Shape Rule Types and Spatial Search

Rudi Stouffs(✉)

National University of Singapore, Singapore, Singapore
stouffs@nus.edu.sg

Abstract. Searching for spatial objects in CAD tools is mostly based on the ability to compare properties of different objects. Instead, the matching mechanism(s) underlying a shape grammar interpreter offers a much wider potential for search, including the emergence of shapes that were unanticipated at the point of specification. This paper provides an overview of different rule types that can be discerned in the context of shape grammars, and explores the impact these have on the ability for search. It specifically considers two alternative matching algorithms, either determining a transformation matrix or an association of graphical elements, the latter complemented with constraining predicates, applying over different data types, e.g., shapes, shapes augmented with attributes, and descriptions, to provide for a wide range of spatial search variations.

Keywords: Spatial search · Shape rules · Description rules · Rule types

1 Introduction

A rule-based design generation scheme can be considered to consist of a set of generative rules that can be applied to a set of initial objects in order to generate a variety of solutions or design objects. The set of rules, together with the initial object(s) and the vocabulary of elements from which the rules are composed, is termed a grammar and the objects resulting from a generative application of such a grammar constitute the derived language. Specifically, a shape grammar is a formal rewriting system for producing languages of shapes [1].

However, the paradigm of design as search [2, 3]—within which rule-based design generation fits particularly well—is more fundamental to design than its generational form alone might imply. Any mutation of an object into another one, or parts thereof, whether as the result of a transformation or operation, constitutes an action of search. A rule can be deemed to constitute a particular compound operation or mutation, that is, a composition of a number of operations and/or transformations that is recognized as a new operation and can be applied as such. Similarly, a grammar may be viewed as just a collection of rules or operations that yields a certain set (or language) of design objects given an initial object (or set thereof). As such, the creation of a grammar is only a tool that allows a structuring of a set of operations that has proved its applicability to the creation of a certain set of objects, rather than a framework for generation.

A rewriting rule can generally be expressed in the form *lhs* → *rhs*, with the left-hand-side (*lhs*) of the rule specifying the part to be recognized in the current design and the right-hand-side (*rhs*) of the rule specifying the part to replace the recognized part

J.-H. Lee (Ed.): CAAD Futures 2019, CCIS 1028, pp. 474–488, 2019.
https://doi.org/10.1007/978-981-13-8410-3_33

with. Thus a rewriting rule specifies two actions, an action of search or recognition specified by the left-hand-side and an action of mutation specified by the right-hand-side with respect to the left-hand-side. Recognizing the left-hand-side of the rule in the current design involves determining a transformation under which the left-hand-side becomes part of the design, that is, denoting the current design d and the transformation f, determining f such that $f(lhs) \leq d$. Once f is found, the mutation involves replacing the left-hand-side with the right-hand-side under the same transformation, that is, the resulting design is the result of $d - f(lhs) + f(rhs)$. An additive rule is a rule that only adds elements to the current design, that is, $lhs \leq rhs$. Similarly, a subtractive rule only removes elements from the current design, that is, $lhs \geq rhs$.

Transformations may be of different kinds [4]. A shape grammar commonly considers transformations of similarity, allowing for translation, rotation, reflection and uniform scaling. However, a more narrow or broad definition of transformations can be considered as well. For example, Stiny and Mitchell's Palladian grammar [5] can be considered to allow for only isometric (or Euclidean) transformations, including translation, rotation and reflection, but no scaling, as the rules potentially operate on a simple grid of rectangular shapes of equal size. In practice, as exemplified by the underlying grid for the Villa Malcontenta [5], the size of grid cell may vary implying the inclusion of non-uniform scaling or, instead, the specification of parametric shape rules. Theoretically, in a parametric shape rule, the left-hand-side and right-hand-side of the shape rule can have parameters specified and constraints over these parameters [6]. However, no implementation of a general parametric shape grammar interpreter exists that allows for the explication of parameters, unless for description rules. So-called parametric shape grammar interpreters instead allow for the recognition of associations between spatial elements that serve to constrain rule matching. For this reason, we prefer the term *associative* (or parametric-associative) shape rules. Applied to shapes of line segments, associations that are recognized may include intersecting lines, parallel or perpendicular lines, or segments of equal length. Depending on the associations recognized, a rectangle may be a set of four line segments, either with twice two parallel segments that are perpendicular with respect to one another, or with twice two segments of equal length and with diagonals of equal length as well.

It is obvious that the types of allowable rules greatly impact the expressiveness and applicability of a grammar system. In this paper, we present an overview of different types of rules (not entirely exhaustive) that can be discerned in the context of shape grammars. Subsequently, we narrow our focus to the left-hand-side of the shape rule and the related action of search or shape recognition, in the context of different types of rules. That is, we focus our attention on search not as a paradigm of design but as a technique of recognizing shapes within a design. Specifically, we explore to what extent the shape recognition mechanism underlying a shape grammar interpreter can support graphical search within a Computer Aided Design (CAD) environment. Currently, searching and finding spatial elements and objects in CAD software tools, if available, is mostly based on the ability to compare properties of different objects, for example, object type, area, etc. The shape recognition or matching mechanism(s) underlying a shape grammar interpreter offers a much wider potential for search, including the emergence of shapes that were unanticipated at the point of specification.

