any time during the exploration process, whenever this may be needed. Unlike previous approaches, in which new rules are to be defined in a separate editor environment, it is possible to manually adapt the current design using common CAD functionality — thus without the use of rules. In particular, the designer is able to switch from a 'grammar' mode to a 'manual' drafting mode, in which the current design can be adapted manually. This means that either new shapes can be added or erased, or their properties can be modified. After the manual intervention, it is possible to store this action in the form of a new shape rule that is subsequently added to the grammar. As a result, new rules can be created during, instead of before, the exploration process. The step of capturing a manual action in the form of a new rule cannot be performed in an entirely automatic manner, because it is not possible to determine the pattern shape of the rule without additional information being provided from the user. If the current design is manually modified, only the transformation of a possible pattern shape to a replacement shape is explicitly defined, but the pattern shape itself cannot be determined. Once this pattern shape has been specified by the user (by selecting the parts in the current shape), a new rule is fully specified and may eventually be added to the grammar for later recall in (new) design situations.

Nevertheless, the on-the-fly generation or modification of new rules remains a difficult and delicate issue. Chakrabarti et al. [2011] even consider the inability of current shape grammar implementations to support iterative development of the grammar as a main roadblock to achieve wider impact — especially in conceptual design. Indeed, grammar rules must be 'knowledge engineered', which can be a laborious and error-prone task because the designer needs to determine the important features and how to formulate the rules. An intuitive rule editor interface, such as the one proposed here, provides a first step to enable designers in creating and modifying rules during the exploration process. Several other approaches exist to ease the process for the designer in developing grammar rules; for example, by providing systematic analysis during rule development [Königseder and Shea, 2014] or by learning grammar rules in an automatic manner using machine learning techniques [Ruiz-Montiel et al., 2013; Talton et al., 2012]. Machine learning is a subfield of AI that deals with algorithms that learn from examples, instead of following static predefined commands. The machine learning method described in Appendix B addresses the problem of learning to classify architectural designs belonging to a particular corpus, based on a set of examples. This method can be used (1) to select a consistent corpus from which it is easier to extract rules (grammar formulation), and (2) to assess the output of a grammar and determine, in a quantitative way, whether the rules extracted are working (grammar evaluation).

6.1.4 Software prototype

The software prototype proposed is based on the underlying JAVA development environment for graph rewriting, called AGG[1]. The existing algorithms for automatic rule matching and rule application of AGG have been used. Figure 6.3 shows the interface of AGG, which contains a list of the grammar rules (left), a visualization of the current graph (center), and the node and edge type sets (right). As these graphs and graph rules are used only for representation and calculation purposes, this window is by default hidden for the end-user. Instead, a new interface has been developed on top of the underlying graph rewriting framework to enable users to develop and explore grammars. This interface consists of two main panels — the first panel includes the external recall and database functionality, and

[1]`http://user.cs.tu-berlin.de/~gragra/agg/`

the second panel includes the functionality of generation and navigation in the explicit design space. Figure 6.4 shows the database system, in which existing designs can be searched based on metadata that is associated with the designs (left). Both this metadata and a visual representation of this design are shown (right). Figure 6.5 shows the explicit design space (right), the current grammar (upper left) and the current design state (bottom left). The explicit design space is visualized as a whole, following the schematic visualization shown in Figure 6.1.

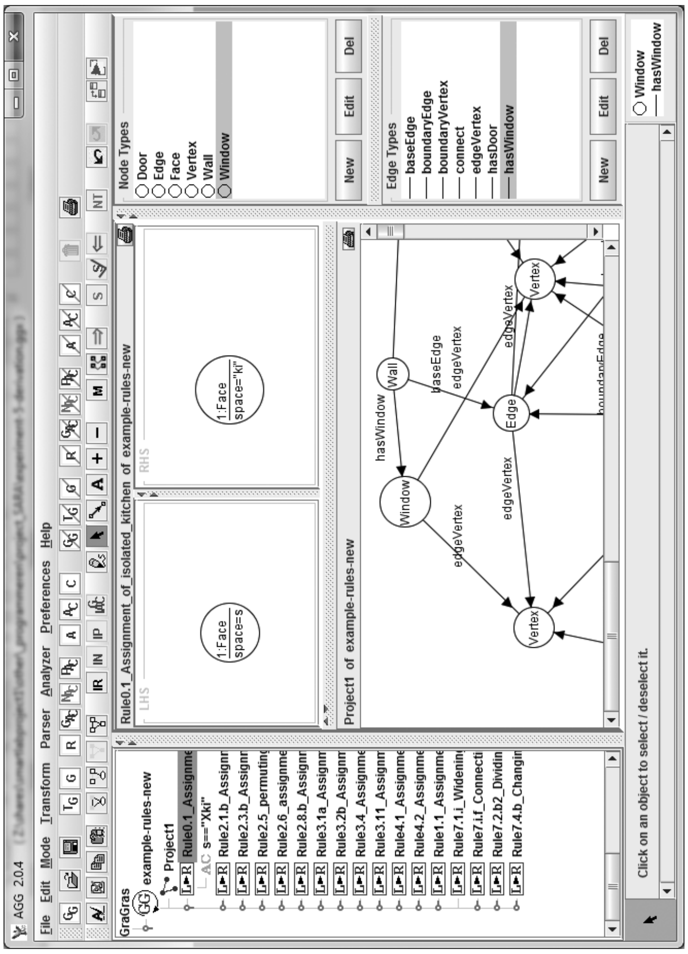
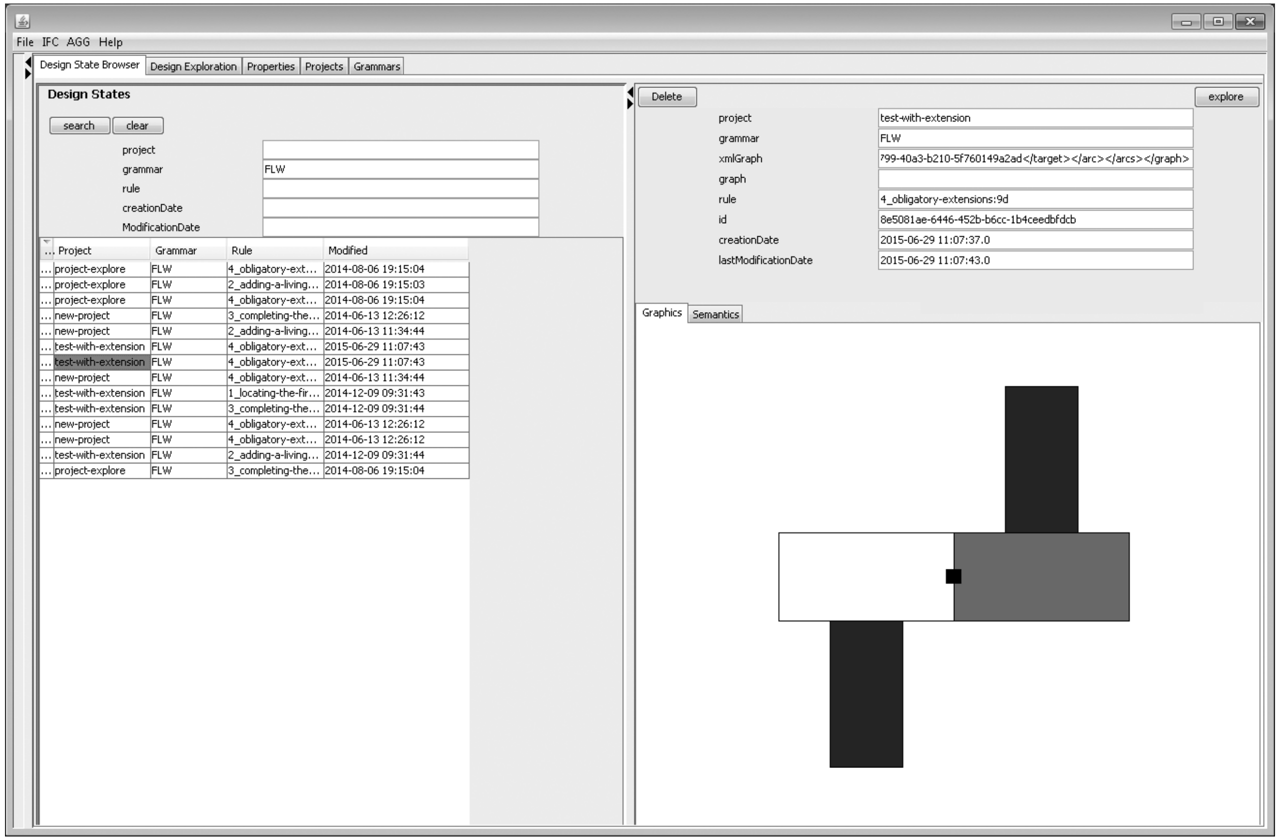


Figure 6.3: *User interface of AGG, which contains a list of the grammar rules (left), a visualization of the current graph (center), and the sets of node and edge types (right).*



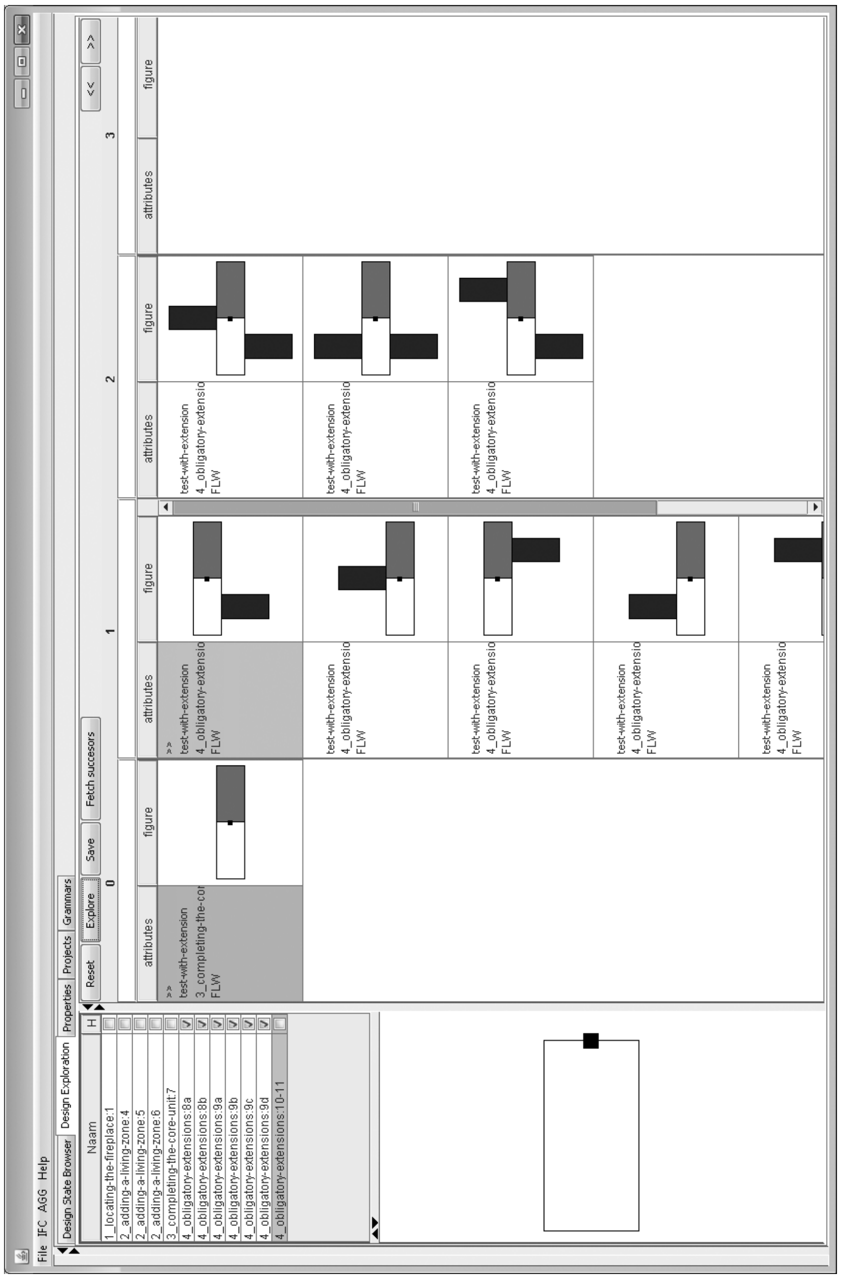
Design States

project	
grammar	FLW
rule	
creationDate	
ModificationDate	

… Project	Grammar	Rule	Modified
… project-explore	FLW	4_obligatory-ext…	2014-08-06 19:15:04
… project-explore	FLW	2_adding-a-living…	2014-08-06 19:15:03
… project-explore	FLW	4_obligatory-ext…	2014-08-06 19:15:04
… new-project	FLW	3_completing-the…	2014-06-13 12:26:12
… new-project	FLW	2_adding-a-living…	2014-06-13 11:34:44
… test-with-extension	FLW	4_obligatory-ext…	2015-06-29 11:07:43
… test-with-extension	FLW	4_obligatory-ext…	2015-06-29 11:07:43
… new-project	FLW	4_obligatory-ext…	2014-06-13 11:34:44
… test-with-extension	FLW	1_locating-the-fir…	2014-12-09 09:31:43
… test-with-extension	FLW	3_completing-the…	2014-12-09 09:31:44
… new-project	FLW	4_obligatory-ext…	2014-06-13 12:26:12
… new-project	FLW	4_obligatory-ext…	2014-06-13 12:26:12
… test-with-extension	FLW	2_adding-a-living…	2014-12-09 09:31:44
… project-explore	FLW	3_completing-the…	2014-08-06 19:15:04

project	test-with-extension
grammar	FLW
xmlGraph	799-40a3-b210-5f760149a2ad</target></arc></arcs></graph>
graph	
rule	4_obligatory-extensions:9d
id	8e5081ae-6446-452b-b6cc-1b4ceedbfdcb
creationDate	2015-06-29 11:07:37.0
lastModificationDate	2015-06-29 11:07:43.0

Figure 6.4: Database system of the software prototype. Existing designs can be searched based on metadata (left), and both this metadata and a visual representation are shown (right).



Naam	H
1_locating-the-fireplace:1	☐
2_adding-a-living-zone:4	☐
2_adding-a-living-zone:5	☐
2_adding-a-living-zone:6	☐
3_completing-the-core-unit:7	☐
4_obligatory-extensions:8a	✓
4_obligatory-extensions:8b	✓
4_obligatory-extensions:9a	✓
4_obligatory-extensions:9b	✓
4_obligatory-extensions:9c	✓
4_obligatory-extensions:9d	✓
4_obligatory-extensions:10-11	☐

>>
test-with-extension
3_completing-the-cor
FLW

>>
test-with-extension
4_obligatory-extensio
FLW

test-with-extension
4_obligatory-extensio
FLW

test-with-extension
4_obligatory-extensio
FLW

test-with-extension
4_obligatory-extensio
FLW

test-with-extension
4_obligatory-extensio
FLW

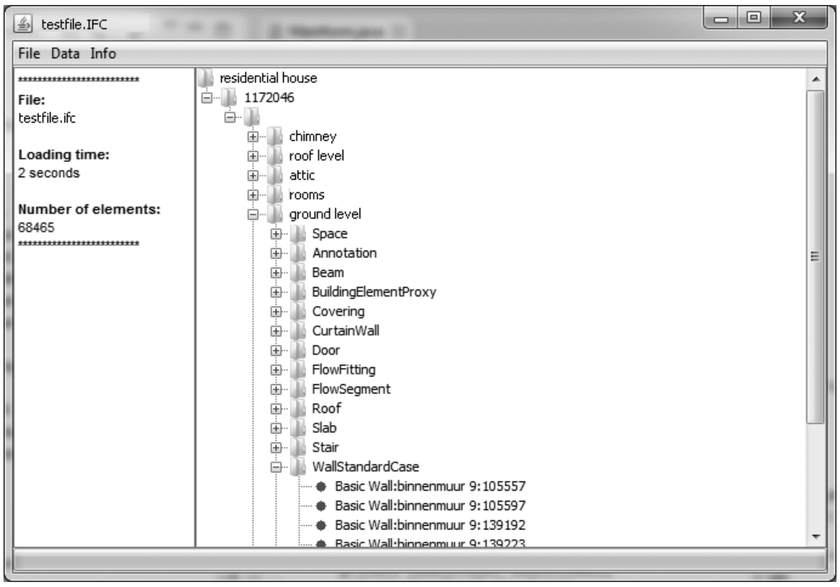
test-with-extension
4_obligatory-extensio
FLW

test-with-extension
4_obligatory-extensio
FLW

test-with-extension
4_obligatory-extensio
FLW

Figure 6.5: Visualization of the design space as a two-dimensional grid (right), the current grammar (upper left) and the current design state (bottom left).

In order to enable the designer to create design models in their familiar CAD environment, it is possible to import these models into the database system, by converting them into attributed part-relation graphs, after which they can be further explored in the design space. Currently, two-dimensional CAD drawings can be imported using the Drawing interchange format ($.dxf$), and more complex three-dimensional building information models (BIM) can be imported using the Industry Foundation Classes (IFC) data model. For the IFC data model, geometric and other entities are first added as the nodes of the part-relation graph, after which the connections and attributes are determined. In fact, the entities that are imported include $IfcCartesionPoint$, $IfcPolyline$, $IfcSpace$, $IfcWallStandardCase$, $IfcDoor$, and $IfcWindow$. Figure 6.6 shows the 'IFC-import' window, in which an IFC model can subsequently be opened, viewed, and added to the database system. Figure 6.7 shows an example of an IFC model that has been added to the database system. Since this design state is imported as a fresh starting point, the rule field remains blank until it is used in a future exploration process.



