Design genes

by

Robert F. Woodbury

48-18-90 *6.3*

7 '

Design genes

Robert F. Woodbury
Department of Architecture / Engineering Design Research Center
Carnegie Mellon University,
Pittsburgh, Pennsylvania, USA, 15213.
Tel: 412-26S-S853
Fax: 412-268-7819
E-mail: rw^cad.cs.cmu.edu

Appeared in the proceedings of the International Round-Table Conference on Modeling Creativity and Knowledge-Based Creative Design, Queensland Australia, Dec. 11-14, 1989.

Abstract. This paper proposes an organization for a *genetic design system* (CDS) that is derived from two sources: (1) the search paradigm in computer-aided design, (2) the structure of natural evolution. Both sources are presented in an abstract form that exposes their mechanism. The components of the search mechanism are related to those of natural evolution, and a system design based on this correspondence is advanced. A critical discussion of the design and a proposed research program are presented.

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

Design genes

Robert F. Woodbury
Department of Architecture / Engineering Design Research Center
Carnegie Mellon University,
Pittsburgh, Pennsylvania, USA, 15213.
Tel: 412-268-8853
Fax: 412-268-7819
E-mail: rw@cad.cs.cmu.edu

Abstract. This paper proposes an organization for a *genetic design system* (GDS) that is derived from two sources: (1) the search paradigm in computer-aided design, (2) the structure of natural evolution. Both sources are presented in an abstract form that exposes their mechanism. The components of the search mechanism are related to those of natural evolution, and a system design based on this correspondence is advanced. A critical discussion of the design and a proposed research program are presented.

1 Introduction

Two distinctly different research programs coexist in computer aided design (at least in architecture). I shall call these respectively: *cognitive modeling* and *design mechanics*. The former takes as its goal the explanation of human designing in information processing terms. The latter seeks computable mechanisms for making designs, and is indifferent to their source. To be sure, these strands of research overlap and enrichen each other. For example, cognitive models of design are extremely useful if one proposes to build an effective human computer interface. Conversely, computational mechanisms provide the substrate upon which cognitive simulations are built. The results of both are necessary in building 'symbiotic' human computer systems in which labor is divided between humans and computers according to their respective capabilities. A problem arises when the two emphases are confused and particularly when an attempt to simultaneously advance both of them is made. Put bluntly, human cognition is a poor sole basis upon which to build automatic design systems. Humans and computers have fundamentally different information processing architectures, the latter much more malleable than the former. Blindly adopting a human model only hamstrings the search for mechanisms that take full advantage of computation.

With this opening blast, I reveal my initial bias to the question of creativity in computer based design. I am baldly interested in mechanisms that create, irrespective of their source[1]. I will judge a mechanism interesting by its invention of the unexpected, by its computational clarity, and by the absence of heuristics at its core. From these three criteria I argue for several constraints on any mechanism that might be proposed:

- The first constraint is that of the use of constructive rules. In this constraint lies my sole use of human cognitive models of design. Two arguments support the constraint:

  1. Appreciation of computational clarity is a function (however partial) of human cognitive structure. A clear model for creativity is likely to be used over a murky one. The rule-like behaviour of human problem solvers seems to me to be a crucial constraint to be respected in searching for clear mechanisms.
  2. Human cognitive models have been a rich source of analogy for design mechanics in its search for new computer aided design approaches. As design mechanics has matured, it has become more discriminating. Some parts of the analogy have been adopted and transformed, and have become a part of the foundations of design mechanics. Other parts have been (or should be) discarded as the need for the crutch of analogy has diminished. A principal retained concept from the analogy to human cognition is that of rule-like behaviour. Rules appear to have distinct advantages when compared with other means of organizing generative knowledge. (1) Rules support the incremental build-up of knowledge. (2) Rules can be applied in unexpected circumstances. (Serendipity seems to be important to creativity.) (3) Rules act locally, yet their consequences may be global. Thus they can potentially create the unexpected.

- Using rules would appear to imply that a form of *search* is the main mechanism for design. Rules specify allowable moves, and these moves form a *derivation tree.* Moving through the derivation tree to find designs requires search. In architectural design, the enormity of the derivation tree for any significant problem would appear to eliminate any search procedure that depends solely upon exhaustive search, however discriminating its pruning rules might be. Obvious extensions to exhaustive search, for example, hierarchical decomposition, seem to me to embody heuristics that are too intertwined with the method. I prefer a heuristic free mechanism, so must adopt a different form of search.

[1] **After finding a mechanism that can create, I would hope to look at leveraging its powf r through interaction with humans.**

- Architecture is concerned with the creation of spatial compositions, so geometry must be a main preoccupation of any search. Representations for geometry are thus needed, and such representations appropriate to search remain rare in the literature. It seems to me that any representation of geometry must have a certain 'richness' (about which I shall be more precise later) if it is to be used at the core of a creative system. It must be capable of representing many if not all configurations that might be of interest in a particular problem. Especially, it must be able to model the appropriate *spatial coincidences* that seem so pervasive in architecture. This last point appears to require less abstract representations than are usual, for example, in layout problems.

- A form of generate-and-test search mechanism seems inevitable, for the following reasons:

  1. Designs are composed in terms of their spatial (and other physical) properties, but are evaluated according to a range of criteria, that rely upon (but are not defined by) these properties. It is beyond the range of current theory (at the least) to devise synthesis procedures that simultaneously create form directly from a variety of performance criteria.
  2. Serendipity seems to be an important aspect of creative design. Finding a concept by accident is only likely to happen when a wide variety of 'accidents' are floating about.