2 Types of Rules

We base our exploration on an overview of different types of rules, as these support different approaches to search. Shape grammar rules can be distinguished, among others, by data type, e.g., shape and description rules, by the relationship between the rule's left-hand-side and right-hand-side, e.g., addition and subtraction rules, and by matching algorithm, parametric-associative and non-parametric rules. In the context of search, where the emphasis lies on the left-hand-side of the shape rule, the first and last categorization are the most interesting, as the right-hand-side of the rule doesn't affect the search process. We limit our exploration to these two categorizations.

2.1 By Data Type

Most examples of shape grammars rely on labeled shapes, a combination of line segments and labeled points (in two dimensions) [1]. Krishnamurti [7, 8] extends the underlying maximal element representation to plane segments and Stouffs [9, 10] to volume segments and higher-dimensional hyperplane segments. Jowers [11, 12] considers the application of shape grammars to curves, such as quadratic Bezier curves.

Next to labels, other non-geometric attributes have been considered for shapes. Stiny [13] proposes numeric weights as attributes to denote line thicknesses or surface tones. Knight [14, 15] considers an extension to the shape grammar formalism that allows for a variety of qualitative aspects of design, such as color, to be integrated in the rules of a shape grammar. Though not specific to colors, the resulting grammar is called a *color grammar* and notions of transparency, opacity and ranking are introduced to regulate the behavior of interacting quality-defined areas or volumes. Stiny [16] also proposes to augment a shape grammar with a description function in order to enable the construction of verbal descriptions of designs. Although most authors do not consider descriptions as attributes to shapes, Beirão [17] specifically considers descriptions as attributes to spatial objects. Other kinds of attributes, or even variations in the specification of a kind of attribute, can also be considered. For example, colors can be specified in different ways, as a three-dimensional RGB or HSV (Hue, Saturation, Value) space, or in an enumerative way as Knight [14, 15] considers.

Sortal grammars [18, 19] are a class of shape grammar formalisms that enable a single *sortal* grammar interpreter to support all of the above shape grammars, and any variations thereof. Adopting an algebraic abstraction enabling the algebraic derivation of combinations of basic shape algebras with attribute algebras, this abstraction at the same time serves as a procedural abstraction, giving insights into the modular implementation of a general shape grammar interpreter for different grammar forms [20]. Representationally, *sortal* grammars utilize *sortal* structures as representational structures, where these structures are defined as formal compositions of other, primitive, *sortal* structures, termed *sorts* [21]. *Sortal* grammars benefit from the fact that every component *sort* specifies a partial order relationship on its individuals and forms, defining both the matching operation and the arithmetic operations for rule application. The SortalGI *sortal* grammar interpreter [22], developed as a library and API in the Python programming language, implements this approach and supports most data types and their combinations identified above. Specifically, SortalGI operates on points, lines and planes, line segments and (rectilinear) plane segments, circles and ellipses, circular and elliptical arcs, quadratic Bezier curves, labels, weights, colors, enumerated values, dates, and descriptions, and any combination thereof, within both 2D and 3D.

2.2 By Matching Mechanism

As stated above, applying a rule $lhs \rightarrow rhs$ onto a design d requires determining a transformation f such that $f(lhs)$ is a part of d, $f(lhs) \leq d$. The mechanism that supports this shape recognition process is called the *matching mechanism*. It generally relies on the identification of distinguishing elements within the rule's left-hand-side and the design under investigation, and mapping distinguishing elements of the same kind between both shapes in order to determine the corresponding transformation. Distinguishing elements are commonly points that form part of the shape or points of intersection among infinite lines that carry line segments forming part of the shape [23, 24].

As noted above, transformations may be of different kinds [4] and the kind of transformation necessarily impacts the matching mechanism. In three dimensions, three distinguishable points from the rule's left-hand-side and three mapped counterparts from the design under investigation uniquely determine up to two possible similarity transformations (under reflection). Evaluating each possible transformation can take two steps: a first step focused on the selected distinguishable points and a second step involving the entire left-hand-side shape. The first step looks at invariants under similarity transformations, e.g., angles and length ratios, that can be compared between the two sets of distinguishable points. The second step applies the respective transformation to the entire left-hand-side in order to check its embedding in the design under investigation. Note that it is assumed that the embedding relationship for shape grammars supports emergence. For example, when matching for a square of line segments, any square of line segments from the given shape will do, even if these line segments extend beyond the corner points of the square. The same applies when matching for a rectangle, however, only rectangles with the same ratio between length and width will be matched.

If the allowable transformations are reduced from similarity to, e.g., isometric transformations, the number of invariants will increase, thus constraining the matching process and, potentially, reducing the number of possible matches [4]. However, in the case of isometric transformations, or whenever rotations are allowed, three distinguishable points remain necessary in three dimensions and, therefore, the same matching mechanism, with the additional constraints, may be adopted. Relaxing the allowable transformations is, in principle, also possible, although a so-called parametric matching process may become more appropriate. Woodbury [25] presents the mechanisms of a shape schema grammar, a grammar specifying parametric shape rules that operate on parametric shapes. The matching mechanism is one of constraint satisfaction. However, the algorithm is intractable, no shape grammar interpreter yet exists, and there is no indication as of yet of how designers would use shape schema grammars or rules.