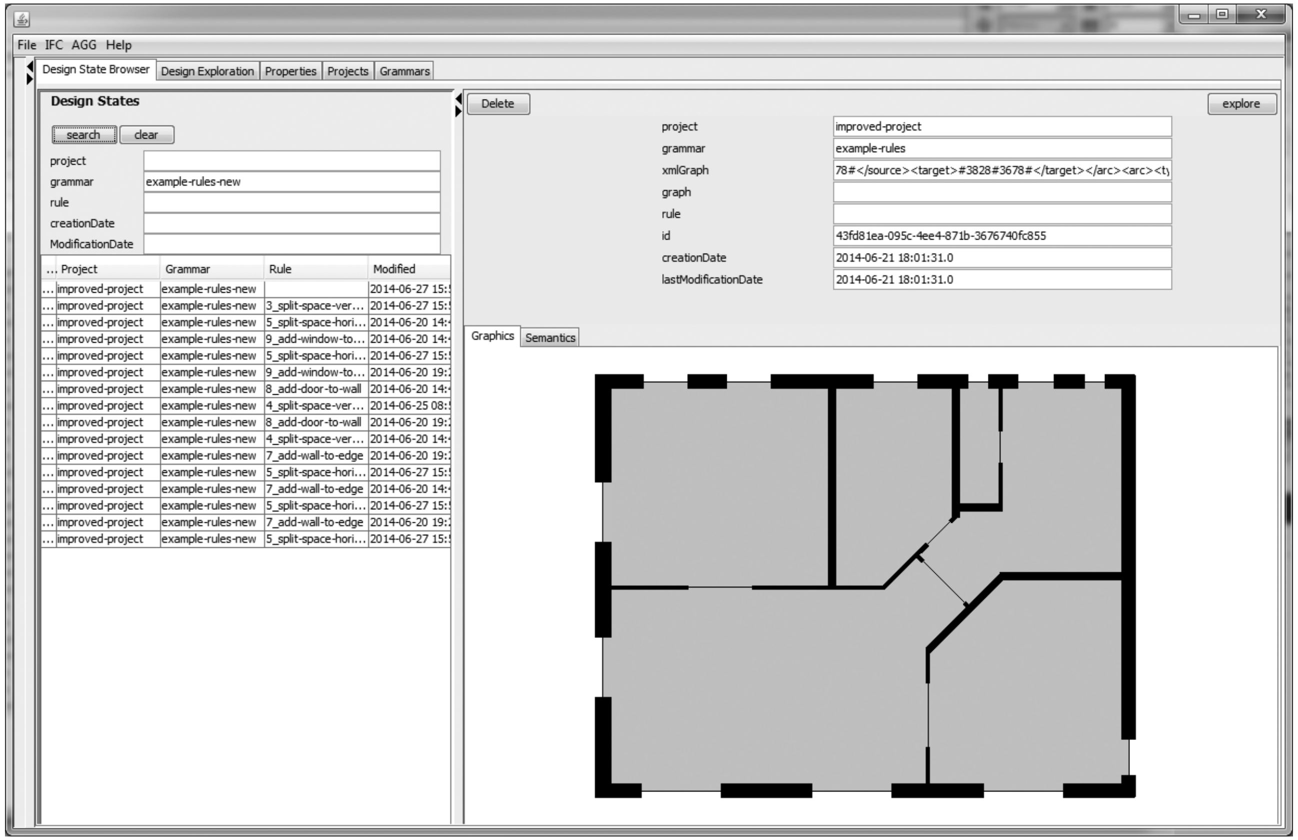
```
************************
File:
testfile.ifc

Loading time:
2 seconds

Number of elements:
68465
************************
```

```
residential house
⊟ 1172046
   ⊟
      ⊞ chimney
      ⊞ roof level
      ⊞ attic
      ⊞ rooms
      ⊟ ground level
         ⊞ Space
         ⊞ Annotation
         ⊞ Beam
         ⊞ BuildingElementProxy
         ⊞ Covering
         ⊞ CurtainWall
         ⊞ Door
         ⊞ FlowFitting
         ⊞ FlowSegment
         ⊞ Roof
         ⊞ Slab
         ⊞ Stair
         ⊟ WallStandardCase
            ● Basic Wall:binnenmuur 9:105557
            ● Basic Wall:binnenmuur 9:105597
            ● Basic Wall:binnenmuur 9:139192
            ● Basic Wall:binnenmuur 9:139223
```

Figure 6.6: An IFC model can subsequently be opened, viewed, and added to the database system.



project	
grammar	example-rules-new
rule	
creationDate	
ModificationDate	

...	Project	Grammar	Rule	Modified
...	improved-project	example-rules-new		2014-06-27 15:
...	improved-project	example-rules-new	3_split-space-ver...	2014-06-27 15:
...	improved-project	example-rules-new	5_split-space-hori...	2014-06-20 14:
...	improved-project	example-rules-new	9_add-window-to...	2014-06-20 14:
...	improved-project	example-rules-new	5_split-space-hori...	2014-06-27 15:
...	improved-project	example-rules-new	9_add-window-to...	2014-06-20 19:
...	improved-project	example-rules-new	8_add-door-to-wall	2014-06-20 14:
...	improved-project	example-rules-new	4_split-space-ver...	2014-06-25 08:
...	improved-project	example-rules-new	8_add-door-to-wall	2014-06-20 19:
...	improved-project	example-rules-new	4_split-space-ver...	2014-06-20 14:
...	improved-project	example-rules-new	7_add-wall-to-edge	2014-06-20 19:
...	improved-project	example-rules-new	5_split-space-hori...	2014-06-27 15:
...	improved-project	example-rules-new	7_add-wall-to-edge	2014-06-20 14:
...	improved-project	example-rules-new	5_split-space-hori...	2014-06-27 15:
...	improved-project	example-rules-new	7_add-wall-to-edge	2014-06-20 19:
...	improved-project	example-rules-new	5_split-space-hori...	2014-06-27 15:

project	improved-project
grammar	example-rules
xmlGraph	78#</source><target>#3828#3678#</target></arc><arc><t
graph	
rule	
id	43fd81ea-095c-4ee4-871b-3676740fc855
creationDate	2014-06-21 18:01:31.0
lastModificationDate	2014-06-21 18:01:31.0

Figure 6.7: An example of an IFC model that has been imported and added to the database system.

6.2 Some visual examples of exploration

In order to demonstrate how the proposed shape grammar implementation tool can be used in (architectural) design practice, several visual examples of design space exploration have been performed using the prototype. Many of the shape grammars found in the literature have been developed on paper, solely, and relatively few of them have been implemented on a computer system. The examples include both an original grammar (for the generation of spatial configurations) and two analytical shape grammars (for the generation of Chinese lattices and traditional prairie houses).

6.2.1 Example 1: Chinese lattices

The shape grammar for Chinese lattice designs, originally developed by Stiny [1977], is one of the first (analytic) applications of shape grammars. Chinese lattices are traditional, mostly ornamental, window and grille designs that were constructed between 1000 BC and 1900 AD. An overview of these lattice designs is given in the catalogue *A Grammar of Chinese Lattice* by Dye [1937]. Most of the designs exhibit some periodicity or regularity, but a particular subset — called *ice-ray* — exhibits a more complex structure, at first sight. Such ice-ray lattices contain patterns that resemble the lines formed by cracking ice. Stiny demonstrated, using shape grammars, how the compositional logic of these seemingly complex ice-ray designs can be captured in a few simple (parametric) shape rules [Stiny, 1977]. The rules needed to create ice-ray designs are shown in Figure 6.8. Each rule divides a particular shape (a triangle, quadrilateral, and pentagon, respectively) into smaller shapes. The rules are parametric, in the sense they can be applied to any geometric realization of these shapes, under all kinds of transformation (translation, rotation, scaling, and so forth).

The ice-ray grammar has not been implemented on a computer system, so far. In order to explore the language of this shape grammar, the rules have been implemented using the implementation approach discussed in Chapter 4. This means that the shapes and rules have been translated to part-relation graphs and graph rewriting rules, respectively. Unlike the computer implementation of the Rabo-de-Bacalhau transformation grammar, in which semantic node types and different kinds of application conditions are used (Appendix A), the computer implementation of the ice-ray grammar is more straightforward. In particular, two geometric node types (point and line) have proven to be sufficient in representing the shapes as part-relation graphs (see Figure 4.11), and no particular geometric constraints are needed, because the rules are fully parametric.

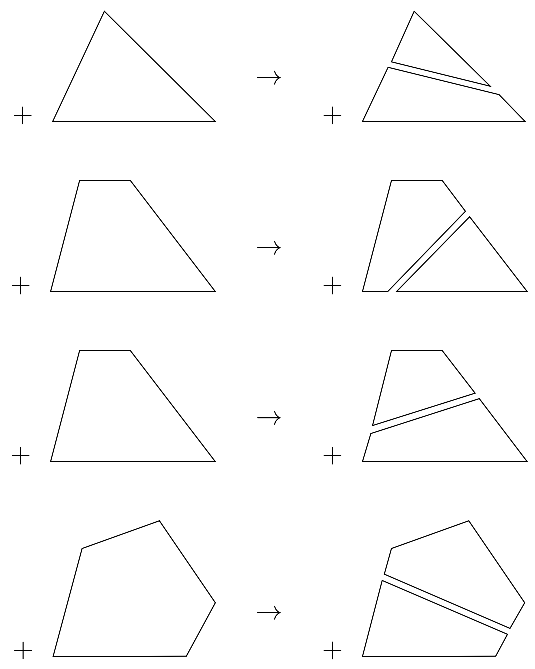


Figure 6.8: Four parametric shape rules of the ice-ray grammar. Reproduced from the original image appearing in Stiny [1977].

After the graph rewriting rules have been specified, it is possible to explore new and existing Chinese lattice designs, either by manually applying rules step by step, or by automatically generating random designs. The stopping criterion of this random design process is either when the parts are too small to be divided, or when a maximum number of rules has been applied. The results shown in Figure 6.9 include five randomly generated designs and one design (bottom right) that has been generated step by step. The latter design is also included in the catalogue of existing Chinese lattices by Dye [1937], thus demonstrating the ability of the ice-ray grammar in capturing the compositional principles of Chinese lattice designs. The other lattice designs in Figure 6.9, which are generated using the same grammar, clearly share the same compositional principles, but they are also different in terms of visual coherence and uniformity. In order to generate new designs that more closely resemble the existing lattice designs, the grammar could be elaborated further by introducing additional application conditions, constraints, or predefined rule sequences. Another way to achieve this goal is by using a goal-directed search strategy, instead of random rule application; for example, using a heuristic search strategy

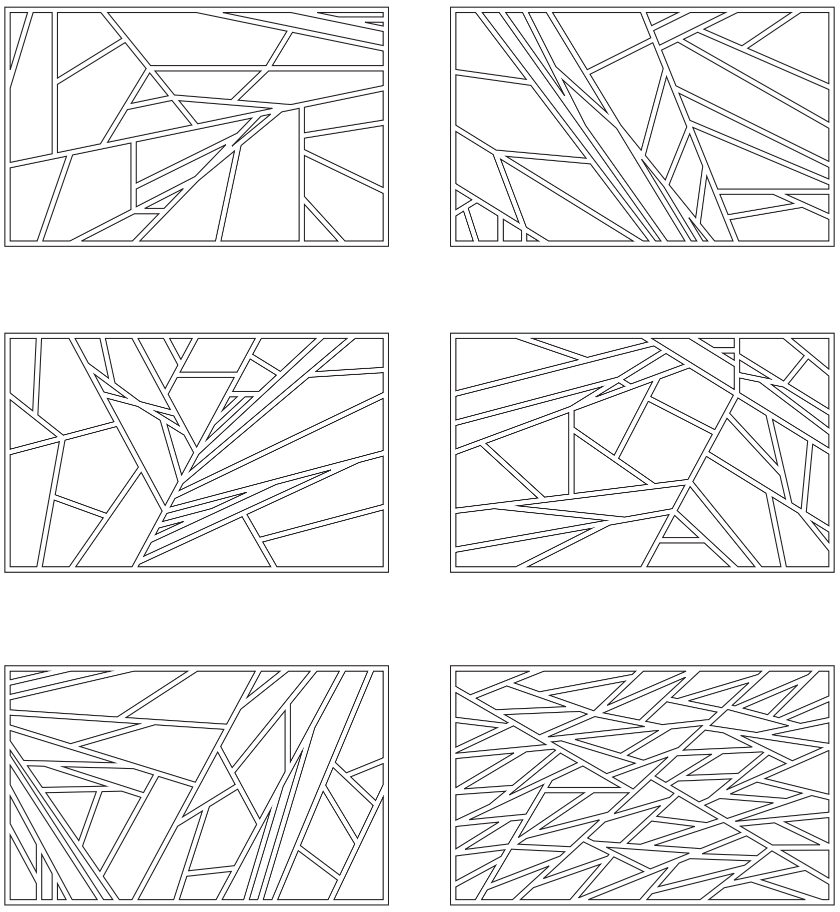


Figure 6.9: Lattice designs in the language of the implemented ice-ray grammar. The designs are generated using random rule application, except for the bottom right design, which is generated manually.

that evaluates visual uniformity. These types of approaches would limit the number of designs that can be generated, thereby constraining the design space. In the context of original design, however, this might come at the expense of design freedom, thus lowering the chances for discovering unexpected or surprising designs.

An important aspect of design space exploration is the ability to come up with new rules. These new rules might make new paths in the design space available for future exploration. In the case of Chinese lattice designs, some of the existing designs in the catalogue contain axial pat-

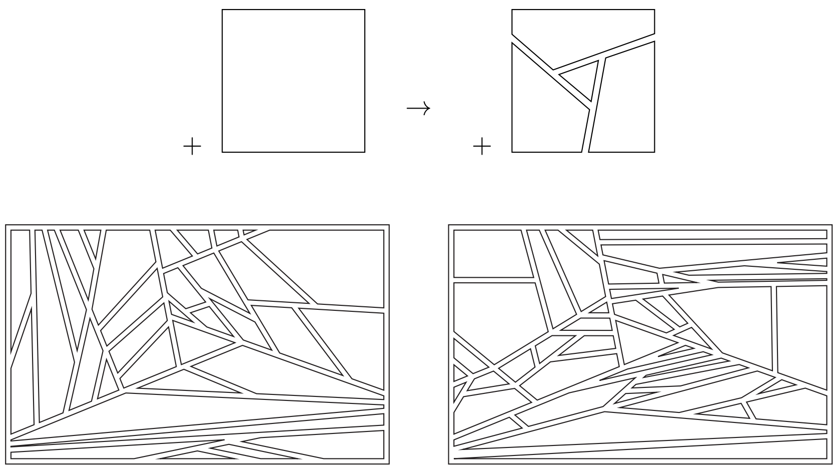
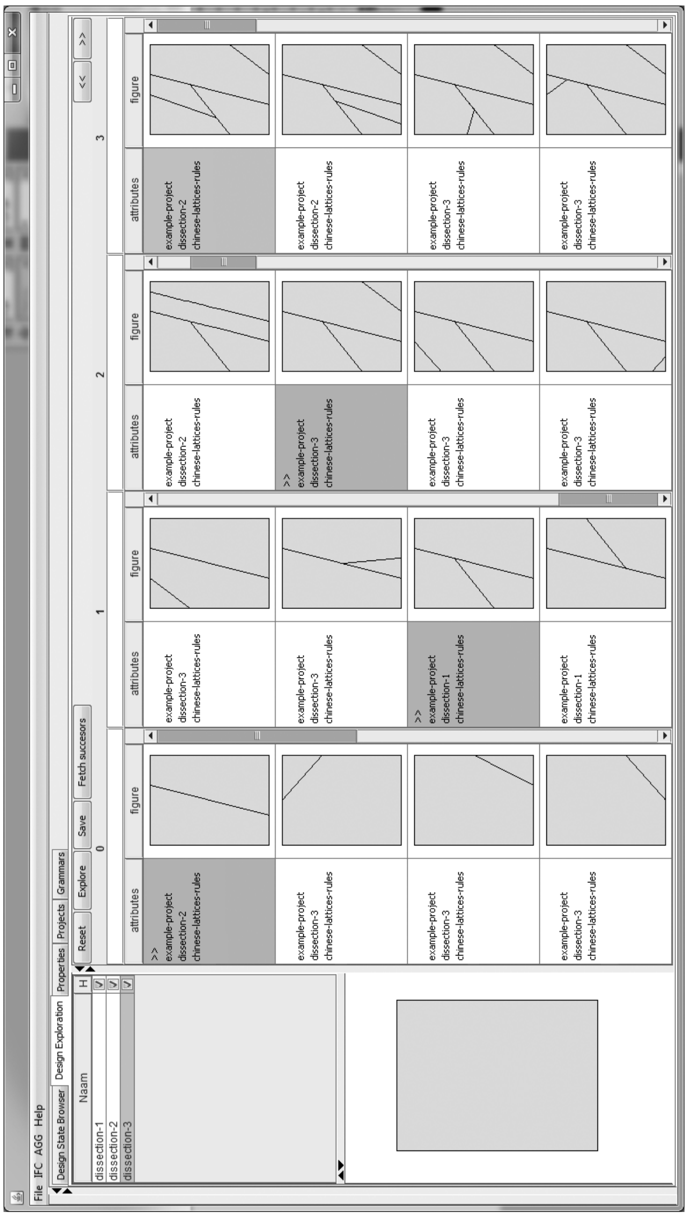


Figure 6.10: An additional shape rule of the ice-ray grammar (top) and two resulting designs (bottom).

terns, which cannot be generated with the rules in Figure 6.8 alone. For this reason, it is desirable to enable designers to create new rules during, instead of before, the process of design space exploration. In doing so, the new rules might change, enlarge, or sometimes reduce the design space in which exploration can be performed. For example, Figure 6.10 shows an additional shape rule of the ice-ray grammar (top) and two resulting designs (bottom). The newly added rule divides a quadrilateral shape according to an axial pattern. Of course, other kinds of rules can easily be devised and added to the grammar in a similar way. As a result, while the ice-ray grammar was initially developed for analytical purposes, it forms the basis for a generative or exploratory process, in which existing and new rules co-exist. Figure 6.11 shows the stepwise exploration of (part of) the Chinese ice-ray design space, as performed on the software prototype.

6.2.2 Example 2: Spatial configurations

In creative design practice, shapes often play an important role in describing form and spatial relations. In some cases, however, the focus is more on semantic or topological aspects of designs, rather than geometric aspects. Especially in architectural sketch design phases, designers tend to explore spatial configurations without having specific geometric realizations in mind. Typical examples include so-called ‘bubble diagrams’ or sketches in which architectural spaces are represented as symbols with re-



	0	1	2	3
	>> example-project dissection-2 chinese-lattices-rules	example-project dissection-3 chinese-lattices-rules	example-project dissection-2 chinese-lattices-rules	example-project dissection-2 chinese-lattices-rules
	example-project dissection-3 chinese-lattices-rules	example-project dissection-3 chinese-lattices-rules	>> example-project dissection-3 chinese-lattices-rules	example-project dissection-2 chinese-lattices-rules
	example-project dissection-3 chinese-lattices-rules	>> example-project dissection-1 chinese-lattices-rules	example-project dissection-3 chinese-lattices-rules	example-project dissection-3 chinese-lattices-rules
	example-project dissection-3 chinese-lattices-rules	example-project dissection-1 chinese-lattices-rules	example-project dissection-3 chinese-lattices-rules	example-project dissection-3 chinese-lattices-rules

Figure 6.11: The stepwise exploration of (part of) the Chinese ice-ray design space, as performed on the software prototype.

lations drawn between them. Graphs offer a natural framework to model objects and relations between these objects. The use of graphs to represent spatial configurations is not uncommon in the architectural design domain; for example, see the early work of Steadman [1976], and the use of graphs in space syntax theory [Hillier and Hanson, 1984] to represent spaces and connections.