Together, constructive rules, search, geometric richness, spatial coincidences, and generate-and-test present a tall order for a mechanism for design. However, I believe that just such a mechanism can be proposed by analogy to evolution and genetics. Evolution has produced astounding variation in the natural world, and much of this variation would be called creative were it the construct of man. Vet evolution is an absolutely blind process that proceeds mechanically, with neither guidance nor goal.

In outline, I suggest that an object called a *design space* constitutes the genotype, development process, and selection environment of designs. Genes are captured by grammars; development by search strategies. Selection of genotype occurs on the phenotype; selection of grammars occurs by testing members of their language in simulated environments. Designs are evolved by introducing mutative changes to either the search space operators, the development procs*. or the simulated environment and by selection on the resulting developed dosigns. The product of an evolutionary design process is a design space that ran be 'turned on' to create a set of designed objects.

Before building an analogy to biological evolution, I present a brief sketch of the mechanics of evolution (largely following the popular book by Dawkin^ [Daw87, pll 1-137']).

2 Natural evolution

Evolution occurs by cumulative selection; new structures in organism's emerge through a process of many small changes over generations of reproduction. Each change must be *spontaneously plausible*. Successful changes will be *reproductively effective*. For cumulative selection to work three things must simultaneously exist. (1) There must be entities (*replicators*) that are capable of producing replicas of themselves. (2) There must be a source of error (essentially random in accounts of biological evolution) in the replication process. These errors must be passed on to the replicas, so that they are reproduced when the replica replicates. (3) Entities must be able to exert influence over their likelihood of being replicated. The new entities that are produced be the replication process will be different from their progenitors. Some differences will have no effect on entities' ability to reproduce. Other differences will have have positive or negative effect. Entities *inheriting* changes that have positive effect will, over generations of replication, become prevalent over entities that inherit changes having neutral or negative effect.

Living things (at least the multicellular ones) realize these requirements via a process that employs a genetic code[2] (the *genotype*, stored in *genes*) and a process of embryonic development governed by that code. The genetic code in these systems is akin to a *program* for creating individuals. This program governs a process of cell division, growth, and death that creates an actual living thing (the *phenotype*). Replication is accomplished by *parent(s)* providing a *seed* (or *egg*) that contains a genetic code and an appropriate development environment for its growth into another living thing (a *child*). Errors in replication can occur either in the seed where they are called mutations, or in the development process, where (at least in mammals) they are called congenital defects. Only the former are transmitted through the replication process from one generation to the next. Individuals inheriting changed genetic information are more or less likely to reproduce in the environment in which they live. The successful ones are said to be *selected* to reproduce; the unsuccessful ones produce no descendents. Over time the result is a population of individuals that are increasingly *adapted* to the environment in which they live. Thus the genetic errors that 'improve' the individual are retained from generation to generation. Genotypes are recipes rather than blueprints; they specify a set of directions, not a miniature model for an individual. Through the development process specified (in part) by the recipe in the genotype, a living thing that bears no physical resemblance to the recipe emerges. *Sexual reproduction*, or *crossover* (as it is euphemistically known in the genetic algorithm literature), is a combining of part of the genetic code from each of two parent individuals to produce a seed, different from that of either parent, that, through development, becomes an individual in its own right. Crossover allows a small population of actual individuals to effectively store a very large population of possible individuals through the combinatorial

[2]Stored in DNA molecules.

possibilities of gejjetic composition.

A group.of similar[3] living things constitutes a *species.* Over time (and space) species bifurcate, to create new species; this process is called speqation. If the speciation process is considered as a mathematical graph (the *phytogeny* of species), its properties are those of a a tree: it branches, and branches never merge.

The process of natural evolution is goalless in that it has no final arrival point. It is a continuous process of adapting to an environment, that in itself may also be continually changing. Once started, it requires no guidance, and it depends upon no vision of its future. It produces astonishing variations and marvellous levels of adaption. It does, however, require a certain stability in the environment in which it operates. The environment must not change so quickly or drastically that many individuals fail to survive and replicate. Such chaftges lead to a collapse, from which diversity must emerge anew, if at all.

Before proceeding any further with the analogy I present the a summary of the notion of *design spaces* and arguments for its relevancy. I will use (and italicize) terms that have a precise meaning with respect to design spaces, definitions for which may be found in Appendix A.

3 Design spaces

Informally, a design space is all the machinery required to computationally search for designs. It consists of a *search space* and a *search strategy.* A search space is a way of describing the possible configurations that might be considered as solutions to a design problem. A search strategy is a policy for making the decisions required in search as well as a problem context in which that policy applies.

There are three main reasons for using design spaces as the basis for a genetically inspired design system: preservation of meaning under mutation and crossover, leveraging of change through rules, and explicit description of a search process.

In a design space, the *representation space* is defined by a grammar (or algebra), but is searched by a (possibly) separate set of *search operators.* The representation space itself can be much larger than the set of objects that can be reached by the operators, yet under certain restrictions (those of the *representation scheme),* all of its members retain the ability to actually *repnsent* a design. If the search operators are defined in terms of the representation space grammar, and if mutations are similarly defined, the operators can be mutated without fear that the objects they produce will become malformed in an essential way. This is in contrast to prototype based approaches to representation in which the meaning of the component symbols is internally arbitrary.

[3]There are several measures of sufficient similarity.

