Building and Environment, Vol. 26, No. 1, pp. 61–73, 1991.
Printed in Great Britain.

0360–1323/91 $3.00 + 0.00
Pergamon Press plc.

Searching for Designs: Paradigm and Practice

ROBERT F. WOODBURY†

Most CAD systems operate within a paradigm that makes it impossible for computers to materially assist in design. Yet in the research community, the well-established search paradigm *is technically ready for broader application. This paper describes design search using the devices of set theory and grammars. It constructs a model of* design spaces *as a framework for creating new types of systems, and argues that current professional CAD systems cannot easily adapt the search paradigm. Two approaches to systems that use design search,* potentially exhaustive enumeration *and* spatial grammars *are discussed with reference to existing research efforts.*

INTRODUCTION

FOR MANY YEARS a generative approach to computer-aided design has been quietly developing in research laboratories at universities around the world. Within the research community it has become a paradigm; there are many researchers who accept its premises and methodology and take as their task its further development. Within its bounds there are parallel threads of inquiry that are moved by their own motivations and logic, yet are inherently complementary. The approach considers design as an act of *search* in the realm of *spatial compositions* and sets as its task the discovery of formal and computational machinery appropriate to express and conduct search. It demands rigor and precision, but returns insight and a spirit of exploration.

The *search paradigm* (to give it a name) stands in harmony with, and perhaps partially motivates, the present recurrence of compositional ideas in architecture. The mechanisms provided by the paradigm may even prove to sustain this interest. In contrast, current CAD practice and search share very little in common. Existing CAD systems for architecture have their roots in another approach: the drawing processor. They support drawing, not design. They advance technically to higher speeds and more "intuitive" displays, not theoretically to greater insight into architecture. On the other hand, recent developments in search suggest that we can expect its larger application and dissemination in the CAD industry. The first significant implementations of search based CAD programs are emerging from the research laboratories. These are not just potential tools for industry, they invigorate research by supporting greater exploration of architecture itself.

In this article, the search paradigm is described abstractly as a *design space model*,‡ using concepts from set theory and formal language theory. Into this model specific search systems can be mapped; with it, requirements for new systems can be posed. At its core the model views search as the action of a set of *operators* on a *representation* all being guided by a *search strategy*. When compared to a current understanding of human problem solving the model suggests that a complementary relationship can exist between human designers and computer-based search systems. A hypothetical contemporary professional CAD system is compared against the design space model, and it is shown that the organization of the CAD system presents little potential for integration with the search paradigm. Put simply, new system designs are needed. Two lines of inquiry, mature as research and exemplary of the search paradigm, are described: (1) the *LOOS system* as an advanced development of an approach to layout that emphasizes exhaustive search strategies, and (2) *spatial grammars*, that use constructive rules both to explain existing corpora of designs and to explore new families of designs.

The search paradigm owes intellectual debts to many sources outside of architecture. These include: formal language theory and computational linguistics for grammatical concepts, set theory and graph theory for representations and proof procedures, cognitive psychology for models of human problem solving, and artificial intelligence for representation and search. However, the paradigm is more than an amalgam of these gleanings; it is enlivened by its own logic and is firmly anchored in the field of design.

SEARCH AND DESIGN

There are many accounts of the act of architectural design. Among the most clear and vigorous of these view design as puzzle-making [5], as ill-structured problem solving [6], as an evolutionary process [7], and as decision making [8]. All of these seem to have at their core a sense of form-making that is constructive in its essence. In computational terms such construction is *search*.

Search characterizes design as a path planning problem through a space of possibilities. It presents a broad and

† Department of Architecture/Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.

‡ [1–4] contain other general accounts of the search paradigm.

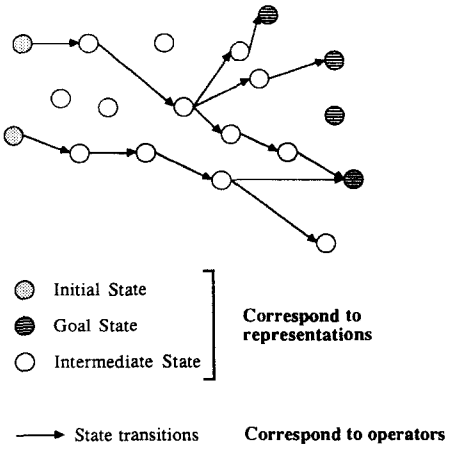

Fig. 1. A generic diagram for search.

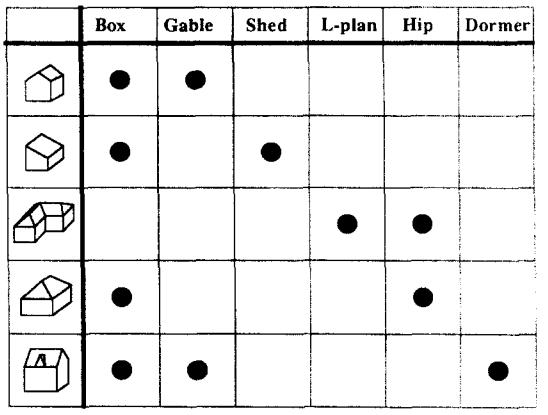

Fig. 2. An abstract representation scheme.

flexible metaphor for design that provides insights into both the act of designing and the structure of designs. It has significant empirical support as the principal mechanism used by human designers. There exists in computer science a well-developed understanding of search and some powerful tools for describing, simulating, and conducting search. Each of these characteristics alone provides, for me, ample reason to seriously consider search as a basis for computer-aided design systems. That they all exist, within one paradigm, is convincing indeed.

If design is search, then design problems have a structure and the act of designing is a process on that structure. Informally and as shown in Fig. 1, design problems consist of a set of *information states*, divided into *initial*, *intermediate*, and *goal states*, and a set of *operators* that move between those states. Each of the states *represents* some design, possibly incomplete. Designing in its simplest form consists of finding a set of operator sequences (or paths) between initial and goal states. More typically in architectural design, any part of the structure may be partly undefined and it is left to the design process to discover that structure. For example, it is common that the program for a building (the initial state) undergoes significant change as building design progresses.

Since my interest is in developing computational support for the search paradigm, it is useful to be more formal. I will begin with a description (following Requicha [9]) of the states of design, particularly how they fulfill their role as *representations of designed objects*. Then I will describe the operators of search (partly following Newell and Simon [10]) and finally how these components are brought together into a single conceptual structure that generically describes design search.

In the following text I have taken care to be precise (yet still abstract) in the presentation of various properties, both *formal* and *informal*. Essentially a formal property of an object is one that can be stated mathematically, in terms of a mathematical description of the object. Informal properties can be stated only imprecisely. The properties I describe are not the only ones that can exist, nor are they of interest for every type of search. They do, however, present an overall picture of the issues that arise when search is taken seriously as a metaphor for design.

Design search, whether done by human or machine, acts on data, not physical objects. These data are finite *symbol structures*, that is, they are collections of related atomic elements called symbols. The collection of all symbol structures that may be considered in a particular design task constitutes a *representation space* R and each member of R is called a *representation*.

Elements in representation space may have interpretations as one or more designed objects. In order to be precise about these interpretations it is necessary to have a precise characterization of designed objects. The concept of a mathematical *modeling space* M is employed for this purpose. A modeling space is an abstract description of some properties of all members of the space. For example, a building may be described in plan as a collection of non-overlapping rectangles, and a modeling space for buildings may be all sets of non-overlapping rectangles. A modeling space captures properties of designed objects. If these are useful properties that allow us to understand the character and utility of a design then the modeling space will be useful. Modeling spaces are used to describe (and discover) properties of designs, for example, the mathematical conditions that guarantee that elements in a set of rectangles do not overlap. It is important to understand that modeling spaces capture only properties of a physical object relevant to a particular design task, for example, a set of rectangles may describe the relative positions and dimensions of rooms, but not the materials, textures, lighting, or construction of an actual building.

