Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2006), *20*, 95–103. Printed in the USA.

DOI: 10.1017/S0890060406060082

RESPONSE TO KEYNOTE

Explicit design space?

RAMESH KRISHNAMURTI
School of Architecture, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

(Received March 16, 2005; Accepted December 13, 2005)

Abstract

This paper examines the need for explicit representations of the design space, in response to Woodbury and Burrow. Specifically, their proposal for a particular search strategy, by means of which one can reuse past experiences explicitly represented by previously traversed paths, is examined. This is done by exploring issues with respect to design search and representation in general, while relating these to specific issues raised by Woodbury and Burrow. The paper concludes by suggesting that their arguments essentially *point* to devising an appropriate "programming language" for design.

Keywords: Data Types; Design Space; Grammars; Representation; Search

1. INTRODUCTION

Proof of evidence is not necessarily evidence of proof. Rob Woodbury and Andrew Burrow in a virtual *tour de force* have critically analyzed and elicited characteristics, some familiar, others perhaps less so, which are vital to a successful exploration of the design space. Their arguments lead them to propose that an explicit representation of the design space, along with an explicit representation of a particular search strategy, is essential for the success of this exploration. It is this conclusion that I principally wish to examine. The Woodbury and Burrow paper is multifaceted, and any adequate response requires more than a single thought. In this essay, I have attempted, within the space limitations set, to lay out a train of thought, mostly in sequence, occasionally digressive but related and without trying to compartmentalize my response. In doing so, I have taken the liberty of dispensing with section headings, avoiding any distractions that they may engender.

Design proceeds from requirement toward goal. Requirement usually comprises several, possibly conflicting, constraints. Likewise, the singular goal, the object of the design, may comprise various outcomes, some perhaps contradictory. Designers do not seek to ascertain truths, to establish dogmas, or to proclaim manifestos; although, if one were to read what designers say of their own work one might conclude otherwise. Designers seek answers to problems refined over many iterations. Through this iterative process, the specification of the design problem and its solution becomes less and less abstract and more and more concrete—real, so to speak.

It is not surprising that a design may respond to an altogether different problem from the one with which the designer started. We express this by the following relationship:

$$\text{design space} = \text{problem space} + \text{solution space} + \text{design process}.$$

Importantly, the design process *binds* a design problem, selected from a world of possible related problems, to its solution, selected from a world of possible designs.

Search is instrumental in this process. Designers search for solutions (and in the process search for the problem that the solution responds to). In this respect, Woodbury and Burrow are correct: design space exploration deserves serious study for the following reasons they cite. Search *is* a compelling model for design and designer action; it *is* a basis for computation, and, as such, results in effective design algorithms. Ömer Akın, who has written extensively

Reprint requests to: Ramesh Krishnamurti, School of Architecture, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890, USA. E-mail: ramesh@cmu.edu

on this subject, has found empirical evidence for *problem restructuring, generate and test*, and *heuristic search* (Akın, 1986). In a recent article (Akın, 2001), he revisits the role of *representation* in this process—in fact, a multiplicity of representations—and examines their importance. On architectural design, Akın has this to say:

> Architecture, like these, accommodates the user along many dimensions: functional, psychological, cognitive, ergonomic, climatic, economical, and so on. Humans live around architecture but also in and on them. The behavior of the users is an integral part of the functionality of the object. Living in a building means that there is a cognitive, ergonomic, psychological and economic interaction with it on a continuous basis. No other artifact of human design can claim a similar range, scope, and directness of use by human . . .
>
> Architects then are faced with handling situated, multifaceted, and multi-media driven representations that tend to persist throughout the entire design process . . . Thus *architecture is a representation saturated problem domain*, more so than any other with which I am familiar. (p. 4)

Following Akın we have the following equation:

$$\text{design process} = \text{design knowledge} + \text{strategy}.$$

Strategy incorporates search; design knowledge refers to the multiplicity of representations that designers both need and find useful. Design representations are manifold: product, metaphors, designer actions, process, design states, and so forth. Woodbury and Burrow concentrate on amplification of designer action as the basis for developing a computational strategy. They list the following ingredients: the qualitative prowess of design representations, codification, explicit space, implicature, speed, backup, recall, and replay. I will not enter into detail, but, with one exception, I am content in stating that they make a strong argument for their inclusion. Notwithstanding, the Woodbury–Burrow case for the explicit representation of the design space needs further examination.

Many factors affect decision making in design, for instance, social interaction, rational choice, advent of technology, and computational support.

Design involves stakeholders, experts, and the designer, in interaction and dialog. Each influences the problem and the solution, and thus, design search. The relationship between social interaction and design search seems to me the most difficult to pinpoint except, perhaps, with respect to designer mediation, which, at least, can be classified in operational terms from a computational standpoint. In other words, each computational design support tool specifies a set of designer actions, although not necessarily the motivation that precedes any particular designer act. In design, such mediation defines the metaphors by which, according to Bruton and Radford (in press), designers contingently bend the rules.

Design is typically ill-structured. In general, designers demonstrate a seemingly rational attitude: one that is deliberate, conscious, and explicable. Herb Simon's way of dealing with ill-structured problems rationally was "satisficing," which has been empirically shown to be a reasonable ingredient of the human decision process (Simon, 1973).