A graph grammar has been created to support the exploration of spatial configurations. The type graph of such grammar contains one node type (space) with a single attribute (the function of the space). A straightforward grammar for supporting exploration of spatial configurations is shown in Figure 6.12. This grammar contains a rule for adding new spaces (top left), a rule for assigning functions to spaces (bottom left), and two rules for changing the relations between the spaces (right). Using this grammar, it is possible to generate any configuration of architectural spaces, even when these spatial configurations would be nonsensical, from an architectural point of view. This kind of grammar can be useful for the purpose of revealing unexpected designs, or comparing designs towards multiple design criteria. In some cases, however, it could be more useful to have a grammar that generates only those spatial configurations that make sense from an architectural point of view. In order to develop such a grammar, additional design knowledge has to be included in the grammar rules on how feasible spatial configurations can be achieved. In other words, the grammar rules have to be 'knowledge engineered', which means that design or expert knowledge on desirable and undesirable spatial relations has to be included in the grammar rules. This kind of design knowledge can be found in building codes, good practice manuals, and of course, in the mind of experienced designers.

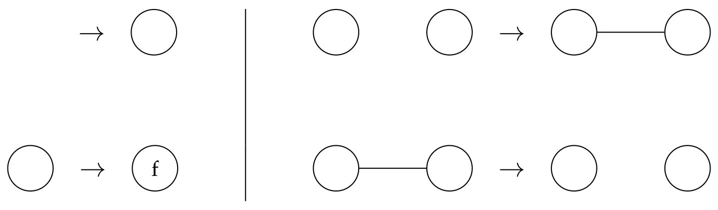


Figure 6.12: Graph grammar for exploring spatial configurations. The grammar contains a rule for adding new spaces (top left), for assigning functions to spaces (bottom left), and two rules for changing the connections between the spaces (right).

For example, several guidelines for floor plan design are collected in the design manual called C2008[2]. These guidelines are stated in a textual manner:

- The entrance room, which acts as a buffer between the living room and the outdoor environment, is strictly required;
- The kitchen should be directly connected to the dining room;
- The storage room should either be connected to the kitchen, or be easily accessible via the entrance; ...

Another way of exploring new spatial configurations is by browsing a library of potentially recoverable designs. The designs in this library might have merit in a new design context; for example, to serve as a source of inspiration, or to learn from (successful) precedents. As these precedents can be developed using a broad range of design media (sketches, several kinds of CAD drawings, and so forth), they should be converted to part-relation graphs, before they can be used in the proposed shape grammar implementation tool. For example, the CVC database[3] contains a collection of 122 scanned floor plan documents. The corresponding part-relation graphs have been generated automatically, using the floor plan recognition tool described in the work of de las Heras et al. [2014]. Subsequently, these spatial configurations have been added to the database, after which they can be recalled in a new exploration process, as a fresh starting point. Figure 6.13 shows several spatial configurations that have been generated from a collection of scanned floor plan documents. Table 6.1 shows an overview of the labels that are used in this figure.

[2]`http://www.vmsw.be/C2008`
[3]`http://dag.cvc.uab.es/resources/floorplans`

li	Living room	ba	Bathroom
ol	Open living room	sh	Bathroom (2)
ki	Kitchen	de	Workroom
en	Entrance	sr	Hall
st	Storage	la	Laundry
ga	Garage	dr	Dressing
ci	Circulation	ou	Outside
wc	Toilet	ea	Dining room
be	Bedroom	hl	Hall

Table 6.1: Labels used for the representation of spatial configurations.

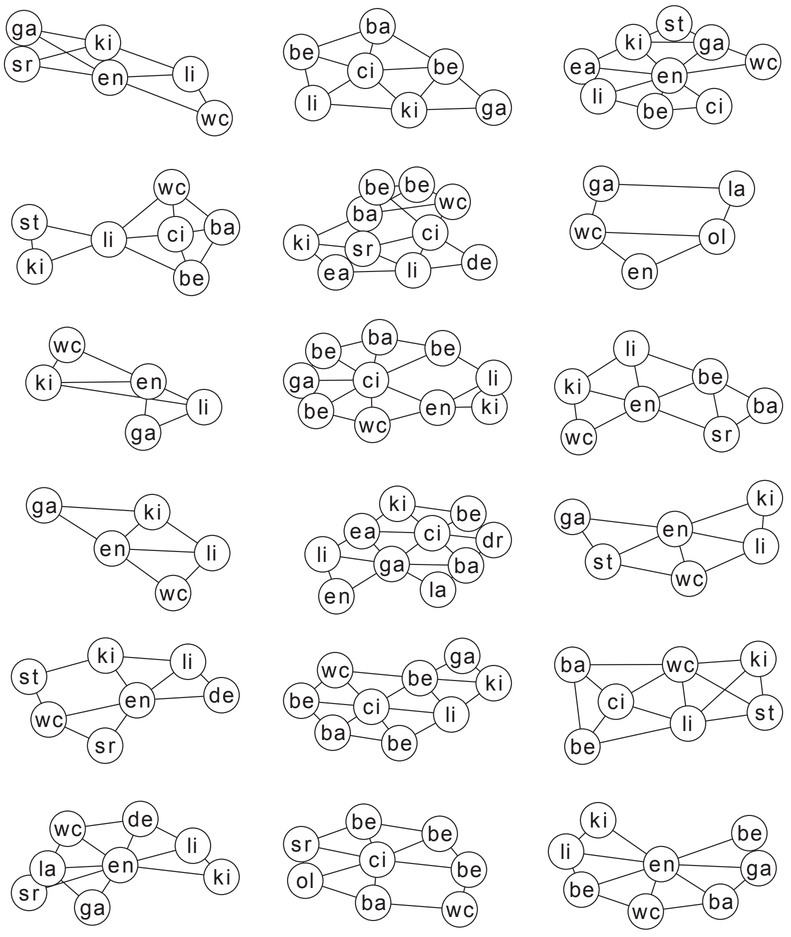


Figure 6.13: 18 spatial configurations that have been generated from a collection of scanned floor plan documents.

6.2.3 Example 3: the Frank Lloyd Wright grammar

The Frank Lloyd Wright (FLW) grammar is a three-dimensional parametric shape grammar, originally developed by Koning and Eizenberg [1981]. The FLW grammar generates the compositional forms, and specifies the function zones of FLW prairie-style houses. Koning and Eizenberg describe the development of the grammar as being based on a corpus of eleven existing designs. Part of this grammar has been implemented, in order to demonstrate how a designer can explore the design space associated with this grammar. In particular, the grammar implemented con-

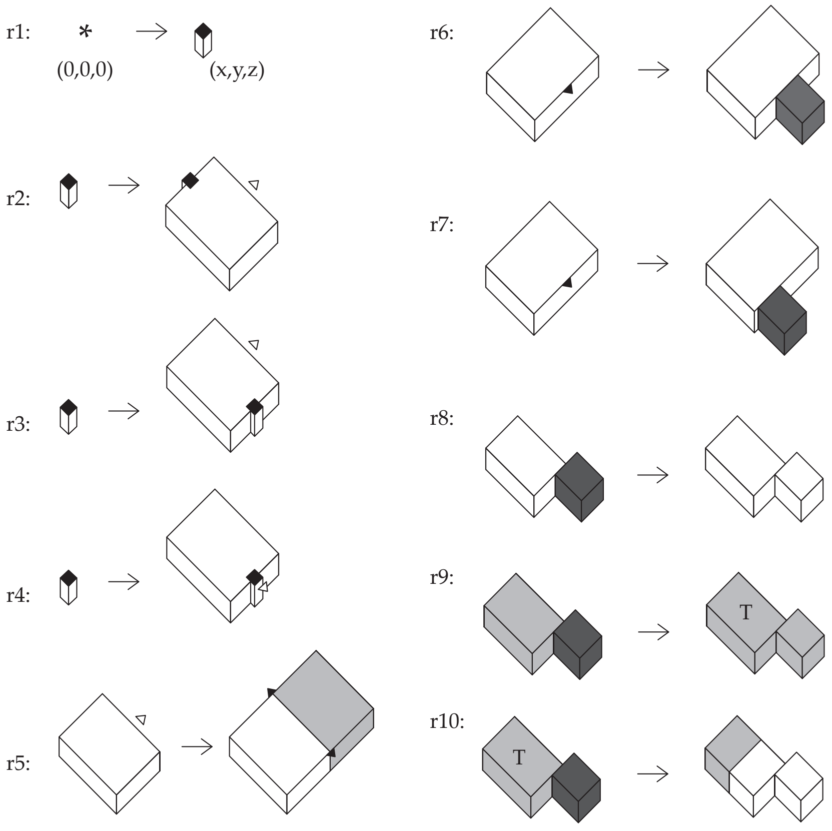


Figure 6.14: Some rules of the FLW grammar [Koning and Eizenberg, 1981] — locating the fireplace (1), adding a living zone (2-4), completing the core unit (5), adding obligatory extensions (6-7), and assigning function zones (8-10). Living zones are indicated in white, service zones in light gray, and the obligatory extension in dark gray.

tains rules to locate the fireplace (1), to add a living zone (2-4), to complete the core unit (5), to add obligatory extensions (6-7), and to assign function zones (8-10). These rules generate all the basic compositions that underlie specific FLW prairie house designs. The resulting designs are specified as three-dimensional objects with labels ($U_{33} \cdot V_{03}$). An overview of the implemented rules is shown in Figure 6.14. There are slight differences between the original grammar and the implemented grammar; for example, rule 1 does not distinguish between a single-hearth and a double-hearth fireplace, and some additional labels have been used to guide the rule application.

The first step involves locating the fire place, after which a living zone can be added in three different ways (determined by rules 2, 3, and 4). This results in a tree structure with three branches denoting the possible alternatives. After manual selection (in the set of alternatives) of the design state to be further explored, a service zone is added by rule 5 in the grammar. This sequence of rules generates the 'core unit' of a prairie house. The current design path "*Adding a living zone and completing the core unit*" can be stored to recall it in a future project (Figure 6.15). In order to recall this sequence from the database in a new design project, a search query can be performed based on metadata, such as the name, the grammar or rule used, or some additional properties that may be associated with the design (sequence).

When the core unit is established, obligatory extensions are added that radiate outwards and are smaller than the core unit, to maximize views and to maintain the fireplace as the hierarchical center of the design. These obligatory extensions are added by rules 6 and 7, resulting in a tree structure with six branches. The selected design state is then further specified by assigning functions to the obligatory extensions. The rules 8, 9, and 10 attempt to group service and living zones together, in order to maximize interior spaciousness. As a result, the rules generate the basic compositions underlying prairie houses, which can further be refined to complete the designs. Figure 6.15 shows some part of the explicit design space of

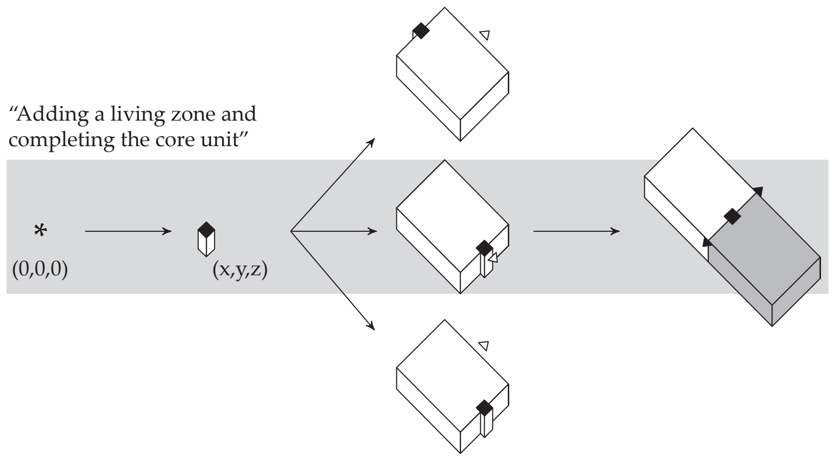


Figure 6.15: Partial derivation of the FLW grammar. The current design path is indicated in light gray. Redrawn from the original image that has been generated using the software prototype.

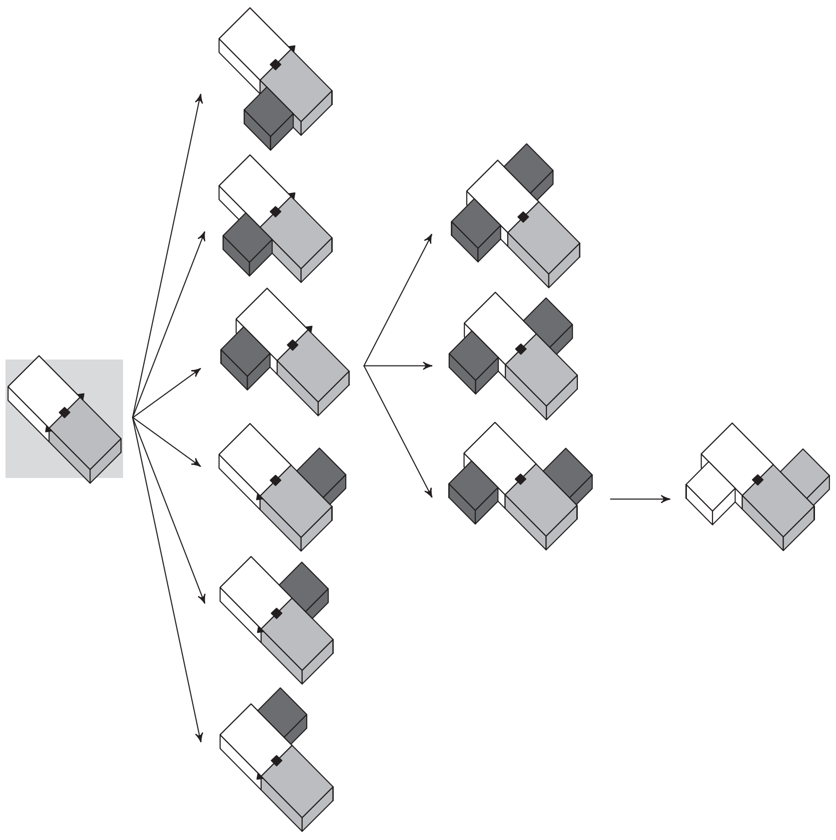


Figure 6.16: Partial derivation of the FLW grammar, starting from a design state that is recalled from a different project. Redrawn from the original image that has been generated using the software prototype.

the implemented grammar — starting from a recalled design state and following one particular design path in the tree structure. The intermediate design states shown in the tree structure may not have intentional value, but they provide access to specific areas of the design space. In this respect, a design is not only characterized by the current state, but also by the derivation history. Through navigation in the explicit design space, new paths are becoming available for future exploration.

As is the case for many shape grammars, the FLW grammar was not implemented on a computer system, yet. In order to do so, the three-dimensional compositions have been implemented using the approach discussed in Chapter 4. In particular, a type graph (or ontology) is used, which

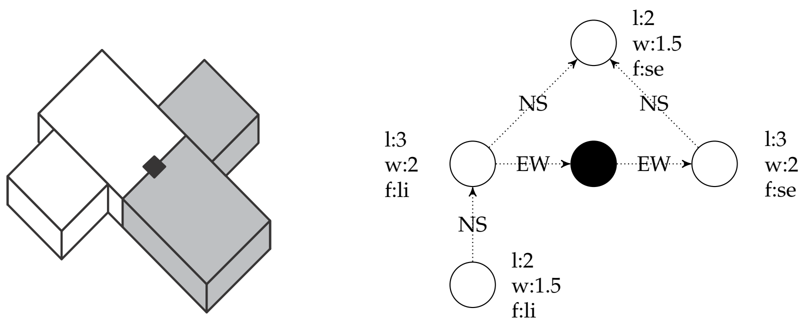


Figure 6.17: Visual representation of a FLW prairie house composition (left) and the corresponding part-relation graph (right). The space nodes are indicated in white and the core unit nodes are indicated in black. The graph edges represent north–south (NS) and east–west (EW) relations between the spaces.

contains two semantic node types (core unit and space), two kinds of relations (north–south and east–west), three attributes (length, width, and function), and some labels to control rule application. Figure 6.17 shows a visual representation of a FLW prairie house composition, together with the corresponding part-relation graph.

7 Conclusion

This thesis started from the observation that information systems for creative design can be roughly classified into three categories — draughtsmen, oracles, and agents (**Chapter 1)**. Given that the impact of draughtsmen and oracles is considered to be limited, a different line of thought is followed in this thesis; namely, how information systems could more closely resemble agents, acting like an assistant or sidekick to the designer. With such agent-like functionality available, it is possible to talk about a mixed-initiative enterprise in which both designers and computers contribute to a result that would have been unviable by either conversant alone. While agent-like design tools may come in many guises, information systems that support — and amplify — the exploration of design alternatives may be of particular interest in architectural design; for example, in the context of an increased emphasis on building performance. The definition of exploration given here involves both searching for previous designs and generating new designs — very similar to the way in which designers perform exploration in a sketchbook. The metaphor of a *digital sketchbook* in which human exploration is mixed with computer amplification strategies is the motivating idea and the central research topic of this thesis.

During the last few years, the topic of design space exploration has gained increasing attention in academic circles and (architectural) design practice alike. While new technologies are now slowly coming to the surface (for example, in visual programming environments), the concept of searching a design space is actually an old idea — parametric modeling was already put at the center of the Sketchpad system in 1963, thereby foreseeing one of the main features of CAD systems to come. In fact, this thesis traces back the origin of the design space to the 1960s, with the birth of artificial intelligence (AI), which initiated the paradigm of searching a problem space for solutions (**Chapter 2**). Through the shared protagonists of AI and architectural design, this shift marks the formulation of design as a form of searching, which can be 'solved' through appropriate heuristics. Such a reductionist definition of design as being 'problem solving, searching, goal oriented, etc.' quickly fell out of favor, making way for renewed insights in the wicked nature of designing. These fresh insights did not lead to the rejection of the design-as-search paradigm, but led to a more subtle formulation and expansion of the design space concept. While searching might be an important part of design explorations, it is not what characterizes design as a distinct kind of behavior. Instead, design space exploration is often characterized as an adventurous, and at times poorly guided, activity. This results in an apparent dilemma between the wickedness of creative design and the structured nature of information systems.