Search spaces *do* not define designs directly; they rather specify *derivation sequences* by which designs can be developed. Simple changes to rules can create complex and far-reaching changes to the designs that they imply. Thus, mutations can be small (randomly plausible), but can have large effect. Search space specification by rules parallels the biological development process governed by genes; both can magnify the effect of small changes.

Search spaces define not a single design, but a *design collection.* A search strategy must control rule applications to guide the way to an individual design or a set of designs that are only a tiny fraction of the entire representation space. A strategy is explicitly part of a design space, and its presence as data means that it too can be subject to change, both mutative and crossover, in the genetic search process.

In the next part of the paper I build an analogy between biological evolution and a proposed mechanism for generating creative designs. As the analogy emerges in the discussion, so will an number of constructs that, in addition to design spaces, constitute the organization of a proposed *genetic design system,* GDS[4]. My construction uses biology more as a point of departure than as a target for complete analogy. Wherever computational capabilities suggest changes I freely make them, subject to the constraint of maintaining a capability that provides for replication, error, and power.

4 The genetic analogy

In building my analogy between biological evolution and a proposed design mechanism, my first step is to posit two representations for any design, one implicit and one explicit.

- The implicit representation is a design space, and corresponds to the biological genotype. Several of the entities in a design space carry genotypic information. The search space operators are the closest analogue to biological genes. These describe the allowable mechanisms for synthesizing various parts of a design. They operate locally, but produce global or emergent form. Three other design space components: the search strategy, the goals, and the evaluation devices, can all be considered to carry genotypic information[5]. The search strategy of a design space is closely analogous to the embryonic development process of biology, in that it is an implicit specification of the process by which a design space is tranlated into a design. However, it is genotypic as well, in that changes to it can be preserved in a copy and can therefore be transmitted between generations. Goals guide the search strategy and questions concerning them are answered by the evaluation devices in a design space. Like operators and search strategies,

[4]Pun intended.

[5]Here is my first departure from natural evolution.

these can be freely copied, but at the expense of changing the environment for which a design is intended.

- The explicit representation is the result of 'executing' a design space, it is a representation of some design or set of designs, and it corresponds to the biological phenotype. Design space 'execution' is thus in analogy to the process of embryonic development. Designs are members of the representation space of a design space, and they correspond (in a mathematical sense) to some possible artifacts. They are generated, through a search strategy, with the intention of meeting the goals in a design space, and this is the second departure from strict analogy with biology. Biological systems have no goals; embryos develop quite independently (modulo environmental poisons, etc.) of the outside environment that they will eventually inhabit. Goal constructs in design are added here to provide: (1) a means of reducing the language of possibilities implicit in the search space to a manageable set, and (2) a mechanism for direct intervention into a genetic design system.

The next part of the analogy equates heredity with copying, not of designs or artifacts[Ste79, [4]p. 79*], but of design spaces. Any design space can be copied to create a new space that is identical with the original. The requisite copying errors are introduced through 'mutation operators' that can act on any (or all) of search space operators, search strategies, goals, and evaluation devices. These mutation operators may be purely random with respect to the selection process, as in they are in biology. More interestingly, and without disturbing the overall organization of the system, they might themselves be knowledge based[6]. When search space operators are mutated, they imply a different language of designs in which the eventual design phenotypes exist[7].

Finally, selection is equated with testing designs against their goals (and using the evaluation devices in the process). The goals and evaluation devices form a *simulated environment* in which designs either 'live' or 'die'[8]. Those that 'live' are selected for the next generation; those that 'die' are simply discarded. Actually, it is not the explicit representation of a design that is selected, it is the design space (the implicit specification).

The entire process is captured in a single infinite loop that respectively:

- Copies and mutates each element of a set A of design spaces. Includes all copies in A.

- Develops the spaces into individual designs.

[6]Another departure from strict analogy. Knowledge based mutations are to CDS as an intrusive God would be to natural evolution.

[7]See [MS85,Kni81,Kni83a,Kni83b,Kni83c,Kni88] for a formal discussion of related ideas in shape grammar systems.

[8]This introduces complexities if the design goals themselves are subject to mutation. I want to include such a potential for completeness and for its suggestion of possibilities, particularly those of environmental and design problem change.

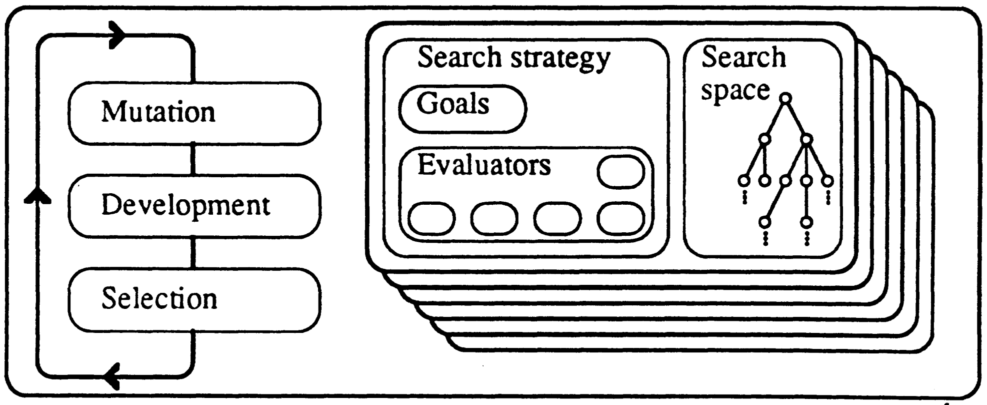


Figure 1: The organization of GDS

- Tests these designs against the design space goals.
- Selects a subset of designs and places their corresponding design spaces in A.

