An introduction to structure and structure grammars

by

C. Carlson, R. McKelvey, R.F. Woodbury

48-20-90 C. 3

An introduction to structure and structure grammars

Christopher Carlson*, Roy McKelveyt, and Robert Woodbury*

Departments of Architecture* and Designt
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

Structures represent relationships between parts in configurations. We define the concepts of *structure* and *structure rewriting* and show in several examples how these are applied to performing computations on designs and describing design languages.

Introduction

A structure is a symbolic representation of discrete parts and their relationships in a configuration. Structures typically represent spatial relationships between graphic motifs or three-dimensional components, although they can represent non-spatial objects as well. By representing parts in configurations symbolically, structures abstract away from the constitutions of the parts themselves, distilling the essence of the relationships between them.

Like Stiny's shapes (1980), structures are a fundamental representation for spatial designs. Their associated computational mechanisms — structure rewriting systems and structure grammars — describe algorithms, languages of design, and design search spaces. Stiny has outlined a formalism called *set grammars* that is similar in spirit to structure grammars (1982); to our knowledge, though, no one has explored their properties. There are at least two reasons that recommend doing so. First, the symbolic nature of structure grammars

This work has been supported by the Engineering Design Research Center, an NSF Engineering Research Center, and by NSF grant MSM 8717307.

makes **them close relatives of production** systems, which are described in a large body of literature. We may draw on that experience to implement efficient, interactive structure grammar interpreters. Second, structure grammars can be put into correspondence with Chomsky's phrase-structure grammars via an analogy of objects to symbols and spatial composition to string concatenation. Thus many of the methods and theoretical results of formal linguistics can be brought to bear on problems of spatial composition and design. As it turns out, this analogy is good enough to suggest intriguing avenues of inquiry, but it is not so complete that it answers all of the questions it raises.

Structure

Formally, a *structure* is a finite set of ordered pairs called *structure elements.* Each structure element consists of a symbol drawn from a finite set *V,* called a *vocabulary,* and an element drawn from a transformation group *F.* We use lower-case Greek letters to denote structures and boldface type to denote the symbols in a vocabulary. The structure a= {(c, /z), (c, -/*), *(s,* v)} may represent many different configurations, depending on the parts and transformations to which its symbols and group elements are mapped. One such configuration is shown in figure 1. The symbols ***c*** and *s* in the structure correspond to column and span motifs in the configuration; the symbols *h* and v to transformations — horizontal and vertical translations — that map the motifs to their positions in the configuration.

Since structures are sets, any operation or relation on sets applies also to structures; we use the same notation to denote both. The union of structures ***a*** and /J, written *a* u j3, is **the set** of structure elements contained in either *a* or *p.* Structure intersection *(anp)* and difference *(a - p)* are defined similarly. We say that a is a *substructure* of *P* *(cc^p)* when a is a subset of /?, and that *a* and *p* are *equal* when each is a substructure of the other.

We define the composition of transformations by *gf(x) = gtf(x))>* ^{anc}* represent the identity transformation by *L* A structure element is transformed by composing a transformation on the left of the element's transformation: *g[(a,f)]* = (a, *gf).* A structure is transformed by transforming each of its elements individually:
g(cc) = {(a, *gf)* | (a,/) e a}. We denote by (V, F)$^+$ the smallest set that contains the set of singleton structures *{{(a, i)} \ ae V)* and is closed under structure union and transformation by elements of *F.* Informally, (V, F)$^+$ is the universe of structures that can be constructed by composing elements of *V* under transformations in *F.* We obtain the set (V, *Ff* by adding to (V, F)+ the empty structure, 0. It is straightforward to verify that (V, *F)** is the power set of *V* x *F>* and that it is closed under structure union, intersection, difference, complement, and transformation by elements of F.