A well-studied approach shown to be capable of bridging this gap is provided in the theory of shape grammars (**Chapter 3**). This thesis points out how shape grammars provide a concise and computable framework to represent a design space by encoding design moves in the form of shape rules. On the one hand, shape grammars are clearly influenced by concepts drawn from the field of AI — such as their formulation as rule production systems. On the other hand, they deviate from traditional AI approaches by avoiding any form of explicit or fixed representation and by incorporating a kind of visual thinking and ambiguity that characterizes creative design. In particular, exploration with shape grammars involves creating, storing, repeating, and copying rules in new design situations — using embedding as the key mechanism for doing so. As a result, they offer an elegant formalism for describing a kind of rule-based design, while at the same time providing a computable representation of the design space, which can be explored in a mixed-initiative enterprise. This is not to say that design can be reduced to calculating with shapes and rules alone, and most certainly not that all aspects of design can be described with shape grammars. Nevertheless, they do incorporate some phenomena described in the field of design theory and thinking (such as reframing and emergence), making them a notable and interesting formal theory. Rather than a reductionist model of design, shape grammars should be considered as being powerful tools for exploration — the large number and diversity of visual explorations shown throughout this thesis serve to illustrate this.

The contribution of this thesis is twofold — it advances the state-of-the-art in the computer implementation of shape grammars, and also, it allows several conclusions to be drawn based on the more general topic of design space exploration. Turning our attention to its contribution to the field of shape grammar theory first, this thesis proposes a particular symbolic representation for shapes, which is suitable for computer implementation, without losing the essential features resulting from the emergent nature of shapes (**Chapter 4**). The development of appropriate symbolic representations is challenging; for example, to support subshape detection and parametric grammars, but is valuable for reasons of automatic generation, parsing, inference, and development. In this thesis, attributed part-relation graphs are introduced and are shown to be a feasible and valuable choice to implement grammars — because they enable (parametric) subshape detection, but they also extend shape grammars by describing semantic objects, instead of purely geometric ones. This results in more compact representations, which positively influences the time needed for rule matching or subshape detection, and also makes it easier to specify rules. Nevertheless, the specification of graph rules and grammars might be less natural (and more error-prone) to designers than their shape counterparts, which implies the need for an intuitive and visual editor built on top of the underlying graphical rule editor. Also, the development of shape grammars and graph grammars has been shown to be complementary, in the sense that both lead to an alternative understanding of the grammar at hand. Finally, the RdB-tf case study (Appendix A) discussed in this thesis has shown that computer-implemented grammars are found to be useful for purposes of enumeration, revelation or comparison — especially for grammars that are more difficult to explore manually.

The second aspect of the thesis' contribution is the implementation of several amplification strategies for design space exploration (**Chapter 5**). While external representation, codification of design moves, and implication are three strategies commonly found in shape grammar theory and its computer implementations, the strategies of replay, recall, backup, and alternatives — all involving the representation of the explicit design space — are only available to a far less extent. The representation of the explicit design space may be valuable to reveal and compare alternatives, to back up and recall prior work, and to replay paths previously discovered in a design space. In this thesis, the concept of tree structures is introduced to keep track of the explicit design space. Unlike previous research efforts in which trees are used, they are not used as devices for searching design solutions, but they are used to implement several aspects of design space exploration, including the generation of design alternatives and the navigation in the explicit design space. Navigation in the explicit design space can be done by navigating from one design state to another along design paths, or by searching stored design states based on design state properties; for example, a name, a timestamp, a rule, or other kinds of descriptive information associated with each design state. In order to enable this kind of navigation, a persistence system (such as a database) is needed to efficiently store and organize design states. As a result, tree structures might enable some aspects of design space exploration, which is a far richer concept than merely searching.

The theory of shape grammars, the graph-theoretic representation of shapes and designs, and the tree structure together form the keystones of a new kind of grammar-based tool for design space exploration — the *digital sketchbook*. The outline of such a digital sketchbook, together with several visual examples, is described in **Chapter 6**. The three main aspects are: (1) representing and visualizing the design space as a whole, (2) enabling backup and recall strategies, and (3) enabling designers to create, change, or delete rules in an intuitive manner. The visual examples in this chapter demonstrate how the proposed shape grammar implementation tool can be used in a variety of situations — ranging from analytic grammars (FLW prairie houses) to original grammars (spatial configurations), and from simple grammars (Chinese lattices) to more complex grammars (Portuguese RdB houses).

7.1 Some future lines of research

While the proposed approach shows merit in several aspects, it only scratches the surface of a new kind of design space exploration tools. There are several future lines of research that can be identified in order to further develop the work set out in this thesis. First, the recall of designs, not based on navigation in the tree, but based on searching on (semantic) design properties, is an interesting future research area worth pursuing. The database system that has been implemented in the software prototype already hints at some potential benefits, however, it is possible to deal with semantics in a much more structured way — a notable example is the semantic web, in which data can be shared and linked over the web. In the context of design recall, an equivalence relation for detecting similarities between designs can be useful for retrieving similar design precedents in the design space — the classification method that is described in Appendix B might be a first step in this direction.

Second, the implementation of the tool for design space exploration on to a (commercial) CAD environment is a key aspect for extending its impact. The software prototype enables designers to import designs that were generated in a particular CAD environment, after which they can be further explored in the design space. An important future research question here is how to deal with semantics that are inherently associated with the data structure of the CAD model imported. It is indeed possible to import a design, by converting it into an attributed part-relation graph, which may contain geometric node types (for representing shapes) and semantic node types (for representing semantic or architectural concepts). However, in the current prototype, it is not possible to transfer modified shapes or designs back to the initial CAD environment.

Third, since the development of grammar rules is often a laborious and error-prone task, the development of intuitive rule editor interfaces is a key aspect that should be further investigated — especially in the context of creative design, where rules should be created and modified on the fly, thus during the exploration process. Two possible approaches for supporting grammar development are providing systematic analysis during rule development and learning grammar rules in an automatic manner. The use of algorithms from machine learning, a subfield of AI that deals with algorithms that learn from examples, is a promising future line of research.

The proposed digital sketchbook, in its current form, can be used on the short term by other researchers in the field who are interested in implementing shape grammars on a computer system. The design tool can then be used to evaluate the correctness of a grammar, and to explore the language of a grammar in a systematic way. However, the lack of integration into a commercial CAD environment and the lack of an intuitive rule editor prevent the tool for being used in architectural design practice. Also, extensive error handling and a more robust implementation would be needed to make the step from prototype to a full-fledged, foolproof software system. While this step might not include additional research difficulties, it is not possible to realize such a system within the timespan of one PhD research period. Given that a more elaborated rule editor, user interface, and robust implementation would be available, the software system could then be used by a larger group of architects, using the feedback and surveys to further elaborate the proposed design tool. This also indicates the need for a more concerted research effort on the topic of shape grammar implementations. This concerted research effort was initiated at a 2010 workshop '*shape grammar implementation: from theory to usable software*', and continues to date. The work discussed in this dissertation provides only a small, but nevertheless an important, step towards this end.

7.2 Concluding remarks on design and computation

In a more general way, this thesis touches upon two key aspects of design and computation — *representation* and *process*. First, representation in most traditional CAD tools involves a set of predefined types (line, polyline, etc.) that can be combined to make 'meaningful' objects. This information structure is strongly hierarchical and combinatoric. As a result, making changes to this structure becomes increasingly difficult for each geometric structure built up. In contrast, the kind of representation described in this thesis does not involve predefined types, but designs can continuously be reinterpreted, according to what parts the designers consider to be of interest (at a particular moment in time). This idea could form the basis for any kind of CAD tool — not only grammar-based tools — which uses specialized algorithms to detect which parts can be of interest to designers. Second, in most traditional CAD tools, the aspect of process involves search and/or optimization in a design space that is limited by a bounded range of alternatives. Search and optimization play an important role in narrowing the design space towards design alternatives that are near-optimal in terms of some predefined goal test. Many designs in a given design space may indeed be nonsensical — for instance, the sentence "*Colorless green ideas sleep furiously*" composed by Noam Chomsky, is an example of a sentence that is grammatically correct, but without understandable meaning that can be derived from it. In this thesis, the focus is not so much on choosing a single (near-optimal) design, but rather on changing the designer's way of thinking through exploration. Exploration is a much richer concept, which involves interaction with the design space through search, generation, and navigation. These aspects are not limited to grammar-based design tools only, but could also be incorporated in any kind of CAD tool.

The latent theme underlying this thesis is the use of AI and other knowledge representations in the domain of creative design. While AI has successfully outperformed humans in several tasks (playing chess, route navigation, etc.), the human ability for creative design seems to be out of reach. At the very best, the proposed tool for design space exploration might generate new, anticipated, or sometimes even surprising results, but this can hardly be called creative behavior. Instead of trying to simulate, or even surpass, human creativity, as pursued by the early AI pioneers, the search for suitable human–machine collaborations might perhaps be less ambitious, but more promising. As Horst Rittel, a stubborn skeptic of AI, already pointed out — "*as my eyeglasses do not see on my behalf but help me to see better, one might use the computer not to think on ones behalf but to reinforce and enhance ones own ability to think.*". It is in this context that the digital sketchbook metaphor should be considered as a mixed-initiative enterprise of human creativity and computer amplification. In other words, the computer is a tool — one of many — and much like any other tool, it is only as good as the person who is using it. The results of such a mixed-initiative enterprise might surpass the results generated by either the designer or the computer alone — which is demonstrated in the examples throughout this thesis. In fact, many of today's designed objects are the result of human–machine collaborations, including this very thesis.

A
Implementation of the RdB-tf grammar

In this appendix, we describe the implementation of the original Rabo-de-Bacalhau transformation (RdB-tf) grammar. In order to do this, the proposed approach of type graph definition, attributed graph construction, and application condition specification is followed. This results in a graph grammar that can be explored on a computer system. This appendix is adapted from the paper that is to be published in *T. Strobbe, S. Eloy, P. Pauwels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. A graph-theoretic implementation of the Rabo-de-Bacalhau transformation grammar. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 2015.*

A.1 The original RdB-tf grammar

The original RdB-tf grammar, developed by Eloy [2012], attempts to provide an answer to the need for mass rehabilitation of the existing housing stock in Portugal. A large part of the existing housing stock shows several constructional and functional problems — resulting in unsuitable housing in terms of contemporary comfort and accessibility standards. In particular, the RdB-tf grammar constitutes a grammar-based methodology to generate alternative housing designs that meet the current comfort and accessibility standards, but that also depend on specific client needs and cost requirements. In order to do this, the RdB-tf grammar encodes several *transformation strategies* to adapt traditional Portuguese Rabo-de-Bacalhau (codtail) houses. These transformation strategies — such as moving the kitchen from its original position to strengthen the relationship between the social and service areas, or maintaining the position of the kitchen while keeping construction transformations to a minimum — contain a number of transformation rules that step by step transform an existing housing design to a transformed housing design. The floor plan of any existing RdB house can be used as the starting point of the grammar, and by applying different rule sequences (the so-called transformation strategies), multiple transformed floor plans can be generated — each adapted to the comfort and accessibility standards, client needs, and cost requirements.

The original RdB-tf grammar uses a compound representation of the designs and the transformation rules (Figure A.1). In particular, the Rdb-tf grammar is defined in the algebras U_{02}, U_{12}, U_{22}, V_{02}, and W_{02}. The different representations of an existing housing design to be transformed include a two-dimensional floor plan (U_{12}), the topological relations ($U_{02} \cdot U12$), and the spatial voids (U_{22}). Labels (V_{02}) are attributed to spaces in the floor plan to associate semantic information not provided by shapes — for example to assign habitable spaces, non-habitable spaces, and so forth. Table A.1 shows an overview of all the labels used in the RdB-tf grammar. Weights (W_{02}) are used to incorporate shape properties and to characterize construction systems — for example brick walls (light gray) structural elements (dark gray), or side walls (black).

nhs	Non-habitable space	co	Corridor
hs	Habitable space	co.p	Private corridor
xba	Existing private bathroom	la	Laundry
xki	Existing kitchen	hl	Hall
xla	Existing laundry	ba	Bathroom
be	Bedroom	ba.p	Private bathroom
be.s	Single bedroom	ba.g	Guest bathroom
be.d	Double bedroom	lh	Lift hall
ki	Kitchen	di	Dining room
li	Living room	ho	Home office
F	Function	D_n	Derivation stage
w	Width	l	Length
Ff	Function of front room	Fb	Function of back room

Table A.1: Labels used for the representation of housing designs and for the transformation rules.

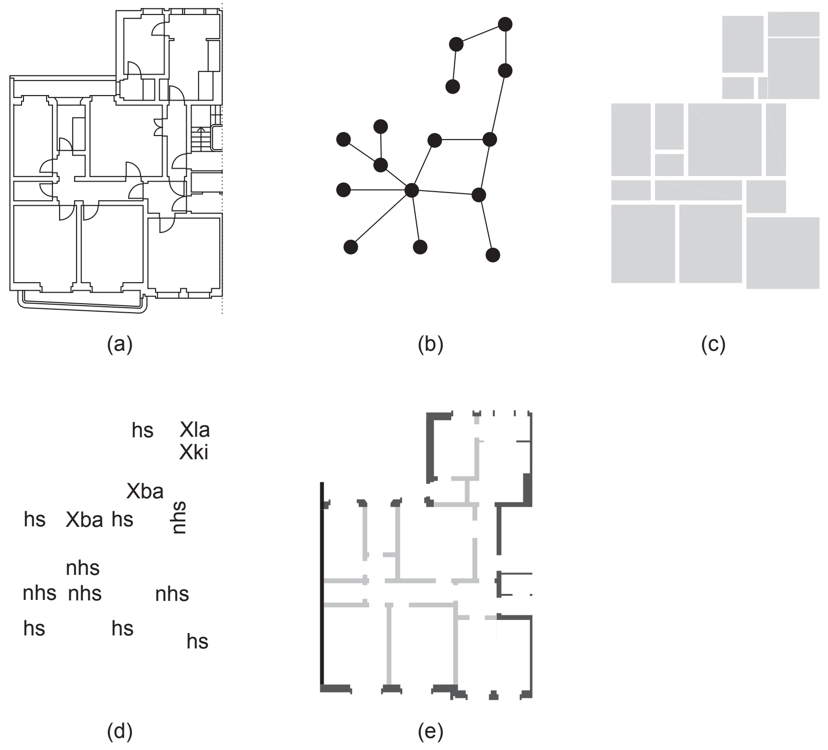
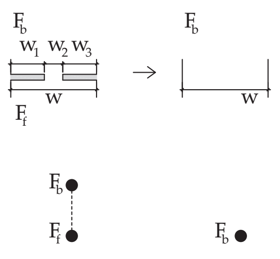


Figure A.1: Compound representation of an existing RdB housing design. (a) Floor plan representation, (b) topological configuration of spaces, (c) spatial voids, (d) labels, and (e) weights. Reproduced from the original image appearing in Eloy [2012].

The rules in the original RdB-tf grammar are defined using the same compound representation. In particular, the rules consist of three parts — a shape part, a conditional part, and a descriptive part. First, the shape part of the rules contains a combination of multiple representations (algebras). At least two representations are needed to specify the rules, but sometimes more or all representations are needed to fully specify the rule. The rule in Figure A.2 contains a representation of a partial floor plan (U_{12}) and topological relations ($U_{02} \cdot U12$). Second, the conditional part of the rules describes dimensional and functional aspects to control rule application towards specific cases. The rule in Figure A.2 describes that it can only be applied if the dimensional and functional conditions are satisfied. A logical notation is used to define these conditions. Third, the descriptive part of the rules contains information that cannot be defined in terms of shapes; for example, to keep track of functions already assigned to a design or to



Conditions:

Dimensions:
w ≤ 2m
0m ≤ w1, w2, w3 ≤ 2m

Function:
Fb ∈ {be.d, be.t, be.s} ∧ Ff ∈ {nhs, co, co.p, co.s, cl}
∨
Fb ∈ {li, di, li/di, ho, mr} ∧ Ff ∉ {be, ba, la, ki}
∨
Fb ∈ {la} ∧ Ff ∈ {Xba, nhs, st}
∨
Fb, Ff ∈ {co,co.s, co.p} ∧ Fb = Ff

Description (abbreviated):
R7.1.b <D7: Fb, Ff; w*wub(Fb, Ff)> → <D7: Fb; w*Ø>

Figure A.2: Example of a rule for connecting two adjacent spaces by eliminating a straight wall. The shape part is shown on the left, and the conditional and descriptive parts are shown on the right. Reproduced from the original image appearing in Eloy [2012].

keep track of spaces still available for assignment. The descriptive part is defined as an operation on a tuple of elements (separated by a semicolon). In this example, the operation has the format $< D_n : F_b, F_f; w * wcs(F_b, F_f) > \rightarrow < D_n : F_b, F_f, ; w' * w_{cs}(F_b, F_f) >$, where D_n is the stage in the derivation, w is the width of the wall, and w_{cs} is the wall construction system.

A.2 Step 1: Defining a type graph

The original RdB-tf grammar is available in the form of rules written on paper, so that it can be explored using manual rule applications. On the other hand, if this grammar should be implemented as a graph grammar, it could be explored on a computer system. The first step to implement such grammar is to define a type graph that specifies which node types, edge types, and attributes can be represented in the graph. The aim is to represent designs and rules with as little graph objects as possible, while maintain sufficient semantic meaning. Compact representations positively influence the time needed for rule matching or subshape detection, and they also make the specification of rules more easy and intuitive. For example, the floor plan could have been represented using nothing but geometric node types (lines and points). Indeed, architectural or semantic elements — such as spaces, walls, doors, and windows — can all be rep-

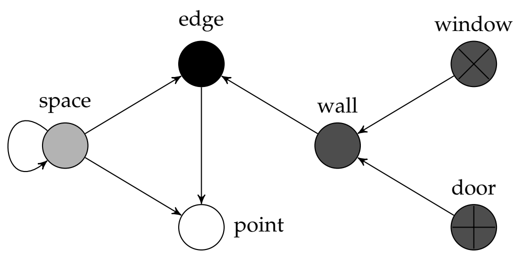


Figure A.3: Type graph for the construction of part-relation graphs, including six node types (space, edge, point, wall, door, window) and six attributes (function f, coordinate geometry x and y, construction system cs, width w, length l).

resented as shapes, but this would result in overly large graphs, which is disadvantageous for automatic subshape detection. Moreover, this would result in ambiguity, because architectural elements can be drawn according to different conventions (which is a major problem in automatic floor plan recognition). As a result, architectural and semantic elements are treated as symbolic entities in the implemented graph grammar. Figure A.3 shows the chosen type graph, which contains six node types (space, edge, point, wall, door, window), and seven edge types (space-edge, space-point, edge-point, wall-edge, door-wall, window-wall, adjacent-space).