By building pairwise associations between elements of modeling and representation spaces, the *semantics* of representations are defined. Formally, these are described as a representation scheme, a relation $\Theta: M \rightarrow R$ on the sets M and R. Figure 2 informally demonstrates a representation scheme for the massing of houses that simply uses elements from a set of descriptive words as the representation space.

This description of a representation scheme would be sufficient if both of the sets M and R were available to us in their entirety. We could then simply construct the Cartesian product $M \times R$, and select directly from it the appropriate pairings of models and representations.

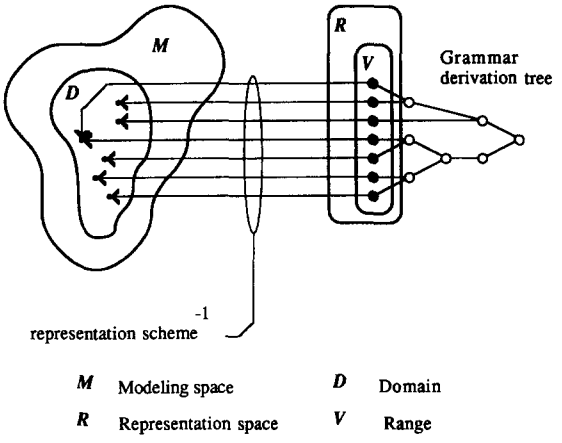

Fig. 3. A more operational depiction of a representation scheme.

Sadly, M and R can seldom be so expressed, and for quite different reasons.

The set R contains those computational objects that will be generated and manipulated by a design process. In interesting design problems, the number of alternatives that could be considered is large enough to defeat any exhaustive enumeration (it could easily be infinite). Even if it were practically possible to directly enumerate all of the alternatives, it would be quite inefficient, therefore undesirable. The set R is typically (but not always, as will be shown in the discussion of LOOS later in this paper) specified directly, as a grammar (informally a set of primitive symbols with syntax rules for their combination) or as an algebra (informally a set of objects and a collection of operators). All members of R are generated by grammar rules (alternatively, by algebra expressions) and are therefore *syntactically correct.*

The set M is not specified in advance; indeed, the generation of some of its members is the task of design. Further, the set M is usually existentially characterized, that is, it is a mathematical abstraction of some of the properties of the actual object. For example, in a modeling space of non-overlapping sets of orthogonally oriented rectangles it is not important to physically generate all such sets of rectangles, it suffices to know that for each pair of rectangles an overlap cannot occur simultaneously in both of the principal directions.

Figure 3 shows a different way of defining Θ; it takes the form of its inverse relation Θ^{-1}, for mapping members of R to members of M (such members of R are said to *model* or *represent* members of M). It is useful to described Θ^{-1} as a *characteristic predicate*,

$$P(r) = \begin{cases} 1 & \text{if } \Theta^{-1}(r) \in M, \\ 0 & \text{otherwise} \end{cases}$$

that determines if a symbol structure in R represents an element of M. This predicate can be thought of as a test that can be implemented as computer program.† As it would be impossible to implement a computer program to test any $m \in M$, Θ is defined as $\Theta^{-1^{-1}}$. The domain of Θ, denoted by D, is the set of all elements of M that have corresponding elements in R. The codomain of Θ is R. The range of Θ, denoted by V, is the set of all members of R (by definition syntactially correct), that correspond to elements in D.

With these preliminaries in place it is possible to more formally describe certain properties of a representation scheme, namely: *extent of domain* (*expressiveness*), *syntactic validity*, *well-formedness*, *completeness* (*unambiguousness*), *uniqueness*, and *abstractness*.

When compared to the entire modeling space M the size of the domain, D, of a representation scheme is a measure of the descriptive power of the scheme. D is that part of the modeling space that is accessible by construction of representations in R. If $D = M$ the representation scheme is *semantically exhaustive.*

Every element of V (the range of the representation scheme) is considered to be *valid*, as it is both syntactically and semantically correct (i.e. it can be constructed by the rules that define R and has corresponding elements in D). If $V = R$ then the representation scheme is *syntactically valid*, as every syntactically correct representation corresponds to an element of D.

If, in addition to syntactic validity, a representation scheme is semantically exhaustive, then the scheme is *well-formed.* A consequence of well-formedness is that the characteristic predicate of Θ^{-1} is always TRUE. With a well-formed scheme, it is theoretically possible to generate a representation that corresponds to an arbitrary member of the modeling space, using only the syntax rules that define the representation space. As we shall see in a later section, well-formedness is an essential quality for exhaustive search strategies.

A representation $r \in V$ is *unambiguous*, or *complete*, if it corresponds to a single element in D. It is *unique* if its corresponding objects in D have no other representations in V. Intuitively a valid representation is ambiguous if it models several objects in D, and an object in D has non-unique representations if it corresponds to more than one element of V. A representation scheme is unambiguous, or complete, if all members of its range are unambiguous. Similarly, a representation scheme is unique if all members of its range are unique.

Related to unambiguousness is the property of *abstractness.* In constructing representation schemes for design it is very useful to keep the size of either or both of the representation space R or the range V of the representation scheme as small as possible. This implies a smaller space to search. Given a particular modeling space M, a common way of achieving this is to introduce a kind of controlled ambiguity into the representation scheme. Figure 4 provides an example, the LOOS system [11] (discussed in more detail later), in which the modeling space is the set of all arrangements of loosely packed, non-overlapping, orthogonally orientated rectangles in two dimensional euclidean space $\mathbf{R}^2$. The representation space consists of a set of graphs, where the nodes of the graph denote rectangles and the arcs denote the spatial relations *to-the-right-of*, *to-the-left-of*, *above*, and *below*. Each graph in the representation space represents an entire class of rectangles in the modeling space that differ in the dimensions and locations of the constituent rectangles but are the same with respect to the spatial

† Remember, the only things that we can compute upon are members of R, therefore all computable tests must be written in their terms.

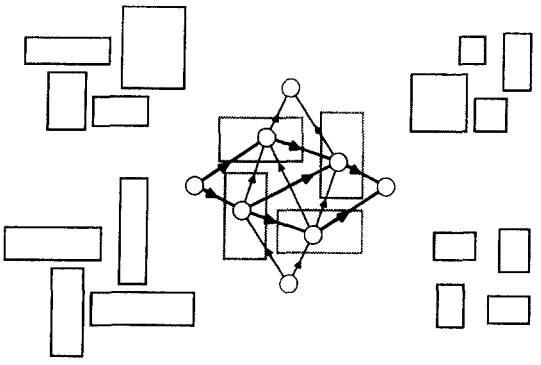

Fig. 4. Ambiguity in the representation scheme of LOOS.

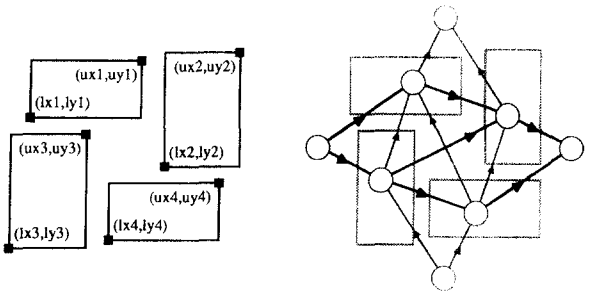

Fig. 5. Abstractness as a property of a modeling space.

relations specified in the graph. Thus the representation scheme for LOOS is decidedly ambiguous, as every representation corresponds to an infinite set of rectangles, yet the ambiguity is precisely controlled, since specific spatial relationships are faithfully modeled. This feature is used in LOOS to divide layout problems into two parts, corresponding to discrete and continuous spatial variables, only the first of which uses search as it is described in this paper.

In some cases abstractness may be separated from unambiguousness by considering it a property not of the representation scheme, but of the modeling space. Figure 5 shows that, in LOOS, if the modeling space M is considered to be the set of all arrangements of *generic rectangles*, that is, rectangles whose vertex coordinates are expressed as variables rather than numbers, and whose spatial relations are specified as equations, then the representation scheme could be unambiguous with respect to M. In this case all of the abstractness in the system has been transferred to the modeling space itself.

