Innovative Design Systems: Where are we, and where do we go from here?*

D. Navinchandra

CMU-RI-TR-90-01 2

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

12 January 1990

*Title Idea taken from R. Davis' 1983 Expert Systems Tech. Report

Table of Contents

List of Figures

Innovative Design Systems: Where are we, and where do we go from here?

This report was last updated in March 1989

1. Introduction

This report reviews the state of the art in systems that innovate. The review concentrates more on the techniques used in these systems rather than what the system actually did.

This report reviews some of the important issues for DA systems that work in domains requiring non-routine and innovative approaches to problem solving. This part of the report covers issues such as analogy & metaphor, exploration & discovery and creativity.

Readers who wish to get a broader sense of the techniques used in DA systems may refer to the following papers [Gross 86, Stefik 80, Tong 86a, Mostow 85, Nilsson 80, Sriram 86, Ullman & Dietterich 87, Tong 86b].

2. What is Innovative Design?

Before talking about tools and techniques, we will first try and characterize innovative designs. Earlier definitions of innovative design have tended to finess over details. For example, innovative design has been defined as any design that is: new or different or elegant or uses new ideas or is an improvement over its peers [Dixon 65]. Such a definition is correct, but it does not tell us how to measure "newness". How new is something? In this section we shed a little more light on the problem of trying to distinguish between innovative and routine design. No definition will ever be complete or accurate, but something is better than nothing.

The real purpose of trying to define innovative design is to try and identify the characteristics of innovative design. With a list of such characteristics we can better focus our research efforts. If DA systems that emulate some of the characteristics of innovative design can be built, then, such systems can be said to have some form of innovative behavior.

3. Characterizing Design

The first question to ask is: "When does a design qualify as innovative (as opposed to routine) ?" Ironically, the answer is that there is no absolute measure of innovativeness. It does not make sense to label a design as being innovative, because the perception of an artifact as being novel or not, lies not in some inherent property of the artifact but in the eyes of the beholder. For a given observer or a given group of observers (e.g. Mechanical Engineers) there is a set of design styles they are accustomed to seeing in the artifacts designed by their peers. For example, a construction equipment designer will view a construction robot (e.g. an excavation robot) as being innovative. On the other hand, a Robot designer will view the same construction robot as being yet another application. The robot's sophisticated vision, tactile and position sensing systems are not new to the Robot designer and are part of the ***design culture*** he works within. For the construction engineer, on the other hand, an excavation robot is novel because robotics is outside his design culture. *A design culture is defined by the common practices, design styles and technologies used by people who operate within the culture. It defines the context they operate within, it's their weltanschauungen.* The notion of design culture gives us a datum from which designs can be characterized, where, each design culture has its own way of viewing designs as being routine or innovative.

Here are three different types of design:

The simplest type of design is ***procedural.*** A design process is said to be procedural when all the steps are known beforehand and all the critical decisions have been made. In such a situation, the designer just follows instructions blindly. For example, the design of a concrete beam which involves the calculation of beam depth, width and the amount of steel reinforcements is procedural in nature.

Routine design. A design process is said to be routine when all the steps are known beforehand, but all the decisions are not known. The designer may have to search a space of design alternatives to arrive at the final design, however, as the steps are known there are no surprises. All possible outcomes are known and appropriate responses are also known. All knowledge used in problem solving is from within the original design culture. Let's take an example from simple TTL logic design. Given a logic specification there is a set of simple steps that will convert the specification into a circuit. The design process is routine: Specification --> Truth table --> Karnaugh map --> Boolean equations --> TTL logic diagram. A designer who uses the above process to design simple TTL logic circuits **only** (e.g. binary adders or decoders) can be said to be doing routine design. However, it is not necessary that a routine design process will always produce routine designs. For example, if we gave our TTL designer the specifications of a ROM or a Programmable logic array, then he could follow the same old routine process to come up with the combinational logic of a complex product that would be very different from any of the designs previously attempted him. Where does the novelty lie? In the process or in the specifications? There is no clear cut way of classifying a design process, even within the context of a given design culture.

The novelty of a design can be characterized along three dimensions: the initial ***specification***, the design ***process*** and the final designed ***product.*** The Figure shows eight different types of design within the framework of routine design. This table goes to show that there is no objective demarcation between routine, non-routine, and innovative design.

Specs	Proc.	Prod.	Example
M	M	M	Routine Design. (Beam design)
D	M	M	Where the application is novel but the solution is quite mundane (eg Office automation of a Church)
M	D	M	A new faster/cheaper process (Faster beam design)
M	M	D	Not possible
D	D	M	Similar to DMM. (e.g. A new Systems analysis technique gives the same old solution)
M	D	D	A new method for solving an old problem. (eg A Microprocessor controlled toaster.)
D	M	D	A different spec. passes through a mundane process to become a novel product.
D	D	D	A truly non-routine design situation.

LEGEND:
Specs = Specifications to the design process.
Proc. = The design process.
Prod. = The product, the final design.
M = Mundane
D = Different, not-mundane

In the Figure above, the word "different" has been used to denote how different a particular dimension is with respect to the norms of the given design culture. The word is used loosely. In reality there are several shades of difference between a design and the design culture it is a part of. Let's examine this aspect even further.

3.1. Shades of Difference

All new ideas have their origins in old ideas. A human being's ability to understand some new concept is predicated on him/her already possessing sufficient support knowledge for the new idea. Likewise, all new designs involve the use of previously acquired knowledge. A designer acquires such knowledge in particular from his design experience and his time in college. This body of knowledge defines the design culture he resides in. In a design situation, designers normally draw upon this knowledge. However, some design situations cannot be solved by the design knowledge possessed by the designer. Under these conditions he will have to either learn more about the design culture he is operating within or try to use knowledge from outside the current design culture. The designer can reason analogically from his experiences in situations other than just design. The designed artifacts we see around us are a mixture of knowledge of the design culture the artifact belongs to and knowledge drawn from various other sources. For example, a biomedical engineer, given the task of designing a non-surgical method for removing urinary calculii[1] is faced with a task requiring innovations drawn from outside his design culture. He has to find some way of reaching the calculii and removing it without surgery. The requirement of finding a non-surgical method reminds him of catheters. This idea is from within his design culture. Now that he has a way of reaching the calculii, he has to find a way of grabbing it. This reminds him of kitchen tongs and she decides to put small clips at the end of the probe. Next, he has to pull out the calculii without scratching the insides of the urinary tract. This can be done by expanding the tract. The word "expanding" reminds him of a balloon and he decides to introduce a small balloon at the end of the probe. This will aid in pulling out the calculii. This example, taken from a real design problem, shows how remindings can occur from within and without the design culture. As the designed artifact drew upon knowledge from outside the design culture of bio-medical engineering it was viewed as being innovative. This is the essence of measuring difference. Let's expand on this idea.

Conceptual distance. To measure the difference between two ideas, one has to have some measure of their conceptual closeness. For example, nuclear engineering is closer to mechanical engineering than say, biology or psychology. Conceptual closeness can be measured if one has a conceptual clustering of the different domains. Such a clustering might be in the form of a tree and the number of links between two domains can be used as a measure of closeness. For example, all engineering fields could be under one heading and all humanities could be under another. The number of links to be traversed from nuclear engineering to mechanical engineering is less than the number of links between mechanical engineering and psychology. Every time a new innovation is made, this tree will have to be reorganized.

Let us draw up a spectrum for characterizing designs as a function of the difference between the knowledge used in solving a design problem and the design culture of the design domain. For brevity, let us assume that new designs are generated by drawing knowledge only from one source at a time. The

---

[1] A calcium deposit in the urinary tract

spectrum has the following two dimensions:

the **difference** between the knowledge used (base knowledge) to solve a problem (target problem) and the knowledge subsumed by the design culture of the target problem. The greater the difference between the base and target the greater the perceived innovativeness of the product.

the **level** of knowledge drawn from the base culture. The higher the level of knowledge used, the easier the transfer of knowledge. A low-level concept is a basic principle or law in some domain. A high-level concept is some final result or equation from the domain. For example, the author once came across a thesis that applied control theory to highway maintenance. The student had not derived new results in control theory but had only applied some of the existing results of control theory to highways. He had used high-level concepts borrowed from control theory. If, on the other hand, he had derived some new results, he would have had to draw upon low-level concepts in control theory and mathematics.