A.3 Step 2: Constructing attributed graphs

The second step is to construct the attributed part-relation graphs. Attributes represent different kinds of information that are not topological in nature, for example numerical features or textual descriptions. In this case, attributes are associated with the graph nodes to fix coordinate geometry, to characterize construction systems for walls, to describe geometric properties of windows and doors, and to assign functions to spaces. In particular, the attributes 'x' and 'y' define the point coordinates, the attribute 'cs' describes the wall constructions, the attributes 'w' and 'l' define the width and length of door and window objects, and the attribute 'f' describes the function of spaces.

With both the type graph and the attributes specified, a housing design and the shape part of the rules can be represented using such an attributed part-relation graph. For example, Figure A.4 shows the part-relation graph of the initial housing design shown in Figure A.1. The resulting part-relation graph contains 113 graph nodes, 294 graph edges, and 126 attributes. For illustrative purposes, the attributes are not shown, except for

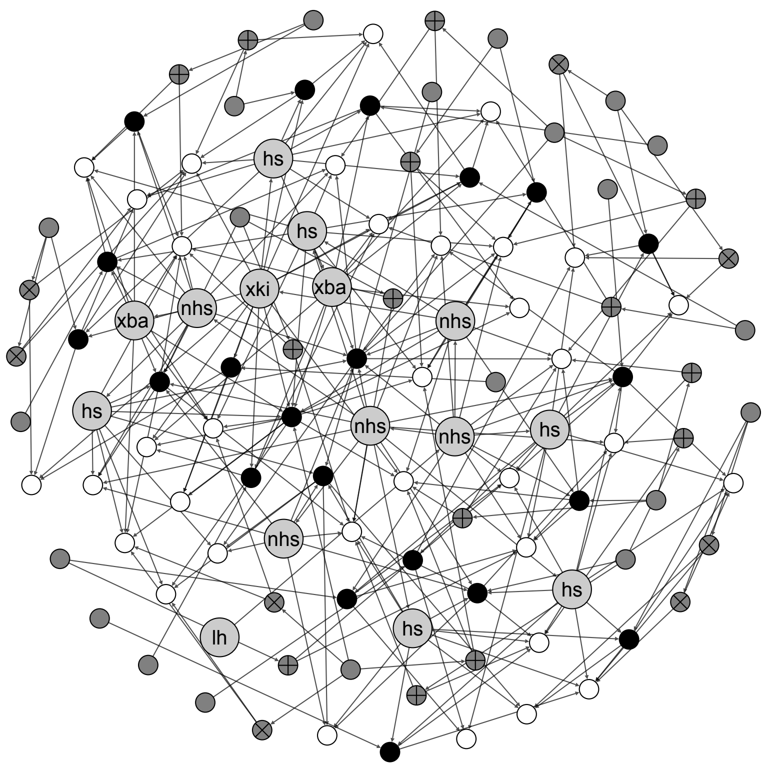


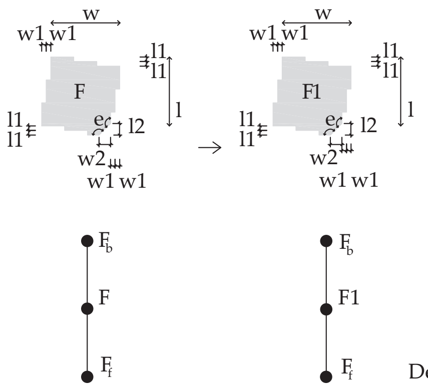
Figure A.4: Attributed part-relation graph of an existing RdB housing design. The attributes are not shown here, except for the function attribute of space node types.

the function attribute of space node types. Note how the information contained in the resulting graph is equivalent to the five representations used in the original RdB-tf grammar. This attributed part-relation graph serves as one possible starting point of the RdB-tf grammar, but in the context of this grammar, the floor plan of any other existing RdB housing design might serve as the starting point of the transformation process. Manually constructing the part-relation graph of each floor plan would prove to be a laborious and error-prone task. In practice, these initial floor plans are usually drawn in a traditional CAD environment. For this reason, a conversion tool was developed to convert floor plans from a traditional CAD environment to an attributed part-relation graph. As a result, users can specify the designs and rules in their familiar CAD environment, while the underlying graph for the representation of shapes, rules, and the generation of design alternatives remains hidden for the user.

A.4 Step 3: Defining graph rules

The rules in the RdB-tf grammar encode multiple strategies to transform an existing housing design so that it meets the current standards, client needs, and cost requirements. These transformations involve assigning (new) functions to spaces, connecting spaces by removing (parts of) walls, and dividing spaces by adding walls. In order to specify these kinds of transformations, three main rule types are used in the TdB-tf grammar — which are assignment rules, connection rules, and division rules. Other rule types include rules to permute functions of spaces, to integrate technical devices, and to change the derivation stage in the grammar.

A first rule type is used to assign functions, required by a given client program, to spaces in the existing design. For example, the rule in Figure A.5 transforms a non-habitable space (nhs) to a hall space (hl) by modifying the label from this space — both in the floor plan and graph representation. The shape part of the rule contains a parametric shape to match spaces with different geometry in the floor plan. The conditional part of the rule describes dimensional and functional conditions that must be satisfied to apply the rule. The descriptive part of the rule is described as an operation on a four-tuple with the format $< D_n : F_b, F_f; F; Z'; E > \rightarrow < D_n : F_b, F_f; F1; Z' + \{F1\}; E - \{F\}, E + \{F1\} >$, where D_n is the stage in the derivation, Z' is the set of spaces already assigned, and E is the set of existing spaces.



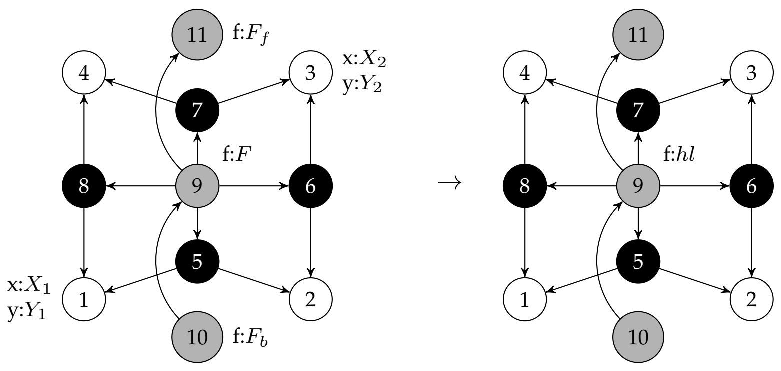
Conditions:

Dimensions:
l, w ≥ 0.9m
$1m^2 \leq F \leq 20m^2$
0m ≤ l1, w1 ≤ 1m
l2, w2 ≥ 0m
e ∈ {135°, 180°}

Functions:
F ∈ {nhs}
Fb ∈ {lh} ∧ F(passage_to(lh)) = true
Ff ∈ {nhs} ∧ F(passage_to(nhs)) = true
⇒ F1 ∈ {hl}

Description (abbreviated):
R1.1 <D1: lh, nhs; F; Z'; E> →
<D1: lh, nhs; hl; Z'+{hl}; E - {nhs}; E + {hl}>

Figure A.5: Rule from the original RdB-tf grammar for the assignment of a hall space. Reproduced from the original image appearing in Eloy [2012].



Attribute conditions:
$F == nhsF_b == lh$
$F_f == nhs$
$X_2 - X_1 \geq 0.9m$
$Y_2 - Y_1 \geq 0.9m$
$(X_2 - X_1) * (Y_2 - Y_1) \leq 20m^2$

Negative application condition:

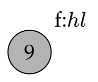
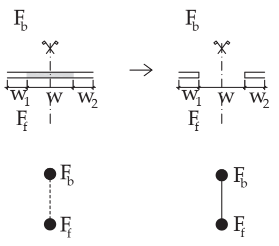


Figure A.6: Graph-theoretic representation of the hall assignment rule. The rule contains a pattern graph (left), a replacement graph (right), attribute conditions, and a negative application condition. The numbers indicate the rule morphism.

The graph-theoretic representation of the hall assignment rule is shown in Figure A.6. The pattern graph contains three nodes (light gray) to represent the space to be transformed (F) and two adjacent spaces (F_b and F_f). Also, geometric nodes (point and line) define the topology and geometry of the space to be transformed. In the original rule, the space to be transformed is drawn as a parametric shape to match spaces with different geometry. In the implemented graph rule, the pattern graph can be matched to all parametric variations of a rectangle. The transformation performed by the rule is to modify the attribute 'f' of the space F, leaving the topology of the graph unchanged. Additional attribute conditions define the necessary dimensional and functional conditions of the original rule. These attribute conditions are specified as logical expressions, which are evaluated once the rule is applied to a given graph. In order to avoid a hall function being assigned to more than one room in the floor plan, a negative application condition is added to the hall node in the replacement graph. Due to this negative application condition, the particular rule can only be applied if no hall function can be found in the graph, thereby avoiding duplicate assignments.

A second rule type is used to eliminate a part of a straight wall, thereby connecting or enlarging spaces. For example, the rule in Figure A.7 connects two adjacent spaces, if several dimensional and functional conditions are satisfied. The shape part of the rule is defined using two representations — a partial floor plan and the topological relation between a front space (F_f) and back space (F_b). The conditional part of rule describes that only specific adjacent spaces can be connected, for example a bedroom (be) with a corridor (co) or non-habitable space. The descriptive part of the rule is described as an operation on a tuple with the format $< D_n : F_b, F_f; w * w_{cs}(F_b, F_f) > \rightarrow < D_n : F_b, F_f, ; w' * w_{cs}(F_b, F_f) >$, where w is the width of the wall and w_{cs} is the wall construction system. In this specific rule, a part of the existing brick wall (w_{ub}) is removed to make a door opening between two spaces.



Conditions:

Dimensions:
w1 + w + w2 ≥ 1m
w ∈ {0.8m, 0.9m, 1m, 1.2m, 1.6m}
w1, w2 ≥ 0m

Function:

Private areas:

Fb ∈ {be.d, be.t, be.s} ∧ Ff ∈ {nhs, co.p, co} ∧ Fb(passage_to(x) = false, ∀x ∈ {nhs, co})
⇒ w ∈ {0.8m, 0.9m, 1m}

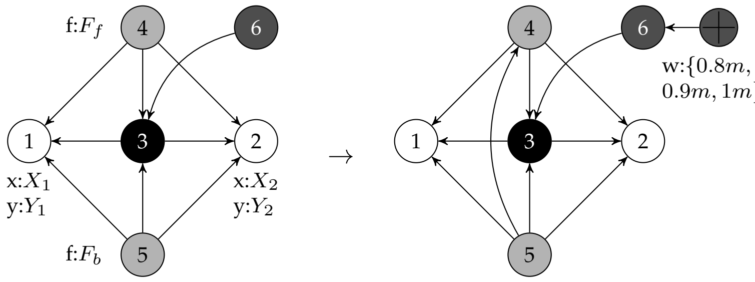
Fb ∈ {be.d, be.t, be.s} ∧ Ff ∈ {cl, ba.p} ∧ Ff(passage_to(x) = false, ∀x ∈ {Z, hs, nhs})
⇒ w ∈ {0.8m, 0.9m, 1m}

Fb ∈ {ba.p, cl} ∧ Ff ∈ {nhs, co.p, co} ∧ Fb(passage_to(x) = false, ∀x ∈ {Z, hs, nhs})
⇒ w ∈ {0.8m, 0.9m, 1m}

Description (abbreviated):
R7.1f <D7:Fb, Ff,w*wub(Fb, Ff) →
<D7: Fb, Ff, w*Ø>

Figure A.7: Rule from the original RdB-tf grammar to connect two adjacent spaces by eliminating a part of a wall. Only the conditions for private spaces are shown. Reproduced from the original image appearing in Eloy [2012].

The graph-theoretic representation of the connection rule is shown in Figure A.8. The pattern graph contains two nodes (light gray) to represent two adjacent spaces (F_b and F_f), an edge node (black) connected with two point nodes (white) to represent the shared edge of the two adjacent spaces, and a wall node (dark gray) connected with the edge node. The transformation performed by the rule involves adding a new node that represents a door object, and adding a new 'adjacent-space' edge between the two spaces to connect them. The dimensional and functional conditions are defined as attribute conditions, using a similar logical notation as in the original rule. In particular, only specific combinations of rooms can be connected, and only if the wall is sufficiently large to make an opening in it. A negative application condition is added to the replacement graph to avoid that the two spaces are already connected, or in other words, to ensure that the wall does not yet contain an opening. Finally, the implemented rule — just like the original rule — contains one parametric attribute (w) that can take three different values, depending on the selection of the user.



Attribute conditions:
$(X_2 - X_1) \geq 1m \vee (Y_2 - Y_1) \geq 1m$
$(F_b == \{be.d, be.t, be.s\} \wedge F_f == \{nhs, co.p, co\})\vee$
$(F_b == \{be.d, be.t, be.s\} \wedge F_f == \{cl, ba.p\})\vee$
$(F_b == \{ba.p, cl\} \wedge F_f == \{nhs, co.p, co\})$

Negative application condition:

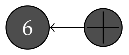


Figure A.8: Graph-theoretic representation of the connection rule. The rule contains a pattern graph (left), a replacement graph (right), attribute conditions (bottom), and a negative application condition. The numbers indicate the rule morphism.

Other rules in the original RdB-tf grammar include rules to divide spaces by adding a wall, to permute functions of spaces, to integrate technical devices in the housing design, and to change the derivation stage in the grammar. These rules can be implemented in a similar way as the two examples (assignment and division rule types) discussed — without additional significant difficulties. Table A.2 shows the rules that have been implemented using the AGG graph transformation tool [Taentzer, 2004]. This set of rules corresponds to one of the two transformation strategies encoded in the original grammar. In particular, the implemented transformation strategy involves maintaining the position of the kitchen, and relocating other spaces, thereby keeping construction transformations to a minimum effort.

A.5 Conclusion

This appendix presents a summary of an approach for the graph-theoretic implementation of a shape grammar, originally developed on paper, on a computer system. A practical step-by-step approach is given for the translation of a shape grammar to an equivalent graph grammar. The RdB-tf grammar is used to demonstrate the details of this approach and to evaluate the feasibility. In particular, two relevant types of rules used in the

Description	Reference
Assignment of isolated kitchen	Rule 0.1
Assignment of hall	Rule 1.1
Assignment of double bedroom	Rule 2.1b
Assignment of single bedroom	Rule 2.3b
Permuting bedroom assignment due to area criteria	Rule 2.5
Assignment of main private bathroom	Rule 2.6
Assignment of second private bathroom	Rule 2.8b
Assignment of living room	Rule 3.1a
Assignment of dining room	Rule 3.2b
Assignment of isolated home office	Rule 3.4
Assignment of guest bathroom	Rule 3.11
Assignment of private corridors	Rule 4.1
Assignment of corridors	Rule 4.2
Widening the connection between two rooms (by eliminating walls on both sides of a door opening)	Rule 7.1.i
Changing room dimension by moving a wall	Rule 7.4b

Table A.2: The implementation of RdB-tf grammar rules corresponding to one particular transformation strategy. For each rule, a short description is given, together with a reference to the original rule [Eloy, 2012].

RdB-tf grammar are discussed — assignment rules and connection rules. In order to evaluate the feasibility of the implementation approach, a part of the RdB transformation grammar is implemented, using a JAVA development environment for graph rewriting. This implementation is shown to be both feasible and valuable in several aspects. First, the proposed approach contributes to the existing state-of-the-art on the graph-theoretic representation of shape grammars. Second, the work presented in this paper can be considered as an example of how shape grammars are implemented on a computer system, which might in the turn increase the impact of grammars on design practice. In particular, the development of a (semi)automated methodology to support mass housing refurbishment is described. Finally, the proposed approach is embedded within a commercial CAD environment to make the shape grammar formalism more accessible to students and practitioners. An elaborated discussion of the benefits and drawbacks of the proposed approach can be found in the original article *T. Strobbe, S. Eloy, P. Pauwels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. A graph-theoretic implementation of the Rabo-de-Bacalhau transformation grammar. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 2015.*

B
Classification of architectural designs

In this appendix, we describe a method for the automatic classification of architectural designs, belonging to a particular architectural corpus. This method can be used complementary to generative and grammar-based methods, by enabling the designer (1) to select a consistent corpus from which it is easier to extract rules (grammar formulation), and (2) to assess the output of a grammar (grammar evaluation). This appendix is adapted from the paper that is to be published in *T. Strobbe, F. wyffels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. Automatic architectural style classification using one-class support vector machines and graph kernels. Automation in Construction, 2015.*

B.1 Introduction

In this appendix, we address the problem of automatically classifying architectural designs, belonging to a particular architectural corpus. Given a particular architectural corpus and a new design — does this new design belong to the corpus; does it exhibit some patterns or influences of this corpus; or should it be considered as a completely different design? In the field of AI, this problem is known as the classification problem. A method for automatic classification can be used complementary to generative and grammar-based methods, especially in the context of analytic shape grammars. On the one hand, such a method could enable the designer to select a consistent corpus from which it easier to extract rules (grammar formulation). Even a consistent design corpus may contain designs that are unexpected, making rule formulation a lot harder. For example, in the analysis of the Malagueira corpus that led to the formulation of the Malagueira shape grammar, Duarte [2005b] identified several housing designs that deviate from the other designs. On the other hand, such a method could enable the designer to assess the output of the grammar, and determine, in a quantitative way, whether the generative rules extracted are working (grammar evaluation). In other words, it could be possible to determine to what extent the designs generated fit the corpus from which the grammar was extracted.