All other (proposed) implementations of a parametric shape grammar rely on a graph-based representation to underlie the matching process [26–29]. Although the exact graph representation may differ from one implementation to another, commonly, intersection points between line segments, or between the infinite lines carrying these segments, are considered as distinguishable points. As such, graph-based implementations can be used to find polygons of a specified number of sides, while additional constraints can be specified on (the coordinates of) the vertices, in a similar way to the shape schema grammar. In that sense, graph-based implementations are much more general than Stiny's [6] parametric shape grammar implies. Where Stiny adds parameters to loosen the matching constraints, graph-based implementations can ignore geometric coordinates altogether and, instead, require constraints to be added where invariants should apply. However, rather than requiring such constraints to be explicated, graph-based implementations generally consider implicit constraints that can be automatically recognized from associations between spatial elements. For example, Grasl and Economou's *GRAPE* shape grammar interpreter considers equal length as an associative relationship among line segments [26]. As such, an equilateral triangle will only match onto equilateral triangles and an isosceles triangle onto isosceles triangles. Any other triangle will match any triangle, irrespective of its shape. The lengths of diagonals are also considered to apply under this associative relationship. Therefore, any regular polygon only matches onto a regular polygon with the same number of sides.

Instead, the SortalGI *sortal* grammar interpreter considers associative relationships of parallelism and perpendicularity among lines and planes, instead of length. As such, an equilateral or isosceles triangle of line segments will match any triangle of line segments, as there are no parallel or perpendicular lines involved. Only a right-angled triangle will limit any matches to right-angled triangles. Similarly, a rectangle of line segments will only match rectangles of line segments, although, a square of line segments will also match all rectangles of line segments, as equal lengths are not recognized. To allow for further fine-tuning of the matching process, SortalGI allows for additional constraints to be specified in the form of predicates [30, 31]. We elaborate on the roles of associative relationships and predicates below.

3 Shape Recognition and Search

In the sequel, we narrow our focus to the left-hand-side of the shape rule and the related action of search or shape recognition, while reflecting on the different rule types identified above. Specifically, we investigate to what extent the shape recognition mechanism underlying a shape grammar interpreter can support graphical search within a CAD environment. If available, searching for spatial objects in CAD software tools is mostly based on the ability to compare properties of different objects, for example, object type, area, etc. Instead, the shape recognition or matching mechanism(s) underlying a shape grammar interpreter offers a much wider potential for search, including the emergence of shapes that were unanticipated at the point of specification.

We adopt the SortalGI shape grammar interpreter [22] as the reference point for our exploration of spatial search. As a modular implementation of a generalized shape grammar interpreter for different grammar forms, it supports both parametric-associative and non-parametric shape grammars, including points, line and plane segments, circular and elliptical arcs, quadratic Bezier curves, labels, weights, colors, enumerated values, and (parametric) descriptions, in 2D and 3D. Emergence is naturally supported. As such, it supports both different types of rules by data type and by matching mechanism, at least to some extent.

The SortalGI shape grammar interpreter exists in the form of a library and API in the Python programming language, as well as a Rhino/Grasshopper plug-in. As such, the SortalGI library can be accessed and employed in at least three different ways: within a Python development environment, within the Rhino 3D modeling environment and within the Grasshopper algorithmic modelling environment. Note that the API and plug-in necessarily limit the extent of geometric and non-geometric element types that are supported, due to the need to graphically visualize the data within the Rhino 3D modeling environment. Nevertheless, we adopt the plug-in as a reference for this study for reasons of simplicity and reproducibility. The SortalGI Grasshopper plug-in encapsulates the SortalGI library and supports the specification and application of shape rules, both non-parametric and parametric-associative, including points, line segments, plane segments, circles, ellipses, (circular) arcs and quadratic Bezier curves, descriptions and attribute descriptions (or labels), and the generation of single or multiple rule application results. It also includes a node that, given a shape rule and a shape under investigation, returns a list of all possible matches, without rule application. As such, it can be used for spatial search, depending on the left-hand-side of the rule and irrespective of the right-hand-side of the rule.

3.1 Emergence

The SortalGI plug-in offers two different components to create a rule object. The Rule component constructs a non-parametric rule object from a rule name and brief explanation of the rule, a left-hand-side shape object and a right-hand-side shape object. While the right-hand-side of a rule may be left empty, the left-hand-side must have a minimum set of geometry and/or descriptions present. The pRule component takes the same inputs as the Rule component but returns a parametric-associative rule object.