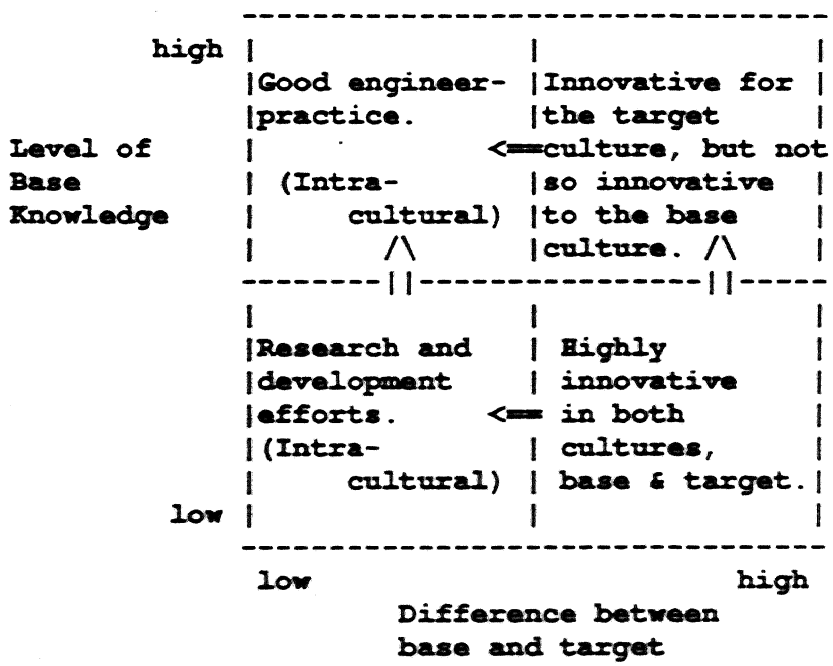

Figure 3-1: Design Spectrum

Figure 3-1 shows a spectrum of designs along two dimensions. The higher the difference and the lower the level of knowledge used the greater the perceived inventiveness of the product. Here are some examples that will help explain the above table. Consider the following scenario: a Civil Engineer acquires and programs a robot for laying bricks. The project will be viewed by civil engineers as very innovative idea. Robotics researchers, however, will view the project as "yet-another-application". This is because the brick laying robot is just some standard robot arm programmed to perform a new task. In this example a well developed, high-level concept from a base domain (robotics) was transferred to a very different target problem (construction). This kind of design falls in the upper-right corner of the spectrum.

If, however, the engineer had decided to work on a much harder robotics project things would be different. Consider this scenario: the engineer decides to develop a general-purpose window cleaning robot for high-rise buildings. This task is so hard that no off-the-shelf, well-developed, high-level concepts in robotics are available. The window-cleaning robot would need a good 3-D navigation system with a good vision system that can recognize windows, pay attention to dirty spots and operate under almost any lighting conditions. The engineer will have to design a robot from basic principles of robotics, working with less developed, low-level ideas. The product of this second type of design will be viewed as inter-culturally innovative (lower-right box). The other two types are intra-cultural. In an intra-cultural setting, the level of knowledge used in design determines the innovativeness of the design. A designer who uses high-level knowledge of a domain is said to be a good engineer (not a scientist). He uses the well established results of the domain, and as he is not involved in any kind of innovative design, he is happy with access to knowledge only from within the design culture he belongs to. Conversely, engineers who work from the first principles of the target problem's design culture can be viewed as doing research and development.

In the examples above, we made two major assumptions which have the following implications:

1. To draw out the characterization, we assumed that innovative designs are derived from the application of one base concept to a target problem. In actuality, however, designs involve all types of approaches. Parts of the design may be innovative while other parts might require routine or procedural design.

2. Another aspect of design is: that which is innovative today ceases to be innovative in the future. A new design idea that draws on some extra-cultural base concept will end up becoming part of the target domain's design culture. It is for this reason that keeping one's design innovative and competitive is a constant struggle for newer ideas.

3. In all our discussions we have assumed that innovation comes from applying some base knowledge to a target problem. Innovation also stems from trial-and-error processes, both random and informed.

3.2. In Conclusion

In order to emulate innovative behavior in a DA system, one has to consider the following questions:

- Can the program reason analogically?
- Can it reason at several levels? Can it use principles and results of several domains?
- Does the program have access to knowledge outside it's original design culture? If so, how can it use this outside knowledge. How should one represent and index knowledge?
- Can the design generate alternatives? Can it explore new ideas? Does it ask the right questions?

These issues are open research problems in AI and DA. The rest of this this paper reviews work that has been done towards answering the questions listed above.

4. Analogy

4.1. Introduction

Experienced engineers have the ability to utilize knowledge gained from previous experiences to provide novel solutions to a wide range of problems. Hence, this type of reasoning should be incorporated in programs that attempt to emulate the intelligence of engineers. A wide spectrum of problem solving techniques have been developed by AI researchers during the last few decades. Some of these techniques have been incorporated in Knowledge-based Expert Systems (KBES). However, the techniques used in current day KBES do not adequately exploit the reasoning process that went into solving previous problems.

The purpose of this section is two fold: 1) to provide an overview of a problem solving technique - analogical reasoning - that can play an important role in building KBES for engineering; and 2) to provide an intuitive understanding of the role analogues, heuristic-rules and first-principles play in technical problem solving.

The rest of the section is organized as follows. Section 4.2 provides an overview of Analogy and introduces a model of Analogy. Section 4.3 reviews the literature and Section 4.4 introduces the different types of knowledge used for solving engineering problems.

4.2. Analogy: Some Fundamentals

4.2.1. What is Analogy?

Analogical Reasoning (AR) involves the use of past experience to solve problems that are similar to problems solved before. A formal definition is provided by Carbonell:

> "Analogical problem solving (reasoning) consists of transferring knowledge from past problem solving episodes to new problems that share significant aspects with corresponding past experience - and using the transferred knowledge to construct solutions to the new problems." [Carbonell 86]

4.2.2. A Model of Analogy

The various components that would be involved in a computational model of AR are shown in Figure 4-1. This model is based on the work reported in [Gentner 83] and views AR as a four stage process: *Retrieval*, *Elaboration*, *Mapping*, and *Justification*.

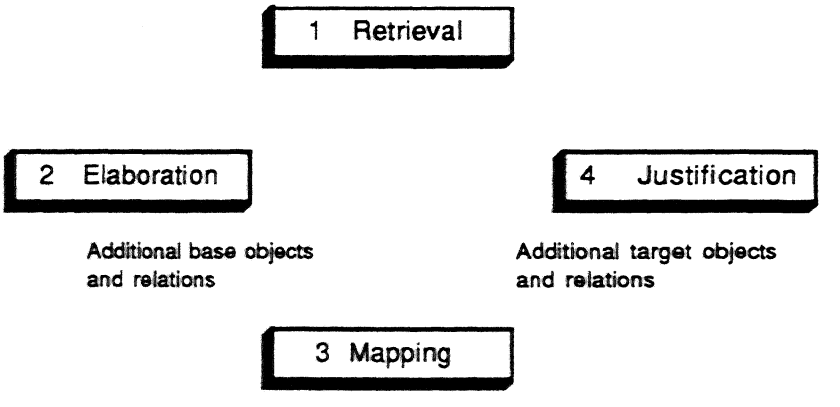

Figure 4-1: A Model of Analogy

The connotation of the above components is described below through the following metaphor taken from the building design domain: *The building is like a beam.*

1. **Retrieval:** Given the target, this process notices and retrieves a potentially analogous state and places portions of the base and target in correspondence. In the above example there is no need to retrieve the base (beam) since it is provided in the metaphor.

2. **Elaboration:** Given the base and the knowledge about the base, derive additional attributes, relations, and complex causal chains involving the base. A derived attribute in the above example may be *a beam resists forces in transverse direction*, a derived relations could be *the beam is made of steel*, and a derived causal relations could be *IF the beam is made of steel THEN the modules of elasticity is 36 ksi.*

3. **Mapping:** Given base attributes, relations, and causal chains, map selected ones over to the target (with modifications). Here the attributes/relations that are salient to a beam can be mapped. For example *the beam resists forces in the transverse direction* can be mapped into *the building resists forces in the transverse direction.*

4. **Justification:** Given mapped target attributes, relations and causal chains, justify they are, in fact, valid. Modify these if needed. In the above example justify that the building can resist transverse forces.

For a design automation system the AR steps involve:

1. Determining a Cue based on a given design representation.
2. Retrieving a Design.
3. Synthesizing parts of the old design in the new one.
4. Verification and Debugging the result.

The rest of this section discusses these steps. Before embarking on the discussion, we provide some background of analogical problem solving methods.

4.3. Analogical Reasoning and Problem Solving: Background

4.3.1. Early Work on Analogical Reasoning.

One of the first interesting programs was called ANALOGY [Evans 68]. This program could solve simple visio-spatial problems like the one in Figure 4-2.

One of the important ideas introduced by Evans was the use of generalizations in the derivation of the transformation rules. For example, in Figure 4-2 we can describe the transformation rule for mapping A to B as "Rotate the inner-most triangle". The ANALOGY program can generalize the rule to "Rotate the inner-most object".

Kling [Kling 71] introduced a program ZORBA, that proved theorems analogically. The program is given a theorem to prove. It is then given an analogous base theorem to use to prove the target. The program is interesting, in that, it proves a theorem efficiently by using an analogous theorem and proof, and thereby reduces unnecessarily search. The program could not perform generalizations, as the ANALOGY program did.