As shown in Figure 1, GDS thus consists, in the abstract, of a collection of design spaces, mechanisms for mutation, development, and selection (the latter largely accomplished by design space goals and evaluators), and a infinite loop that implements the overall evolutionary process. In more concrete terms, I would expect an implementation of GDS to use a particular kind of design space that may be characterized as a spatial grammar. Spatial grammars allow the capture, in a set of rules, of knowledge that is explicit and particular about its manipulation of spatial form. They provide, in essence, a programming language for constructive rules and an interpreter for their application. Such rules can be modified (mutated) into other rules. Also, as I shall argue later, spatial grammar systems seem to be the best candidates for capturing the property of *emergent form*.

5 Relation to other work

The biological analogy in architecture is not new; it has a history that is far older than Darwinian theories of evolution. One book length [Ste79] and numerous shorter works have been published in the area. To my knowledge, none of these build an analogy to evolution in the manner that I do here. In particular, the papers I have encountered do not take a computational view of the evolutionary analogy.

There has been much work in genetically inspired methods for optimization and search; [Gol89] presents an overview, a historical survy, and an extensive bibliography. The work and applications that he reports only encounter design tangentially, but provide what appears to be a rich source for insight into mechanisms within the genetic analogy. The basic material on genetic algorithms treats the coding of representations quite informally; as a point of departure, I maintain that the coding sould be as formal as possible, so that the units of genetic code that are manipulated might be, to the greatest extent possible, semantically relevant to a design domain.

Lenat[LB84] has built a series of programs that learn by discovery. The earliest of these, AM, combines a frame-like data structure (with inheritance) and a set of constructive rules, indexed to their locations of applicability, that opefate on the frame. The frame-like structures represent mathematical concepts and the rules: (1) refine these or transform (mutate) them to form other concepts, and (2) propose new tasks to be performed . Control of the rules is accomplished by a scheduler that acts upon priority ratings of tasks. AM was able to discover interesting mathematical knowledge, beginning from basic definitions of set theory. A later program, EURISKO, developed similar capabilities for non-mathematical domains (including 3-D VLSI design) by developing frame structures and mutations whose form closely mimicked knowledge of heuristics. A main lesson from Lenat's work is that:

> "... it's important to find a representation in which the form*-content mapping is as natural (i.e., efficient) as possible, a representation that mimics (analogically) the conceptual underpinnings of the task domain being theorized about."[LBS4, p. 276']

Within design research, there have been several forays into a "discovery" approach to design, in which mutation operators are introduced into what are essentially prototype based design representations and processes. Murthy and Addanki [MAST] report modifications to prototypes by moving between nodes in a *graph of models.* They use *modification operators* that capture heuristics for (1) recognizing applicability. (2) calling appropriate analysis procedures, and (3) direct changes to the prototype. Maher and Zhao[MZGS9] propose *analogy and mutation* on search space operators as mechanisms for enlarging the design space of a prototype based design system. They note that their mutation operators tend to be domain specific. Both of these works appear to gain much of their capability from the structure of the prototypes and operators that are available to the system, neither of which are addressed formally. To me, this is a crucial question in the search for a clear design mechanism; if much of the mystery is buried in a structure of knowledge that can be critiqued only by example, tlun how much insight is really gained? In this paper I attempt to set some "ground rules*' for more explicit capture of meaning.

6 Implications of the analogy

It appears to me that the organization of GDS presents a possible mechanism for the generation of designs that could be judged creative. In this Section I present, in no particular order, a number of observations and arguments that support my contention, as well as some of the problems that I see.

At the beginning of this paper I set out three criteria for a proposed creative design system: invention of the unexpected, computational clarity, and absence of heuristics. The organization of GDS is, I believe, remarkable in its achievement of the latter two. (1) The mechanism I propose is, at its highest level, simple. It consists of nothing more than an infinite loop, and a small set of conceptually simple mechanisms for mutation, development, and selection. It does require design space machinery, but several exemplars of this have, to greater or lesser degrees, been created, for example shape, structure, and solid grammars, and rectangular layout systems. (2) The mechanism, by itself, contains no heuristics, although there is ample opportunity to introduce them, and to imagine hybrid human-computer systems, without disturbing the organization in any essential way. Search space operators, search strategies, mutation operators, goals, and evaluators are the chief vehicles for such insertions.

Whether GDS can meet my first criterion, creation of the unexpected, can only be determined experimentally. However, natural evolution provides a strong existence demonstration of the potentials of the GDS organization.

Rules appear to have a magnification effect on designs. As demonstrated in [Kni83a,Kni83b,Kni83c], changes to rules lead to substantial changes in the corpora of designs specified by those rules. As a specific example consider Figure 2. The initial rule *(a)* simply rotates a square about its centroid by 45 degrees. Some of its (well-known) derivations are shown. A simple mutation of this rule involves adding a translation of less than $\sqrt{2}/8$ times the side length of the square along the vector (1,1). When combined with the original rule[9], the mutated rule produces a new sequence of derivations whose geometry is very different from that of the first derivation set. Such magnification gives me hope that insights into creative acts in terms of simple rule changes could be a welcome serendipitous result of an implementation of the GDS organization.