For this classification task, we apply a technique from Machine Learning, which is a scientific domain that deals with algorithms that can learn from data. In particular, one-class Support Vector Machines (SVM) [Cortes and Vapnik, 1995; Schölkopf et al., 2000; Vapnik, 1995] are applied for automatically identifying an architectural corpus, after which new designs can be classified, either as similar of different to this corpus. One-class SVMs are learning models with associated algorithms that recognize patterns from a set of coherent training examples. They are commonly used for similarity detection, which involves deciding whether a new example is similar to the training examples (it is an *inlier*, or in the same architectural corpus), or should be considered as different (it is an *outlier*, or outside a particular corpus). In contrast to conventional classification algorithms, one-class SVMs do not need training examples of all classes, making them a suitable way to perform the classification task, because collecting a general set of other design corpora would be time-consuming, or even impossible. In the context of this research, designs are represented using a representation method commonly used by architectural designers — the *two-dimensional floor plan*. In fact, a graphical representation of floor plans is used that describes both morphological (geometrical), topological, and functional features of designs. With the training examples represented as floor plan graphs, the one-class SVM can be combined with graph kernels [Vishwanathan et al., 2010] to efficiently perform the classification task. The use of SVMs combined with graph kernels is popular in a wide variety of biological and other applications [Noble, 2006] but, to the extent of our knowledge, has not been applied in the field of architectural design before. In general, little research has been carried out in the field of automatic architectural design identification and classification [Mathias et al., 2012; Romer and Plumer, 2010].

The remainder of this appendix is structured as follows. First, we describe how architectural designs, or two-dimensional floor plans, can be converted to (labeled) graphs. In particular, the corpus of Malagueira houses, designed by the architect Álvaro Siza Viera, is used as a case study to illustrate the concepts discussed in this appendix. The next section describes some of the concepts of one-class SVMs and graph kernels. Subsequently, we train and optimize the one-class SVM for the given Malagueira dataset, using four-fold cross-validation, while also holding out part of the available data as a test set to avoid overfitting. In the final section, we apply the trained and optimized model to the test set, in order to evaluate the performance in classifying new designs, which are not in training dataset.

B.2 Design features of two-dimensional floor plans

The design process of architectural floor plans involves different kinds of data; including the generation of sketches, CAD drawings, solid models, and written reports. These documents are typically generated using different media and conventions, and as a result, they rely on different information representations. In order to apply Machine Learning techniques, such as one-class SVMs, the dataset of examples must be described using a unique representation that reflects the design features that we attempt to learn. In the case of architectural floor plans, two important design features are (1) the function of individual spaces and (2) the topological relationships between these spaces. Geometric design features (such as size, proportion, etc.) are typically secondary to the functional and topological design features. Especially in early sketch design phases, designers tend to develop several spatial configurations, without having specific geometric realizations in mind. One possible representation that reflects geometrical, topological and functional design features are graphs, which contain nodes and edges to represent objects and relations, respectively. In doing so, spaces in floor plans can be represented by graph nodes, and the relations between these spaces can be represented by graph edges. Also, each graph node can be associated with a specific label that represents the function of the space. Geometrical features can also be added as attributes of the nodes or edges, including coordinate geometry, angular constraints, etc. By using graphs to represent architectural floor plans, the task of classifying designs of floor plans can now be considered a classifying a set of labeled graphs. This can be performed using one-class SVMs combined with graph kernels, which are discussed further in this paper.

We demonstrate the feasibility of the proposed classification method on a specific case study — the Malagueira houses designed by the architect Álvaro Siza Viera. The work of Duarte [2001] describes the corpus of houses that were developed in the Malagueira plan. The corpus consists of several houses, each of which is adapted to a given site and user constraints. In an earlier research effort, Duarte [2005a] has manually captured, or encoded, this corpus in the form of a shape grammar, which contains an elaborated set of 162 rules. Using such a grammar, it is possible to classify a new design as being or not being in the Malagueira corpus, by determining whether or not this new design is in the 'language' of the shape grammar. The approach proposed in this appendix involves an automatic way of classifying designs as being or not being in the Malagueira corpus. In order to do this, the Malagueira floor plans are first translated to labeled graphs, by representing the functional spaces in the floor plans as graph nodes, and by connecting the appropriate nodes with graph edges. Figure B.1 shows the visual representation (left) and the corresponding graph representation (right) of one of the floor plans in the corpus. The conversion from floor plan images to graphs is done in an automatic manner, using the method proposed in the work of de las Heras et al. [2014]. After this automatic conversion has been completed, several smaller rooms are merged into larger functional spaces; including living spaces 'li', sleeping spaces

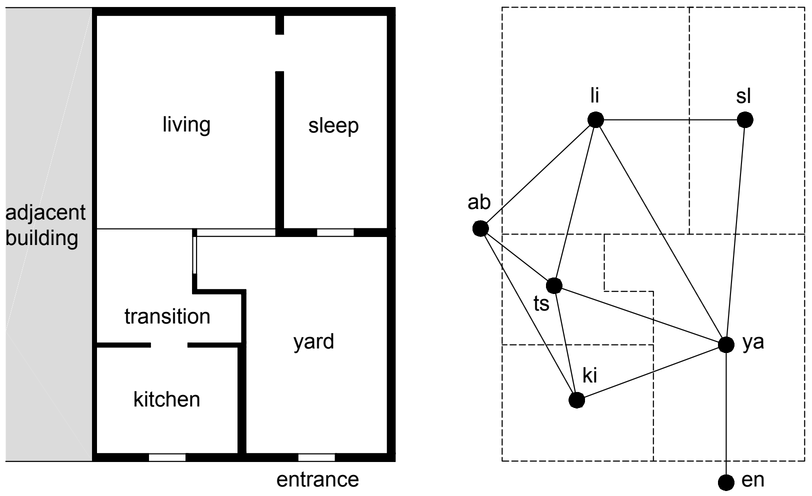


Figure B.1: A visual representation (left) and graph representation (right) of one floor plan design in the Malagueira corpus.

'sl', kitchen spaces 'ki', transition or laundry spaces 'ts', and yard 'ya'. The grouping together of rooms in functional zones is also proposed in the work of Duarte [2005a]. Second, additional entrance nodes 'en' and adjacent building nodes 'ab' are added to the graph representation to include information about the location of the entrance and the adjacency of other neighboring buildings or houses, respectively. As a result, the context of the house is also taken into account during the training process.

The resulting Malagueira dataset generated contains 128 labeled graphs with an average of 7 nodes and 1.56 edges per node. Additionally, an equally large dataset of 128 outlier floor plans has been generated. This dataset of outlier floor plans has been generated in a procedural way, by randomly creating functional zones and adding topological relations between them. As a result, this outlier dataset contains 128 randomly generated floor plans that are not in the Malagueira corpus. This random dataset serves to evaluate the accuracy of the trained SVM in correctly classifying outliers.

B.3 One-class SVMs with graph kernels

The one-class support vector machine

A one-class SVM [Cortes and Vapnik, 1995; Schölkopf et al., 2000; Vapnik, 1995] is a classification technique that seeks to distinguish one specific class (the inliers) from a broad set of classes (the outliers). In contrast to standard classification techniques, a one-class SVM is trained with examples of the target class. This makes one-class SVMs particularly useful when examples of the other classes are hard to obtain. More specifically, one-class SVMs construct a frontier, which separates the inliers from the outliers in a certain given feature space, thereby delimiting a sub-area of this feature space. If new examples lie within this frontier, they are considered to be similar to the training examples. Otherwise, these examples are classified as outliers. This is visually demonstrated in Figure B.2, in which the frontier is shown as an ellipse that geometrically separates the inliers from the outliers in a two-dimensional feature space. Typical applications are anomaly detection; for example, to detect atypical genes [Metzler and Kalinina, 2014], and network intrusion detection [Li et al., 2003]. In our case, the goal is to 'learn' an architectural corpus from a single set of coherent designs. Since it is too time consuming and even impossible to collect a sufficiently rich design set of all other styles, one-class SVMs are preferred over the more common multi-class SVMs.

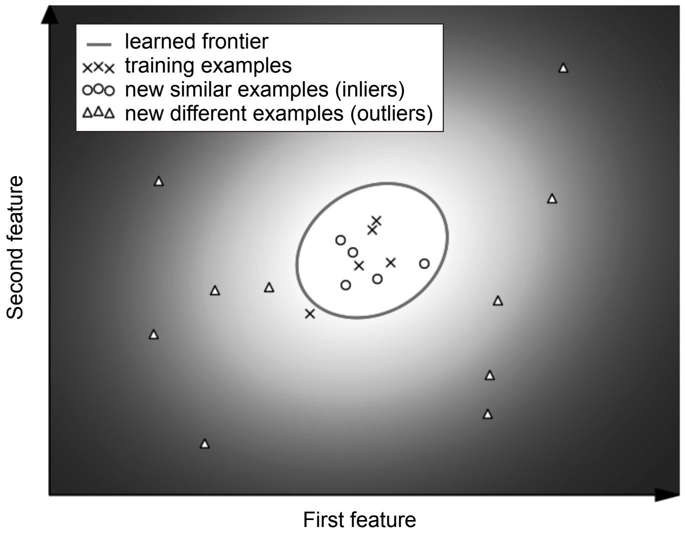


Figure B.2: The frontier separates the inliers from the outliers in a two-dimensional feature space, and delimits a sub-area of this feature space. The learned frontier does not classify all training examples correctly, because of the regularization parameter ν.

In many cases, the examples to discriminate can not be linearly separated in the given feature space. In the example given in Figure B.2, the inliers can not be separated from the outliers using a linear hyperplane in the given feature space. Therefore, the initial finite-dimensional feature space $\mathbb{R}^p$ is mapped into a (much) higher-dimensional feature $\mathbb{R}^n$ (where $n > p$), using a function Φ. This makes the separation presumably easier in that space. Figure B.3 shows the mapping of the previous two-dimensional feature space to a three-dimensional space, in which the examples become linearly separable with a hyperplane. This hyperplane corresponds to the non-linear hypersurface in the initial lower-dimensional feature space; for example, the ellipse in Figure B.2. As a result, one-class SVMs can also perform classification on examples that are non-linearly separable.

For many classification problems, it is not possible to fully separate the inliers from the outliers, using a hard boundary. Therefore, a 'soft margin' is defined in order to allow some classification errors, without affecting the final results. In the case of one-class SVMs, Schölkopf et al. [2000] introduced a user-specified parameter ν ($0 \leq \nu \leq 1$), which is the lower and upper bound on the number of examples that are support vectors — input vectors that just touch the boundary of the margin, and that lie on the

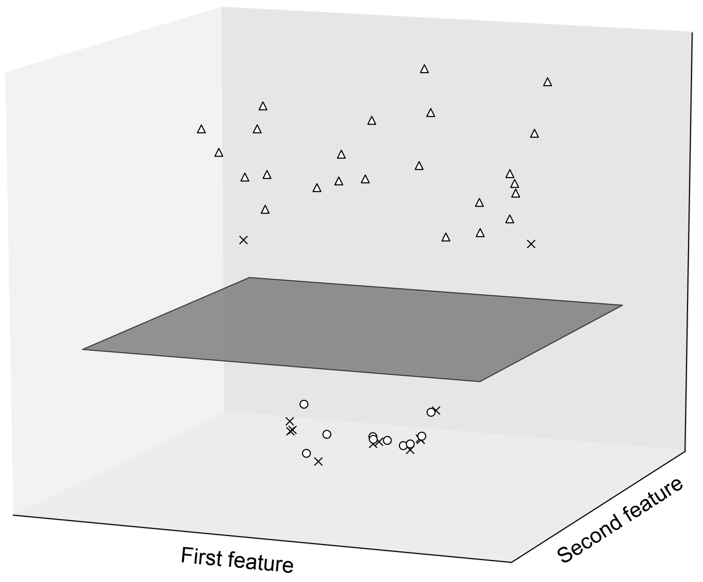


Figure B.3: The previous two-dimensional feature space is mapped to a three-dimensional feature space, in which the examples become linearly separable with a hyperplane.

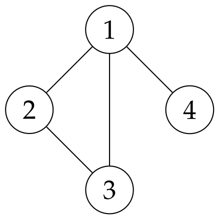
wrong side of the hyperplane, respectively [Chen et al., 2005]. In layman terms, a number of classification errors of the training examples are deliberately allowed during the training process, in order to reduce the amount of outliers that are falsely classified as inliers (*false positives*). Consequently, the amount of *false negatives* (i.e. inliers that are classified as outliers) will increase. Hence, the value of the ν parameter must be chosen carefully, in order to find a good trade-off between the amount of false positives and false negatives. In other words, the ν regularization parameter is used to avoid 'overfitting' of the model, which means that the trained model would perform very well on classifying the training examples, but would fail to correctly classify unseen data. An overfitted model specializes on particular training examples, but is less able to generalize to unseen examples.

The graph kernel

In many cases, the explicit mapping of low-dimensional feature spaces into higher-dimensional feature spaces is computationally expensive. In order to overcome this, kernel functions are used that implicitly map the original feature space to a higher-dimensional feature space. By doing so, it is possible to separate the inliers from outliers based on some measure of similarity that is introduced by the kernel function. In other words, a kernel function $k(\mathbf{x}, \mathbf{x}')$ can be considered as a similarity measure between $\mathbf{x}$ and $\mathbf{x}'$. The benefit of using kernel functions is that an explicit mapping Φ of the initial feature space to a higher-dimensional feature can be omitted, because the similarity measure may be computed easily in terms of the initial feature space. This is known as the *kernel trick*, which is a powerful feature of SVMs in classifying non-linearly separable data.

The Malagueira dataset is represented as a collection of labeled graphs, which can in turn be represented as a set of adjacency matrices (as shown further). The challenge is thus to find a computationally efficient kernel function that captures the topological and functional information that is implicitly represented in the adjacency matrix. Graph kernels, as initiated by Kondor and Lafferty [2002], are able to do this. A recent overview of graph kernels is given in the work of Vishwanathan et al. [2010]. The use of graph kernels to study relationships between graphs has already been applied in several domains, including shape classification [Dupé and Brun, 2009], bio-informatics [Mahé et al., 2005; Ralaivola et al., 2005], and social networks [Kumar et al., 2010]. A common approach is the use of random walk graph kernels [Kashima et al., 2003]. In order to calculate the similarity between two graphs, a random walk graph kernel performs random walks of different length on both graphs, and counts the number of matching walks. In other words, the number of matching walks gives an indication of the similarity between the two given graphs. Random walks of different length are performed in order to take into account both local similarities (short random walks) and more complex similarities (long random walks). In order to calculate the random walk graph kernel, the following steps are needed;

First, the graphs are represented using an adjacency matrix. Given two undirected graphs G and G' of respectively m and n nodes, the adjacency matrices $\mathbf{A} \in \mathbb{N}^{m\times m}$ and $\mathbf{A}' \in \mathbb{N}^{n\times n}$ are symmetric matrices in which the diagonal entries are zero and the non-diagonal entries A_{ij} are the number of edges from node i to node j. For example, the adjacency matrix $\mathbf{A}$ of the following undirected graph is:



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

The second step consists of calculating the direct product graph of the two given graphs. The adjacency matrix $\mathbf{A}_\times \in \mathbb{N}^{mn\times mn}$ of the direct product graph $G_\times$ is calculated as the Kronecker product of $\mathbf{A}$ and $\mathbf{A}'$:

$$\mathbf{A}_\times = \mathbf{A} \otimes \mathbf{A}' = \begin{bmatrix} A_{11}\mathbf{A}' & A_{12}\mathbf{A}' & \cdots & A_{1m}\mathbf{A}' \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1}\mathbf{A}' & A_{12}\mathbf{A}' & \cdots & A_{mm}\mathbf{A}' \end{bmatrix}.$$

The corresponding graph $G_\times$ is a graph over pairs of nodes from graphs G and G', in which two nodes in $G_\times$ are neighbors if and only if the corresponding nodes in G and G' are both neighbors. As a result, performing a random walk on the direct product graph $G_\times$ is equivalent to performing a simultaneous random walk on both graphs G and G' [Vishwanathan et al., 2010]. In our case, the graph nodes in G and G' are associated with labels from a finite set $\{1, 2, ..., d\}$. These labels correspond to the different functional spaces of the architectural designs that can be assigned to the graph nodes. Therefore, the adjacency matrix $\mathbf{A}_\times$ should have a nonzero entry if and only if an edge exist in the direct product graph, and the corresponding nodes in G and G' have the same label. In this case, we use the filtered adjacency matrix of the graphs ${}^{l}\mathbf{A}$ to calculate $\mathbf{A}_\times$:

$$\mathbf{A}_\times = \sum_{l=1}^{d} {}^{l}\mathbf{A} \otimes {}^{l}\mathbf{A}'.$$

The third step consists of calculating the random walk graph kernel $k_\times(G, G')$ directly from the direct product graph, according to the following equation (for the complete derivation, we refer to [Vishwanathan et al., 2010]):

$$k_\times(G, G') = \textstyle\sum_{i,j} (\mathbf{I} - \lambda \mathbf{A}_\times)^{-1},$$

in which, $\mathbf{I} \in \mathbb{N}^{mn\times mn}$ is an identity matrix, and λ is a decay parameter that ensures the convergence of the equation. The value of the decay parameter λ has to be chosen carefully in order to obtain a good performance of the graph kernel. In practice, a very low value for λ is often chosen, but this makes the contribution of long random walks negligible. As a result, the impact of local similarities between graphs (or short random walks) would be larger than the impact of complex similarities between graphs (or long random walks). The kernel $k_\times(G, G')$ can be calculated efficiently in $O(n^3)$ time using several optimizations discussed in the paper of Vishwanathan et al. [2010].

B.4 Hyperparameter optimization

The test set and validation set

Training one-class SVMs on a set of examples, and testing the performance on the same set of examples leads, in many cases, to false conclusions. For particular hyper-parameter values, the trained model would be able to perfectly classify the training examples, but would fail to classify yet-unseen data. It is important that the classifier should be able to classify unseen designs correctly. In order to optimize the hyper-parameters correctly, a subset of the available examples is withdrawn from the training procedure and used as a test set to evaluate the performance of the trained model afterwards. In this experiment, we randomly hold out 28 of 128 valid examples for future performance evaluation. Also, an additional 28 outliers are withheld from the outlier dataset to evaluate outlier accuracy. As a result, the test set contains 56 examples, half of which are Malagueira houses and the other half are outliers.