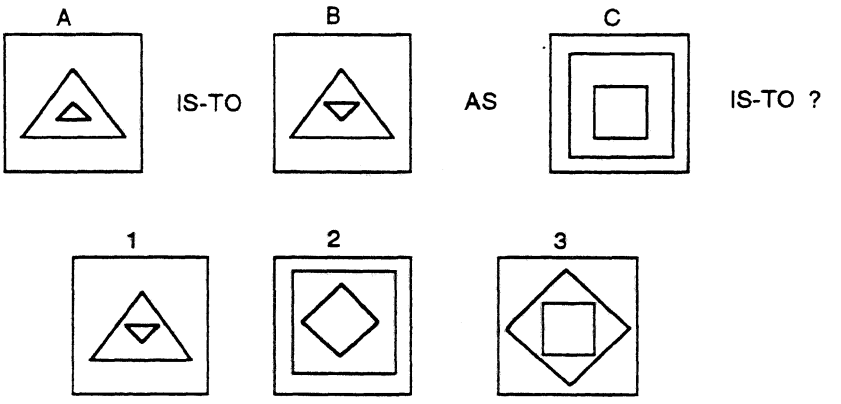

Figure 4-2: The ANALOGY program

4.3.2. Recent Work on AR

Carbonell. Carbonell introduced a program that solved problems by combining analogical reasoning and means/ends analysis [Carbonell 83a]. Given an initial state (base) and a final state (target), the program searches the space of solutions using transformation operators. This approach was called Transformational Analogy.

A limitation of Transformational analogy was the fact that problems that share some characteristics need not share problem solving strategies. This led to work on Derivational analogy [Carbonell 83b]. The mapping of base to target is done by transferring the base's reasoning process (derivation) to the target.

Winston. Winston [Winston 80, Winston 81, Winston et.al. 83] has made a considerable contribution to Analogical Reasoning. Much of his work uses a characteristic representation scheme. Mapping is carried out by matching the links of the base and target representations.

Structure Mapping, Causals and Explanations One of the limitations of early work is the belief that base and target should be matched using the feature sets [Tversky 77]. This approach is limited compared to the Structure Mapping theory [Gentner 83] and the Systematicity principle [Gentner & Toupin 86]. The systematicity principle is used to map base to target by concentrating on the causal relations while regarding all other relations as independent. An interesting example builds the analogy. "The atom is like the solar system" (Figure 4-3).

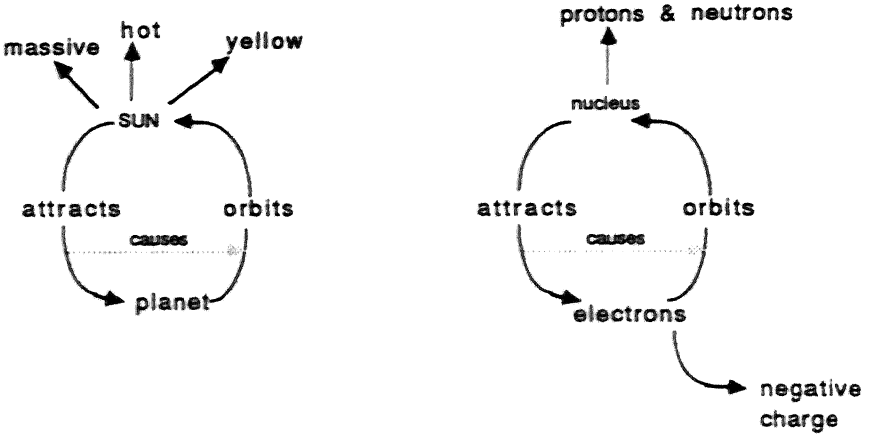

Figure 4-3: The atom is like the solar system

An important assumption of this approach is that, the given causal network of the base is assumed to be

relevant to making the analogy. The trick is to develop an explanation of the base that suits the needs of analogy being drawn . This idea is called purpose-directed analogy [Kedar-Cabelli 85a].

4.3.3. Requirements for a Computational Model

The AR model should be able to satisfy a number of requirements, some of which are outlined below [Kedar-Cabelli 85b].

1. AR is knowledge-intensive. Hence, a model of AR must be knowledge-based. It must have some representation of the domains to be put in correspondence beyond the features of the target and base supplied as input to the analogy.

2. AR requires the ability to recall previous experiences. Hence, a model of AR must have mechanisms for organizing experiences in memory, augmenting memory, and retrieving experiences from memory

3. AR requires the ability to retrieve a potential base solution from a large store of situations. A potentially analogous situation might be dissimilar by surface features from the current situation, or might be represented at a different level of detail than the current situation. Hence, in the model of AR, retrieval cannot be based on a partial match of surface features.

4. In AR, retrieval might be based on shared abstractions between base and target. Hence, mechanisms for generalizing knowledge in base and target would be useful.

5. AR prefers mapping over a system of connected knowledge, rather than independent facts. Hence in a model of AR preference for mapping systematic, causal networks of relations must constrain the possible consistent mappings from base to target.

6. In AR, the interpretation of an analogy may depend on knowing the purpose for which the analogy is being stated. Hence, in a model of AR, explicit representation of the intended purpose of analogy may be used to constrain the possible consistent and systematic mappings from base to target.

7. The model of AR should know what to match and what not to match.

8. AR is performed with limited time and resources. Hence. a model of AR must be computationally effective.

Issues relevant to the above requirements will be discussed in the following sections. The discussion is in the context of building design automation systems.

4.4. Design Representation: Determining what to index on.

In the past, design systems needed only to use knowledge drawn from one domain. Consequently they used domain-specific knowledge and artifact representation methods. In order to build systems that can reason analogically from precedents from within and without the current design domain, we need better canonical knowledge representation schemes. In the example presented in Section 3.1 (page 3) we saw how a biomechanical engineer was reminded of a balloon while trying to solve a urinary disorder. This means that the balloon precedent has to be represented in such a way that it could be used for very

different purposes. One easy way out would be to represent the balloon precedent such that it can be retrieved by a few distinct indices such as: expanding, soft, thin etc. This will work for some limited cases but does not capture the fact that we can retrieve precedents to solve problems in ways we had never imagined before. This means that a precedent should be represented such that it can be viewed in many different ways and can thus be used to serve different purposes.

A popular representation technique is the use of predicate logic. The work on explanation based learning and purpose directed analogy uses this formalism [Kedar-Cabelli 85c]. People working on DA systems for VLSI circuits use standard circuit description languages that allow for abstraction and reasoning about functional behavior. In the mechanical domain, however, the problem of representation is critical. There is a lot of semantic difference between an artifact representation, its function and its behavior. For example, a stapler can be used as a paper weight, as a nut cracker, as a hammer, its spring-action could be used to launch projectiles such as pencils, open it up and it could be used as a set of crude weighing scales.... How does one represent the stapler that it can be retrieved for different purposes?

Two research efforts on representing physical artifacts are trying to address this issue: learning physical domains [Forbus & Gentner 86] and the EDISON system [Dyer et.al. 86]. The first effort bases precedent representation on a qualitative process (QP) theory. In QP theory, a physical situation is modeled as a collection of objects and relationships among them, with processes responsible for causing changes. The continuous parameters of an object, such as temperature and pressure, are represented by quantities that are denoted by an amount and a change-derivative. In the EDISON effort, artifacts are represented in terms of parts, spatial relations, connectivity, functionality, and processes. For example, when describing the relative orientation of two objects, EDISON uses predefined notions such as: coaxial, colinear, adjacent, overlapping etc. Using a set of production rules which know about the behavior of these different types of connections, spatial relations etc., the system is able to simulate the working of the artifact.

We believe that cases should be represented at several levels. This will allow an AR system to appropriately extract information from the precedent case. Following is a proposed representation for design cases.

There is a host of questions that arise regarding representation and indexing of cases in the context of a creative and synthetic task, such as design. What constitutes a "design case"? What information should be incorporated? What features could be used as indices? How are these features extracted from the input? How can the various reasoning levels be captured in an expressive and efficient manner enabling a problem solver to move between those levels during the problem solving process? How is numerical and analytical information incorporated into cases? Should it be incorporated in the case memory or in a different library? How are constraints and tradeoffs represented and indexed?

The case representation we propose is multi-layered. An important aspect of the representation is the existence of mechanisms which allow one layer to be mapped onto the next. We currently use the following layers to handle mechanical engineering related cases:

1. **Linguistic.** Linguistic indices are best suited for direct indexing based on a matching linguistic index. Case attributes provide such indices. For example, a simple household water tap can be indexed in terms of its function, to control water flow; its components, pipe, nozzle, handle, valve and seal; the material out of which it is made, brass; the type of device it is, mechanical; the places where it is intended to be used, kitchen, bathroom, water tank.

Previous research in index determination in the CBR literature [Hammond 86, Kolodner 88, Sycara 87] has identified goals as well as object attributes and scenes as general classes of features that can be used as indices. Since artifacts always have an associated intended function as the purpose/goal of the artifact, it is clear that the intended functional specification should give rise to a set of related indices. In addition, features that capture the physical description of the device (object features) need to be included.

2. **Functional Description.** Devices can be viewed as black-boxes which take inputs and produce desired outputs. In the physical domain, three types of inputs and outputs have been identified: signals, energy and materials [Pahl & Beitz 84]. A characterization of the relationships between the input and the outputs is the device behavior. Device behavior can be represented at several levels of detail. At the highest level, the behavioral description contains the overall inputs and outputs, whereas at more detailed levels domain principles, such as Bernoulli's theorem, Newton's laws and conservation laws could be used.