Rubenstein (1998) quotes Simon, who suggests that, in his version of bounded rationality the following are basic questions for which answers are sought. Although Rubenstein writes about economic models of bounded rationality, Simon's remarks apply equally to design:

> What are the kinds of reasoning procedures that people actually use, and why (in terms of knowledge of their psychological makeup)? What are the effects of social environment and social history on the procedures used? To what extent are other procedures used? In what way does the introduction of computers into [*design*] change these procedures?[1]
>
> What are the consequences of their using these procedures and not others? In what respects are current *design* models deficient in the assumptions they make about reasoning procedures? (p. 192)

Simon posits that inquiry into design deserves no less attention than scientific inquiry, and must be subject to the rigors of establishing empirical evidence.

One factor that must be considered in (the analysis of) any decision making process for design is the *sunk cost fallacy* (Hastie & Dawes, 2001); it is the time, effort, or choices invested by a designer, which are essentially nonrefundable, that are honored in subsequent design decisions. Rationally, sunk cost should not affect outcomes; in reality, it does. There are obvious instances of sunk costs, for example, the time and effort of learning new computer-aided design software, choice of conventions and metaphors used, in-house rules, and so forth. In time, such efforts desist from being sunk costs, and instead acquire utility value. There are sunk cost considerations for software designers also; this is reflected in the design of updates to their products. Indirectly, consideration of potential sunk cost affects design aesthetics, which seem to be reflected in the aesthetics of the output of design support tools. How else can one explain, if one ignores logo and grille, that, on a few models, a Honda and BMW 3 series are indistinguishable?[2] I would wager that both companies, at one time, worked with the same design software. Every *AutoStation*

[1]The word *economic* was used instead of *design* in Simon's original writing.

[2]I am sure one of the car manufacturers is flattered by the comparison.

https://doi.org/10.1017/S0890060406060082 Published online by Cambridge University Press

design looks as if it was produced in *AutoStation*.[3] Even noted designers are not immune.

However, considerations of sunk cost in investigations of design search, design representation, or design process in general, are nontrivial. I am convinced that sunk cost plays an implicit role in the *breadth first, depth next* strategy practiced by experienced designers (Akın, 1999). Sunk cost considerations have implications for computational design research and teaching in general. On the one hand, for "highbrow" research or methods to succeed, to have a utility value, these techniques have to find implementation in "lowbrow" commercial software in use in practice. On the other hand, rationality might prevail.

Last, but not least, there is no simple way of characterizing design knowledge. It falls into constraints and conventions that a designer *has* to adopt as well as the constraints and rules that the designer *chooses* to adopt. There is no general way of establishing which is which, or when. In architectural design, cost constraint typically serves as a guide to designers: pricing, change orders, and so forth all have cost implications; the rejection to a change in the design by a design decision support tool, based on cost consideration, is generally overridden. A stakeholder on a limited budget might not be as accommodating. In contrast, no designer would ever deny a law of physics; instead, she would find ways of working around it.

One way of looking at design is as a game played by the designer with a design system wherein the designer always wins. This does not necessarily imply that the designs are good or work well. Almost 20 years ago, I investigated such a paradigm where designs are the outcome of dialog between the designer and the design support medium (essentially, its knowledge base; Krishnamurti, 1986). I wrote the following (p. 187):

> If the knowledge base represents statements in some first-order logic, then dialogue theory for design is simply a theorem prover for deduction in that logic. However, in real design, any new assertion about a design may have the effect of invalidating any previous assertion about the design. In other words, designing is essentially nonmonotonic. Consequently, dialogue theory for design must be equipped with decision procedures based on rules that allow new assertions and invalidate old deductions.

The fact that the design process involves problem redefinition implies a declarative approach to design spaces: one can conceivably argue that this requires an explicit representation of the design space. In that paper, I go on to suggest that ". . . dialogue can be supported by a common representation for the description of the spatial and nonspatial elements in design."

[3]For the reader who might be somewhat puzzled, I believe in the Michael Jordan philosophy on endorsing products that are updated, renewed, or even replaced every so often.

Furthermore (Krishnamurti, 1986),

> . . . design support rules are essential if we are to verify the validity of design descriptions against some model of the world. Moreover, design rules should have some form of flexibility built into them. (p. 188)

Rules may be weak or strong. Weak rules serve as guides, for example, where cost constraint is not a maximum; a change introduced by the designer may have the effect of an acceptable design increase in cost that would otherwise be rejected by the decision procedure that handles costs. Structural consistency is an example of a strong rule that has the power to invalidate changes that yield physically unsupportable designs. This differs from Woodbury and Burrow's notion of strong and weak representations. If we accept their classification, rules become doubly weak, doubly strong, weakly strong, or strongly weak, depending on whether the rules are cognitive (or situationist), enforceable or not. Weak or strong, as Woodbury and Burrow rightly point out, design rules and representations require exogenous properties.

Rudi Stouffs and I have been collaborating on an approach to design descriptions, termed *sorts*. The premise here is that whenever individuals are confronted with information, they naturally classify it according to their own needs and understanding: sort it out, so to speak. "Sortal" descriptions may be shared. These descriptions have implications for information transfer, information coverage, and information loss. They also have implications for problem solving.