The SortalGI library, underlying the plug-in, only supports transformations of similarity for non-parametric rules, allowing for translation, rotation, reflection and uniform scaling. As such, a rule's left-hand-side specifying a simple square composed of four line segments will match any square of four line segments, irrespective of size, orientation and position. In addition, any of these four line segments may be part of longer line segments that extend beyond the corner points of the square, or they may (originally) be drawn as multiple smaller pieces of line segments that extend one another or even overlap (Fig. 1). Internally, the matching mechanism will automatically reduce any line segments that overlap or extend one another into the corresponding maximal line segment and match maximal line segments, in the left-hand-side of the rule, onto (parts of) maximal line segments, in the shape under investigation. This mechanism specifically supports emergence, the square emerges from any number of line segments that together cover the four sides of the square and, possibly, more. The same applies to any other shape of line segments.

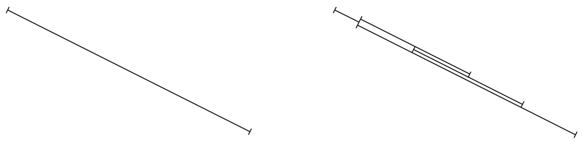


Fig. 1. A single line (left) may constitute one or more line segments that touch or overlap (right). The matching mechanism automatically reduces any such line segments to the single maximal line segment (left).

To constrain the matching process, the rule application node additionally accepts a subshape as input. While rule application always applies to the entire shape specified, the matching mechanism compares the left-hand-side of the rule only to the subshape, if specified, thereby ignoring any spatial elements that do not belong to the subshape.

3.2 Shape Descriptions

While the non-parametric matching mechanism only supports similarity transformations, it is not impossible to require only a subset of transformations, such as Euclidean transformations (omitting scaling). However, the process will not be as straightforward. In principle it is possible to use shape descriptions to constrain rule application. Shape descriptions are verbal descriptions accompanying a shape; these can be structured or unstructured [32, 33]. When included in the left-hand-side of the shape rule, they may include parameters that are matched onto part of the respective descriptions of the shape under investigation, as well as conditional expressions that constrain these matches. Such conditional expressions can include explicit references to shapes and shape rules. Such a reference must minimally specify the spatial data kind and its numeric property but, in the case of multiple elements of the same spatial data kind, must additionally specify a filter to distinguish the specific spatial element.

In the SortalGI plug-in, the spatial data kinds can be identified as 'point3D', 'lineSeg3D', 'planeSeg3D', 'circle3D', 'ellipse3D', 'arc3D', 'bezier3D', respectively, for points, line segments, plane segments, circles, ellipses, circular arcs and (quadratic) Bezier curves. Note that these names are specific to non-parametric rules and are slightly different within parametric-associative rules. Attribute labels as a data kind are identified as 'labelD' (or a variant thereof). As such, the length of a line segment with label "l1" can be retrieved using the partial description 'lineSeg3D.length:labelD.-value = "l1"'. The colon should be read as 'where', as in the length of a lineSeg3D where the value of labelD equals l_1. However, this will only work if at least one of the line segments of the square has been labelled uniquely with respect to the other line segments of the square. Unfortunately, any rule to do so would necessarily match the same square four times, as each of the four line segments may receive the label. Therefore, a better approach may be to add an extra spatial data kind to the rule, such as a plane segment covering the square, that can be used to constrain the allowable matches (Fig. 2). Given that the plane segment retains the full symmetry of the square, a rule that adds such plane segment to a square will only match each square once. The plug-in includes a component 'Apply All Together' that applies a rule in parallel according to every match found. As each square specifies a single plane segment, no additional label is necessary to uniquely identify the plane segment. Given a description specifying the exact area of the square to be matched, a description rule 'prescribed_area? = planeSeg3D.area → prescribed_area' will constrain the rule matching appropriately. Here, *prescribed_area* is a parameter matching whatever (numeric) value is specified in the description accompanying the shape under investigation, and will constrain rule matching to squares that have the exact area, thus omitting scaling as an allowable transformation. Note that the rule necessarily also specifies a right-hand-side. In this case, the rule leaves the description unchanged.

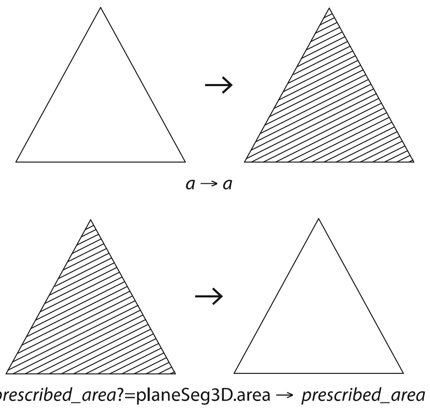


Fig. 2. A preparatory rule to add a plane segment to the triangle (top), and the search rule constraining matching to triangles with a given area (bottom). Each rule combines a shape rule and a description rule, although the description rule of the preparatory rule may be omitted.

In general, description rules with conditional expressions referencing shape rules can assist in constraining rule matching, but may require a preparatory rule that is performed first and matches a larger set of shapes, and adds additional information to the shape in the form of spatial elements or attribute labels, or both, before a second rule including an appropriate description rule can be applied to retrieve only the desired matches.