Another, less formal, way of looking at the concepts of abstractness and unambiguousness is to employ the ideas of *instance* and *class*. An instance is a single object and unambiguous representation schemes can be said to model instances in modeling space. A class is a group of objects and an abstract scheme models classes, where all instances in a class have some (hopefully relevant) common properties. With the ideas of instance and class another relation, the *same-class* relation $\Psi : M \rightarrow M$, can be constructed between elements of the modeling space. Two objects are in Ψ if both correspond to the same representation in V. If Ψ is an equivalence relation† then all representations in V unambiguously denote *blocks* (alternatively *pieces*) of a *partition* of M.

The formal properties of a representation scheme can be used to describe properties of the search operators that act in representation space. Remember that search is a process of applying operators that move between design states (points in representation space). For any design search problem a set of operators, denoted O, is needed and these are best described in the context of another formal construct, the *search space*.

A *search space* S is comprised of a modeling space M, a representation space R, a representation scheme Θ, a set of operators O, and a set of initial representations $I \subseteq R$. An *operator application* within S consists of an operator from O applied to a representation from R. More formally, when an operator from O is applied to a representation $r \in R$ to yield another representation $r' \in R$, then r is said to *directly derive* r' in O, or symbolically $r \underset{O}{\Longrightarrow} r'$. If there exists a sequence of direct derivations using operators from O, such that $r_0 \underset{O}{\Longrightarrow} r_1 \underset{O}{\Longrightarrow} \cdots \underset{O}{\Longrightarrow} r_n$ then r_0 *derives* r_n in O, $r_0 \overset{*}{\underset{O}{\Longrightarrow}} r_n$.

Operators in a search space may individually or as a set have the properties of *closure* and *monotonicity*. Other properties of search space operators, *completeness* and *non-redundancy* are defined only on the search space itself. Another formal property of search spaces, *fixed order*, exists, though it appears specific to a particular search strategy (that used in LOOS).

An operator is *closed* in V if its application to any member of V can never result in a representation not in V. A search space is closed in V if all of its operators are closed in V. If operators are closed, and begin from elements in V, then the characteristic predicate of the representation scheme, $P(r)$, need never be applied to test for representational validity.

An operator can be *monotonic* with respect to both properties in M (*modeling space monotonicity*) and to the symbol structures in R (*representational monotonicity*). If a set of properties P_M of any $m \in M$ cannot be altered by the application of an operator, then the operator is monotonic with respect to P_M. If an operator can only add to, but otherwise never alter, the symbol structures of R, then it is monotonic with respect to R. A search space is monotonic in either sense if all of its operators are monotonic in that sense. These two types of monotonicity are quite different and do not imply each other.

A search space S is *complete* in the domain of its representation scheme Θ if, by application of any combination of operators from O starting from any elements of I, representations in V sufficient to model all of D can be reached. If Θ is semantically exhaustive and the operator set is complete in D, then the operator set is complete in M. Informally, completeness in M means that every conceivable solution can be reached by some sequence of operator applications from O. A search space S is *non-redundant* if there is at most one sequence of operator applications beginning from $i \in I$ that can generate any $r \in R$.

A search space exists in the absence of any specific

† An equivalence relation is *reflexive*, *symmetric*, and *transitive*.

design context; it is simply a description of possibilities. By applying operators beginning at initial states it is theoretically conceivable that one might eventually *visit* any design within the space. But such unguided wandering is unlikely to be interesting. To pursue design requires more: a way to choose operator applications, a sense of where in the space one wishes to go, and a means of knowing when one has arrived at a goal. These are accomplished by a *search strategy*.

A search strategy is a policy; a way of making decisions. Under its guidance it is possible to move through a search space purposefully, visiting new states and remembering or forgetting them, that is, making them *active* or *inactive* until an appropriate design is found (or is not found). Each visitation of a state is called a *step* and typically occasions four types of decisions.

1. Is the design problem solved?
2. Which from among the active designs will be *selected* next?
3. Which search space operator will be *applied to* the selected design?
4. Which of the active designs will remain active? (Which will be made inactive?)

To make these decisions requires two additional components in a search strategy: *design goals* and *evaluation devices*. Design goals are statements of intent; they describe in some way the characteristics possessed by a successful solution. They may be precise, for example, the program for a speculative apartment building may require a total of 200 units evenly divided between 1 and 2 bedroom apartments. Alternatively they may be quite vague, for example, a design review board may require the same apartment building to conform to a very loosely stated set of visual criteria. As suggested by the above example, design problems have multiple and often conflicting goals. Designs are compared against goals as they are reached by the operators and these comparisons are used in making the decisions at each step in design.

Comparing a design against goals brings to the surface what is both a central feature of the search paradigm and a reason for our interest in it: *designs are composed with one set of variables and evaluated with another.* In architecture (and in other disciplines in which geometry is a prime design issue) this appears inevitable. At least with present understanding, designs are created by specifying their physical properties, that is, by developing the form that an object will take and the materials (and perhaps processes) that will be used to build it. *Design variables* thus specify physical objects. On the other hand, designs are of real interest to us only when we posit their interaction with an environment (which in the case of buildings includes their users). By simulating a design in its context we understand its *performance variables* each providing a measure according to some *criterion* or point of view. To make matters worse, the relation between design and performance variables is *many-to-many*. One design variable may affect a number of performance variables; a change in a window location might affect view, ventilation, and facade balance. On the other hand a single performance variable may require many design variables for its computation; heating demand is a function of building size, window area and orientation, construction, use, etc. Thus we must generate before we can test; we must have a design before we can critique. The search paradigm, by separating generation (operator application) and evaluation (design step decision making) fits well with this seeming inevitability.

To understand performance, a design is tested (according to various criteria) against its predicted context. A set of such tests, one for each criterion, together with a means for understanding their collected results constitutes the evaluation devices of a search strategy. The tests alone are not enough, for designs perform according to many different criteria, and these cannot be treated separately. It is commonly the case that one performance measure conflicts with another, for example, that it is impossible to improve a view without increasing heat loss. Making decisions in the face of these conflicts requires an understanding of possible tradeoffs and ultimately judgements of relative value [8, 12].

Many different search strategies have been developed over the years in cognitive science and artificial intelligence. These are lucidly explained in numerous publications [13–16]. I do not treat these in detail here, rather later in this paper I introduce and compare two exemplary approaches that must use very different search strategies.

With search strategies our portrait of the search paradigm is complete. To search requires a space and a strategy. A search space is composed of a representation scheme, a set of search operators and a starting point. It provides an implicit specification of a world of possibilities. A search strategy is a decision making policy and its associated machinery. It provides a means to move purposefully through a search space. To coin one last term, a *design space* consists of a search space and a strategy.

It is only with the concept of a design space in place that a crucial idea can be introduced. A set of operators in a design space is *semantically relevant* if its members correspond to "meaningful moves" in design. For example, in beaux-arts composition a set of operators that supports the creation of axes, cross-axes, symmetrical placement of facade elements, proportioning between elements, operations on poche, etc. is much more semantically relevant than a set that describes how to construct a building from its physical components (steel members, blocks, bricks, etc.). As suggested by the example, semantic relevance depends on not just the search space, but also the goals and evaluation devices of the search strategy.

DESIGN SPACES, HUMANS, AND COMPUTATION

Traditionally, architects have worked "on the boards". Now more and more work on machines. I will argue here that this presents a difference of kind, yet currently only a difference of degree has been accomplished in the professional mainstream. To do so requires a brief look at human cognition and computer organization.