To illustrate an instance of sortally solving a problem using exogenous properties, imagine designing a new utility network amidst other existing utility networks (Stouffs & Krishnamurti, 1996*a*). Further assume that these networks belong to different utility firms, and that the information made public is kept to a minimum—at best, center line data and bounding radii. Figure 1 shows, in plan, the case for networks belonging to utilities, *m* and *n*. For convenience, each straight line length of pipe is identified by its network utility name, and numbered 1, 2, 3, . . . Gray circles indicate pipe intersections; of these, the darker circles signify possible interference.

An easy way to solve the design problem is to define, exogenously, a "view" of labeled points of intersection as follows: each such point is associated with labels identifying the pipes' network whenever the pipes do intersect. In this view, these "sorted" intersection points are labeled by one or both networks (i.e., labeled in the special way). Clearly, checking for pipe interference reduces to counting doubly labeled points of intersection in this created view. This information is used in the search, but is not needed again once a valid design has been found.

The preceding example also illustrates another feature of exogenous properties, namely, of only utilizing just those aspects necessary for the problem on hand. In another paper

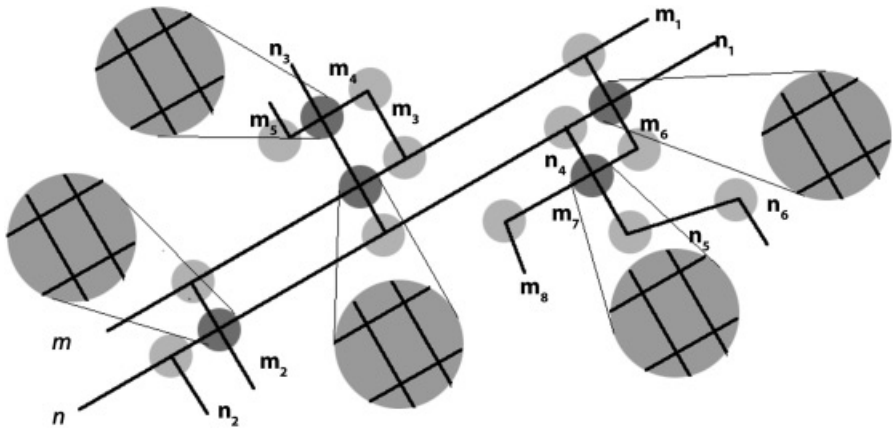

Fig. 1. Two pipe networks with possible interference.

(Stouffs & Krishnamurti, 1996*b*), we discuss this through the sortal layering of features. Figure 2 illustrates this nicely when we need the ability to represent geometries of different dimensionalities.

In building construction, a design is realized as a composition of building components (essentially solid elements). However, in the design and/or evaluation process, the dimensionality of the individual components are not always essential; indeed, an abstraction is often more useful. For instance, in a structural evaluation of a design, a wall may be represented as a plane with simple attributes; unless approximated, its true three-dimensional model may be too complicated. In general, each building component may be multiply represented as elements of different dimensionalities, each projecting information for a specific application such as structural or performance evaluation. Sortal views can be further dealt with sortally. Figure 3 illustrates that the joint condition on two composite walls needs to be dealt with sortally.

This trait of including exogenous properties in knowledge representations to solve design problems, indeed, for any search based problem, seems fairly common.