Realizations

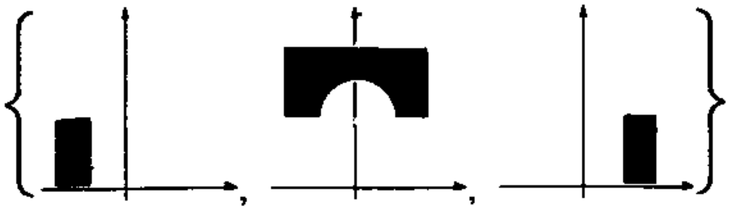
A depiction such as figure 1 necessarily involves mapping a structure's symbols to images and its group elements to transformations of the images. We call such a pair of mappings a *realization* and the result of applying them to a structure a *realization of the structure.* In general, realizations may be defined in terms of the action of any group on a set. Group actions abound in spatial contexts; common examples are the action of translations on the

real number line, the action of affine transformations on the plane, and the action of direct isometries on components positioned in a 3-dimensional space. But since realizations of structures are not restricted to spatial domains, we define realization in a more general sense.

Consider a group *H* that acts on a set 5. We denote the action *ofHonS* by rand the transformation of *S* that corresponds to an element *he H* by Tfc. If *H* is the group of 3x3 homogeneous matrices and *S* is the set of points in the plane, then *Xh* is the transform$ti<5n of the plane corresponding to the matrix *h*. A realization p maps structures in a universe (V, F)* to sets of transformed subsets of *S*. It does so via two mappings: a group homomorphism ft *F* -> //, and a mapping a: *V* -> 2^s from *V* to subsets of *S*. The realization of a structure *a e (V, F)** is the set

P(a) = *{*0(f)[o(*)]* I («»/) e *a).*

Informally, *p(a)* is the set obtained by mapping structure element symbols to subsets of 5, mapping abstract group elements to transformations of S, and applying the transformations to the subsets. In the realization of figure 1,5 is the set of points in the plane; *H* is the group of plane translations; (7 is a mapping of the symbols *c* and *s* to column and span images; 0is a mapping of the group elements *h* and v to plane translations; and *th* and r_v are the translations that position the images in the figure. The realization of the structure is a set of transformed images:



These are combined by taking their union to yield figure 1.

A realization's group homomorphism may, of course, be the identity. It may, for example, be convenient in a structure grammar implementation to use the same groups in structures as are used to calculate their realizations. But by uncoupling abstract groups from particular groups via a homomorphism, we achieve greater purity in the abstract formalism. This raises the question whether structure grammar implementations can in principle work exclusively with abstract groups, mapping them to particular groups only to produce realizations of structures. Whether or not they can hinges on the ability to determine the equality of structures algorithmically. But as a consequence of the decidability of the word problem for groups, a composition of abstract group elements can be reduced algorithmically to canonical form given the generators of the group (see, for example, Le Chenadec, 1986); it follows that the equality of abstract structures can also be determined algorithmically.

Structure grammars

New structures may be derived from old ones by means of *structure rewriting rules.* A rewriting rule consists of a *precedent,* which specifies the conditions under which the rule applies to a target structure, and a *consequent,* which specifies the effect of applying the rule to the target. The precedent divides further into *inclusive* and *exclusive* parts. All three parts of a rule are themselves structures.

A rewriting rule r with inclusive precedent α, exclusive precedent δ, and consequent β is denoted r: $(\alpha, \delta) \rightarrow \beta$. When the exclusive precedent is the empty structure, we also write r: $\alpha \rightarrow \beta$. The rule r is said to *apply* to a structure γ via transformation f when

$$f(\alpha) \subseteq \gamma \quad \text{and} \quad f(\delta) \cap \gamma = \varnothing,$$

that is, when $f(\alpha)$ is a substructure of γ and $f(\delta)$ has no elements in common with γ. Under this condition, r may be applied to γ to derive a new structure ω by removing the inclusive precedent from γ and adding the consequent:

$$\omega = [\gamma - f(\alpha)] \cup f(\beta).$$

Grammar and language

The action of a set of rewriting rules on an initial structure defines a set of structures called a *language*; the rules and initial structure constitute the *grammar* of the language. A grammar is a compact representation of a potentially infinite language of structures; its rules may, in addition, reveal significant aspects of the structure of the language.