Figure 6 presents a well-known and empirically validated model of the human information processing system (IPS) [10, 17]. Human thought is modeled as a number of processors and memories linked together in a constant

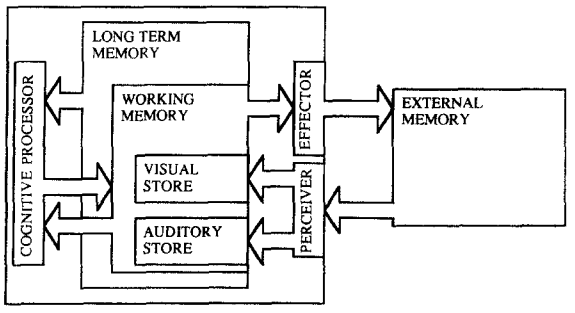

Fig. 6. A model of human cognition.

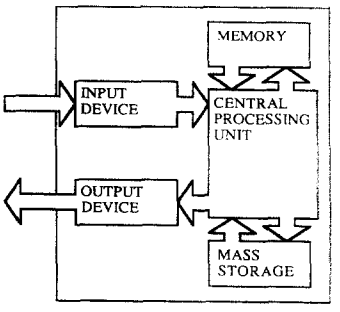

Fig. 7. A schematic computer organization.

structure. Each processor has a characteristic *cycle time*, or interval in which it can complete a simple information process. Each memory† is described by five parameters: storage capacity (the number of items that can be simultaneously stored), decay time (the time it takes a remembered item to be forgotten), read time (the time it takes for the human processor to access a memory element), write time (the time it takes to store a memory element), and code type (which can be physical, visual, acoustic, or semantic). Included in the model is an *external memory* that is actually outside of the human body yet is essential to any explanation of human behavior in a complex task. Books, drawings, charts, and computers can all act as external memory. Several properties of the cognitive model are of interest here:

- *Seriality.* It is empirically incontestable that human thought is serial. Humans do one thing at a time; the time it takes to do any cognitive task increases proportionately to the complexity of the task.
- *Speed.* The human processor is, by computer standards, depressingly slow. A typical mean speed for a single information processing task is 70 ms, several orders of magnitude slower than modern computer systems.
- *Rule-like processor behavior.* Production systems are sets of rules with an automatic recognition and activation mechanism. A production system model of the human processor appears to fit well with observations of human performance.
- *Goal-oriented processor behavior.* Humans act towards achieving goals, and such behavior surfaces in many ways. In particular goals organize problem solving and their level of satisfaction is a large determinant of behavior.
- *Short term memory (STM) parameters.* Two main classes of internal memory, short and long term, exist. Short term memory is very limited in capacity and has very short read and write times compared to long term memory. It and that part of external memory in direct foveal view are effectively the only memories that the human processor can immediately read. The short term memory itself is the only memory to which the processor can quickly (relative to cycle time) write.
- *Long term memory (LTM) structure.* Long term memory is an effectively infinite, associative memory. By associative is meant that memory items are stored as an interlinked network, the nodes being items and the arcs being relations. Memory is accessed by following the arcs from one item to another. Items in LTM are not forgotten, though the arcs connecting them to other parts of the network may disappear. A crucial characteristic of LTM is its write time which is about 5–10 s of processor effort.
- *External memory (EM) structure.* External memory is also effectively infinite, with highly variable organization, decay period, and read and write times. It is the only part of the IPS that is subject to structural change and its characteristics can greatly alter human performance on a task.

Comparison of the human cognitive model with the search paradigm yields several provocative observations:

- Short term memory bounds effectively constrain the types of search in which unaided human problem solvers can indulge. In particular: the systematic exploration of design problems would appear to be extraordinarily difficult, and search strategies that involve saving large numbers of states are effectively impossible.
- Human goal orientation as well as processor and memory limits make the simultaneous consideration of a wide range of design concerns difficult.
- Designed objects are complex and their descriptions correspondingly so. Human designers have an absolute reliance on external memory to record and manipulate these descriptions.

Figure 7 demonstrates a second model; a highly simplified view of computer organization‡ that I expect will be quite familiar to all readers. In this model, a computer has a single processor, two different kinds of memory, and separate input and output devices. Like the model of human cognition, each of these is characterized by a set of parameters: speed for the processor and capacity, read and write time for memory. This model describes at some level almost all of the machines used in the building field today, new developments in parallel processing and massively parallel computers notwithstanding. Several properties of this model are of interest:

- *Seriality.* Like human cognition, computers of this type are inherently serial devices. Multiple processes

† Cognitive theories differ in the location of working memory. For present purposes these differences are irrelevant.

‡ Also known as computer architecture; from the point of view of the readership of this journal an unfortunate choice of wording.

can exist at one time within a machine, but these are executed piecewise consecutively by the central processing unit (CPU).

- *Speed.* Current CPUs are blindingly fast when compared to a human processor even when the great difference in elemental operation complexity is acknowledged. As of September 1989, speeds of 15 millions of instructions per second (MIPS) are available on moderately priced workstations.
- *Processor behavior.* An array of different programming language paradigms is available for computers, allowing these machines to behave in many different patterns, most not matched by normal human cognition. Great precision of behavior is the norm but must be accomplished by rigorous programming. Current understandings of machine learning, adaptive and goal-oriented behavior are in their infancy, and such behaviors will not be seen soon outside of research laboratories.
- *Memory structure.* That two memories exist in the model is a technological and economic artifact. *Primary memory* must have sufficiently fast read and write times to keep up with processor speeds, but is only needed for data and programs that must be immediately available to the processor. Currently, relatively expensive silicon chips are the most common technology. *Secondary memory* is provided by less expensive devices (typically disks). With modern virtual memory techniques, the existence of secondary memory is hardly noticed by most users. Available memory capacities are large, with tens of megabytes of primary storage and gigabytes of secondary storage available for moderately priced workstations.

When compared to the search paradigm this model of computation yields observations complementary to those for the human cognitive model:

- Search spaces for real design problems are vast and increase in size exponentially as the complexity of design problems increases. No amount of processor speed and memory capacity can defeat this fact.
- Memory is quickly available in large quantities to a computer processor, permitting search strategies that require large (but in practice not exponential) memory resources.
- Accomplishing design search with a computer processor requires an implementation of a design space as a computer program. Theoretical rigor is a necessity. Part experience indicates that such programs are likely to be large and complex.†

When the two models are put together, such that the outputs from one are attached to the inputs of the other, a model of a single entity appears. As shown in Fig. 8, the human is connected to the machine and the machine to the human. Each can perform part of the task and

† Such programs have been usually written in imperative languages that are not "natural" to the search paradigm. Perhaps the "right" language would go a long way to making search implementations simple.

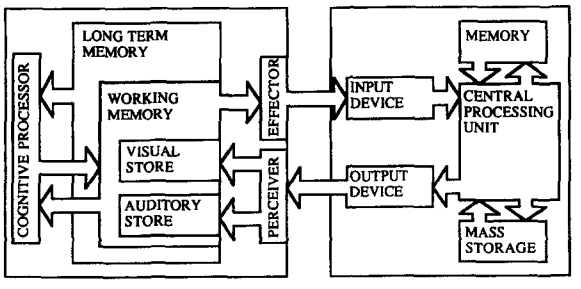

Fig. 8. Human and computer in a single model.

communicate its results to the other. It seems that something akin to a "symbiosis" is possible, with each part assuming those parts of the task for which it is most suitable. Such a vision is far from new and has been central in the growth of computation in society, but so far it has hardly been realized in architecture. Current systems are founded on theory that inevitably leaves all of the search strategy on the human side of the model.

If computers are to perform any of the search, then a realization of at least part of the machinery presented above must exist on the computer side. The entire machinery is not necessary and is not even desirable for some approaches. At a minimum, the operators and means for their automatic invocation seem to be required. Current theory and implementation in research labs has achieved some of this capability, yet almost none is in evidence in professionally available systems. Until it is, at least one of the potentials of computation in architecture will remain unrealized.

CURRENT CAD SYSTEMS AND SEARCH

It is interesting to analyze existing CAD systems within the models just presented. Consider a typical CAD system of the type that is currently used for working drawing production. An actual system could be chosen for this comparison, but that both is unnecessary and would be unfair to the system, as all such systems have much common underlying structure. Call this hypothetical system *Archidraw*.