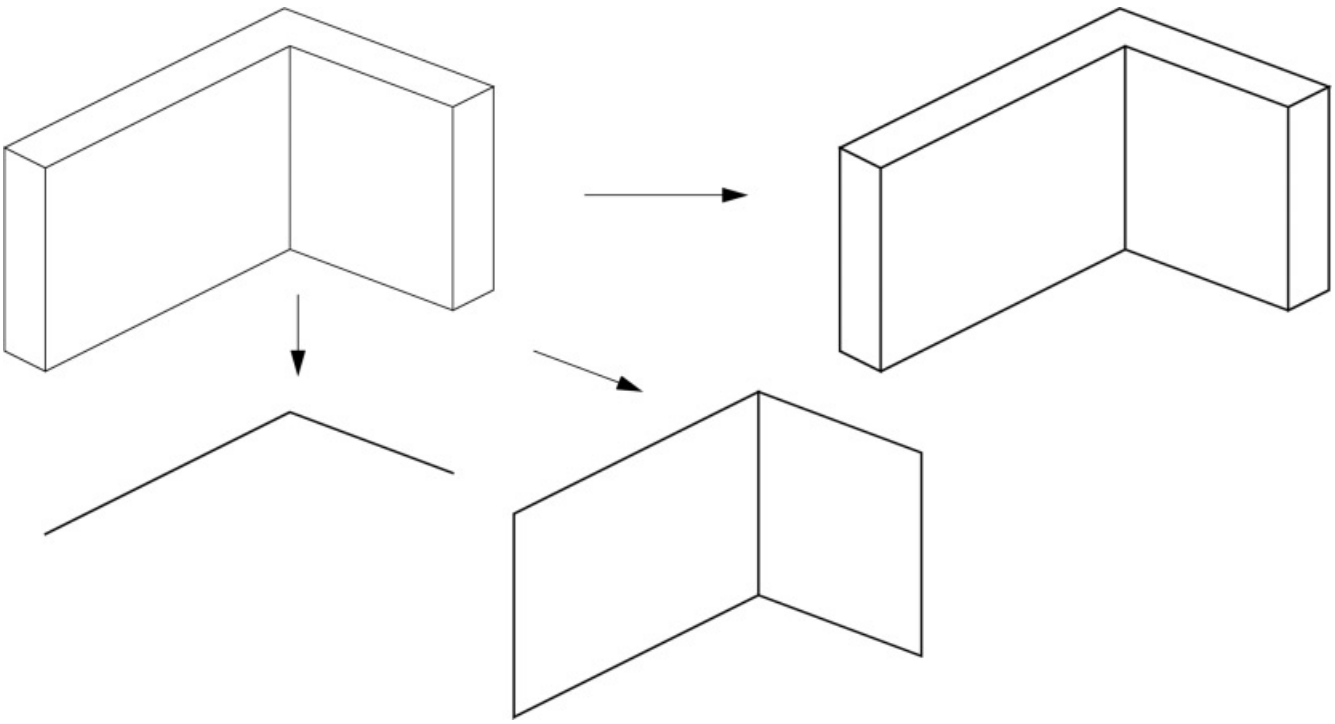

Fig. 2. A wall represented at once as a volume, plane, or line.

https://doi.org/10.1017/S0890060406060082 Published online by Cambridge University Press

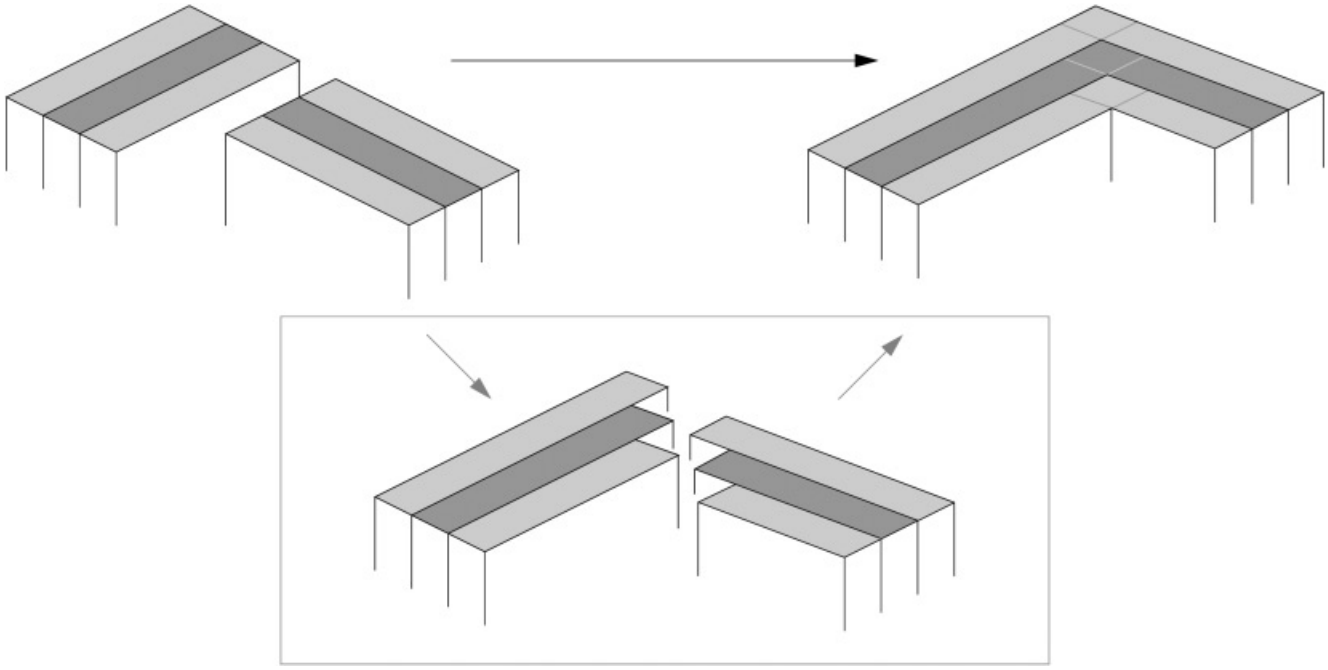

Fig. 3. A solid composition of walls as layers with different material properties.

The idea for *sorts* originated from a concern about information coverage and data loss between design representations (Stouffs et al., 1996). Using a subsumption relation defined on well-known solid models—boundary solid representations (Baumgart, 1975; Mäntylä, 1988; Paoluzzi et al., 1989) and the maximal element representation (Krishnamurti, 1992)—we were able to show that information loss between some of these solid models is inevitable. What is true for nonequivalent solid representations also holds for our *alter egos*, those computer programs we write that employ such models. No amount of syntactic or semantic sugaring can alter this. The idea for subsumption, in turn, derives from partial order and lattice theory, which seem natural for knowledge representation (Scott, 1976; Cardelli, 1984).