Formally, a *structure grammar* G consists of five components:

Vff a vocabulary of *non-terminal* symbols;

VT a vocabulary of *terminal* symbols, with Vy n Vjv = 0;

F a group of transformations;

P a set of rewriting rules whose structures are drawn from *(V, F)*$_9$*

where *V* = V>/ u *Vj* ;

K) an initial structure in (V, F)*.

The vocabulary V and group *F* contain the building blocks of the structures the grampiaf generates. The vocabulary is partitioned into terminal and non-terminal vocabularies in order to distinguish completed structures, which belong to the grammar's language, from intermediate structures, which do not.

To define the concept of language more precisely, we first define two relations on structures in (V, F)*. The relation ^ holds between structures /and *co* whenever a rule in *G* can be applied to /to yield ax We write y^? *co,* and say that *ydirectly derives co* in G. The relation *derives,* written ^?, is the reflexive and transitive closure of ^?. That is, ^? holds between structures *y*and *co* when a sequence of zero or more rules in *G* can be applied to *y*to yield *co.* The *language L(G)* generated by *G* is the set of structures that derive from the initial structure and contain only terminal symbols:

L(G)={cce (V$_T$,F)\yo%<x).-*

The application of these concepts in practice is demonstrated by the following grammar, which was inspired by the stylized sports figures designed by Otl Aicher for the 1972 Olympic Games (Jacob & Kazumie, 1969). Aicher's use of a limited vocabulary of geometric body parts that are combined in canonical ways led to remarkable success in satisfying seemingly contradictory objectives: each of his figures is clearly distinguishable from the others, yet all of the figures belong, unmistakably, to the same family.

The rules of the grammar are depicted in figure 2. The inclusive parts of rule precedent* are rendered in black, the exclusive parts in gray. Rules that share a common precedent are combined into a single diagram with the consequents separated by vertical lines. The grammar's terminal vocabulary comprises the solid elements of the rules — a head, a torso, an arm, and a leg; the non-terminal vocabulary consists of a single "U"-shaped marker that indicates the attachment points of extremities to the torso.

Rule 1 rewrites the initial structure, a torso and head marker, to a torso with a head and markers for the limbs. By removing and adding markers, the rule ensures that the head is attached to a figure before the arms, which is required for the arm rules to function properly. Rules 2 and 3 attach arms, as does rule 4, which uses an exclusive precedent to prevent the attachment of an arm when it would overlap the head. Rules 5 and 6 complete a figure by attaching the legs; the exclusive precedent of rule 6 prevents both legs from being attached horizontally at the bottom of the torso, a configuration that is graphically awkward. The derivation of a running figure is shown in figure 3. The language generated by the grammar is shown in its entirety in figure 4.

Observant readers may ask why figures in the language have limbs on both sides when the grammar's rules appear to attach limbs only to the right sides of torsos. The answer lies in the representation of the torso. We represent the torso as two mirror-symmetric halves; the union of their graphic representations yields the black rectangle shown in the figures. Since the composite torso is mirror symmetric, the rules that attach limbs apply to both halves: to the right half directly, and to the left half via a mirror reflection. Thus one rule suffices to handle both cases. Rather than split the torso in two, we could also have jidfled explicit rules to the grammar to attach limbs to the left sides of figures. But it is generally more elegant to embed the symmetry of a figure in its representation than in the rules that apply to it We use the same technique in subsequent examples without comment

Structure rewriting systems

A set of structure rewriting rules encodes an algorithm that computes over structures. When it is packaged with an initial structure, such an algorithm defines a language of structures. But it is also useful to regard sets of rewriting rules as algorithms in their own right, algorithms that compute relations and functions on the domain (V, *F)**. We call such rule sets *structure rewriting systems.*