Archidraw works by allowing users to enter, delete, and modify data that denote *geometric entities* such as points, lines, polygons, curves, etc., in 2 or 3 dimensions, and associated properties of those entities. It maintains an internal representation of these entities that typically corresponds exactly to operations used to create the data. Thus a line segment that is entered as three separate sections joined end to end is stored as three entities, not as a single geometric entity. Geometric entities may be grouped together with two mechanisms, *layers* and *groups*. Every geometric entity belongs to precisely one layer. Layers are typically used to separate information pertinent to different subdisciplines in design. For example, it is usual to store structural design information on a different layer from electrical wiring. Groups are different creatures altogether; they are used to create sets of geometric entities that are used repeatedly in a design. For example, a group containing line segments that "represents" a door may be used wherever in a design a door is required. Groups may in turn contain groups;

a group "representing" a bathroom may contain subgroups for the tub, sink, and toilet. Layer information is preserved within groups, that is, every geometric entity used in a group belongs to a possibly different layer. Archidraw has representation support for the so-called *non-geometric data* that is used to attribute geometric entities with properties such as line type, color, manufacturer, cost, etc. Finally, dimensions may be represented as distances (either horizontal, vertical or direct line) between any two points or as angles between any three points. Dimensions in Archidraw are *associative*; when the points upon which dimensions depend are moved, the values stored in the dimensions are automatically updated. These devices, geometric entities, layers, groups, non-geometric data, and dimensions are the only means that Archidraw has to store and organize its information and may be called its *representation*.

Acting on Archidraw's representation are a set of operators that allow the user to explicitly construct geometric entities, to compute relevant geometric properties (such as line center-points and intersections), to assign geometric entities to groups and layers, to enter non-geometric information, and to select dimensions. Archidraw has a wide variety of different commands that are constantly being changed and upgraded by the system's designers. All of these commands are available to the user through an interactive interface that makes their application quick and "intuitive".

By themselves the commands are limiting because they are at a very "low level"; a large number of them may be required to perform an operation that "makes sense" to a designer. Archidraw thus has a *macro language* that is used to collect comments together into a logical structure. This language supports some of the features of imperative programming languages such as C and PASCAL, most notably subroutines and iteration.

It is easy to see that Archidraw leaves the human with all of the burden of design. It provides neither representation scheme, operators, nor search strategy, and cannot easily be coerced into doing so.

The representation of Archidraw is that of drawings; essentially of lines that can be combined with a set of drawing operations, thus it is a representation space. It lacks a clear semantics as no precise (and therefore implementable) representation scheme, or mapping from objects to the things that represent them, can reasonably be claimed. Although humans are adept at interpreting drawings as design concepts or physical objects,† attempts to give computers similar capabilities remain less than convincing. This by itself is not fatal, indeed the most extensively published design search theory, shape grammars, has this feature, but in its absence the evaluation devices of a search strategy must be explicitly constructed [18].

Humans using the operators of Archidraw can become quite adept at creating drawings, but in the absence of a representation scheme, operators can have few meaningful formal properties. With respect to the crucial yet informal property of semantic relevance, I would argue that these essentially mark-making operators are of little significance in design; they are merely a recording device for the relevant operators that remain implicit in the human brain. Of equal significance is the inability of Archidraw to recognize when and where its operators can be applied. This is death, and it is eloquently argued by Stiny [19] that current CAD data structures cannot accommodate matching that meets our spatial intuitions.

Archidraw has no search mechanism; it provides a largely passive repository for input information. To be sure, clever input strategies and display operations exist, but these are only devices to place and review already known data. This last shortfall is the most damning of all, for without a search strategy, however partial, design all remains up to the human.

If current CAD systems, of which Archidraw is exemplary, hold out little promise that they can be adapted to the search paradigm, then what can be done? The answer is "Much". In the next sections I will describe two quite different approaches within the search paradigm, tracing a bit of their history, sketching their organization and theory, and describing their current status. No attempt at coverage or classification of all of the extant research work is made. Others [2, 20] have done so at various times, and I shall not repeat their work.

The LOOS system

LOOS, by Flemming [11, 21], is a system for generating designs that can be meaningfully approximated by a set of non-overlapping rectangles. It is exemplary in the present discussion in two regards: (1) it implements a search strategy that appears to be a high water mark of an exhaustive approach to search, and (2) its search space, upon whose properties the validity of the search strategy depends, is both highly formalized and intuitively appropriate.

LOOS stands within a paradigm for research on formalizations for geometric layout that began with [22, 23], and has continued its development through [24–27] to [11, 21]; these last two are the primary literature for LOOS. The essential character of the paradigm lies in a separation between so-called *qualitative* and *quantitative* variables. Qualitative variables make discrete (and hopefully semantically meaningful) distinctions between designs; the spatial (or topological) relations between parts are important examples. By making these distinctions, an uncountably infinite world of designs may be divided into a large, but countable, set of *classes* of designs. All of the designs in each class are partially described by the same set of qualitative variables. But note that this must be a partial characterization, for designs within a class are not the same, and it falls to the quantitative variables in each class to express these differences. Quantitative variables describe the continuously varying features of designs; dimensions and location are typical examples. Within a class of designs, the qualitative variables impose certain mathematical relations on the quantitative ones, and these relations precisely describe the bounds of variation allowed by the class. Both the imposed relations and the quantitative variables (not just their bound values), differ between classes. The paradigm calls for a *formalized representation* for qualitative variables, and in such formalizations we find instances of search spaces.

† And at making creative use of ambiguities that arise in the interpretive process.

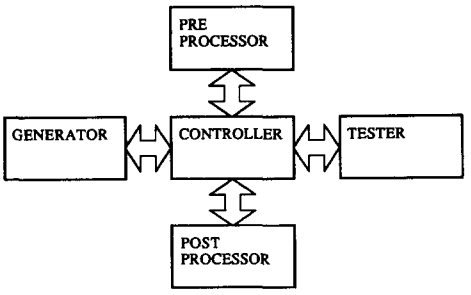

Fig. 9. The LOOS system architecture.

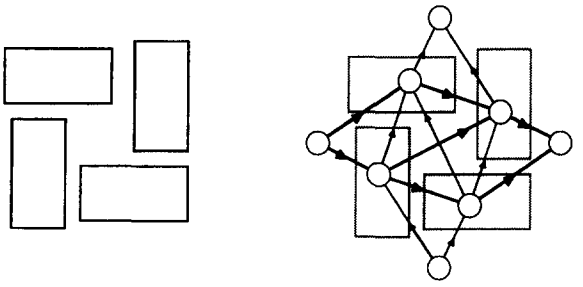

Fig. 10. A loosely packed arrangement of rectangles and a corresponding orthogonal structure.

The organization of LOOS as a system, diagrammed in Fig. 9, provides a clear conceptual separation between design space components. Each of the five parts of LOOS contributes meaningfully to system behavior. Two of them, the generator and controller, are central to this discussion and will be given greatest emphasis.

The generator realizes the search space of LOOS. Its modeling space is the collection of all sets of non-overlapping rectangles in $\mathbf{R}^2$. Such rectangle sets are called *loosely-packed arrangements of rectangles* (*LPARs*). Its representation space is the set of directed, arc-colored (from the set $\{R, A\}$) graphs (*orthogonal structures*) that satisfy a precisely stated set of conditions on the interconnections of their nodes.† Informally, these conditions state that: (1) any two arbitrary nodes in the graph (excepting one specially identified *external node*) must be joined by only one type of uniformly colored path, (2) the external node is joined to every node by paths of both colors, and in both directions, and (3) each of any two parallel paths contains at least three vertices. An orthogonal structure represents an LPAR (and the [LPAR, orthogonal structure] pair is thus in the representation scheme) iff there exists a one-to-one correspondence between nodes in the orthogonal structure and rectangles in the LPAR such that an R or (A)-colored arc between two nodes corresponds to a *to-the-right-of* (or *above*) spatial relationship between two rectangles (see Fig. 10). Proofs exist [11, 21] that the representation scheme is well-formed. It is certainly not unambiguous with respect to LPARs, but it is abstract and this is required by the layout paradigm. It is not a unique scheme as there exist multiple representations of certain LPARs.‡

The generator implements a set of operators on orthogonal structures that have been proven [11, 21] to make the search space closed, complete, modeling space monotonic with respect to spatial relations from {*to-the-right-of*, *to-the-left-of*, *above*, *below*}, non-redundant, and to have fixed-order.§ These operators produce from *parent* orthogonal structures their *children*, orthgonal structures with one node more than the parent. Given any orthogonal structure as input, the generator produces as output all possible *children* of the structure.