For example, a household water tap takes a material input (water) and outputs the water in response to a signal (open/close). Taking this a step further, a tap takes the input signal *theta*, and the input water flow rate of *Qin* and produces the output flow rate of *Qout*. The temperatures of the input and outputs are *Tin* and *Tout*. The tap may be represented as shown in Figure 4-4. The figure shows the tap as a box with the following qualitative relationships: (1) The inflow of water (*Qin*) monotonically increases (*M+*) with the signal *theta*. (2) The inflow is equal to the outflow (*Qin* = *Qout*), (3) Temperature does not change (*Tin* = *Tout*). The bottom two statements are qualitative boundary conditions: (4) when *theta* is zero, there is no flow through the tap, and (5) when *theta* is 2π, then the flow is maximum.

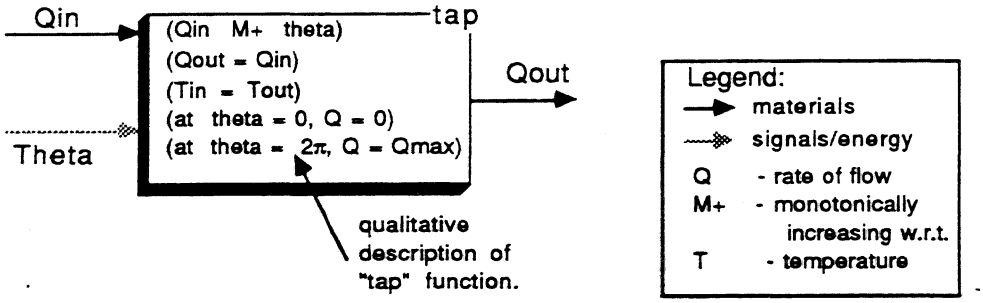

Figure 4-4: Qualitative Device Description of a Tap

3. **Causal Explanation of Behavior.** The Qualitative Device description is at a high level of detail, it does not capture how the tap works. The device behavior can be viewed as a causal explanation of how the structure of a device enables the accomplishment of its functional specifications. We propose to explore the use of causal networks augmented with relations expressed in the language of qualitative physics [Forbus 84, Kuipers 86] to capture the causal behavior of devices. We believe that the augmentation of causal explanations with qualitative relations captures the dynamics of device behavior.

A causal explanation of the tap is shown in Figure 4-5 (the tap's components are shown in Figure 4-7). The representation is a semantic network of components and attributes with causal links between the nodes in the network. We have augmented the causal links by qualitative constraints [Kuipers 86]. The figure is built of several primitives. It has a definition of flow across an orifice with links relating orifice size and pressure drop to the resulting flow rate. It also shows how a screw mechanism causes the tap's cylinder to go up and down, changing the orifice size as a result of the movement. The network shows objects (e.g. the cylinder) and parameters (e.g. orifice size) linked by relations. We have found three type of relations to be useful: (1) Attribute relations (e.g. the orifice has a "size" attribute), (2) Positional relations (e.g. the orifice has pressure P1 to it's left), and (3) Causal relations. The causal relations capture the notion that some parameter (e.g flow) is dependent upon having a positive pressure difference across the orifice. The causal relations are augmented by constraints which qualify the causal relation. For example, the flow rate monotonically increases with respect to the size of the orifice: (*Q M+ size*). In addition to the behavioral constraints, the boundary conditions are also specified (not shown in the figure). For example, the relation that flow monotonically increases with respect to the pressure difference is true only if the orifice is open, that is, *size* > 0 . These boundary conditions are derivable from the configuration space diagram (next section).

4. **Qualitative states and Configuration Spaces.** The causal relationships that describe the behavior

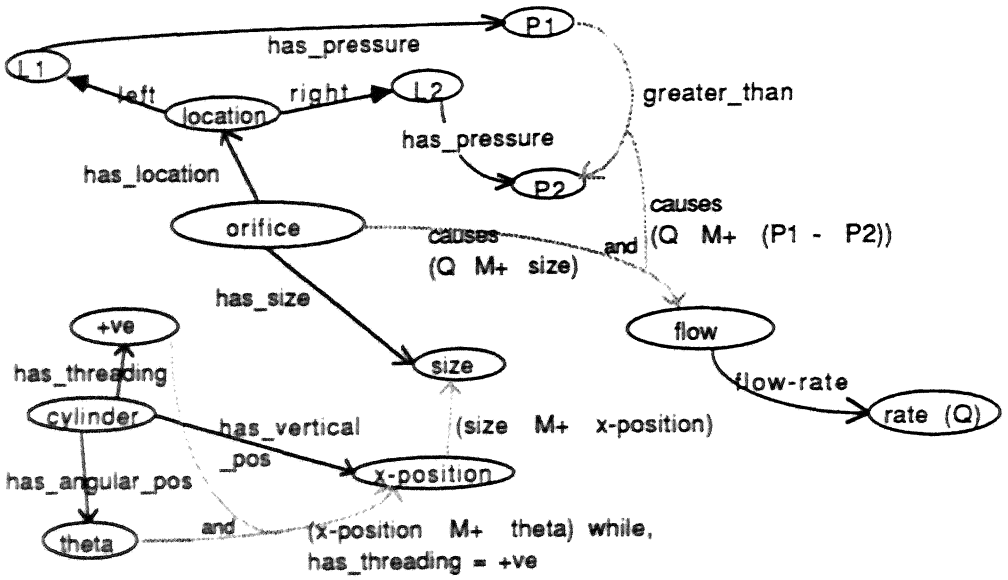

Figure 4-5: Causal Explanation of the Tap Function

of an artifact refer to specific artifact components and relate status conditions such as position and size of the components. The next step is to associate the objects and their status directly to the object geometry. This may be done through Qualitative State descriptions. Qualitative states provide a vocabulary for describing the device behavior. Transitions between the states in the vocabulary are expressed in the causal explanation. The qualitative states can be derived from the geometry through a special type of representation called the *Configuration Space*.

The configuration of a single object is a vector of six parameters, three positions and three orientations, that uniquely define the object's position and orientation in space. Now consider a mechanism with two links. If regarded individually, the two links have a total of two times six, i.e. twelve degrees of freedom. However, because of the fact that two objects cannot overlap in space, some configurations for each link become illegal. The illegal region is also called the *no-go* region. The plot of all *go* and *no-go* regions for any two parameters of a mechanism is the configuration space plot of the mechanism [Lozano-Perez 83, Faltings 89, Joskowicz & Addanki 88]. All regions within the configuration space represent qualitative states of the mechanism. The extreme points of regions in the configuration space correspond to the boundary conditions on behavior.

For example, the tap can take three states that are qualitatively significant: closed, partially open and fully open (Figure 4-6). These states provide limit cases for qualitative simulation of the causal network. The figure also shows three qualitative state and the corresponding boundary conditions. These states are directly derivable from the configuration space, while the configuration space can be derived from the device geometry [Bourne et.al. 89].

5. **Structural features.** These are the structural features of an artifact that are visually prominent. They are always present in the input and allow the reasoner to retrieve cases based on surface similarity. There is strong evidence in the cognitive science literature [Holyoak 87, Gentner 85] that, in a large number of experiments, although subjects rated relational similarity as the basis for judging the soundness of a match, the overwhelming majority of retrievals that occurred most readily were based on surface similarity.

The tap has several structural features (e.g. pipe, nozzle, cylinder) and relations (e.g. cylinder is across the pipe, cylinder is between the input and the nozzle). Structural relations that determine the visual form of the artifact are also used as indices (e.g. the relative positions of the nozzle and the cylinder).

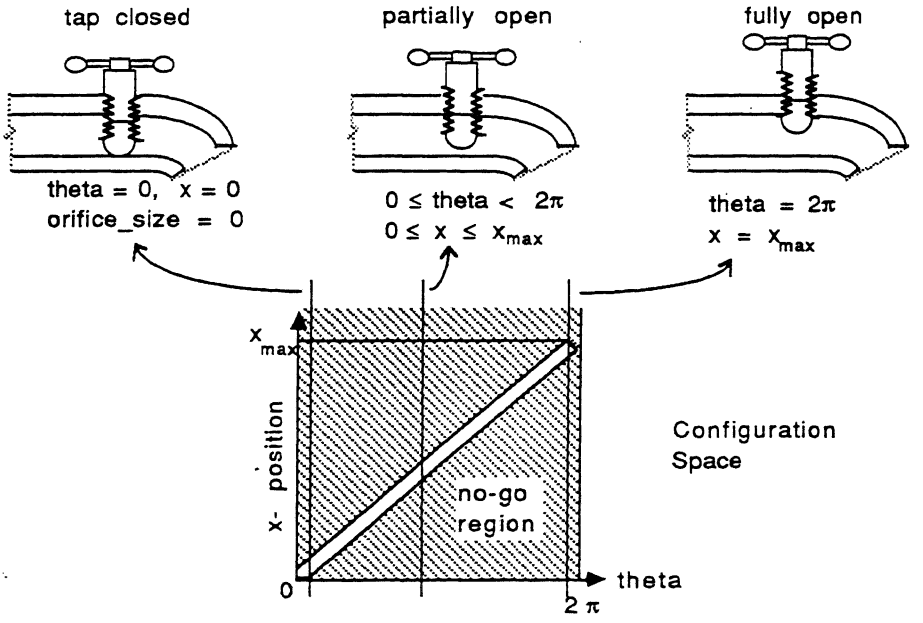

Figure 4-6: The configuration space and qualitative states of a tap

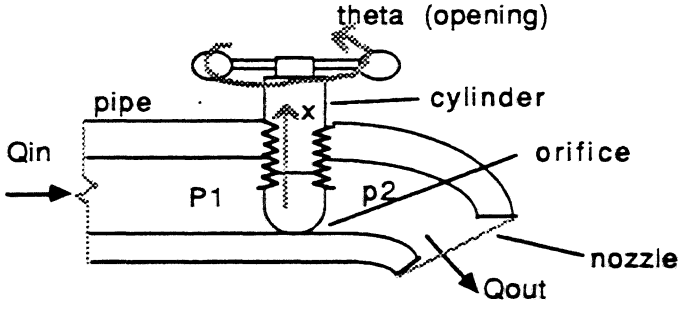

Figure 4-7: A water tap

The above levels or design representation covers a wide spectrum of ways in which a design case may be re-used.