Design representation is about formal semantics; for our purpose, these are meanings associated with design artifacts and processes. Historically, I would credit the first formalization to Scott–Strachey's theory of *denotational semantics* (Stoy, 1977), which applies to programming language design. Scott's work directly influenced Chris Carlson's dissertation, which is perhaps the finest attempt to date at providing a functional semantics for rule-based design space exploration (Carlson, 1993). Unfortunately, symptomatic of such work, is the difficulty in realizing a practical, workable, usable programming language for design. Nonetheless, Carlson's endeavor is a step in the right direction, and represents a goal that we must achieve. Scott's work has influenced numerous others, for example, Aït-Kaći (1984), on a calculus of partially ordered types (attribute/value pairs) and subtype inheritance, whose work, in turn, influenced Carpenter (1992) on type feature structures. Burrow (2006) follows in this fine tradition through his dissertation in applying type feature structures to design space exploration through the reuse of design experience, that is, through a record of designer actions, and reasoning from design history.

Typed feature structures present a particular formalization of feature structures, which are a recordlike data structure for representing partial information that can be expressed in terms of features and their values. Key to typed feature structures is the notion of partial information structures and the existence of a unification procedure that determines if two partial information structures are consistent, and if so, combines them into a single, new (partial) information structure. Typed feature structures further consider a type hierarchy and a description language, where each type defines a corresponding description. Then, the generating procedure relates feature structures with a description (or type) these satisfy, and the subsumption relation between feature structures extends the subsumption ordering on types inherent to the type hierarchy. The fact that the generating procedure monotonically generates more complete information structures could be interpreted as excluding the possibility for information loss and thus making design states reversible. Typed feature structures formalism allows for the specification of an efficient design space, a generating procedure that incrementally generates more complete design structures, and of a subsumption relation that enables a richer form of exploration than ordinarily found in generative systems (Woodbury et al., 1999).

Subsumption is a powerful mechanism for comparing alternative representations. If a representation is subsumed

https://doi.org/10.1017/S0890060406060082 Published online by Cambridge University Press

by another, all designs represented using the former representation can also be represented using the latter representation, without any data loss. Typed feature structures, like most logic-based formalisms, link subsumption directly to information specificity, that is, a structure is subsumed by another, if this structure contains strictly more information than the other. One consequence of (logical) subsumption is that the absence of information in a design representation does not necessarily imply the absence of this information in the design, that is, representations are automatically considered to be incomplete. As a result, when searching for a design (representation) that satisfies certain information, less specific representations cannot automatically be excluded (Baader et al., 2003).

$$\text{semantics} = \text{syntax} + \text{behavior}.$$

In functional semantics, if two functions exhibit the same behavior they are the same. In type feature structures, if two paths exhibit analogous properties, they represent analogous design (or reuse) cases. However, difficulties arise when dealing with information of different types in a uniform way. For instance, at the representational level, operations that may otherwise seem trivial, such as adding or removing data elements or figures, become resolutely nontrivial; for instance, the addition of two numbers when these represent cardinal values (e.g., a number of columns that is increased) and when these represent ordinal values [e.g., for a given space, determining the minimum distance to a fire exit or the (maximum) amount of ventilation required given a variety of activities]. Similarly, additive versus subtractive colors, depending on whether these refer to the mixing of surface paints or colors of light, respectively.

Of course, there is more than one approach to formally modeling design knowledge. Stouffs and I have taken a slightly different tack. *Sorts* too are based on a part relationship that specifies a partial order; in this case, it gives rise to a semiconstructive algebraic formulation (Stouffs & Krishnamurti, 2002). An important ingredient of *sorts* is behavioral specification.

Behavioral specification is a prerequisite for the effective exchange of data between various representations. Fortunately, it is reasonably limited to the common arithmetic operations of addition, subtraction, and product. It turns out that the more common computer-aided design operations of *creation* and *deletion*, and *selection* and *deselection*, can all be expressed as some combination of addition and subtraction from one design space (*sort*) to another. The complex operations of *grouping* and *layering* can be treated likewise.

The simplest specification of a part relationship corresponds to the subset relationship on mathematical sets. This part relationship particularly applies to points and labels; for example, a point is part of another point only if the two are identical, and a label is a part of a collection of labels only if it is identical to one of the labels in the collection. Then, operations of addition (combining elements), subtraction, and product (intersecting elements) correspond to set union, difference, and intersection, respectively. Explicit designer action is required to alter any data element. Only when two elements are identical can these combine as one.

Another kind of behavior arises when we consider the part relationship on line segments. A line segment is an interval on an infinite line carrier; in general, one-dimensional quantities such as time may be considered as intervals. An interval is a part of another interval if it is embedded in this interval; intervals on the same carrier that are adjacent or overlap combine into a single interval. Specifically, interval behavior can be expressed in terms of the behavior of the boundaries of intervals (Krishnamurti & Stouffs, 2004). This behavior also applies to infinite intervals, provided there is an appropriate representation of both (infinite) ends of its carrier.

Behaviors also apply to composite *sorts*, that is, a part relationship can be defined for its component data elements belonging to a composite *sort* defined under a conjunction (attribute operator) or disjunction. The composite inherits its behavior from its components in a manner that depends on the compositional relationship.