3.3 Parametric-Associative Rules: Searching Triangles

The parametric-associative matching mechanism supports a much wider set of transformations, although the matching transformation is not specified numerically, e.g., in the form of a transformation matrix. Instead, the transformation associates specific spatial elements in the given shape with respective spatial elements in the left-hand-side of the rule. For example, revisiting Stiny's [6] grammar for Chinese ice-ray lattice designs, one rule splits (almost) any triangle into a triangle and a quadrilateral by placing a single line segment between two of the original triangle's edges. As stated before, any triangle that does not include a right angle will match any other triangle, including right-angled triangles. At a minimum, searching for arbitrary triangles within a given shape only requires the specification of a non-right-angled triangle as the left-hand-side of a rule (ignoring the right-hand-side of the same rule). However, Stiny's grammar is not intended to apply to any triangle, but only to triangles of a certain minimum size that have not been the subject of a rule application—have not been split —before. Addressing the latter condition first, one solution is to label the edges of each polygon and include the labels as parametric descriptions using the same parameter in the left-hand-side of the rule. However, this procedure works fine within the context of a grammar application/derivation, but is rather useless as a search mechanism. Instead, SortalGI supports Liew's [30] *void* predicate, prescribing the area to be devoid of any spatial elements. The specific area is necessarily a polygonal area specified by its vertices, i.e., a list of position vectors. While these position vectors are defined in absolute coordinates in order to achieve a compact description, it is of utmost importance that the vertices coincide with points, line segments or intersection points between line segments, or between the infinite lines carrying these segments. Such associations will be automatically recognized by the matching algorithm and applied in the matching process, instead of the actual coordinates. As a necessary consequence, note that only the inside of the area must be devoid of spatial elements, the vertices themselves and the edges of the polygon may coincide with, overlap or partially bound a spatial element in the given shape.

To address the other condition, we can apply the plane segment trick previously described to compute the area of the triangle and constrain this area to be larger than a specified value. However, adding a plane segment will violate the void clause and open the possibility to match a larger triangle composed of smaller triangles. One solution to this problem is to assign each plane segment a unique label, such that the individual plane segments remain distinguishable, rather than possibly combining into a maximal plane segment. The parametric description assigned to the corresponding plane segment in the left-hand-side of the rule will match only a single label, limiting the matching to an individual plane segment. It should be noted that while this is a perfectly valid solution it is not a very efficient one. The matching mechanism will initially focus on matching the triangle, before checking the presence of the plane segment and its label. This means that all possible triangles will be exhaustively found and tested. While this is no different from the void rule (without the plane segment), the trick with the plane segment necessarily requires the void rule to be applied previously, adding the plane segment with a unique label within the right-hand-side of the rule. Improving efficiency could be achieved by labelling the edges of the triangle, in addition to or instead of the plane segment. The edge labels will already play a role in the matching of the triangle, before the plane segment is matched. Of course, this comes at the cost of even more additional information added to the shape. Another solution to the problem may be to use a variant of the void predicate as provided by SortalGI. The void predicate additionally allows one to specify a type of spatial element that should be absent from the area, while spatial elements of other types are ignored. In the case of searching for a triangle as defined by three line segments, specifying a void predicate to apply only to 'lineSegP3D' (the parametric-associative equivalent of 'lineSeg3D') spatial elements, would be able to complement the plane segment in the left-hand-side of the rule and the corresponding matching process. Note that this doesn't address the efficiency problem but it does avoid cluttering the shape with numerous labels (Fig. 3).

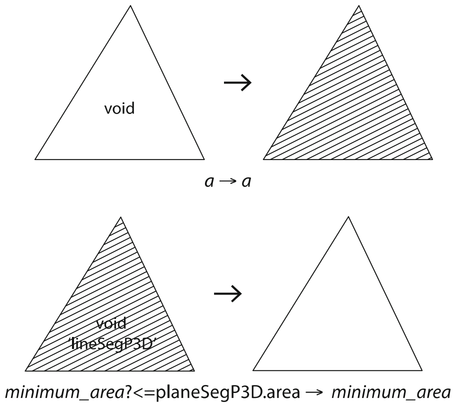


Fig. 3. A preparatory rule (top) and search rule (bottom) to identify an empty triangle with a specified minimum area. Each rule combines a shape rule and a description rule, although the description rule of the preparatory rule may be omitted.

So far, we have focused on searching for arbitrary triangles, but we might also be interested in more specific kinds of triangles. Searching for right-angled triangles is straightforward. If the left-hand-side of a rule constitutes a right-angled triangle than the rule will match only right-angled triangles. The matching mechanism automatically recognizes parallel and perpendicular associations between spatial elements (of the same kind). However, if the left-hand-side of a rule constitutes an equilateral triangle then the rule will still match an arbitrary triangle, as equal length associations are not automatically recognized. For this purpose, we can make use of a description that collects the lengths of the respective line segments and compares them. Unfortunately, this requires a preparatory rule to collect the lengths of the respective line segments, before the search rule can compare the different lengths.

SortalGI allows spatial elements within the left-hand-side of a parametric-associative rule to be tagged. Tags may be considered similar to labels, to some extent, but tags only exist within the matching process of the rule. The ability to tag a spatial element is mainly intended to support predicate specifications. As predicate specifications form a separate input, the tag serves to identify the spatial element the predicate is referencing. As such, tagging is only available in parametric-associative rules. However, tags as references can also be used in description rules that accompany parametric-associative shape rules. Thus, the description rule can identify different line segments by their tag, without these tags to be assigned as labels in another preparatory rule application. Instead, assigning tags as actual labels would not only require an additional preparatory rule but also have the labels persist unless another rule is used to remove the labels once again.