4.5. Indexing and Retrieval

Indexing precedents in memory is one of the toughest issues facing systems that reason by analogy. As an illustration, try answering the following question: "Think of ten things you can do with a spent printer ribbon-cartridge". In answering this question, notice how you can retrieve precedents by using properties of the cartridge as cues into memory. Another question: "Who is the most famous person you ever met?" In answering this question you have to search memory because it is highly unlikely that you have all meeting-precedents indexed in decreasing levels of "famousness". Here is a trace of the process I followed in order to answer the question: "Where could I have met famous people,.. Who are famous people, actors, scientists, politicians... Who are the scientists I have met?... Where could one meet scientists?... Whom did I meet at the last AAAI conference?... Who are the famous AI people? ... and so on." Memory has to be indexed such that precedents can be reached in many ways. In addition, the memory is constantly reorganizing indices and making generalizations [Schank 82].

The work of Kolodner on the CYRUS system provides a good indexing mechanism [Kolodner 81]. CYRUS uses knowledge structures called Episodic Memory Organization Packets (EMOPS). An EMOP can have several episodes under it. The EMOP contains generalized information characterizing its episodes. The individual episodes are discriminated from each other by their differences. When many new episodes are added to an EMOP, then new sub-EMOPS are created. This process eventually forms a tree with the most generalized concept at the root with specific episodes residing at the leaves.

The EDISON system uses this discrimination tree strategy to organize devices [Dyer et.al. 86]. In EDISON, devices are organized under a general index and then discriminated by their differences. Indexing is done on function, topology and context of use of the device. For example, a magnet and a suction cup are both methods of semi-permanent connection and are indexed under one node, they are however differentiated by the principles used; namely, magnetism and vacuum.

4.5.1. The Chicken-n-Egg Problem of Indexing

There are ways of indexing precedents/episodes based on their characteristics. The CYRUS program indexes episodes based on actual enumerable attributes of the episodes. In an analogical reasoning system, we earlier argued, there needs to be a method by which precedents can be retrieved to fulfill purposes that they were not originally intended for. Purpose-directed analogy can help explain a precedent as performing some required function or not. For example, explaining how a stapler can be used as a nutcracker. It is not clear how to index memory to retrieve precedents for given purposes.

To find relevant cases we need to develop a case indexing mechanism which will allow one to retrieve cases which are analogically related to the current problem. The question is: "As one can determine the analogical relevance of a case only after it is retrieved, how does one know which case to retrieve in the first place"? This is the chicken and egg problem of memory indexing. Surprisingly, people are very good at this, a "property of memory that always seemed technically mysterious is its uncanny aptitude for making metaphorical connections, of causing us to recollect things that turn out relevant only after reflection and reformulation (Minsky)".

One cannot go through memory searching for precedents that can be explained as serving some given purpose. While answering questions such as "Think of all the things that can be used as a nutcracker" we use a functional and structural description of a nutcracker to search memory with. A functional description might be based on functions and causal relations among the different parts of the nutcracker. Does this mean that memory should be organized not only by part attributes but also by function and the types of causal relations among the different parts? Maybe so. Here is a question that illustrates the point about having causal relations as indices too: "Name five things that change color when you touch or press them?" This question does not index on attributes but on some relation among attributes. A technique for answering questions that require indices that do not exist in memory is required. We have several choices: we can either re-organize memory based on a question; reformulate the question; index memory with many redundant indices and hope for the best; not retrieve the precedent at all; or just go through all the precedents trying to explain the precedents as serving some purpose.

In the preceding paragraphs we have been emphasizing the idea of being able to recognize a particular precedent in many different ways. This ability requires a level of understanding on the part of the design system that will allow it to recognize (explain to itself) an artifact as capable of fulfilling some given function, even if the artifact's original purpose was very different. Here is an example that will help illustrate this point: take a moment and look around the room you are in right now, ask yourself the following question: "What items in this room could I use as an ice-cream scoop?" The performance of this task is based on your ability to "explain" how the different items you see around could be used as a scoop. It is important that a design system be able to explain precedents as being able to serve a required function. In order to do this the system needs to have sufficient domain knowledge. For example, a mechanical design system has to know about force, motion and other laws of naive physics. The program CYCLOPS uses explanations but does not generate explanations based on a given purpose. Explanations, though complex, are pre-coded and are static.

Then there is generalization. Generalization plays an important role in learning engineering design. For example: almost all engineering courses require students to work through several exercises. This is done with the assumption that students will generalize from the specific exercises and learn the underlying principles. The ability to generalize is linked to a learner's understanding of the situation. A system can be said to understand, if it can explain a new situation in terms of the concepts it already possesses. References to Explanation based learning and generalization are found in [DeJong 81, Mitchell et. al. 86]

A possible strategy for solving this problem is to use extensive cue transformation heuristics to find cases. The hypothesis is that, cue transformation could possibly find related cases which, upon inspection, turn out to be relevant. One of the techniques of interest is question reformulation. We will see later, how question transformation lies at the heart of innovative design.

4.6. Synthesis and Debugging

Synthesis lies at the heart of design, it is the process by which an artifact is refined or generated. The process by which the artifact actually takes form. In order to perform a synthesis, one has to have things to synthesize. The synthesis process involves first finding what to synthesize, then deciding how to synthesize, followed by an evaluation and finally debugging or redesign.

4.6.1. Synthesis in the Physical Domain

In many systems, it is already known what alternatives are available for synthesis e.g. ALL-RISE [Sriram 86] and MEET [Steinberg et.al. 86]. In goal directed systems the programs generate lists of possible candidates for synthesis at each stage of the design process. A design problem is broken into subfunctions and then components which can fulfill the different subfunctions are retrieved. Conventionally, these components could be retrieved from a global database or local list attached directly to a refinement operator. In an analogical reasoning system the synthesis alternatives could be drawn from within and without the current design culture. Where, each subfunction provides us with a purpose or index to search memory with. After precedent case are retrieved, parts of the cases are extrated and synthesized into new designs. This transfer of knowledge from base to target is one of the least investigated problem in design automation and probably the hardest. Work on case based planning has shown how plan snippets can be retrieved and incorporated in new plans. This work on snippet synthesis has been developed in problem domains where the representation of the case and the solution is of the same type. In mechanical design, however, decisions relating to how certain functions are achieved might be taken at a linguistic or qualitative level. Considerable complication arises from the fact that although a design might be verified to be correct at these levels, simulation at the physical level might fail. We have to find ways in which the problem solver can synthesize snippets at one level of abstraction while making sure the parts will work together in physically correct ways [Sycara & Navinchandra 89]. A further complication is that the parts (also called case "snippets") may not be from the same domain and may need substantial modifications becore synthesis. This is the cross-contextual synthesis problem.

4.6.2. The Cross-Contextual Snippet Synthesis Problem

Consider, for example, a biomedical engineer is given the task of designing a non-surgical method for removing urinary calculii (a calcium deposit in the urinary tract) without surgery. The requirement of finding a non-surgical method reminds him of catheters. This idea is from within his domain. Now that he has a way of reaching the calculii, he has to find a way of grabbing it. This reminds him of kitchen tongs and he decides to put small clips at the end of the probe. His next sub-goal is to pull out the calculii without scratching the insides of the urinary tract. This can be done by expanding the tract. The word "expanding" reminds him of a balloon and he decides to introduce a small balloon at the end of the probe. This will aid in pulling out the calculii. This example, taken from a real design problem, shows how designers are able to use known cases from a variety of domains to solve a given problem. The issue is, how does one make a correspondence, find the relevant parts of the case, make an abstraction, and appropriately instantiate it in a different domain.

The cross-contextual synthesis of design snippets could take place in the following ways:

- The physical form of a precedent case (the base) and current problem (the target) match. In this situation one can cut out part of an old design and transfer it to another. For example, a special gear train used in some machine could be re-used in the design of another machine.
- In some situations, even if the physical form is the same, snippets being moved from base to target may need modifications. For example, a surgical drill may be designed based on an industrial drill.
- Snippets may need to be adapted when being moved from base to target. Adaptation may in itself be a design problem involving the retrieval and use of cases.
- Ideas from the base may be extracted in a purpose-directed fashion and re-embodied in the context of the target problem. We have no way of doing this automatically.