Other researchers[CRRG87] have drawn distinctions of creativity in designs by the method of design generation employed. These methods appear to me to be based upon the behaviour of human designers, and to provide plausible and interesting explanations of such behaviour. To me they do not prescribe how creation by computer might best be accomplished, but do introduce a lot of machinery in the process. GDS is more parsimonious in this regard; it uses a single method for all design generation.

The GDS organization is not guaranteed to produce creative designs, nor are tests of completion or exhaustion likely to be easy (or possible) to find. In this matter it is very much like human designers, who also can provide no guarantees

[9] **Admitting that genes in nature are mutated and replaced.**

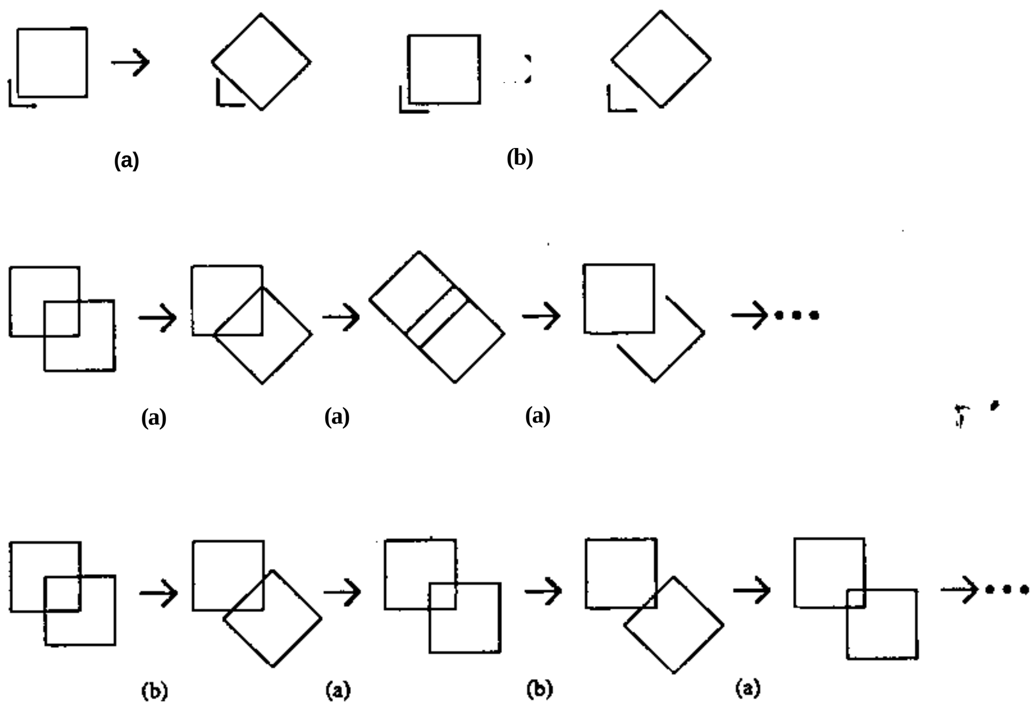


Figure 2: A mutation of a single rule can have a large effect on the derivations it might generate.

of creativity. I suspect that an implementation would be a tool, qualitatively different from any that exist, for exploring vast design spaces, and would be quite likely to find creative solutions.

GDS provide opportunities for experimentation with knowledge based approaches to increasing its capabilities. Through changes to search space operators, search strategies, mutation operators, goals and evaluation devices, heuristics can be introduced, without in any way changing the overall structure of GDS. I would contend that this distinguishes GDS from current proposals for extending systems based upon exhaustive enumeration[CoyS9].

Formal extensions to GDS in terms of grammars that act upon mutations seem plausible.

Different types of reproduction can be imagined within GDS. Design spaces for different designs can be merged, resulting in 'genotypes' that are widely different from their parents. For these the criteria for survival can be relaxed[10] for several generations, until new successful adaptations emerge. An implication of merging

[10]In contrast to the situation in natural evolution.

of design spaces is that the 'phylogeny' of designs can be reticulated and need not be limited to the tree that must occur naturally. This would bring the behaviour of GDS in accord with observed 'cultural evolution' of designs[Ste79, 'p.101'].

New technologies can be introduced as new rule sets for forming and combining parts. Thus the GDS organization can be responsive to technological change in a manner uniform with its basic mechanisms.

The process requires mimimal bootstrapping. As long as designs that can be evaluated are produced, then the process starts and is thereafter self-sustaining.

GDS presents a goal-less process in the sense that it wanders wherever 'fitness for survival' takes it. The goals and evaluation devices of design only provide a simulated environment. The larger process has no goals; and is utterly mechanistic.

With correct technical formulation, the process will work. There is in biological evolution an existence demonstration (if not proof) for it. The vast spans of time required for biological evolution can be greatly reduced because: (1) designs (at least in architecture) are much simpler than living things, (2) the generation cycle can be greatly shortened, and (3) the mutation rate can be increased.