Moreover, overfitting can occur during the optimization of the regularization parameter ν and the decay parameter λ (see below). If examples from the test set are used to evaluate the performance of the parameters, the trained model would be optimized to classify the test examples instead of new unobserved examples. In order to overcome this problem, another part of the remaining dataset should be held out as a so-called validation set. In this case, training is performed on the training set, after which evaluation is done on the validation set, and final evaluation can be done on the test set. However, partitioning the dataset of 128 valid examples into three sets would seriously reduce the size of the training set. For this reason, we apply a well-known technique called 'k-fold cross-validation', by which a validation set is no longer needed [Stone, 1974]. In general, this technique involves splitting the training examples into k smaller sets. For each of the folds, the model is trained using $(k - 1)$ of the smaller sets, and validated on the remaining part of the examples. In this experiment, four-fold cross-validation is used — the model is consequently trained and validated four times on 75 and 25 examples of the training set, respectively. Also, the model is each time validated on a set of 25 outlier examples, in order to prevent the model in classifying all examples to be inliers.

Hyper-parameter optimization through grid-search

In order to evaluate the performance of the trained model, the average accuracy of the four folds is calculated. Accuracy is a well-known performance measure in Machine Learning that indicates the rate of correctly classified examples. For one-class SVMs, we distinguish between validation accuracy, outlier accuracy and total accuracy. The validation accuracy indicates the rate of validation examples that are correctly classified as similar to the training examples (true positive). The outlier accuracy indicates the rate of outlier examples that are correctly classified as different from the training examples (true negative). The total accuracy is defined as the average of the validation and outlier accuracy, and gives a good indication of the performance of the trained model.

As mentioned earlier in this appendix, the values for the regularization parameter ν and the decay parameter λ must be optimized in order to obtain a good performance of the trained model. These parameters are often referred to as hyper-parameters, because they are not optimized during the learning procedure of one-class SVMs, but have to be optimized beforehand. In order to do this, a two-dimensional parameter space is defined that consists of the two hyper-parameters ν and λ. For each value in this parameter space, a one-class SVM is trained using the specified hyper-parameters. The validation accuracy is measured on the validation set (using four-fold cross-validation) and the outlier accuracy is measured on the set of random outliers. In particular, we define a logarithmic x-scale, ranging from 10^{-5} to 1, that specifies several values for ν. Also, a logarithmic y-scale specifies several values for λ, ranging from 10^{-5} to 1. This technique is known as grid-search optimization. Figure B.4 shows the total accuracy, which is the average of the validation and outlier accuracy, of the grid-search that has been performed. An optimal accuracy of 84.5% is found for the hyperparameter values indicated with '$\times$' in the figure.

B.5 Evaluation of the trained SVM model

In order to evaluate the generalization performance of the trained SVM model, it is applied to the test set that was withdrawn from the training procedure. As a result, 49 out of 56 houses are correctly recognized either as Malagueira designs or outliers, corresponding to a total accuracy of 87.5%. These findings are in the same line of the total accuracy found on the validation sets during the optimization (84.5%) — thereby demonstrating that the trained model is able to generalize well to new, unobserved examples. Figure B.5 shows four examples from the Malagueira test set. The top left floor plan is a housing design from the Malagueira corpus, which is not recognized as an inlier (false negative). One of the reasons that the left floor plan is not recognized as a Malagueira design may be that this specific spatial composition is less frequent in the training examples. Indeed, houses in the Malagueira corpus typically have a sleeping

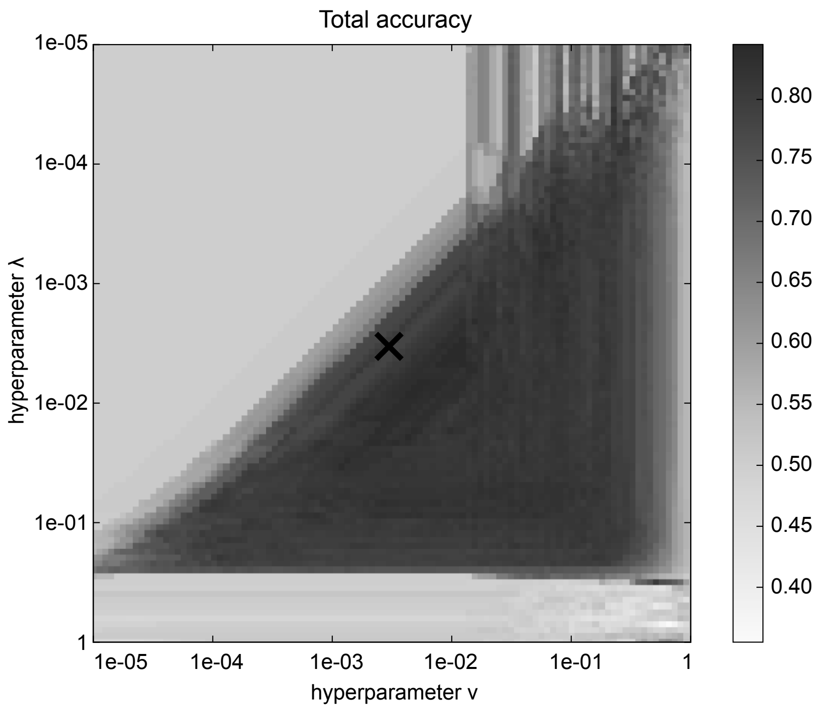


Figure B.4: Two-dimensional parameter space that indicates the total accuracy for all the hyper-parameter values. The best total accuracy (0.845) is achieved for the combination of ν and λ indicated with '$\times$'.

space on the ground level, which is not the case in this example. The top right floor plan is a housing design from the Malagueira corpus, which is correctly classified as an inlier (true positive). The bottom left floor plan is a randomly generated floor plan, but is recognized as an inlier by the trained SVM model (false positive). While this floor plan shows several patterns that are atypical for Malagueira houses, it also shows several patterns that are similar to housing designs in the Malagueira corpus, such as the relationship between the kitchen and transition space. The bottom right floor plan is a randomly generated floor plan, which is correctly classified as an outlier (true negative). Indeed, this floor plan shows a linear spatial composition that is both infeasible and atypical for the corpus.

The examples given in Figure B.5 indicate that trained one-class SVM is able to generalize well to new unobserved floor plans, and is able to distinguish between floor plans that are either very similar or different from the training examples. However, some floor plans are more difficult to classify,

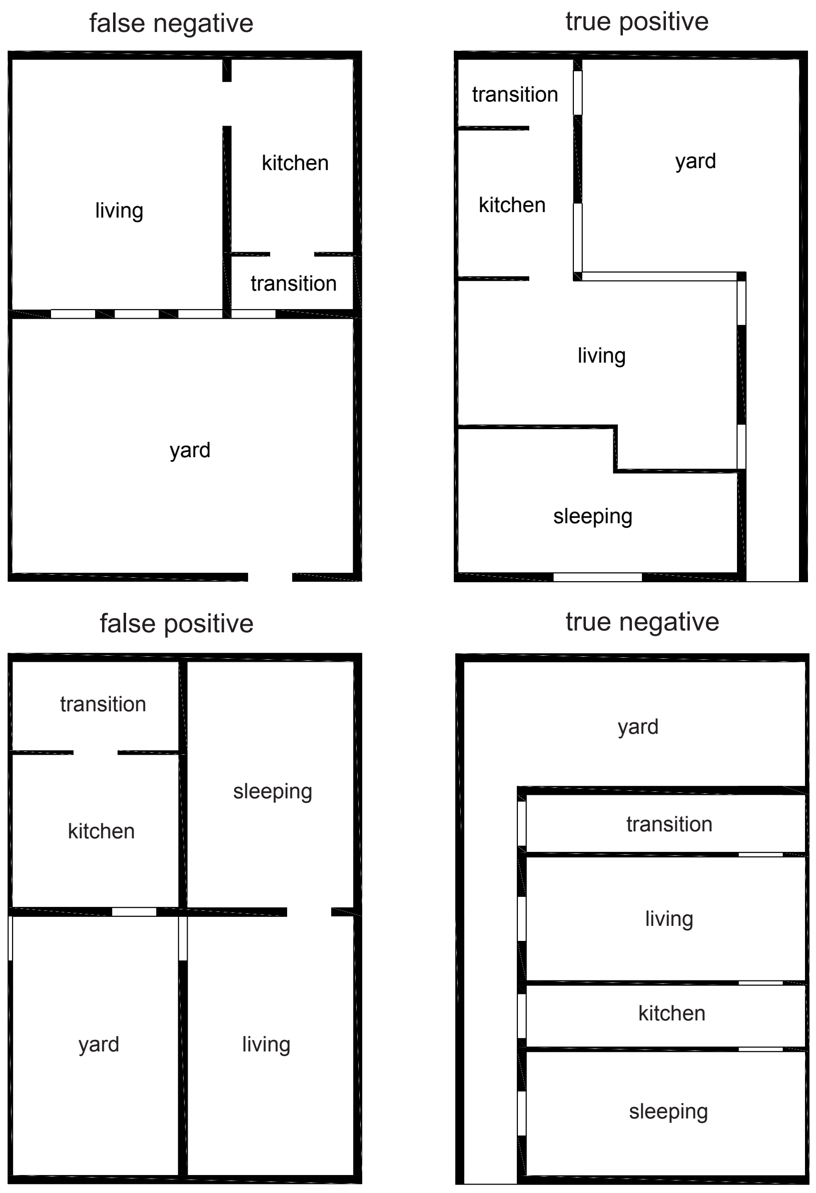


Figure B.5: Four floor plans from the test set that were withdrawn from the training procedure. The two top floor plans are examples from the Malagueira corpus, while the bottom floor plans are randomly generated examples.

because they have some design patterns in common, while other patterns are different. A possible solution to overcome this limitation, is by using a scoring parameter; in this case, the distance of the examples from the separating frontier. In the case of SVMs, Platt scaling [Platt, 1999] is often used to transform the outputs of the SVM into a probability distribution. Unlike other kinds of supervised models, SVMs do not directly provide such probabilities, but they are calculated using cross-validation. This probability indicates a certain confidence level that a given floor plan is either an inlier or an outlier. The higher or lower this scoring parameter, the more likely a given floor plan is an inlier or an outlier, respectively. As a result, the scoring parameter, using Platt scaling, gives a good indication of how similar new examples are to the training examples.

B.6 Conclusion

In this appendix, we investigated whether or not it is possible to 'learn' an architectural corpus from a set of examples, and classify new designs as similar or different from these examples. We proposed the use of one-class support vector machines with graph kernels, and demonstrated the feasibility on the case study of Malagueira houses. After the optimization of the hyper-parameters, a best total accuracy of 84.5% was achieved. With an accuracy of 87.5% on the test set, we illustrated that our model is able to generalize to designs that have not yet been observed during training. Finally, we used a scoring parameter to calculate the confidence level that a given floor plan is either an inlier or an outlier. An elaborated discussion of the proposed approach and future lines of research can be found in the original article *T. Strobbe, F. wyffels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. Automatic architectural style classification using one-class support vector machines and graph kernels. Automation in Construction, 2015.*

References

A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7:39–52, 1994.

M. Agarwal and J. Cagan. A blend of different tastes: The language of coffee makers. *Environment and Planning B*, 25(2):205–226, 1996.

R. Aish and R. Woodbury. Multi-level interaction in parametric design. In A. Butz, B. Fisher, A. Krüger, and P. Olivier, editors, *Smart Graphics. Lecture Nodes in Computer Science*, volume 3638, pages 151–162, 2005.

Y. Aksoy, G. Çagdas, and O. Balaban. A model for sustainable site layout design of social housing with pareto genetic algorithm: Sspm. In *Proceedings of the 16th International Conference on Computer Aided Architectural Design (CAADFutures)*, 2015.

C. Alexander. *Notes on the synthesis of form*. Harvard University Press, 1964.

C. Alexander. The state of the art in design methods. *DMG Newsletter*, 5 (3):3–7, 1971.

C. Alexander. *A pattern language: towns, buildings, construction*. Oxford University Press, 1977.

C. Alexander and M. Manheim. HIDECS 2: computer program for the hierarchical decomposition of a set with an associated graph. Technical report, MIT Civil Engineering Systems Laboratory, 1962.

J.E. Allen, I.G. Curry, and E. Horvtz. Mixed-initiative interaction. *Intelligent systems and their applications, IEEE*, 14(5):14–23, 1999.

N. Bayazit. Investigating design: A review of forty years of design research. *Design Issues*, 20:16–29, 2004.

W. Bekers, R. De Meyer, and T. Strobbe. World war I naval camouflage : an evaluation through image analysis. In *Intellectuals and the Great War*, 2014.

C. Bleil de Souza. Contrasting paradigms of design thinking: the building thermal simulation tool user vs. the building designer. *Automation in Construction*, 22:112–122, 2012.

M. Borges and R. Fakury. Structural design based on performance applied to development of a lattice wind tower. In *Proceedings of the 16th International Conference on Computer Aided Architectural Design (CAADFutures)*, 2015.

J. Cagan and W. J. Mitchell. Optimally directed shape generation by shape annealing. *Environment and Planning B*, 20(1):5–12, 1993.

G. Çagdas. A shape grammar: the language of traditional turkish houses. *Environment and Planning B*, 23(5):443–464, 1996.

A. Chakrabarti, K. Shea, R. Stone, J. Cagan, M. Campbell, N.V. Hernandez, and K.L. Wood. Computer-based design synthesis research: An overview. *Journal of Computing and Information Sciences in Engineering*, 11 (2):021003, 2011.

S. Chase. A model for user interaction in grammar-based design systems. *Automation in Construction*, 11:161–172, 2002.

S. Chase. Shape grammar implementation the last 35 years. Technical report, Design Computing and Cognition workshop, 2010.

H. H. Chau, X. Chen, A. McKay, and A. de Pennington. Evaluation of a 3D shape grammar implementation. In *Design Computing and Cognition 04*, pages 357–376, 2004.

P. H. Chen, C. J. Lin, and B. Schölkopf. A tutorial on ν-support vector machines. *Applied Stochastic Models in Business and Industry*, 21(2):111–136, 2005.

N. Chomsky. *Syntactic Structures*. Mouton, The Hague/Paris, 1957.

C.W. Churchman, R. L. Ackoff, and E. L. Arnoff. *Introduction to operations research*. Wiley, Oxford, England, 1957.

J. Conklin. *Dialogue mapping : building shared understanding of wicked problems*. Wiley Publishing, 2006.

R. Correia, Duarte J.P., and A. Leitao. Malag: a discursive grammar interpreter for the online generation of mass customized housing. In *4th International Conference on Design Computing and Cognition*, 2010.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20: 273, 1995.

F. Coutinho, J. P. Duarte, and M. Kruger. Constructing a shape grammar, the Ducal palace façade. In *Proceedings of the 32nd eCAADe Conference*, 2014.

N. Cross. Descriptive models of creative design: application to an example. *Design Studies*, 18:427–455, 1997.

D. Davis. *Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture*. PhD thesis, School of Architecture and Design, RMIT University, 2013.

X. De Kestelier. Recent developments at Foster + Partners' specialist modelling group. *Architectural Design*, 83(2):22–27, 2013.

L. de las Heras, S. Ahmed, M. Liwicki, E. Valveny, and Sanchez G. Statistical segmentation and structural recognition for floor plan interpretation. *International Journal on Document Analysis and Recognition*, 17(3):221–237, 2014.

M. Delghust, T. Strobbe, R. De Meyer, and A. Janssens. Using BIM-based parametric typologies to supplement single-zone calculations for official performance assessment with multi-zone calculations for predictions on real energy use. In *14th International Conference of the International Building Performance Simulation Association*, 2015.

H. Dreyfus. *What ccomputer can't do*. MIT Press, 1972.

J. P. Duarte. *Customizing mass housing: a discursive grammar for Siza's Malagueira houses*. PhD thesis, Massachusetts Institute of Technology, 2001.

J. P. Duarte. A discursive grammar for customizing mass housing: the case of Siza's houses at Malagueira. *Automation in Construction*, 14(2):265–275, 2005a.

J. P. Duarte. Towards the mass customization of housing: the grammar of Siza's houses at Malagueira. *Environment and Planning B*, 32:347–380, 2005b.

F. X. Dupé and L. Brun. Tree covering within a graph kernel framework for shape classification. In *Image Analysis and Processing–ICIAP 2009*, pages 278–287. Springer, 2009.

D.S. Dye. *A Grammar of Chinese Lattice*. Harvard University Press, 1937.

E. C. e Costa and J. P. Duarte. Tableware shape grammar: Towards mass customization of ceramic tableware. In John S. Gero, editor, *Design Computing and Cognition 14*, 2014.

A. Economou and S. Kotsopoulos. From shape rules to rule schemata and back. In *Design Computing and Cognition 14*, 2014.

H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.

S. Eloy. *A transformation grammar-based methodology for housing rehabilitation: meeting contemporary functional and ICT requirements*. PhD thesis, TU Lisbon, 2012.

S. Eloy and J. P. Duarte. Inferring a shape grammar: translating designers knowledge. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing*, 28:153–168, 2014.

C. Ertelt and K. Shea. Shape grammar implementation for machine planning. In *4th International Conference on Design Computing and Cognition*, 2010.

J. Fenn and M. Raskino. Understanding Gartner's hype cycles. Technical report, Gartner, 2013.

P. Fitzhorn. Formal graph languages of shape. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 4(3):151–163, 1990.

A. Fleisher. Grammatical architecture? *Environment and Planning B*, 19(2): 221–226, 1992.

U. Flemming. More than the sum of parts: the grammar of Queen Anne houses. *Environment and Planning B*, 14(3):323–350, 1987.

U. Flemming. Get with the program: common fallacies in critiques of computer-aided architectural design. *Environment and Planning B*, 21(7): 106–116, 1994.

R. Geiß and M. Kroll. On improvements of the varró benchmark for graph transformation tools. Technical report, Universität Karlsruhe, 2007.

R. Geiß, G. V. Batz, D. Grund, S. Hack, and A. Szalkowski. GrGen: A fast spo-based graph rewriting tool. In *Graph Transformations, Third International Conference, ICGT*, volume 4178, pages 383–397. Springer Berlin Heidelberg, 2006.

J. S. Gero. Towards a model of exploration in computer-aided-design. In Gero and E.Tyugu, editors, *Formal Design Methods for CAD*, pages 315–336, North-Holland, 1994.

J. S. Gero and V. A. Kazakov. An exploration-based evolutionairy model of a generative design process. *Computer-Aided Civil and Infrastructure Engineering*, 11:211–218, 1996.

M. Gerzso. On the limitations of shape grammars: comments on aaron fleisher's article "grammatical architecture?". In *Annual Conference of the Association for Computer Aided Design In Architecture*, 2003.

P. Geyer. Multidisciplinary grammars supporting design optimization of buildings. *Research in Engineering Design*, 18(4):197–216, 2008.