The disjunctive operator distinguishes all operand *sorts* such that each data element belongs explicitly to one of these sorts. For example, a sort of points and lines distinguishes each data element as either a point or a line. Consequently, a data element is part of a disjunctive data collection if it is a part of the partial data collection of elements from the same component sort. In other words, data collections from different component sorts, under the disjunctive operator, never interact; the resulting data collection is the set of collections from all component sorts. When the operation of addition, subtraction, or product is applied to two data collections of the same disjunctive sort, the operation instead applies to the respective component collections.

Under the attribute operator a data element is part of a data collection if it is a part of the data elements of the first component sort, and if it has an attribute collection that is a part of the respective attribute collection(s) of the data element(s) of the first component sort it is a part of. When data collections of the same composite sort (under the attribute operator) are pairwise summed (differenced or intersected), identical data elements merge, and their attribute collections combine, under this operation. Elements with empty attributes are removed and the composite behavior is that, in the first instance, of the first component sort.

When reorganizing the composition of components sorts under the attribute operator, the corresponding behavior may be altered in such a way as to trigger data loss. Consider a behavior for weights (e.g., line thickness or surface tones; Stiny, 1992) as becomes apparent from drawings on paper: a single line drawn multiple times, each time with a different thickness, appears as if it were drawn once with the largest thickness, even though it assumes the same line with

https://doi.org/10.1017/S0890060406060082 Published online by Cambridge University Press

other thickness. When using numeric values to represent weights, the part relation on weights corresponds to the less than or equal relation on numeric values. Thus, weights can combine into a single weight, which has as its value the least upper bound of all the respective weight values, that is, their maximum value. Similarly, the common value (intersection) of a collection of weights is the greatest lower bound of all the individual weights, that is, their minimum value. The result of subtracting one weight from another is either a weight that equals the numeric difference of their values or zero (i.e., no weight), and this depends on their relative values.

Now consider a sort of weighted entities, say points, that is, a sort of points with attribute weights, and a sort of pointed weights, that is, a sort of weights with attribute points. A collection of weighted points defines a set of nonidentical points, each having a single weight assigned (possibly the maximum value of various weights assigned to the same point). These weights may be different for different points. The behavior of the collection is, at first instance, the behavior for points. On the other hand, a collection of pointed weights, which is defined as a single weight (which is the maximum of all weights considered) with an attribute collection of points, adheres, at first instance, to the behavior for weights. In both cases, points are associated with weights. However, in the first case, different points may be associated with different weights, whereas in the second case, all points are associated with the same weight. In a conversion from the first to the second *sort*, data loss is inevitable. An understanding of when and where exact translation of data between different *sorts*, or representations, is or is not possible, becomes important for assessing data integrity and controlling data flow (Stouffs & Krishnamurti, 1996*a*).

Behavioral specification is a prerequisite for a uniform handling of different and a priori unknown data structures. The behavior of such data can be expressed through a number of operations chosen to match the expected behavior. When an application receives data along with its behavioral specification, the application can then correctly interpret, manipulate, and represent this information without unexpected data loss. The part relationship that underlies the behavioral specification for a sort enables matching to be implemented for this sort; because composite sorts inherit their behavior and part relationship from their component sorts, any technical difficulties in implementing matching apply just once, for each primitive sort.

Logic-based models essentially represent knowledge; in contrast, sorts represent data; any reasoning is based purely on present or emergent information. Recognizing information, especially of the emergent kind, is invaluable, rather, necessary, should the information be, subsequently, the subject of action. From a creativity standpoint, design relies on an ability to restructure emergent information. Data recognition and subsequent manipulation can be considered part of a single computation:

$$s - f(a) + f(b).$$

Here, s is a data collection; a is a representation of the data pattern; f is a transformation under which a is a part of s; $f(b)$ is the data replacing $f(a)$ in s; and $s - f(a) + f(b)$ is an expression of computational change, which can be written as a design rule $a \rightarrow b$. We have yet another equivalence:

$$\text{design change} = \text{design rule} + \text{rule application}.$$

Rule application consists of replacing the emergent data corresponding to a, under some allowable transformation, by b, under the same transformation.

Formally, rules may be grouped as a *grammar;* a device for specifying the set of all designs generated by the rules collectively. Each generation of a design in the language starts from an initial design, and uses the rules to create a design that contains elements from a given terminal vocabulary. Rules and grammars specified as such, lead naturally to the generation and exploration of possible designs. According to Mitchell (1993) and Stiny (1993), in the case of creative spatial design, spatial elements that emerge under a part relation are highly enticing to design search. Woodbury and Burrow (section 1) echo this sentiment as well.

However, Woodbury and Burrow argue that standard notions of grammars hamper navigation in the design space. They state the following in section 4:

> A flaw in standard rule-based accounts of design space exploration in implicit space is that the usual formulation of rules cast navigation solely in terms of derivation; thus putting the landscape of the explicit space forever beyond the sight of the navigator.

They go on to add:

> Recasting the devices of navigation from rules to operations that make explicit reference to underlying structure permits us navigators to know more about the paths we have taken.