Several properties of design spaces seem to be required by the GDS. *The 'genotype' of designs must be potentially expressive.* It is important to be able to represent a wide range of variations (if not all of them). Thus the representation scheme of the design space needs to be highly expressive. For realistic systems this would seem to preclude simple attribute selection schemes that are obvious when naively using frames. *The search space operators must be semantically relevant.* If the behaviour of GDS is to be transparent to humans, then the search space operators in a design space should specify plausible 'design moves'. Of all current formal approaches, that of spatial grammars seem to best fit this requirement. A consequence of using rules of this type is that properties of the phenotype cannot in general be predicted, but I do not see this as a problem, in that 'genotypes' will change constantly as 'evolution' in GDS progresses. *A representation scheme appears to be a necessity.* The representations manipulated by GDS must have a correspondence with real designs. If they did not, the necessary evaluation mechanisms could not be built and the loop at the heart of GDS could not be closed. This requirement appears to pose certain problems for drawing based representations (but see [Sti81]). *The search space operators should be based on a grammar (or algebra) that makes a strong sense of emergent form possible.* The stronger the sense of surprise in the process of generating the 'phenotype' from the 'genotype', the more leverage the 'embryonic development process' will have. Shapes from shape grammars (being individuals) and the algebra of r-sets (being based on point sets) are two extant examples. *The mutation operators must be highly redundant.* It must be possible to mutate one form into a large variety of others by a series of small mutations. In other words, the transitive closure of the mutation relation between design spaces must be as dense as possible.

A problem with the evolutionary analogy is its absolute requirement for some mutations that present survival enhancement (or at least neutrality) at every step. Without this, the mechanism breaks down. Thus, in its most naive form, GDS would display some of the shortcomings of hill-climbing search strategies[11], although the pertubations introduced by random mutation should partially surmount this problem.

7 Features of a research program on genetic design systems

To my knowledge, only toy genetically inspired design systems have been implemented (see [MZG89] and [LB84]), and few direct theoretical results have been achieved (see [Gol89]). However, much of the requisite formal machinery, e*p<*-cially that related to design spaces, does exist in some form. A research program on genetic design systems would be constrained by this state of affairs.

I see two somewhat conflicting issues that should be addressed in a research program on genetic design systems. The first is a requirement for further formalization. It would be very useful to precisely describe the structure of a GDS in some mathematical form. Having definitions for each class of computational object and precise abstract descriptions of the overall process would greatly aid both understanding of the known theoretical problems and implementation of prototype systems. The second issue is the importance of having an experimental laboratory for GDS research. It is through a working implementation[12] that insights to the crucial research questions will arise. An implementation would also provide measures of the performance of the idea, and these would not-easily be found in another way. Genetic design systems essentially perform search in a space of rule-sets, and are sufficiently abstract that insights, at least in the beginning, will come most easily through empirical (in contrast to analytic) means.

I propose then a research program with two parallel (but interconnecting) threads. The first (*THEORY)* would aim at mathematical description, and would meet its a major milestone with the production of a set of precise definitions of all of the components in a GDS. The second *{APPLICATIOX)* would aim initially to produce a minimal GDS, containing all components of the architecture, and making the necessary technical compromises to quickly achieve an operational system. Care would be taken to make this initial application as modular as possible, so parts of it could be independently replaced. At this point the two threads would hopefully begin to inform each other. From the implementation, *THEORY* would learn what questions are important and what theories should be formulated and proved; these would be the second major task for *THEORY.* From the theoretical results, *APPLICATION* would fine-tune its implementation, rewriting some modules and replacing others, but this would not be its major second

[11] Genetic design systems can be viewed as performing hill-climbing search in a space of dcM^n spaces; their power may lie entirely in their realization of a meta-level.

[12]Such an implementation would need to need to achieve an entire mechanism; it would IK* important to have a base case of purely automatic behaviour.

task. *APPLICATION* would embark on a series of experiments in design, posing problems, observing results, and gaining insight into the operation of the genetic system. At some point, all of the implementation of *APPLICATION* would be discarded, and the two parts of the research program would come together with the goal of creating and using a theoretically sound implementation.

The research of each thread would have different criteria against which success would be measured. For *THEORY the* criteria are the relevance and quality of its formal results. For *APPLICATION,* the criteria must be more vague; a partial list is: performance in terms of example generated designs, discovery of new mechanisms and effects, and the quality of the system design.

A The design space formalism

lf

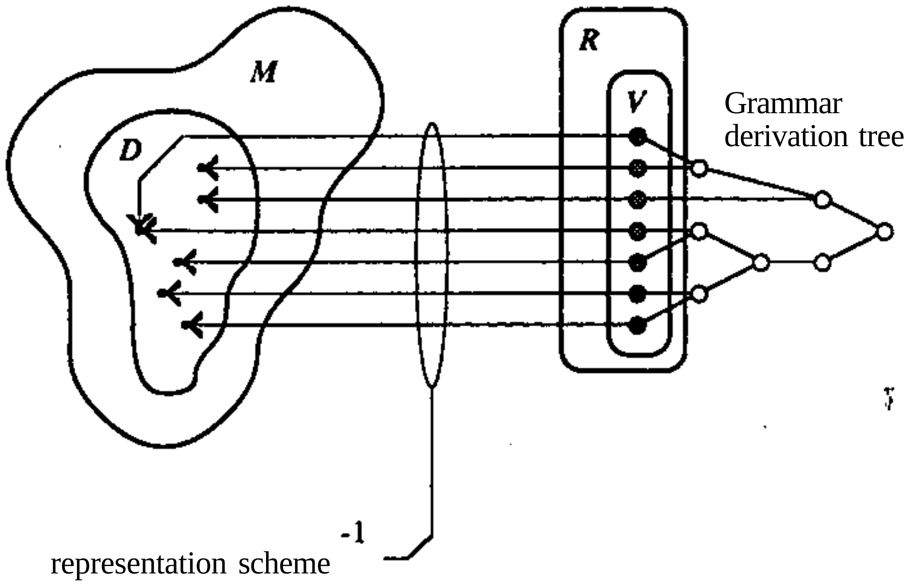
The view of design as search is well-known and needs no introduction here. In this section I present, in summary form, a set-theoretical characterization of the design search view. A more lengthy account may be found in [\Voo90][13].