Assuming the line segments of the triangle are tagged as #l1, #l2 and #l3 (tags are recognized by the '#' symbol), the description rule to collect the lengths of the line segments may take the form 'a → (#l1.length, #l2.length, #l3.length)' where 'a' is a parameter that will be assigned any and all content of the description, to be subsequently ignored and replaced by a tuple of three length values. Next, the description rule to compare the lengths of the line segments can take the form '(l1, l2? = l1, l3? = l1) → nil'. Here, the tuple '(l1, l2, l3)' matches the tuple available in the description and the conditional expressions assigned to 'l2' and 'l3' ensure all lengths are the same. *Nil* is a keyword to indicate an empty value, however any other value might be assigned to the description as well.

As there may be any number of triangles in the shape under investigation, any number of descriptions may be created as a result of the parallel application of the preparatory rule. To link each description with the respective triangle, a labelled plane segment is created. However, rather than labelling the triangle and adding this label into the description, instead, we assign the description itself as attribute to the plane segment. The same applies to the description rule (at least the left-hand-side of the rule) comparing the lengths (Fig. 4).

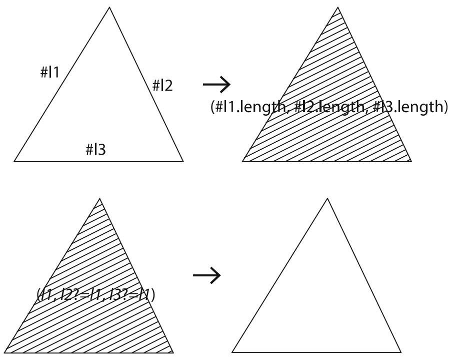


Fig. 4. A preparatory rule (top) and search rule (bottom) to identify an equilateral triangle. The preparatory rule adds a plane segment and adds the edge lengths as an attribute description to the plane segment. The search rule checks whether the edge lengths have equal values.

The search rule (Fig. 4 bottom) could be easily adapted to search for isosceles triangles by removing one of the conditional expressions. Instead of line lengths, it is also possible to identify equilateral and isosceles triangles by computing and comparing angles between respective line segments. The right-hand-side of a description rule may include an expression specifying an angle function that calculates the (counterclockwise) angle between two direction vectors (from the first to the second vector), e.g., 'a → (angle(#l1.direction, #l2.direction), angle(#l2.direction, #l3.direction), angle(#l3.direction, #l1.direction))'. However, it should be noted that the direction of a line segment is not always univocal. In principle, the direction vector reflects on the direction the line was drawn in, from one end to another. However, if the maximal line segment is the result of multiple overlapping or touching line segments, then the direction may be overwritten by the maximalization process, which specifies that the tail of the line segment has the lowest x-value, or if both x-values are equal then the lowest y-value, etc.

3.4 Parametric-Associative Rules: Searching Quadrilaterals

To further clarify the search and matching process, let us consider quadrilaterals instead of triangles. If the left-hand-side of a rule specifies a convex quadrilateral without any parallel or perpendicular line segments, then this rule will match any convex quadrilateral, irrespective of whether the matching quadrilateral has parallel or perpendicular segments. If, instead we want to search for concave quadrilaterals, we must specify a concave quadrilateral as the left-hand-side of the search rule. The matching process identifies all points of intersection among infinite lines that carry line segments forming part of the shape as well as their ordering along the infinite lines. In the case of a convex quadrilateral, all intersection points either define the vertices of the quadrilateral or lie outside of the quadrilateral (Fig. 5). In the case of a concave quadrilateral, the two intersection points each lie on a line segment in between its two vertices (not coinciding with any vertex).

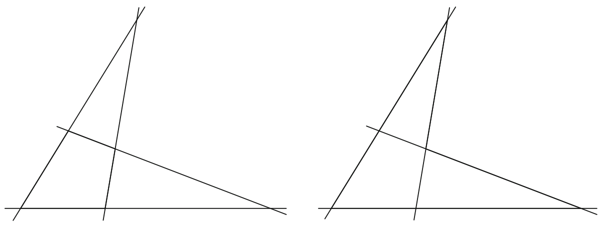
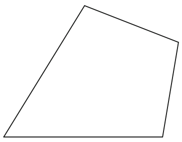
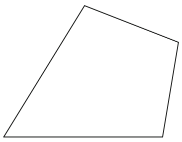
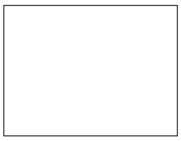
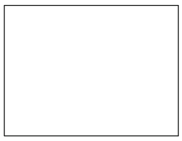
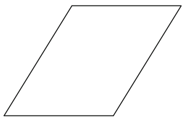
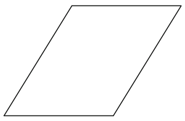
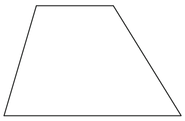
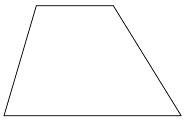
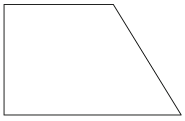
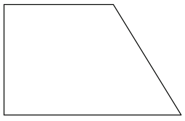
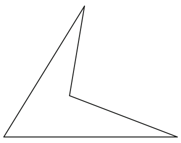
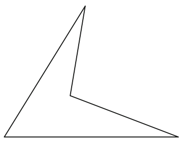
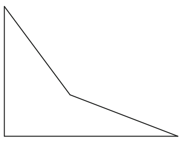
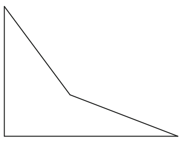
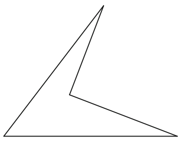
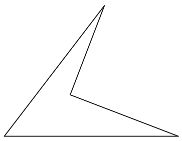