At this juncture, it is important to point out that the concept of search is fundamentally greater than any generational form alone might imply. A mutation of a data collection into another, or parts of others, constitutes an action of search. As such, a rule may be considered to specify a particular composition of operations and/or transformations that is recognized as a new, single, operation and applied as such. Rules can serve to facilitate common operations, for example, for the changing of one data collection into another or for the creation of new design information based on existing information in combination with a rule. Likewise, a grammar is more than a framework for generation; it is a tool that permits the structuring of a collection of rules or operations that have proven their applicability in the creation of a certain set (or language) of designs or in the

https://doi.org/10.1017/S0890060406060082 Published online by Cambridge University Press

derivation of certain information. In this sense, a grammar encapsulates explicitly a representation of the design space. However, a grammar does not inherently provide an explicit representation of the paths traversed in the design space, although it is not difficult to envisage any implementation of grammars maintaining a history of derivations. In this respect, Woodbury and Burrow's notion of replay needs further examination.

According to Woodbury and Burrow, replay invokes recombination, which in their notion is cognizant of the memory of past actions influencing future search. They have developed a resolution mechanism by means of which one can reuse past experiences explicitly represented by previously traversed paths. However, replay also involves designer mediation, which, in turn, involves memory of possibly restructured descriptions. As Earl (1999, p. 285) remarks:

> Generative systems change designs and descriptions. Generative history of a design provides the constraints and context for descriptive history of evolving structure and meaning. Descriptions are woven through the generative history of a design; created, composed and discarded.

Paths in the design space represent sequences of changes. Each change is a change to the design state, that is, a change to component, feature and assembly description, more precisely, an update of the design representation.

Partial ordering allows us to treat a design representation as a discrete topology (see, e.g., Kuratowski, 1972, for standard definitions in the mathematical literature). Informally, a topology is a hierarchical division of a structure (set, shape, etc.) under a closure relation. Closed structures are closed under meet and join. Every substructure is contained in a smallest closed structure. For our purpose, closed structures serve as identifiable types or objects. Then, as Stiny (1994) has so shown, it is possible to retroactively restructure descriptions along a path in the design space in such a way that continuity of change is maintained. An alternative way of stating this is that design rationale can be maintained by retroactive restructuring of design descriptions. Each such change applies to a known entity (or collection of entities) in the design description. *If this retroactive restructuring can be effected, then true replay is possible*. As it stands, if one employs an object-oriented approach to design representation, then such replay does not take into account changes that were made to emergent forms (or objects). Moreover, Stouffs and I showed that by taking an object-oriented approach to design representation, for continuity of change to be preserved, the representation must anticipate a priori all emergent forms (and objects) that are so changed (Krishnamurti & Stouffs, 1997).

Ultimately, design search is about the gelling together of syntax and semantics. Design is storytelling. More importantly, any design strategy (in which search is central) should reify the process as a viable design tool. The telling is syntactic whereas search imbues the story with meaning. A generative mechanism is simply a means to an end. Design search directs that means towards a "proper" end. Design search exploration, as Woodbury and Burrow (sections 3.2, 4, and 5) seem to imply, as Carlson (1993) demonstrated in his dissertation, is about devising an appropriate "programming language" for design. This is no easy task.

It is popularly said (in all probability, by Kierkegaard) that, "If it is edifying, it isn't scholarly. If it is erudite, it isn't understandable." Design remains both an intellectual challenge and a practical enterprise and will continue to be so. On the one hand, design conjures up theories of form, function, and esthetics; on the other hand, it offers up a real ability to create artifacts of beauty, functionality, and ingenuity. Through all of this, *search* remains the one true common thread that binds the dreamer to her dream. Woodbury and Burrow should be commended for exploring a new kind of navigation, a renewed look at search, while at the same time creating a practical forum to channel debate among kindred spirits.

ACKNOWLEDGMENTS

The author gratefully acknowledges the support from the National Science Foundation through Grant CMS 0121549. Any opinions, findings, conclusions, or recommendations presented in this paper are those of the author and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Aït-Kaći, H. (1984). *A lattice theoretic approach to computation based on a calculus of partially ordered type structures (property inheritance, semantic nets, graph unification)*. PhD Thesis. University of Pennsylvania.

Akın, Ö. (1986). A formalism for problem structuring and resolution. *Environment and Planning B: Planning and Design 13(2)*, 223–232.

Akın, Ö. (1999). Variants of design cognition. In *Knowing and Learning to Design Conference* (Eastman, C., McCracken, M., & Newstetter, W., Eds.). New York: Elsevier. Accessed at www.andrew.cmu.edu/user/oa04/Papers/Variants.pdf on June 14, 2005.

Akın, Ö. (2001). "Simon Says": Design is representation. *Arredamento*, July. Accessed at www.andrew.cmu.edu/user/oa04/Papers/AradSimon.pdf. on June 14, 2005.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge: Cambridge University Press.

Baumgart, B.C. (1975). A polyhedron representation for computer vision. In *National Computer Conference 1975*, pp. 589–596. Montvale, NJ: AFIPS Press.

Bruton, D., & Radford, A.D. (in press). *Bending Rules: Grammar, Contingency, Art and Design*. San Francisco, CA: Morgan Kaufmann.

Burrow, A. (2006). *Type feature structure and design exploration*. PhD Thesis. University of Adelaide.