J. Gips. Computer implementation of shape grammars. In *NSF/MIT Workshop on Shape Computation*, volume 55, 1999.

V. Goel and P. Pirolli. The structure of design problem spaces. *Cognitive Science*, 16:395–429, 1992.

G. Goldschmidt. The dialectics of sketching. *Creativity Research Journal*, 4 (2):123–143, 1991.

G. Goldschmidt. The backtalk of self-generated sketches. *Design Issues*, 19 (1):72–88, 2003.

G. Goldschmidt. Quo vadis, design space explorer? *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 20:105–111, 2006.

G. Goldschmidt. *Linkography - Unfolding the design process*. The MIT Press, 2014.

G. Goldschmidt and D. Tatsa. How good are good ideas? correlates of design creativity. *Design studies*, 26(6):593–611, 2005.

V. Granadeiro, J. Duarte, R. Correia, and M.S L. Vitor. Building envelope shape design in early stages of the design process: Integrating architectural design systems and energy simulation. *Automation in Construction*, 32:196–209, 2013.

T. Grasl. Transformational palladians. *Environment and Planning B*, 39(1): 83–95, 2012.

T. Grasl. *On Shapes and Topologies: Graph theoretic representations of shapes and shape computations.* PhD thesis, TU Vienna, 2013.

T. Grasl and A. Economou. From topologies to shapes: parametric shape grammars implemented by graphs. *Environment and Planning B*, 40(5): 905–922, 2013.

T. Grasl and A. Economou. Towards controlled grammars, approaches to automating rule selection for shape grammars. In *Proceedings of the 32nd eCAADe Conference*, 2014.

N. L. R. Hanson and A. Radford. On modelling the work of the architect Glenn Murcutt. *Design Computing*, 1(3):189–203, 1986.

J. Haugeland. *Artificial Intelligence: The Very Idea*. MIT Press, 1985.

J. Heisserman. *Generative geometric design and boundary solid grammars*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1991.

J. Heisserman. Generative geometric design. *IEEE Computer Graphics and Applications*, 14(2):37–45, 1994.

B. Hillier and J. Hanson. *The Social Logic of Space*. Cambridge University Press, 1984.

F. Hoisl and K. Shea. An interactive, visual approach to developing and applying parametric 3-d spatial grammars. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, pages 333–356, 2011.

J. H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, 1992.

J. H. Holland. *Emergence: From Chaos to Order*. Addison-Wesley, 1999.

J.C. Jones. How my thoughts about design methods have changed during the years. *Design methods and theories*, 11(1):48–62, 1977.

J.C. Jones and D. Thornley. *The Conference on Design Methods: papers presented at the conference on systematic and intuitive methods in engineering, industrial design, architecture and communications*. Pergamon Press, London, 1962.

I. Jowers, D.C. Hogg, A. McKay, H. H. Chau, and A. de Pennington. Shape detection with vision: implementing shape grammars in conceptual design. *Research in Engineering Design*, 21(4):235–247, 2010.

J. Jowers and C. Earl. Implementation of curved shape grammars. *Environment and Planning B: Planning and Design*, 38:616–635, 2011.

H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

H.Y. Keles, M. Özkar, and S. Tari. Embedding shapes without predefined parts. *Environment and Planning B*, 37:664–681, 2010.

S. Keller. *Systems Aesthetics: Architectural Theory at the University of Cambridge, 1960-75*. PhD thesis, Harvard University, 2005.

S. Keller. Fenland tech, architectural science in postwar Cambridge. *Grey Room*, 23:40–65, 2006.

T. Knight. The generation of Hepplewhite-style chair-back designs. *Environment and Planning B*, 7(2):227–238, 1980.

T. Knight. Languages of designs: from known to new. *Environment and Planning B*, 6:213–238, 1981.

T. Knight. Computing with emergence. *Environment and Planning B*, 30: 125–155, 2003.

R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *International Conference on Machine Learning*, pages 315–322, 2002.

C. Königseder and K. Shea. Analyzing generative design grammars. In *Design Computing and Cognition 14*, 2014.

H. Koning and J. Eizenberg. The language of the prairie: Frank Lloyd Wright's prairie houses. *Environment and Planning B*, 8(3):295–323, 1981.

R. Krishnamurti. The construction of shapes. *Environment and Planning B*, 8:5–40, 1981.

R. Krishnamurti. Explicit design space? *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 20:95–103, 2006.

R. Krishnamurti and C. Earl. Shape recognition in three dimensions. *Environment and Planning B*, 19:585–603, 1992.

R. Krishnamurti and R. Stouffs. Spatial grammars: Motivation, comparison, and new results. In U. Flemming and S. Van Wyk, editors, *CAAD Futures*, pages 57–74, 1993.

D. Krstic. Language of the rascian school: Analyzing rascian chuch plans via parallel shape grammar. In *Design Computing and Cognition 14*, 2014.

R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *Link mining: models, algorithms, and applications*, pages 337–357. Springer, 2010.

B. Lawson. Oracles, draughtsmen, and agents: the nature of knowledge and creativity in design and the role of it. *Automation in Construction*, 14: 383–391, 2005.

A. Li, H. H. Chau, L. Chen, and Wang Y. A prototype system for developing two- and three-dimensional shape grammars. In *14th Int. Conf. Computer-Aided Architectural Design Research in Asia*, 2009.

K. L. Li, H. K. Huang, S. F. Tian, and W. Xu. Improving one-class svm for anomaly detection. In *International Conference on Machine Learning and Cybernetics*, volume 5, pages 3077–3081. IEEE, 2003.

B. Logan and T. Smithers. Creativity and design as exploration. In J. S. Gero and M.L. Maher, editors, *Modelling Creativity and Knowledge Based Creative Design*, 1992.

P. Mahé, N. Ueda, T. Akutsu, J. L. Perret, and J. P. Vert. Graph kernels for molecular structure-activity relationship analysis with support vector machines. *Journal of chemical information and modeling*, 45(4):939–951, 2005.

M.L. Maher and J. Poon. Modelling design exploration as co-evolution. *Microcomputers in civil engineering*, 11:195–209, 1996.

M. Mathias, A. Martinovic, J. Weissenberg, S. Haegler, and L. Van Gool. Automatic architectural style recognition. In *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 171–176, 2012.

T. Maver. CAAD's seven deadly sins. In *Proceedings of the International Conference on Computer Aided Architectural Design (CAADFutures)*, 1995.

J. McCarthy, M. Minsky, N. Rochester, and C. Shannon. A proposal for the Dartmouth summer research project on artificial intelligence. Technical report, Dartmouth College, 1956.

A. McKay, K. Shea, S. Chase, A. Li, T. Trescak, F. Hoisl, I. Jowers, C. Ertelt, and R. Correia. Shape grammar implementation: from theory to useable software. Technical report, DCC '10 workshop, 2010.

A. McKay, S. Chase, K. Shea, and Hau Hing Chau. Spatial grammar implementation: From theory to useable software. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 26(2):143–159, 2012.

S. Metzler and O.V. Kalinina. Detection of atypical genes in virus families using a one-class svm. *BMC genomics*, 15(1):913, 2014.

W. J. Mitchell, A.S. Inouye, and M.S. Blumenthal. *Beyond productivity: information technology, innovation, and creativity*. National Academic Press, Washington, DC, USA, 2003.

R. Morash. Computer sketchpad. MIT - Lincoln Laboratory, 1964.

V. Mueller and T. Strobbe. Cloud-based design analysis and optimization framework. In R. Stouffs and S. Sariyildiz, editors, *Computation and Performance - Proceedings of the 31st eCAADe Conference*, volume 2, pages 185–194. Delft University of Technology, 2013.

P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. *Association for Computing Machinery*, pages 614–623, 2006.

N. Negroponte. *The Architecture Machine*. M.I.T. Press, 1970.

A. Newell, J. Shaw, and H. Simon. Empirical explorations with the logic theory machine. In *Western Joint Computer Conference*, volume 15, pages 218–239, 1957.

A. Newell, J. Shaw, and H. Simon. The processes of creative thinking. In *Symposium on creative thinking*. University of Colorado, 1958.

A. Newell, J. Shaw, and H. Simon. Report on a general problem-solving program. In *International Conference on Information Processing*, page 256264, 1959.

W. S. Noble. What is a support vector machine? *Nature biotechnology*, 24 (12):1565–1567, 2006.

P. Pauwels, P. Present, and T. Strobbe. A pragmatic approach towards software usage in construction projects : the port house in Antwerp, Belgium. In G. Gundason and R. Scherer, editors, *Ework And Ebusiness in Architecture, Engineering and Construction*, pages 509–512. CRC Press/Balkema, 2012.

P. Pauwels, T. Strobbe, J. Derboven, and R. De Meyer. The role of conversation and critique within the architectural design process. In *Sixth International Conference on Design Computing and Cognition*, pages 141–176, 2014a.

P. Pauwels, T. Strobbe, J. Derboven, and R. De Meyer. Analysing the impact of constraints on decision-making by architectural designers. In K. Zreik, editor, *Architecture, City & Information Design*, pages 97–111, 2014b.

P. Pauwels, T. Strobbe, S. Eloy, and R. De Meyer. Shape grammars for architectural design: The need for reframing. In *Computer-Aided Architectural Design Futures. The Next City - New Technologies and the Future of the Built Environment*, page 507526, 2015.

J. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10: 61–74, 1999.

G. Polya. *How to solve it*. Princeton University Press, 1945.

E. L. Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65(2):197–215, 1943.

A. D. Radford and J. S. Gero. On optimization in computer aided architectural design. *Building and Environment*, 15:73–80, 1980.

L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005.

H.J.W Rittel and M.M Webber. Dilemmas in a general theory of planning. *Policy Science*, 4(2):155–169, 1973.

C. Romer and L. Plumer. Identifying architectural style in 3d city models with support vector machines. *Photogrammetrie - Fernerkundung - Geoinformation*, 14:371–384, 2010.

M. Ruiz-Montiel, J. Boned, J. Gavilanes, E. Jimenez, L. Mandow, and J. L. Perez-de-la Cruz. Design with shape grammars and reinforcement learning. *Advanced Engineering Informatics*, 27(2):230245, 2013.

S. Russel and P. Norvig. *Artificial Intelligence - A modern approach*. Prentice Hall, 2010.

A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

J. Schaefer and S. Rudolph. Satellite design by design grammars. *Aerospace Science and Technology*, 9(1):81–91, 2005.

B. Schölkopf, A. Smola, R. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

D. Schön. *The Reflective Practitioner: How professionals think in action*. Temple Smith, 1983.

D. Schön. Designing: rules, types and worlds. *Design studies*, 9:181–190, 1988.

C. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(314), 1950.

K. Shea and J. Cagan. Languages and semantics of grammatical discrete structures. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 13(4):241–251, 1999.

N. Shireen, H. Erhan, D. Botla, and R. Woodbury. Parallel development of parametric design models using subjunctive dependency graphs. In *27th Annual Conference of the Association for Computer Aided Design in Architecture*, pages 57–66, 2012.

B. Shneiderman. Creating creativity: user interfaces for supporting innovation. In *ACM Transactions on Computer-Human Interaction*, volume 7, pages 114–138, 2000.

H. Simon. Rational choice and the structure of the environment. *Pshychological Review*, 63:129–138, 1956.

H. A. Simon. *The Sciences of the Artificial*. The MIT Press, Cambridge, Massachusets, 1969.

H. A. Simon. The structure of ill-structured problems. *Artificial Intelligence*, 4:181–201, 1973.

S. Skiena. *The Algortihm Design Manual*. Springer Science & Business Media, 2009.

T. Smithers, A. Conkie, J. Doheny, B. Logan, K. Millington, and M. X. Tang. Design as intelligent behaviour: an AI in design research programme. *Artificial Intelligence in Engineering*, 5(2):78–109, 1990.

P. Steadman. Graph-theoretic representation of architectural arrangement. In L. March, editor, *The architecture of form*. Cambridge university press, 1976.

M. W. Steenson. *Architectures of Information: Christopher Alexander, Cedric Price, Nicholas Negroponte & MITs Architecture Machine Group*. PhD thesis, Princeton University, 2014.

G. Stiny. Ice-ray: a note on Chinese lattice designs. *Environment and Planning B*, 4(1):89–98, 1977.

G. Stiny. Introduction to shape and shape grammars. *Environment and Planning B*, 7(3):343–351, 1980.

G. Stiny. Shape rules: closure, continuity, and emergence. *Environment and Planning B*, 21:478, 1994.

G. Stiny. *Shape - Talking about Seeing and Doing*. The MIT Press, London, England, 2006.

G. Stiny. What rule(s) should i use? *Nexus Network Journal*, 13:14–47, 2011.

G. Stiny and J. Gips. Shape grammars and the generative specification of painting and sculpture. In C. V. Freiman, editor, *Information Processing*, volume 71, pages 1460–1465. North-Holland, 1972.

G. Stiny and W. J. Mitchell. The Palladian grammar. *Environment and Planning B*, 5(1):5–18, 1978a.

G. Stiny and W. J. Mitchell. Counting Palladian plans. *Environment and Planning B*, 5(2):189–198, 1978b.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974.

R. Stouffs. Towards a formal representation for description rules. In *Proceedings of the 32nd eCAADe Conference*, 2014.

T. Strobbe and R. De Meyer. Generative systems in architectural design. In *FEA PhD Symposium*, 2013.

T. Strobbe, P. Pauwels, R. Verstraeten, and R. De Meyer. Metaheuristics in architecture : using genetic algorithms for constraint solving and evaluation. In P. Leclercq, A. Heylighen, and G. Martin, editors, *Proceedings of the 14th International Conference on Computer Aided Architectural Design (CAADFutures)*, pages 866–867, 2011a.

T. Strobbe, P. Pauwels, R. Verstraeten, and R. De Meyer. Metaheuristics in architecture. In J. Van Wittenberghe, editor, *Sustainable Construction and Design*, volume 2, pages 190–196. Ghent University, Laboratory Soete, 2011b.

T. Strobbe, P. Pauwels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. Optimization in compliance checking using heuristics : Flemish energy performance regulations (EPR). In G. Gundason and R. Scherer, editors, *Ework And Ebusiness in Architecture, Engineering and Construction*, pages 477–482. CRC Press/Balkema, 2012.

T. Strobbe, R. De Meyer, and J. Van Campenhout. A generative approach towards performance-based design : using a shape grammar implementation. In R. Stouffs and S. Sariyildiz, editors, *Computation and Performance - Proceedings of the 31st eCAADe Conference*, volume 2, pages 627–633. Delft University of Technology, 2013.

T. Strobbe, P. Pauwels, R. De Meyer, and J. Van Campenhout. Design space exploration using a shape grammar implementation. In *Sixth International Conference on Design Computing and Cognition*, pages 79–80, 2014.

T. Strobbe, R. De Meyer, and J. Van Campenhout. A semi-automatic approach for the definition of shape grammar rules. In *Real time - Proceedings of the 33rd eCAADe Conference*, 2015a.

T. Strobbe, S. Eloy, P. Pauwels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. A graph-theoretic implementation of the Rabo-de-Bacalhau transformation grammar. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 2015b.

T. Strobbe, P. Pauwels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. Towards a visual approach in the exploration of shape grammars. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 2015c.

T. Strobbe, R. Verstraeten, M. Delghust, J. Laverge, R. De Meyer, and A. Janssens. Using a building information modelling approach for teaching about residential energy use and official energy performance. In *14th International Conference of the International Building Performance Simulation Association*, 2015d.

T. Strobbe, F. wyffels, R. Verstraeten, R. De Meyer, and J. Van Campenhout. Automatic architectural style classification using one-class support vector machines and graph kernels. *Automation in Construction*, 2015e.

I. E. Sutherland. *Sketchpad, A man-machine graphical communication system*. PhD thesis, Massachusetts Institute of Technology, 1963.

I. E. Sutherland. Structure in drawings and the hidden-surface problem. In N. Negroponte, editor, *Reflections on Computer Aids To Design and Architecture*, New York, 1975. Petrocelli/Charter.

G. Taentzer. AGG: A graph transformation environment for modeling and validation of software. In J.L. Pfaltz, M. Nagl, and B. Böhlen, editors, *AGTIVE*, pages 446–453. Springer-Verlag Berlin Heidelberg, 2004.

J. Talton, L. Yang, R. Kumar, M. Lim, N. Goodman, and R. Mech. Learning design patterns with bayesian grammar induction. In *25th annual ACM symposium on User interface software and technology*, pages 63–74, 2012.

M. Tapia. A visual implementation of a shape grammar system. *Environment and Planning B*, 26:59–73, 1999.

O. Teboul, I. Kokkinos, L. Simon, P. Koutsourakis, and N. Paragios. Shape grammar parsing via reinforcement learning. In *Computer Vision and Pattern Recognition*, 2011.

T. Tidafi, N. Charbonneau, and S. K. Araghi. Backtracking decisions within a design process: a way of enhancing the designers thought process and creativity. In Ann Heylighen Pierre Leclercq and Genevive Martin, editors, *14th International conference on Computer Aided Architectural Design.*, pages 573–587, 2011.

T. Trescak, M. Esteva, and I. Rodriguez. A shape grammar interpreter for rectilinear forms. *Computer-Aided Design*, 44:657–670, 2012.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

G. Varró, A. Schürr, and D. Varró. Benchmarking for graph transformation. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2005.

S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *The Journal of Machine Learning Research*, 11:1201–1242, 2010.

B. Whitehead and M.Z. Eldars. The planning of single-storey layouts. *Building Science*, 1(2):127–139, 1964.

S. Wolfram. *A new kind of science*. Wolfram Media, 2002.

R. Woodbury. *Elements of Parametric Design*. Taylor and Francis, 2010.

R. Woodbury. Design flow and tool flux. *Architectural Design Smart*, 1: 102–111, 2013.

R. Woodbury and A. Burrow. Whither design space? *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 20:63–82, 2006.

R. Woodbury, R. Aish, and A. Kilian. Some patterns for parametric modeling. In *27th Annual Conference of the Association for Computer Aided Design in Architecture*, pages 222–229, 2007.

T. A. Wortmann. Representing shapes as graphs: a feasible approach for the computer implementation of parametric visual calculating. Master's thesis, Massachusetts Institute of Technology, 2013.

K. Yue. *Computation-friendly shape grammars: With application to determining the interior layout of buildings from image data*. PhD thesis, School of Architecture, Carnegie Mellon University, 2009.

K. Yue and R. Krishnamurti. Tractable shape grammars. *Environment and Planning B*, 45:576–594, 2013.

K. Yue and R. Krishnamurti. A paradigm for interpreting tractable shape grammars. *Environment and Planning B*, 41:110–137, 2014.