We believe we need some way in which a design "IDEA" can be extracted and communicated from one domain to another. In the above example, the clips at the end of the probe probably do not look like tongs at all, but share an underlying idea. The notion of "idea" is related to work done my Mostow on the operationalization of advice. It is also related to the work of Owens on the interpretation of abstract ideas. We will have to extend these notions to handle advice across domains.

4.6.3. Synthesizing Design Plans

When solving a new design problem by analogy to a previous case, we might transfer the solution used in the base to solve the target problem. In some cases instead of transferring the final solution from base to target, it is better to transfer the design plan (process) from base to target. This approach was suggested by Carbonell [Carbonell 83a, Carbonell 83b]. The first method transfers the actual steps performed from base to target. This is called a Transformational analogy. The second, improved method, transfers the reasoning process used in the base to the target problem. This is called Derivational Analogy as it uses the underlying reasoning steps in the base and target. To find a match the system first starts solving the target, after some progress is made, the reasoning steps in the target are matched against those of the base. If the match is found, the rest of the plan from the base is transferred to the target. This method, though a desirable capability for DA systems, needs improvements. It is not clear how far the target should be solved before the analogy can be drawn and the rest of the plan can be transferred from the base to the target. This issue, among others, is discussed in [Kedar-Cabelli 85d, Mostow 86].

The idea of using design plans has been applied to VLSI design in a system called Argo [Huhns 87]. Argo uses a hybrid mechanism borrowing ideas from MACROPS of STRIPS [Fikes & Nilsson 71] and the Explanation-based Generalization method [Mitchell et. al. 86]. The Argo program learns operators from solving design problems. Whenever it solves a design problem, Argo creates a tree of the rules used in the solution. The tree is represented as a rule-dependency graph (RDG). Macro-operators are generated by dropping the most specific rules from the RDG. By doing this many times over Argo generates design plans that are more and more generalized. In order to learn these Marco-operators, Argo regresses through the actual rules of the plan. In so doing, the pre- and post-conditions of the macro-operators end up with variables in their patterns rather than references to specific objects.

While using these operators, the program starts with the most specific macro-op, failing which, it proceeds to use more and more abstract macro-ops. As the system uses generalized operators it will be difficult for it to recover from the wrong application of macro-ops. There is no guarantee that in all domains MACROPS (derived the way Argo does) will work even if all preconditions are tested before application. However, Argo seems to run well in the domain it was built for.

5. Exploration and Discovery

A designer is forced to innovate whenever he is faced with a problem that cannot be solved in some previously known way. He is forced to innovate by exploring within his current design culture or by using ideas drawn from other domains. The process of exploration can be either syntactic or context driven, further, it can be either goal driven or data driven.

Whenever a designer reaches a dead end in a design process he can start exploring new alternatives by making syntactic changes (mutations) to the artifact he is designing. He may make such changes in the hope of finding a novel combination that satisfies the given design goals. Exploration can also be done in a data driven fashion. In this case, the designer has no explicit goals, but is trying to find some regularity, some hidden principle in the alternatives generated by exploration. This process is called Discovery. The discovery process does not rely on explicit goals for testing alternatives but uses heuristics to recognize interesting patterns, conceptual clusters etc.

A designer can also explore new alternatives in ways other than making random syntactic changes. When faced with a design problem he can perform an analyses of the problem and then make modifications to the artifact in direct response to the results of the analysis. In this case, the designer is reasoning about the nature and direction of exploration and hence, is using a context-driven approach. In this section we will examine the above issues in detail, with references to research that address these issues.

5.1. The State Space

All artifactual design systems have a target technology. A target technology comes with a standard representation formalism for the artifact in question. For example, in VLSI design the design representation language consists of objects such as gates, connections, adders etc. which are governed by a grammar that defines how the objects may be put together.

The set of all possible, legal (with respect to the grammar) combinations of the objects is called the state space of the design problem. For all practical purposes, the state space of the design is infinitely large. For example, the state space for the Chess game is a number (10^{120}) larger than the estimated number of atoms in the universe (10^{75})! In order to deal with such large spaces search techniques use the governing design constraints to prune away large parts of the search space. Techniques such as least commitment, constraint propagation and hierarchical design have been developed for keeping the search space manageable [Stefik 80].

If we are given a design problem with an artifact representation language and a set of design constraints, then, the set of all combinations in the state space that satisfy the constraints is called the ***solution space*** of the design problem. The aim of techniques such as search and constraint handling is to find any point in the solution space. Systems that optimize over the state space have the harder task of finding the best point in the solution space.

Two problems that one can face while searching the state space for a solution are: first, the possibility that there is no solution space and that the problem is completely over-constrained; second, it is possible that none of the solutions found are acceptable or interesting. To alleviate this problem, exploration techniques are used to aid in looking beyond the current solution space. In building computer programs that explore design spaces we need:

- to know when, and in which direction to explore,
- to resolve the conflict between trying to keep the search space small (for efficiency) and the need to expand the search space (to explore),
- have methods or operators with which the exploration can be conducted,

- ways for using knowledge to recognize promising alternatives during an exploration phase, and,
- operators and techniques to use knowledge to change the state space itself!

One of the assumptions about exploration is that, the best design is somewhere in the state space, even if it is not in the solution space. Let us call this the ***static state space*** assumption. It also possible to modify the representation language in order to expand the state space itself. These, and other issues are discussed in the following subsections.

5.2. Exploring Design Alternatives

This section presents three exploration methods and compares them. The methods are:

1. Searching for novel combinations.
2. Using Relaxation and Mutation heuristics.
3. Analytical Invention.

A design system can generate alternatives by producing legal combinations of parts in its target technology. To avoid doing a blind search, these systems use design goals to guide the search. The relaxation and mutation approachs are different in that they try to generate alternatives that are closely related to the existing solutions. The idea is to mutate existing solutions or to slightly relax a governing constraint to modify the solution. The third technique, is the most goal-directed. The idea is to use analytical relations and equations about the design to produce new designs. This technique is a lot more reliable, but may not be able to produce totally novel solutions like the other exploration methods.

In addition, while exploring a state space we should not miss promising alternatives. The next section describes techniques used to evaluate alternatives generated by the exploration process. Evaluation is particularly important because exploration techniques tend to produce too many design alternatives. If the evaluation are slow or inadequate one may miss a truly novel design. It is as much a part of innovative design to recognize what is innovative as it is to synthesize the design in the first place.

5.2.1. Searching for Novel Structural Combinations

Consider a system that can search the state space of structural combinations of a given set of objects using rules of combination. If such a system can be given a set of criterion to test combinations with, it can be set off on a search for solutions. Given a complex set of criteria and sufficient computer time such programs could come up with combinations never thought of before. By following a simple generate (complete or incremental) and test paradigm it is possible to come up with novel combinations (as long as the static state space assumption holds).

One of the first programs in this area was DENDRAL [Buchanan & Feigenbaum 78]. The program had the task of determining the physical structure of organic molecules. The input to the program is a mass spectrograph and the chemical composition of the molecule in question. Using rules about how molecules fragment, the program first generates a set of constraints on the possible structure. It then enters a generate and test process where it generates possible structures and tests them. The testing is done by comparing a theoretically derived spectrograph with the actual spectrograph. The test is successful if a good fit is found between the two graphs.

The generate and test paradigm tends to be very slow. This problem is addressed by using heuristics to search the state space.

Another program that explores structural models is a discovery system called DALTON [Langley et.al. 87]. The goal of the program is to devise a model of chemical reactions that specifies the number of

molecules and atoms involved. Given a list of reactions and components of substances, the program uses heuristic operators to search, in a depth-first fashion, through a state space of possible models. The search proceeds by making assumptions about the number of atoms in a molecule and testing these hypotheses against laws of conservation, combining volumes etc. Here is a trace of how DALTON figures out the equation for the formation of water by combining hydrogen and oxygen (H and O denote molecules, h and o denote atoms):

1. Starting with the reaction: (Hydrogen + Oxygen --> Water)

2. Assume one molecule each ((H) (O) --> (W))

3. Guess internal structure: (single particles are denoted by h and o)
   ((h) (o) --> (W))

4. By conversion heuristic:
   ((h) (o) --> (h o)) - this result was actually arrived by the 18th century chemist DALTON too.

5. Using a heuristic about combining volumes, the following is inferred
   ((H) (H) (O) --> (W) (W)) which lead to
   ((h) (h) (o) --> (W) (W)), - the above (#4) reaction is revised using data about combining volumes.

6. DALTON assumes the oxygen is made of two particles as the last equation above does not pass the conservation heuristic:
   ((h) (h) (o o) --> (h o) (h o))

7. The above reaction is found to be wrong by a heuristic that knows how hydrogen reacts in some other reaction. At this point, backup occurs and a different branch is searched.