A structure rewriting system has all of the components of a grammar except the initial structure. The relations *derives* and *directly derives* are defined for rewriting systems as they are for grammars. We define the *a-language* of a rewriting system *W* as the set of structures that derive from the structure *a* e (V, F)* and contain only terminal symbols:

$$L_\alpha(W) = \{\beta \in (V_T, F)^* \mid \alpha \overset{*}{\underset{W}{\Rightarrow}} \beta\}.$$

We regard W in this sense as an algorithm that computes a function from structures to sets of structures. We may also regard W as computing a relation, the relation $R(W)$ between structures given by

$$R(W) = \{(\alpha, \beta) \mid \alpha \in (V, F)^* \text{ and } \beta \in L_\alpha(W)\}.$$

That is, β is related to α in $R(W)$ when β is in the α-language of W. In some contexts, it may make sense to consider restrictions of $R(W)$ to smaller domains than $(V, F)^*$: we may not want to treat every structure in $(V, F)^*$ as a valid input to the algorithm. When every structure in the domain of $R(W)$ occurs only once as the first element of a pair in the relation, we may regard W as computing a function from structures to structures.

An example of a rewriting system is shown in figure 5. As in the previous example, the inclusive parts of rule precedents are rendered in black and the exclusive parts in gray. The rewriting system enumerates the brick patterns that fill an input grid without leaving voids. More precisely, when α is a rectangular array of cells with a marker-distinguished corner, the α-language of the system is the set of distinct packings of bricks in α.

The rules of the system emulate how a mason might procede, laying courses of bricks from one side of the grid to the other and back again. One such derivation is shown in figure

5(b). **The sole element** of **the** non-terminal vocabulary, a triangular marker, indicates the position of the **next** brick to be laid. Rules 1 and 2 lay horizontal and vertical bricks and advance the marker, rule 3 advances the marker across already-filled grid elements. Rules 4 and 5 detect when the marker reaches the end of a row and advance it to the next higher row. When the marker reaches the end of the top row, rule 6 removes it; the resulting structure is a member of the language of the input grid. If an input grid has an odd number of cells, every derivation will eventually reach a structure that contains the non-terminal marker but to which no rules apply; the language of an odd grid is therefore empty. The language of an even grid contains every pattern of bricks that fills the grid without leaving voids. Some input grids and their languages are shown in figure 5(c)

Parametric structure grammars

Some operations on structures that are useful in practice are awkward or even impossible to formulate with the rule formalism we have described. Consider, for example, a structure representation of Rubik's cube. Each facet is represented by a structure element whose symbol gives its color and whose group element gives its position on the cube. Although in practice the particular configuration of facets on a face does not affect how one rotates the face by 90°, the formalism we have described requires a separate rewriting rule for each configuration. A similar problem exists with structure element transformations. It is not possible, for example, to formulate a rule whose precedent matches the corners of boxes of unspecified dimensions.

To address these shortcomings, we extend the structure grammar machinery by admitting to rules *parametric* structures, structures that may have variables in place of symbols and transformations. Variables are drawn from disjoint vocabularies: one for those that stand for symbols, and another for those that stand for transformations. A parametric rule r: (a, *S,p)* —» *p* has in addition to the usual precedent structures a predicate *p* that tests assignments of values to its variables; the predicate provides a means of attaching constraints to a rule. An assignment that satisfies the predicate is called an *instantiation* of the rule, which we denote by *r:* (a(x), *8(x)) -»p(x),* where *x* represents the tuple of values assigned to the variables. An instantiation is a conventional rule which is applied to a target structure in the usual way. More precisely, a parametric rule r: (a, 5, p) —» /J applies to a structure 7 when

p{x) and *f(a(x))* c 7 and /(#*)) n 7= 0.