Whether done by human or machine, design search operates on *symbol structures,* that is, organized collections of *symbols.* The collection of all symbol structures that might be considered in a design task constitutes a *representation space R* and each member *r* G *R* is called a *representation.* Each r G *R* may have interpretations as one or more actual designs. If these interpretations are to be precise. a crisp characterization of designs must be made, and for this, the concept of a mathematical *modeling space M* is employed. *M* is usually described by predicates, universally quantified over M, that describe properties (of all *m* £ *M).* These predicates capture only certain properties of physical objects, leaving others undescribed. For example, the well-known oriented 2-manifold conditions describe the idealized geometry of physical solids, but do not describe materials, surface textures, reflectance, or a host of other properties. The *semantics* of representations are defined by building a relation 0 : *M* — *R* on the sets *M* and *R* that associates elements of *R* with elements of *M.* If *(m* G *M.r* G *R)* G 0 then *r* is said to *model* or *represent m.* 0 itself is called a *representation scheme.*

In interesting design problems, the set *R* cannot be directly enumerated; its size is typically huge (or infinite). Various techniques of indirect specification, for example, algebra and grammar notations, are typically used instead. All members of *R* are generated by these indirect syntactical methods and are therefore *syntactically correct.* Members of the set *M* are not directly generated at all; they are only known by the existence of members of *R* that can be shown to represent them.

Since only members of *R* are directly available to any computational process it is convenient to define 0 in terms of its inverse relation $0^{\sim 1}$, for mapping members of *R* to members of M_y and this is shown in Figure 3. It is useful to

[13]**T\vo more fundamental sources are [ReqSOj for representation schemes and [NS72] for search.**



M	Modeling space	*D*	Domain
R	Representation space	*V*	Range

Figure 3: An abstract depiction of a representation scheme

describe 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. 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 /?. The codomain of Θ is *R.* The range of Θ, denoted by V, is the set of all members of *R* (by definition syntactically 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 (unambujuousness). uniqueness,* and *abstraetness.*

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[14].

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 corresponds to more than on 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[FCG*89], in which the modeling space is the set of all arrangements of loosely packed, non-overlapping, orthogonally oriented rectangles in two dimensional euclidean space $\Re^2$. 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 rectangular layouts in the modeling space that differ in the dimensions and locations of the constituent rectangles but are the same with respect to the spatial relations specified in the graph. Thus the representation scheme for LOOS is decidely ambiguous, as every representation corresponds to an infinite set of rectangles, yet the ambiguity is precisely controlled, since specific spatial relationships are faithfully modeled.

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 an unambiguous representation scheme can be said to model instances

[14]Well-formedness is an essential quality for exhaustive search strategies.

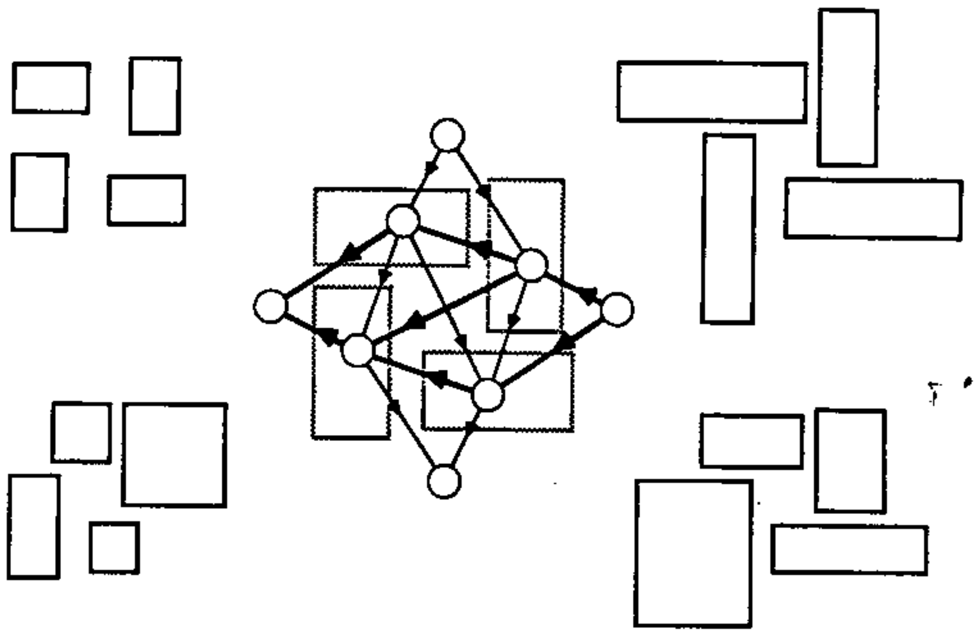


Figure 4: Ambiguity in the representation scheme of LOOS

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 $T : M \rightarrow M$, can be constructed between elements of the modeling space. Two objects are in T if both correspond to the same representation in V. If T 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, but another formal construct, the *search space*, is also required. A *search space S* is a 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 $I \subset R$. 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 $r \underset{O}{\Rightarrow} r'$. If there exists a sequence of direct derivations using operators from O, such that $r_0 \underset{O}{\Rightarrow} r_1 \underset{O}{\Rightarrow} \dots \underset{O}{\Rightarrow} r_{n-1} \underset{O}{\Rightarrow} r_n$ then r_0 *derives* r_n in O.