8. The search proceeds until the following structure is reached
   ((H) (H) (O) --> (W) (W)), which gives
   ((h h) (h h) (o o) --> (h h o) (h h o)), finally
   $2H_2 + O_2 \rightarrow 2H_2O$

Using a very similar technique, a bridge design system, CIDS has been built [Cheyayeb 87]. The program works with objects such as beams, arches and cables. Given a set of functional requirements, the program searches its database for structural elements that can satisfy the given conditions. If no single element is found, combinations of elements is considered. The program follows a beam search technique using number of unsatisfied constraints as a measure of goodness. For example, given the problem of spanning a wide gorge which is too long for a single arch, the program comes up with a solution in the following manner:

The program first puts an arch and then realizes it needs a support for the arch (Figure 5-1). There are several ways of supporting an arch, it finds a cable and suspends the arch's free end from the cable. Next, it needs a place to hang the cable from, again, there are several choices: cable, arch or beam. By going through a process of incremental generation, the program solves the problem as shown in Figure 5-1.

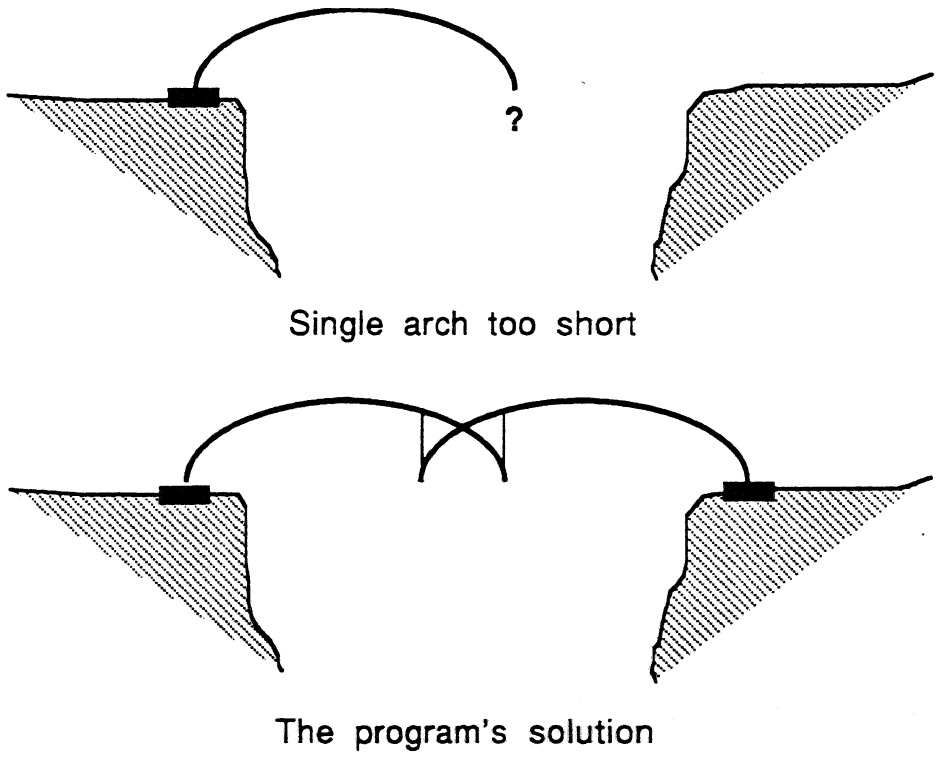

Figure 5-1: CIDS solves a short arch problem

The combinations that the program comes up with are novel.

In conclusion, one can say that innovative design systems can be built using fairly simple heuristics as shown above. The apparent inventiveness of such systems is predicated on the fact that, the search space is so large that a computer can stumble upon combinations never before thought of by humans. This phenomenon holds true, not only for structural combination systems but for ***all*** innovative design systems that search and explore.

5.2.2. Use of Relaxations and Mutations in Exploration

Relaxation operators work on the solution spaces that are bounded by constraints and/or objectives. If a design system has no constraints or objectives, then, any point in the state space is a solution. In effect, the solution space is the state space. If, however, there is a set of criteria (constraints and objectives) bounding the solution space, then one can explore beyond the solution space only by relaxing the bounding criteria.

There are essentially two ways a criterion can be relaxed, it can be either eliminated; or it can be weakened. For example, while preparing a job-shop schedule there are many job due-dates to be adhered to. If a major conflict occurs, the due-date of a job could be relaxed by either pushing the date forward in time or by reducing the penalty for delay. In order to perform such actions on criteria, the computer needs knowledge to determine how to relax a criterion. Currently, syntactic relaxations are most widely used. For example the constraint "Total Delay <= 5 hours" could be changed to: "Total delay <= 6 hours". Such operations can be done using simple arithmetic techniques. In addition to relaxing numerical constraints we need techniques for relaxing non-numeric relationships. For example, an interior designer working with a constraint on compatible colors for walls, drapes and upholstery cannot relax the constraint in a purely syntactic fashion. One way of dealing with such constraints is to attach utilities to the different color combinations with highest utility for the most compatible colors. Having done this, the constraint can be relaxed or intensified by just turning the utility value either down or up. This is how CYCLOPS

relaxes criteria (by using compatibility matrices).

Semantic relaxation is a much harder problem. Here is an illustrative example. Consider the following scenario: there is a city planner trying to layout a new city in a narrow valley. He decides to layout the city as a large circle with radial roads. As he is designing the city he realizes that parts of the city will have very steep slopes. He decides to avoid the steep slopes by using an oblong shaped city. As before, he continues to use radial roads. In this example, the circularity constraint has been relaxed while the essential idea of having a radially laid out city is maintained. A system that can make non-syntactic relaxations need to have a deeper understanding of the purpose of the constraint. It will need the ability to change the superficial form of the constraint while the essence or intent is maintained.

Mutations and Semantic Preservation. The distinction between syntactic and non-syntactic is not very well defined in all domains. A good counterexample to the distinction is the AM system [Lenat 76]. In its search for mathematical concepts, AM essentially was operating as an automatic programming system [Lenat 84]. It searched the state space by making mutations to mathematical concepts. It did this by making syntactic mutations to the LISP code that represented the concepts. The program used a set of rich heuristics for deciding when a concept is interesting or what mutator should be applied to it. For example, the COND clause statement in some piece of LISP code can be generalized by replacing an AND by an OR which, in essence, is equivalent to relaxing the strictness of the function from a conjunction to a disjunction.

AM searches the space looking for example of concepts. It has heuristics about what is interesting (as far as mathematical concepts are concerned). For example the heuristic: "If there are some numbers with unusually small number of divisors, then these numbers are worthwhile investigating." Let's see how AM used this heuristic. Consider the following scenario: at some stage of the search, AM picks up a random example of the concept `numbers-with-3-divisors` and looks for the other properties the example possesses. For instance, the number 49 has the following properties: `perfect-square`, `odd-number` and `numbers-with-3-divisors`. AM then makes a hypothesis that all numbers with 3 divisors are odd and/or perfect squares. AM tries to verify this conjecture on other examples of the base concept and finally discovers that all numbers with 3 divisors are also perfect squares!

AM discovered new concepts by making syntactic changes to LISP code and by using heuristics to recognize if the mutated concept is interesting or not. The program was able to discover concepts such as prime numbers, addition, subtraction, deMorgan's laws etc. AM's success led to the belief, at least initially, that pure syntactic mutation of concepts can yield new meaningful concepts. This rather profound idea, unfortunately, cannot be easily applied to domains other than mathematics. The reason AM worked was because it used LISP as an implementation language. In Lenat's own words: "the density of worthwhile math concepts are represented in LISP that is the crucial factor.... It was only because of the intimate relationship between LISP and Mathematics that the mutation operators turned out to yield... useful new math concepts... (however, when applied to) generating new heuristics, the technique almost always produced useless new heuristic rules" [Lenat 84]. When AM mutated small portions of LISP code which made up mathematical concepts, it produced new concepts. It was because LISP and mathematics are related, that mutations on LISP code maintained the semantic equivalence between the code and the mathematical concept it represented. However, when mutations were made on large sections of LISP code that represented heuristics, new heuristics were rarely generated. It is harder to guarantee semantic preservation over a representation when the representation is not semantically very close to what it represents.

The EURISKO program, (a successor to AM) used a frame like representation for heuristics [Lenat 83]. This approach provided a functional decomposition of the heuristic. Each function could then be independently mutated and the semantic preservation property that AM satisfied fortuitously is recreated by keeping mutations local to functions. This raises some important questions for innovative design. In order to make random mutation meaningful, there should be a strong semantic equivalence between the artifact representation language and the artifact. It is very difficult to achieve this for physical artifacts.

Achieving Semantic Equivalence. The highest level of semantic equivalence is in the domain itself. For example, in the blocks world domain, mutation works best if one moves actual blocks around to change their arrangement. Each mutation gives you a new true state of the world. In building computer models that manipulate artifacts we are dealing with two aspects: representation and reasoning. At one end of the spectrum we could have a representation that is semantically so close to the actual domain that, any mutation to the representation is not meaningless. On the other extreme, one could use a totally ad-hoc representation but make use of the representation in such a way that all actions taken by the control program maintains semantic equivalence. For example, a game program that displays bouncing balls on a screen creates a physical scene using a mathematical representation. The program that sits between the representation and the display device is doing the job of bridging an equivalence between a purely mathematical representation to a simulation we can comprehend. Let us now assume we want to mutate the balls in the bouncing balls domain. For example, a possible mutation might be to cut the balls in half. It is not possible to cut a ball in half by throwing away half the code that generates it but, a new function will have to written to make the change. If fact, this new function will be much harder to write than the first one. In order to build systems that change physical artifacts in meaningful ways we need to develop a balance of the two aspects described above: the representation and the interpretation mechanism that works with the representation.