Fig. 5. A convex (left) and concave (right) quadrilateral. Each infinite (edge) line specifies (at most) three intersection points with the other infinite (edge) lines, two of which are vertices of the quadrilateral. In the case of the concave quadrilateral, the third intersection point lies between the two vertices for two of the edges. In the case of the convex quadrilateral, this never happens.

As indicated before, if the left-hand-side of the search rule specifies a rectangle, the rule will match any rectangle. However, the same applies as well for a square, that is, if the left-hand-side of the search rule specifies a square, the rule will still match any rectangle, because associations of equal lengths are not automatically recognized. Instead a description (and a preparatory rule) can be used to constrain matching rectangles to squares. The same applies to the relationship between a parallelogram and a rhombus. Where a rectangle is uniquely defined by its perpendicular corners (the parallel edges are an automatic consequence), a parallelogram is uniquely defined by its parallel edges. Thus, if the left-hand-side of the search rule specifies a parallelogram, the rule will match any parallelogram. Similar to a square with respect to a rectangle, a rhombus can be conceived as a parallelogram where all four sides have the same length. Thus, if the left-hand-side of the search rule specifies a rhombus, the rule will still match any parallelogram, because associations of equal lengths are not automatically recognized, and a description (and a preparatory rule) can be used instead to constrain matching parallelograms to rhombi. Note that an arbitrary convex quadrilateral with the additional constraint of equal length sides will also match rhombi only, however, the matching process will be less efficient as associations of parallelism and perpendicularity will be considered at an earlier stage, whereas constraints of equal length expressed through a description rule will necessarily need to be checked after the respective quadrilateral has been identified. Similarly, if the left-hand-side of the search

Table 1. Overview of the relationship between the left-hand-side of the search rule and matching shapes for various quadrilaterals. Restricting the matching to a smaller set of shapes necessarily requires the use of descriptions and a preparatory rule.

Left-hand-side of the search rule	**Matching shapes**	**Left-hand-side of the search rule**	**Matching shapes**
A convex quadrilateral without parallel or perpendicular edges 	Any convex quadrilateral 	A rectangle or square 	Any rectangle or square 
Any parallelogram or rhombus 	Any parallelogram or rhombus 	A non-right-angled trapezoid 	Any trapezoid 
A right-angled trapezoid 	Any right-angled trapezoid 	A concave quadrilateral without perpendicular edges 	Any concave quadrilateral 
A concave quadrilateral with a 90 degrees exterior angle 	Any concave quadrilateral with a 90 degrees exterior angle 	A concave quadrilateral with a 90 degrees interior angle 	Any concave quadrilateral with a 90 degrees interior angle 

rule specifies a non-right-angled trapezoid, the rule will match any trapezoid; a right-angled trapezoid will match only right-angled trapezoids. In order to search for isosceles trapezoids, an additional description rule will be required. The same applies if we would like to distinguish trapezoids with an obtuse angle from trapezoids with only acute angles. Any other quadrilateral figures that rely on equal lengths (such as a kite) or particular types of angles require a similar approach (Table 1).

4 Conclusion

The combination of two alternative matching algorithms, either determining a transformation matrix or an association of graphical elements, the latter complemented with constraining predicates, applying over different data types, e.g., shapes, shapes augmented with attributes and descriptions (the latter allowing for parametric description rules) provides for a wide range of spatial search variations, beyond what CAD software tools mostly offer today. Obviously, the elaboration above far from exhausts the possibilities of search based on the shape recognition or matching mechanism(s) underlying a shape grammar interpreter, in general, and SortalGI, in particular.

Acknowledgments. I would like to thank Bui Do Phuong Tung for his development work on the SortalGI library, Bianchi Dy for her development work on the SortalGI plug-in for Grasshopper, and Dan Hou for her inputs regarding predicates.