Figure 11 shows that the LOOS operators fulfill the roles of both the representation space construction operators and the search space operators in the design space model. They can do this as the search space is intentionally blind to any domain; it captures only the properties of layouts of rectangles. This would seem to imply that the operators cannot be particularly semantically relevant in a design space that uses LOOS. Both strong advantages and disadvantages accrue to this fact as will be discussed below.

The controller implements the search strategy of LOOS. The LOOS operators are exhaustive in the domain of sets of rectangles. Left blindly to their own devices they would generate vast numbers of layouts (for any but the smallest of problems), all but a vanishingly few layouts utter nonsense as solutions. The controller's basic function then is to efficiently guide the search for alternatives. It cannot predict directly from which alternative among a developing set a promising design might emerge, for that would appear to demand capabilities that are currently not understood. It can, and does, *prune* those alternatives that are guaranteed to lead to failure. In doing so, current versions of LOOS use a *branch-and-bound* search strategy. In branch and bound, only and exactly those states are expanded that have a score at least as good as any other generated before. Scores are determined by the tester which is described below. States that have worse scores are never touched again (and are thus pruned from the search). To allow consideration of a wide range of criteria the evaluation function of LOOS is structured as a triple:

$$\mathbf{t}(s) = \langle c(s), d(s), e(s) \rangle,$$

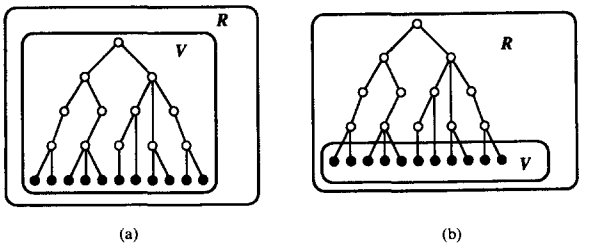

Fig. 11. The operators in LOOS (a) are contained completely within (and define) the range of the representation scheme V. In spatial grammars (b) only members of the language of the grammar are in V.

† Note that the representation space of LOOS is initially specified existentially, not as a grammar. This presents no real problem for our account of representation schemes as a constructive grammar is developed by LOOS's theory.

‡ Non-uniqueness has necessitated small accomodations in the implementation.

§ The fixed-order property states that there is a sequence of rule applications that will generate any graph under an arbitrary order of insertion of its vertices.

where $c(s)$ is the number of constraints, $d(s)$ the number of "strong" criteria and $e(s)$ the number of "weak" criteria violated by a state, s. $\mathbf{t}(s)$ is called the *score* of s. Constraints, weak criteria, and strong criteria support a subjectively designated categorization of goals in a design problem. The controller ranks states *lexicographically*, preferring scores for which $c(s)$ is minimal; when a tie on $c(s)$ exists, a minimal $d(s)$ is preferred, and so on, ...

The controller works precisely because the search space of LOOS is modeling space monotonic. All of the tests to which a state is subject depend on information present and deducible from the representation of the objects and spatial relations in a state. Since knowledge of these, once inserted, can never be retracted by any of the operators, the evaluation function can never return a score for a state that is lower than its parent's score. Thus correct pruning can occur by looking at the parent alone.

The tester contains the evaluation devices of LOOS. LOOS's generator and controller are both domain-independent in that they contain no architectural knowledge beyond the minimal geometric conditions of non-overlap. The other three components contain domain knowledge and must be rewritten for each new domain. Currently the domains of bathroom, kitchen, and high-rise service cores have been implemented. A tester for a domain is a set of *test rules* that can be applied to a partial design (provided the appropriate objects have been instantiated into the state that represents the design). The controller applies all test rules that it can to determine a score for a state.

The strict syntactic nature of the generator permits all of the domain dependent knowledge about designs to be isolated from it and largely placed in the tester. As test rules are independent they can be incrementally modified and new ones can be easily added. This allows expert system techniques of knowledge acquisition to be readily applied.

The pre-processor provides initial states to LOOS. For each specific design problem an initial context is needed, for example, the walls that bound a space. The pre-processor provides an interface between initial contexts stated in terms of a specific domain and the LOOS representation. Once a state is in place LOOS can begin. Note that the pre-processor presents a minor variation of the design space specification, in that it imposes a starting point other than the initial states I of the search space. It can be viewed as providing a very hard constraint, that no states in the space that do not at minimum have its structure can possibly succeed.

The post-processor is a human–computer interface for LOOS. Orthogonal structures are not graphic objects, and must be interpreted to a visual display. In addition, the controller does not necessarily allocate every object for a problem. Some trivial ones are left to be inserted directly. These are the two functions of the post-processor.

As an implementation of a design space LOOS presents several advantages [11, pp. 77–81]:

- Its representation captures significant distinctions between designs. Conversely irrelevant details are suppressed.
- The strict separation of syntactic generation and semantic testing allows the development of a search space that has (in addition to others) the crucial properties of completeness, closure, monotonicity, and representation scheme well-formedness. These permit efficient techniques for exhaustive enumeration to be employed.
- The tester contains knowledge in atomic and easily modifiable form.
- The evaluation function of the controller admits the generation of designs that display significant tradeoffs.
- By exhaustive enumeration it admits the possibility for surprise. If something interesting is out there, LOOS will find it.

These advantages are contrasted with limitations, chief of which appears to be the combinatorial explosion of complete enumeration. Although programs have been implemented for bathrooms, kitchens, and high-rise service cores, these problems require the allocation of modest numbers of nodes. As the number of allocated objects increases, the number of state space possibilities rises explosively. It appears that any exhaustive enumeration strategy will meet the same obstacle. Current research addresses this problem and proposes making the LOOS representation and control hierarchical rather than flat.

Spatial grammars

Parallel and complementary to the research tradition of the layout paradigm is an equally distinguished tradition that has grown from Post's seminal work in computation. *Spatial grammars* seek to express knowledge of design directly as compositional rules. When mapped to the design space model, these rules become search space operators with interesting implications for a search strategy.

The roots of the grammar paradigm go very deep and these have been delved into at various times [2, 20]. The current generation of interest orbits the notion of a grammar as a device for formally describing *rules of design* and their implied *languages*. Informally a grammar is a quadruple $G = (V_t, V_n, R, S)$ where

V_t is a finite set of *terminals*.

V_n is a finite set of *non-terminals* disjoint from V_t. The alphabet of the grammar is $\Sigma = V_t \cup V_n$.

R is a finite set of *rules* of the form $\alpha \rightarrow \beta$ where

α is a *composition* of elements from Σ such that at least one element of V_n is included.

β is a composition of elements from Σ.

$S \in V_n$ is a *start symbol*.