Under this condition, *r* may be applied to 7 to derive a new structure *co* given by

$$\omega = [\gamma - f(\alpha(x))] \cup f(\beta(x)).$$

Figure 6(a) shows the rules of a parametric grammar that generates the language of rectangular partitions of a rectangle.[1] It is not difficult to prove by induction on the number of steps in a derivation that the grammar generates every partition; that, ignoring the last

[1] **This grammar is adapted from a shape grammar by Ulrich Hemming (personal communication).**

"cleanup" rule, every partition has a unique derivation; and that the grammar generates no "dead-ends," derivations that end with a structure that contains non-terminals, but to which no rule applies. The grammar is in these respects an efficient representation of its language.

The grammar's rules have empty exclusive precedents. Black and gray rectangles are non-terminal elements; white rectangles are terminals. The derivation of a partition begins with a single black rectangle and ends with a configuration of white rectangles. Each application of rule 1 increases the number of rectangles by one; rule 2 rearranges rectangles to generate topologically distinct partitions. The black rectangle constrains rules 1 and 2 to apply to the upper left corner of a partition; this restriction maintains sufficient control over the partitioning process to guarantee that each partition is generated uniquely. When rule 3 replaces the black rectangle by a white one, it initiates a cleanup phase that replaces the remaining non-terminal gray rectangles by terminal white ones. An example of a derivation is shown in figure 6(b); some members of the language generated by the grammar are shown in figure 6(c).

The simplicity of this grammar derives from its use of parametric rules. By representing geometric quantities as variables, its rule precedents abstract away from the geometries of particular configurations of rectangles to match infinite classes of topologically identical patterns. Similarly, its consequent variables enable single rules to act on patterns of rectangles with an infinity of topologically identical, but geometrically distinct, effects.

Discussion

Efficiency is a central concern in the construction of systems for interactively exploring spaces of designs. For doing design is more than exploring within fixed spaces; it is also exploring among alternative spaces by manipulating the rules that define them (see, for example, Archea, 1987; Knight, 1983a, 1983b, 1983c). Consequently, a structure grammar implementation must be able to map out new spaces quickly as its users explore alternative rule sets. In the domain of production systems, efficient rule application is achieved through partial match techniques that keep track of partially satisfied rule predecents as a database evolves (Forgy, 1982). Structure grammars are sufficiently close cousins to production systems that they should yield to the same techniques.

Realization mappings support two additional avenues of exploration that are common in design: one of exploring alternative components in a configuration, the other of exploring alternative relationships between the components. Figure 7 illustrates both kinds of exploration applied to the arch structure of figure 1. Stiny has rightly pointed out that describing designs in terms of discrete parts limits the ways in which they may be explored (Stiny, 1987). Nevertheless, it is advantageous in some situations to trade this flexibility against the efficiency and theoretical tractability of a symbolic representation. In these situations, structures are appropriate.

Acknowledgements

We are indebted to Prof. Erwin Steinberg for his guidance in the preparation of this manuscript.

References

Archea J, 1987, "Puzzle-making: what architects do when no one is looking" in *Computability of Design*, Yehuda Kalay, ed. (Wiley, New York)

Forgy C, 1982, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem" *Artificial Intelligence* **19** 17-37

Jacob H, Kazumie M, 1969, "Sign Systems for International Events: Munich, Sapporo, Osaka & Co." *Print* **23**:6 40-49

Knight TW, 1983a, "Transformations of languages of designs: part 1" *Environment and Planning B* **10** 125-128

Knight TW, 1983b, "Transformations of languages of designs: part 2" *Environment and Planning B* **10** 129-154

Knight TW, 1983c, "Transformations of languages of designs: part 3" *Environment and Planning B* **10** 155-177

Le Chenadec P, 1986, *Canonical Forms in Finitely Presented Algebras* (John Wiley & Sons, New York)

Stiny G, 1980, "Introduction to shape and shape grammars" *Environment and Planning B* **7** 343-351

Stiny G, 1982, "Spatial relations and grammars" *Environment and Planning B* **9** 113-114

Stiny G, 1987, "Composition counts: A + E = AE" *Environment and Planning B* **14** 167-182