$r_0 \underset{O}{\wedge} r_n$. The *design collection* of a search space is all representations *r e R* such that $i \underset{o}{\text{i}} r$, i £ /.

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. 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 both to both properties in *M* *(modeling space monotonicity)* and to the symbol structures in *R (representational monotonicity).* If a set of properties P_t*\f* of any m 6 *M* cannot be altered by the application of an operator, then the operator is monotonic with respect to *P\f.* 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 O if. by application of any combination of operators from *O* starting from any elements of /, representations in *V* sufficient to model all of *D* can be reached. If O 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 5 is *non-redundant* in *R* (or *V)* if there is at most one sequence of operator applications beginning from *i* 6 / that can generate any r G *R* (or *V).*

A search space exists in the absence of any specific 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 *secuvh 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 caHcd 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. Design problems typically have multiple and conflicting goals. Designs are compared against goals as they are reached by search space operators and these comparisons are used in making the decisions at each step in design[15].

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 [Mar76,RGS8].

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. 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, if one is doing preliminary design for an airport, it is useful to have operators for placing runways, organizing pedestrian and vehicular traffic flow, developing spatial signage conventions, etc. A set of operators that describes the actual construction of runways and hangars would be much less semantically relevant.

References

[Coy89] R. F. Coyne. *Planning in Design Synthesis: Abstraction-Based LOOS (ABLOOS).* Technical Report, Engineering Design Research Center. Carnegie Mellon University, 1989.

[15]The separation of generation, goals, and evaluation devices in the search paradigm is more than a formal nicety; given current undertanding of designing physical artifacts, it appears to be a necessity: Deriving form from its behaviour has proven to be extremely difficult, even in single-performance design problems. The alternative is to consider the behaviour of forms that are generated in some other terms, and to use the knowledge of behaviour to guide the generation process.

[CRRG87] R.D. Coyne, M.A. Rosenman, A.D. Radford, and J.S. Gero. *Innovation and Creativity in Knowledge-Based CAD,* page . North Holland, Netherlands, 1987.

[Da\v87] Richard Dawkins. *The Blind Watchmaker.* W.W. Norton and Company, New York, N.Y., 1987.

[FCG*89] Ulrich Flemming, Robert F. Coyne, Timothy Glavin, Hung Hsi, and Michael D. Rychener. *A Generative Expert System for the Design of Building Layouts.* Technical Report EDRC - 1989 Report Series. Engineering Design Research Center- Carnegie Mellon University, 1989.

[GolS9] David Goldberg. *Genetic Algorithms in Search Optimization aiufiMachine Learning.* Addison-Wesley, Reading, MA., 1989.

[Kni81] T.W. Knight. Languages of designs: from known .to new. *Environment and Planning B,* 8:213-238, 1981.

[Kni83a] T.W. Knight. Transformations of languages of designs: part 1. *Environment and Planning B,* 10:125-128, 1983.

[Kni83b] T.W. Knight. Transformations of languages of designs: part 2. *Environment and Planning* 5, 10:129-154, 1983.

[Kni83c] T.W. Knight. Transformations of languages of designs: part 3. *Environment and Planning* /?, 10:155-177, 1983.

[KniSS] T.W. Knight. Comparing designs. *Planning and Design,* 15(1):73-110, 1988.

[LB84] Douglas B. Lenat and John Seely Brown. Why am and eurisko appear to work. *Artificial Intelligence,* 23:269-294, 1984.

[MA87] Seshashayee S. Murthy and Sanjaya Addanki. Prompt: an innovative design tool. In *Sixth National Conference on Artificial Intelligence.* pages 637-642, AAAI, Morgan Kaufman Publishers, July 13-17 1987.

[Mar76] Lionel March. *The logic of design and the question of value,* chapter Introduction, pages 1-40. Volume 4 of *Cambridge Urban ami Architectural Studies,* Cambridge University Press, Cambridge, U.K., 1976.

[MS85] Lionel March and George Stiny. Spatial systems in architecture and design: some history and logic. *Environment and Planning* 0, 12(l):31-53, 1985. Paper presented at the Seventh International Conference on Systems Dynamics. University of Brussels, Junel6-lS. 19S2.

[MZG89] M.L. Maher, F. Zhao, and J.S. Gero. An approach to knowledge-based creative design. In *NSF Engineering Design Research Conference*, pages 333–346, National Science Foundation, College of Engineering, University of Mass, Amherst, June 1989.

[NS72] Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall Englewood, 1972.

[Req80] Aristides A.G. Requicha. Representation for rigid solids: theory, methods and systems. *Computing Surveys*, 12(4):437–464, December 1980.

[RG88] Antony D. Radford and John S. Gero. *Design by Optimization in Architecture, Building, and Construction.* Van Nostrand Reinhold, New York, N.Y., 1988.

[Ste79] Philip Steadman. *The Evolution of Designs*. Volume 5 of *Cambridge Urban and Architectural Studies*, Cambridge University Press, Cambridge, U.K., 1979.

[Sti81] George Stiny. A note on the description of designs. *Environment and Planning B*, 8(3):257–268, 1981.

[Woo90] R.F. Woodbury. Searching for designs: paradigm and practice. *Building and Environment*, 1990. in print.