The rules of the grammar (also known as *productions*) describe how to derive one composition from another. Rules are written $\alpha \rightarrow \beta$. When a composition ω can be obtained from a composition γ by the application of exactly one rule from G then γ *directly derives* ω, or $\gamma \underset{G}{\Longrightarrow} \omega$. When ω can be obtained from γ by the application of zero or more rules from G then γ *derives* ω, or $\gamma \overset{*}{\underset{G}{\Longrightarrow}} \omega$. Grammars implicitly specify a *language* that is the set of all compositions containing no non-terminals

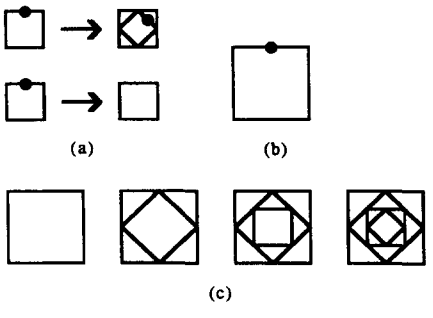

Fig. 12. A shape grammar and some members of its language. Shown are (a) the rules of the grammar, (b) the initial shape, and (c) some of the members of the language of the grammar.

(elements of V_n) that can be derived from the start symbol S. Note that the form of a composition is left undefined here. Different grammar systems operate on alphabets of different kinds and have different means of composition members of these alphabets. For any specific grammar a precise specification of these compositions must be made.

The most developed and used spatial grammar formalism is that of *shape grammars*, largely of Stiny, that are introduced in [28–32],† discussed in [1, 4, 19, 33–37], reviewed in [2], implemented in [38–41], applied in [42–52], and extended in [18, 53–56]. Applications using close variants are [57–62]. Shape grammars produce objects that can be formed from lines by the operations of *transformation* (usually limited to the similarities) and *shape union*; the set of all such objects is called the *universe of shapes* U^*. Each shape grammar determines a subset of U^*. An example shape grammar from [32] that generates drawings of embedded squares is shown in Fig. 12. Note that the grammar rules use distinguished points (called *labeled points* in the literature) to control the language. These are the counterpart of the non-terminals in shape grammars. They all must be removed before a developing shape is admitted into the language of the grammar.

To date, shape grammars have largely been used to characterize existing corpora of designs. In this role they have shown themselves to be a concise means of accurately describing the notion of *style* in architectural form. With implementations of shape grammar interpreters, and these are currently being developed in universities [39–41], it can be expected that shape grammars will find increased use in the generation of new designs, and in the exploration of the possibilities implied by a set of conventions.

Other spatial grammar formalisms exist, though none has been explored or utilized to the same extent as shape grammars. Stiny [34] presents a grammar in which the rules are comprised of sets of shapes instead of shapes. *Structure grammars*, by Carlson [63, 64], act on abstract spatial entities called *structures*; intuitively these are sets of pairs of symbols and transformations. There has been recent interest in grammar systems defined on solid objects, as these promise to produce more complete representations of designs as their output. Fitzhorn [65] presents a graph–grammatical formalization of the Euler operators on boundary representation (B-rep) solids. Woodbury *et al.* [63] report early results of work on a grammar on B-reps, the rules of which are at a higher level than the Euler operators. Outside of architecture, spatial grammars have received considerable attention in biology as models of the morphogenesis of plants; an accessible reference is [66].

The significance of spatial grammars in the present discussion is their mapping to the design space model. Obviously the rules of the grammar correspond to the search space operators of the model and the start symbol is the sole member of the set of initial representations. Grammar rules work by performing compositions on symbols, and the set of all possible compositions is the representation space of the model. The modeling space differs according to the grammar formalism used. In shape grammars, it is the space of all abstract drawings. In solid grammars, it is some set of subsets (or of classes of those subsets) of $\mathbf{R}^3$. The representation scheme, as usual, relates the modeling and representation spaces and it may have (depending on the formalism) some of the formal and informal properties discussed earlier.

The most interesting aspects of spatial grammars relate to the rules, and how they can be used. In spatial grammars, the rules themselves typically prune the representation space. (Thus the operator structure is remarkably different from that of LOOS. See Fig. 11.) Their fascination arises when they are used to describe the hitherto indescribable; when they are used to directly express generative design knowledge. Used in this mode they admit precise insight into the murkiness of architectural design and criticism and allow the explicit capture of design process. By representing "little fragments" of design knowledge, they allow the incremental creation of new theory, and often these "little fragments" themselves provide direct insight into design. Such freedom comes at a cost. It is impossible to know that a set of rules captures all designs of interest. It is difficult to write a set of rules that defines many interesting designs and does not simultaneously define a lot of trash.

Spatial grammars specify languages of designs. If the language is small, it can be generated and examined in its entirety, and some authors, notably Stiny and Mitchell [43, 44] and Flemming [61, 67], have taken pains to create such grammars while managing to retain deep insight.‡ If the language is large, and this will be usual if grammars are used in their most speculative roles, it becomes impractical (and impossible if the language is infinite) to display it in its entirety. Additional pruning is needed and this is the role of the search strategy. Current spatial grammar formalisms are silent on this topic, but it seems to me that it poses difficult issues due to the complexities of: (1) proving the formal properties of operators that are required by efficient search strategies and (2) providing appropriate evaluation mechanisms. An approach to the former problem has been proposed in [68, 69] and to the latter in [18].

CONCLUSIONS

Search (in architectural design at least) is more than a promising approach; it is a well-established paradigm

† I recommend [19, 32] and [57] be read together as an introduction to the formalism.

‡ In claiming a small language size for both of these grammars I ignore dimensional variations (as do the authors) and refer only to the floor plan stage of Flemming's grammar.

that has achieved serious objective results. On top of its theoretical results, a small number of systems have been implemented in university laboratories, and these are beginning to demonstrate that seaching for designs is not only theoretically interesting, but practically feasible. The research field is still very open and much theoretical work, particularly in algorithms and human–computer interface, remains. Searching for designs is at odds with CAD as it is currently seen in the profession, and its successful application will require further development of the new technology. The natural relation of search to current theory of human problem solving behavior suggests that it could greatly leverage human abilities in design, but only under a different attitude towards design. To take advantage of search, we must begin to think in its terms; the unstated "how" and "why" must take their full places beside the "what" and "where".