References

1. Stiny, G.: Introduction to shape and shape grammars. Environ. Plann. B: Plann. Des. **7**, 343–351 (1980)
2. Akin, Ö.: A formalism for problem restructuring and resolution in design. Environ. Plann. B: Plann. Des. **13**, 223–232 (1986)
3. Woodbury, R.F.: Searching for designs: paradigm and practice. Build. Environ. **26**, 61–73 (1991)
4. Wortmann, T., Stouffs, R.: Algorithmic complexity of shape grammar implementation. Artif. Intell. Eng. Des. Anal. Manuf. **32**, 138–146 (2018)
5. Stiny, G., Mitchell, W.J.: The Palladian grammar. Environ. Plann. B: Plann. Des. **5**, 5–18 (1978)
6. Stiny, G.: Ice-ray: a note on Chinese lattice designs. Environ. Plann. B: Plann. Des. **4**, 89–98 (1977)
7. Krishnamurti, R.: The maximal representation of a shape. Environ. Plann. B: Plann. Des. **19**, 267–288 (1992)
8. Krishnamurti, R.: The arithmetic of maximal planes. Environ. Plann. B: Plann. Des. **19**, 431–464 (1992)
9. Krishnamurti, R., Stouffs, R.: The boundary of a shape and its classification. J. Des. Res. **4**, 75–101 (2004)
10. Stouffs, R., Krishnamurti, R.: Algorithms for classifying and constructing the boundary of a shape. J. Des. Res. **5**, 54–95 (2006)
11. Jowers, I., Earl, C.: The construction of curved shapes. Environ. Plann. B: Plann. Des. **37**, 42–58 (2010)

12. Jowers, I., Earl, C.: Implementation of curved shape grammars. Environ. Plann. B: Plann. Des. **38**, 616–635 (2011)
13. Stiny, G.: Weights. Environ. Plann. B: Plann. Des. **19**, 413–430 (1992)
14. Knight, T.W.: Color grammars: designing with lines and colors. Environ. Plann. B: Plann. Des. **16**, 417–449 (1989)
15. Knight, T.W.: Color grammars: the representation of form and color in design. Leonardo **26**, 117–124 (1993)
16. Stiny, G.: A note on the description of designs. Environ. Plann. B: Plann. Des. **8**, 257–267 (1981)
17. Beirão, J.N.: CItyMaker: designing grammars for urban design. Ph.D. thesis, Delft University of Technology, Delft, The Netherlands (2012)
18. Stouffs, R., Krishnamurti, R.: Sortal grammars as a framework for exploring grammar formalisms. In: Burry, M., Datta, S., Dawson, A., Rollo, J. (eds.) Mathematics and Design 2001, pp. 261–269. Deakin University, Geelong (2001)
19. Stouffs, R.: On shape grammars, color grammars and sortal grammars. In: Achten, H., Pavlicek, J., Hulin, J., Matejovska, D. (eds.) Digital Physicality, vol. 1, pp. 479–487. eCAADe, Brussels (2012)
20. Stouffs, R.: Implementation issues of parallel shape grammars. Artif. Intell. Eng. Des. Anal. Manuf. **32**, 162–176 (2018)
21. Stouffs, R.: Constructing design representations using a sortal approach. Adv. Eng. Inform. **22**, 71–89 (2008)
22. Dy, B., Stouffs, R.: Combining geometries and descriptions: a shape grammar plug-in for Grasshopper. In: Kepczynska-Walczak, A., Bialkowski, S. (eds.) Computing for a Better Tomorrow, vol. 2, pp. 509–518. eCAADe, Brussels (2018)
23. Krishnamurti, R., Earl, C.F.: Shape recognition in three dimensions. Environ. Plann. B: Plann. Des. **19**, 585–603 (1992)
24. Krishnamurti, R., Stouffs, R.: Spatial grammars: motivation, comparison and new results. In: Flemming, U., Van Wyk, S. (eds.) CAAD Futures '93, pp. 57–74. North-Holland, Amsterdam (1993)
25. Woodbury, R.: An introduction to shape schema grammars. Environ. Plann. B: Plann. Des. **43**, 152–183 (2016)
26. Grasl, T., Economou, A.: From topologies to shapes: parametric shape grammars implemented by graphs. Environ. Plann. B: Plann. Des. **40**, 905–922 (2013)
27. Wortmann, T.: Representing shapes as graphs. Master's thesis, MIT, Cambridge (2013)
28. Yue, K., Krishnamurti, R.: Tractable shape grammars. Environ. Plann. B: Plann. Des. **40**, 576–594 (2013)
29. Strobbe, T., Pauwels, P., Verstraeten, R., De Meyer, R., Van Campenhout, J.: Toward a visual approach in the exploration of shape grammars. Artif. Intell. Eng. Des. Anal. Manuf. **29**, 503–521 (2015)
30. Liew, H.: SGML: a meta-language for shape grammar. Ph.D. thesis, MIT, Cambridge, MA (2004)
31. Stouffs, R., Hou, D.: The complexity of formulating design(ing) grammars. In: Fioravanti, A., et al. (eds.) Shock! Sharing of Computable Knowledge, vol. 2, pp. 443–452. eCAADe, Brussels (2017)
32. Stouffs, R.: Description grammars: a general notation. Environ. Plann. B: Urban Anal. City Sci. **45**, 106–123 (2018)
33. Stouffs, R.: Description grammars: precedents revisited. Environ. Plann. B: Urban Anal. City Sci. **45**, 124–144 (2018)