Cardelli, L. (1984). A semantics of multiple inheritances. *Proc. Int. Symp. Semantics of Data Types* (Kahn, G., MacQueen, D., & Plotkin, G., Eds.). *Lecture Notes in Computer Science 173*. Berlin: Springer–Verlag.

Carlson, C. (1993). *Grammatical programming: an algebraic approach to the description of design spaces*. PhD Dissertation. Carnegie Mellon University.

Carpenter, B. (1992). The logic of typed feature structures with applications to unification grammars, logic programs and constraint resolution. In *Cambridge Tracts in Theoretical Computer Science*. Cambridge: Cambridge University Press.

Earl, C.F. (1999). Generated designs: structure and composition. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 13(4)*, 277–285.
Hastie, R., & Dawes, R.M. (2001). *Rational Choice in an Uncertain World*. Thousand Oaks, CA: Sage.
Krishnamurti, R. (1986). The MOLE picture book: on a logic for design. In *Design Computing*, vol. 1, pp. 171–188. New York: Wiley.
Krishnamurti, R. (1992). The maximal representation of a shape. *Environment and Planning B: Planning and Design 19(3)*, 267–288.
Krishnamurti, R., & Stouffs, R. (1997). Spatial change: continuity, reversibility and emergent shapes. *Environment and Planning B: Planning and Design 24(3)*, 359–384.
Krishnamurti, R., & Stouffs, R. (2004). The boundary of a shape and its classification. *The Journal of Design Research, 4(1)*. Accessed at http://jdr.tudelft.nl/articles/issue2004.01/stouffs.pdf on June 14, 2005.
Kuratowski, K. (1972). *Introduction to Set Theory and Topology*. Oxford: Pergamon.
Mäntylä, M. (1988). *An Introduction to Solid Modeling*. Rockville, MD: Computer Science Press.
Mitchell, W.J. (1993). A computational view of design creativity. In *Modeling Creativity and Knowledge-Based Creative Design* (Gero, J.S., & Maher, M.L., Eds.). Hillsdale, NJ: Erlbaum.
Paoluzzi, A., Ramella, M., & Santarelli, A. (1989). Boolean algebra over linear polyhedra. *Computer Aided Design 21*, 474–484.
Rubenstein, A. (1998). *Modeling Bounded Rationality*. Cambridge, MA: MIT Press.
Scott, D. (1976). Data types as lattices. *SIAM Journal of Computing 5(3)*, 522–587.
Simon, H. (1973). The structure of ill-structured problems. *Artificial Intelligence 4(2)*, 181–200.
Stiny, G. (1992). Weights. *Environment and Planning B: Planning and Design 19(4)*, 413–430.
Stiny, G. (1993). Emergence and continuity in shape grammars. In *CAAD Futures '93* (Flemming, U., & Van Wyk, S., Eds.), pp. 37–54. Amsterdam: North-Holland.
Stiny, G. (1994). Shape rules: Closure, continuity, and emergence. *Environment and Planning B: Planning and Design 21(1)*, s49–s78.
Stouffs, R., & Krishnamurti, R. (1996*a*). On a query language for weighted geometries. *Third Canadian Conf. Computing in Civil and Building Engineering* (Moselhi, O., Bedard, C., & Alkass, S., Eds.), pp. 783–793, Montreal, Canada, August 26–28.
Stouffs, R., & Krishnamurti, R. (1996*b*). The extensibility and applicability of geometric representations. *3rd Design and Decision Support Systems in Architecture and Urban Planning Conf., Architecture Proc*., pp. 436–452, Eindhoven University of Technology, Eindhoven, The Netherlands, August 18–21.
Stouffs, R., & Krishnamurti, R. (2002). Representational flexibility for design. In *Artificial Intelligence in Design '02* (Gero, J., Ed.), pp. 105–128. Dordrecht: Kluwer Academic.
Stouffs, R., Krishnamurti, R., & Eastman, C.M. (1996). A formal structure for nonequivalent solid representations. *Proc. IFIP WG 5.2 Workshop on Knowledge Intensive CAD II* (Finger, S., Mäntylä, M., & Tomiyama, T., Eds.), International Federation for Information Processing, Working Group 5.2, pp. 269–289, Pittsburgh, PA, September 16–18.
Stoy, J. (1977). *Denotational Semantics*. Cambridge, MA: MIT Press.
Woodbury, R., Burrow, A., Datta, S., & Chang, T-W. (1999). Typed feature structures and design space exploration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 13(4)*, 287–302.

Ramesh Krishnamurti has a BE (Honors) in electrical engineering from the University of Madras, a BA in computer science from the University of Canberra, and Masters and PhD in systems design from the University of Waterloo. He has previously taught and worked in Canada and the United Kingdom and is currently a Professor in the School of Architecture at Carnegie Mellon University in the Graduate Program in Computational Design. His research focuses on the formal, semantic, generative, and algorithmic issues in computational design. Dr. Krishnamurti's past research activities have a multidisciplinary flavor and include spatial grammars, spatial algorithms, geometrical modeling, analyses of design styles, knowledge-based design systems, integration of graphical and natural language, interactivity and user interfaces, graphic environments, computer simulation, and war games. His current research projects deal with sensor-based modeling and recognition, smart building information models, and shape grammar implementations.