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

REFERENCES

1. G. Stiny and L. March, Design machines. *Environment and Planning B* **8**, 241–244 (1981).
2. L. March and G. Stiny, Spatial systems in architecture and design: some history and logic. *Environment and Planning B* **12**, 31–53 (1985). Paper presented at the Seventh International Conference on Systems Dynamics, University of Brussels, 16–18 June 1982.
3. C. F. Earl, Creating design worlds. *Planning and Design* **13**, 177–187 (1986).
4. W. J. Mitchell, Formal representations: a foundation for computer-aided architectural design. *Planning and Design* **13**, 133–162 (1986).
5. John Archea, *Puzzle-Making: What Architects Do When No One Is Looking*. John Wiley, New York. Principles of Computer-Aided Design (Edited by Y. Kalay), Ch. 2, pp. 37–52 (1987).
6. H. A. Simon, The structure of ill-structured problems. *Artificial Intelligence* **4**, 181–201 (1973).
7. P. Steadman, *The Evolution of Designs*, Cambridge Urban and Architectural Studies, Vol. 5. Cambridge University Press, Cambridge (1979).
8. L. March, *The Logic of Design and the Question of Value*, Cambridge Urban and Architectural Studies, Vol. 4, pp. 1–40. Cambridge University Press, Cambridge (1976).
9. A. A. G. Requicha, Representation for rigid solids: theory, methods and systems. *Computing Surveys* **12**, 437–464 (December 1980).
10. A. Newell and H. Simon, *Human Problem Solving*. Prentice-Hall, Englewood (1972).
11. U. Flemming, R. F. Coyne, T. Glavin, H. Hsi and M. D. Rychener, A generative expert system for the design of building layouts. 1989 Report Series, Engineering Design Research Center, Carnegie Mellon University (1989).
12. A. D. Radford and J. S. Gero, *Design by Optimization in Architecture, Building, and Construction*. Van Nostrand Reinhold, New York (1988).
13. J. R. Hayes, *The Complete Problem Solver*. The Franklin Institute Press, Philadelphia (1981).
14. J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading (1984).
15. S. L. Tanimoto, *The Elements of Artificial Intelligence*. Principles of Computer Science, Vol. 11. Computer Science Press, Rockville (1987).
16. Nils J. Nilsson, *Principles of Artificial Intelligence*. Morgan Kaufman, Los Altos (1980).
17. S. K. Card, T. P. Moran and A. Newell, *The Psychology of Human–Computer Interaction*, Lawrence Elrbaum, Hillsdale (1983).
18. G. Stiny, A note on the description of designs. *Environment and Planning B* **8**, 257–268 (1981).
19. G. Stiny, A new line on drafting systems. *Design Computing* **1**, 5–19 (1986).
20. W. J. Mitchell, *Computer-Aided Architectural Design*. Petrocelli/Charter, New York (1977).
21. U. Flemming, More on the representation and generation of loosely packed arrangements of rectangles. *Planning and Design* **16**, 327–359 (1989).
22. J. Grason, A dual linear graph representation for space-filling location problems of the floor plan type. In *Emerging Methods in Environmental Design and Planning* (Edited by G. T. Moore), pp. 170–178. MIT Press, Cambridge (1970).
23. P. Steadman, The automatic generation of minimum-standard house plans. Paper delivered at the Second Annual Conference of the Environmental Design Research Association, Pittsburgh, Pa., U.S.A. (1970).
24. W. J. Mitchell, J. P. Steadman and R. S. Liggett, Synthesis and optimization of small rectangular floor plans. *Environment and Planning B* **3**, 37–70 (1976).
25. U. Flemming, Wall representations of rectangular dissections and their use in automated space allocation. *Environment and Planning B* **5** 215–232 (1978).
26. U. Flemming, Wall representations of rectangular dissections: additional results. *Environment and Planning B* **7**, 247–251 (1980).
27. C. F. Earl, Rectangular shapes. *Environment and Planning B* **7**, 311–342 (1980).
28. G. Stiny and J. Gips, *Shape Grammars and the Generative Specification of Painting and Sculpture*, pp. 1460–1465. North-Holland (1972).
29. G. Stiny, *Pictorial and Formal Aspects of Shape and Shape Grammars on Computer Generation of Aesthetic Objects*. Birkhauser, Basel (1975).
30. J. Gips, *Shape Grammar and Their Uses*. Birkhauser, Basel (1975).
31. G. Stiny, Two exercises in formal composition. *Environmental and Planning B* **3**, 187–210 (1976).
32. G. Stiny, Introduction to shape and shape grammars. *Environment and Planning B* **7**, 343–352 (1980).
33. J. Gips and G. Stiny, Production systems and grammars: a uniform characterization. *Environment and Planning B* **7**, 399–408 (1980).
34. G. Stiny, Spatial relations and grammars. *Environment and Planning B* **9**, 113–114 (1982).
35. G. Stiny, Shapes are individuals. *Environment and Planning B* **9**, 359–367 (1982).
36. G. Stiny, Composition counts: $A + E = AE$. *Environment and Planning B* **14**, 167–182 (1987).

37. G. Stiny, What designers do that computers should. *Computer-Aided Design Education*, CAAD Futures 89, Harvard University (July 1989).
38. R. Krishnamurti, The arithmetic of shapes. *Environment and Planning B* **7**, 463–484 (1980).
39. R. Krishnamurti, The construction of shapes. *Environment and Planning B* **8**, 5–40 (1981).
40. R. Krishnamurti and C. Giraud, Towards a shape editor: the implementation of a shape generation system. *Planning and Design* **13**, 391–403 (1986).
41. S. C. Chase, Shapes and shape grammars: from mathematical model to computer implementation. *Planning and Design* **16**, 215–241 (1989).
42. G. Stiny, Ice-ray: a note on the generation of Chinese lattice design. *Environment and Planning B* **4**, 89–98 (1977).
43. G. Stiny and W. J. Mitchell, The Palladian grammar. *Environment and Planning B* **5**, 5–18 (1978).
44. G. Stiny and W. J. Mitchell, Counting Palladian plans. *Environment and Planning B* **5**, 189–198 (1978).
45. G. Stiny and W. J. Mitchell, The grammar of paradise: on the generation of Mughul gardens. *Environment and Planning B* **7**, 209–226 (1980).
46. G. Stiny, Kindergarten grammars: designing with Froebel's building gifts. *Environment and Planning B* **7**, 409–462 (1980).
47. T. W. Knight, The generation of Hepplewhite-style chair-back designs. *Environment and Planning B* **7**, 227–238 (1980).
48. U. Flemming, The secret of the Casa Giuliani Frigerio. *Environment and Planning B* **8**, 87–96 (1981).
49. F. Downing and U. Flemming, The bungalows of Buffalo. *Environment and Planning B* **8**, 269–293 (1981).
50. T. W. Knight, The forty-one steps. *Environment and Planning B* **8**, 97–114 (1981).
51. T. W. Knight, Languages of designs: from known to new. *Environment and Planning B* **8**, 213–238 (1981)
52. H. Konig and J. Eizenberg, The language of the prairie: Frank Lloyd Wright's prairie houses. *Environment and Planning B* **8**, 295–323 (1981).
53. T. W. Knight, Transformations of languages of designs: part 1. *Environment and Planning B* **10**, 125–128 (1983).
54. T. W. Knight, Transformations of languages of designs: part 2. *Environment and Planning B* **10**, 129–154 (1983).
55. T. W. Knight, Transformations of languages of designs: part 3. *Environment and Planning B* **10**, 155–177 (1983).
56. T. W. Knight, Comparing designs. *Planning and Design* **15**, 73–110 (1988).
57. U. Flemming, *The Role of Shape Grammars in the Analysis and Creation of Designs* (Edited by Y. Kalay), Ch. 12, pp. 245–272. Wiley Interscience, New York. Principles of Computer-Aided Design (1987).
58. A. Mackenzie, C. Sutter, F. Horan, S. Hunter, C. Kokkinos and G. Lidell, A "Language Lab" for architectural design. *1986 University AEP Conference*, pp. II-11–II-31. IBM, IBM Academic Information Systems, Milford, CT 5–8 April 1986.
59. J. L. Kirsch and R. A. Kirsch, The structure of paintings: formal grammar and design. *Planning and Design* **13**, 163–176 (1986).
60. J. R. Mitchell and A. D. Radford, EAVE, a generative expert system for detailing. *Planning and Design* **14**, 281–292 (1987).
61. U. Flemming, More than the sum of parts: the grammar of Queen Anne houses. *Planning and Design* **14**, 323–350 (1987).
62. M. J. Wolchko, *Design by Zoning Code: The New Jersey Office Building* (Edited by Y. Kalay), Ch. 13, pp. 273–292. Wiley Interscience, New York, Principles of Computer-Aided Design (1987).
63. R. F. Woodbury, C. Carlson and J. Heisserman, *Geometric Search Spaces in Design*. North Holland, IFIP WG 5.2 Workshop Series (in press). An earlier version of this paper appeared at the IFIP WG 5.2 Workshop on Intelligent CAD, Cambridge, September 1988.
64. C. Carlson, Stucture grammars and their application to design. Master's thesis, Department of Design, Carnegie-Mellon University (December 1988).
65. P. Fitzhorn, *A Linguistic Formalism for Solid Modeling*, Lecture Notes in Computer Science, pp. 202–215. Springer, Berlin (1987).
66. P. Prusinkiewicz, A. Lindenmayer and J. Hanan, Developmental models of herbaceous plants for computer imagery purposes. *Computer Graphics* **22**, 141–150 (August 1988).
67. U. Flemming, R. Gindroz, R. Coyne and S. Pithavadian, A pattern book for shadyside. Technical Report, Department of Architecture, Carnegie-Mellon University, Pittsburgh (1985).
68. R. Coyne and J. S. Gero, Semantics and the organization of knowledge in design. *Design Computing* **1**, 68–89 (1986).
69. J. S. Gero and R. D. Coyne, Logic programming as a means of representing semantics in design languages. *Planning and Design* **12**, 351–369 (1985).