Modification operators. A system that portrays a mix of using a good artifact representation and a domain specific mutation operators is a design invention system called EDISON [Dyer et.al. 86]. The system is a joining of qualitative reasoning and innovative design strategies. Mechanical devices are defined in terms of part, spatial relations, connectivity, functionality and processes. The representation is rich enough to allow simulation of the working of physical devices. The EDISON team has identified the following strategies for invention: generalization, analogy and mutation, all of which rely on memory organization, indexing and retrieval. The mutation heuristics that EDISON uses are domain specific pieces of code that mutate in semantically correct ways. For example, the figure below shows how EDISON starts with a standard door, mutates it by cutting it in half (a), finds that the second half has no support (b) and adds supporting hinges (c) and finally invents the dual bar-room door.

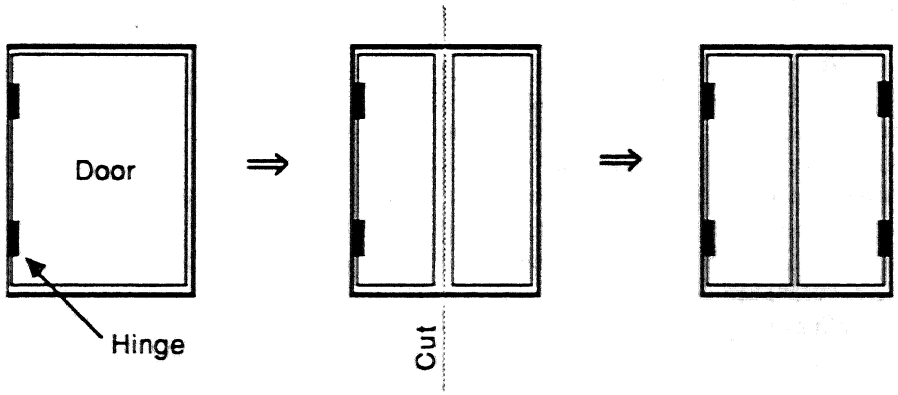

Figure 5-2: EDISON mutates a door

Along a different branch of the search tree, EDISON moved the hinges of the normal door from the side to the top of the door and, in effect, invented a trap-door.

The CYCLOPS program completely circumvents the semantic correctness issue. It's representation is so formal (CLP) that any mutation is guaranteed to produce semantically correct alternatives. For example, it can produce mutations such as: house-on-site, house-on-stilts, stilts-on-house etc. CYCLOPS does not make mutations/relaxations directly to the artifact but to the constraints and objectives that govern the artifact solution space. In effect, criteria mutation is the ***dual*** of artifact mutation. It is for this reason that CYCLOPS can successfully explore. The application of this idea to other artifactual domains is yet to be tried out.

5.3. Invention by Analysis

Bugs give us clues about ways in which a design could be improved. If we explain the bug, then we could use the explanation to appropriately modify the design. An explanation might be in the form of clauses or in the form of analytical equations. The actual method by which a modification is made might be by analogy, or by a heuristic operator. The use of an operator is a special case of analogy. An operator is usually retrieved by matching the bug with the preconditions of the operator. The matching is usually a direct match. If the matching involves abstraction and the retrieved data is in the form of a precedent (rather than an operator) then, an analogy is said to have been drawn.

The decision about what to debug is what gives us an index/pattern with which precedents/operators may be retrieved and applied analogically/directly. The process of deciding what to debug is called bug analysis or failure analysis. Unlike random mutation, analysis involves extensive use of domain knowledge to identify a bug, find its causes and to eliminate the bug. Almost all design systems, routine and non-routine, have some way of recovering from design failures. Earlier in this article we discussed debugging techniques such as advice generation [Mittal 85], here are some other approaches:

Invention by analysis. The PROMPT program is a design system which debugs by performing a qualitative analysis of design problems [Murthy & Addanki 87]. The program performs the analysis on the equations that describe the behavior of the artifact being designed. After the qualitative analysis is completed, modification operators are used to correct the design. One of the design examples PROMPT has been tested on is beam design. Consider the following scenario: the program is given the task of designing a rod which can bear high torsional loads. The program uses the specifications to retrieve a metal rod from the database. The rod is then tested for torsional stress under the specified loading conditions. The program finds the rod is over-stressed. The program then analyses the equation for torsion $Ts = KR^4/L$ and decides to increase the radius of the rod. This modification, however, violates the weight constraint of the rod. The program, through some more analysis realizes that the torsional stress borne by the rod is higher on the periphery and lower towards the center of the rod. PROMPT finally decides to redistribute mass from the center of the rod to the periphery. In this way, PROMPT invents a "hollow torsional member".

PROMPT's ability to analyze problems in such detail is based on a large knowledge base called the "Graph of Models" [Addanki & Davis 85]. This database uses a graph structure to store physics knowledge about principles such as: bending loads, buckling, torsion, vibration etc. Using this knowledge, the program can reason about an artifact from first principles and can derive its behavior from its structure.

PROMPT uses modification operators to eliminate problems. It uses two types of operators: operators that it derives directly from the behavioral equations of the artifact and, operators that are pre-stored in the system. Examples of derived operators are: redistribution of mass by removing mass from regions of low stress and adding mass to high stress regions; changing material distribution and some simple shape changes. Examples of predefined operators are: shape changing operators and topology changing operators. Here are some examples. The figure below shows how a simple shape change in a box beam can change stress concentrations at the edges:

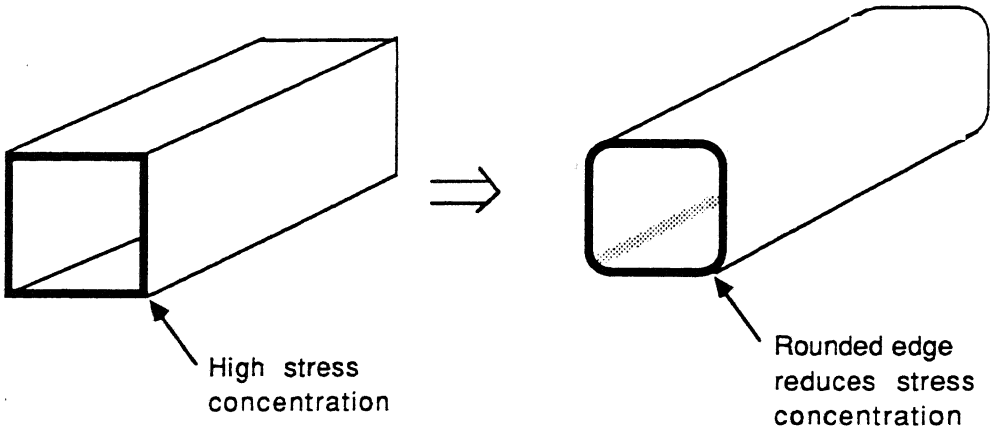

Figure 5-3: Example of a shape change operator

The figure below shows how a strut can be used as a structural brace:

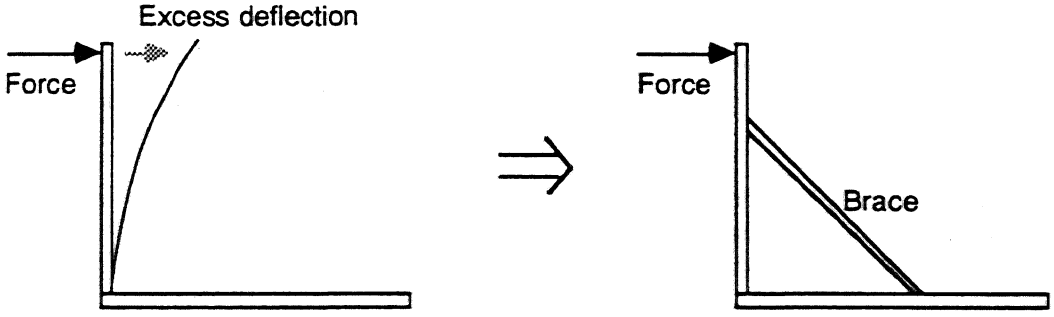

Figure 5-4: Example of a topology change operator

The EDISON program is also capable of reasoning qualitatively about the artifacts it generates through mutations. Using a process reasoning mechanism [Forbus 83], EDISON can simulate the workings of physical artifacts it designs. Device simulation is used by the problem solver for verification and discovery. EDISON can discover constraints by simulating the movement of devices. In this way, process simulation serves as an analysis method for recognizing design problems. For example, if EDISON knows nothing about the way hinges constrain the movement of doors, it can learn such constraints by placing hinges at the top and side of a door (a mutation) and learn that the door will not move when subjected to a simulated push.

Explanation as a form of Analysis. The CYCLOPS system finds innovative solutions by generating explanations of bugs and using the explanations to find solution strategies from a database of previous design cases [Navinchandra 89]. The system performs two basic functions: dependency-tracking and subgoal-matching.

Dependency-tracking involves of the following steps: first, the problem is identified; second, the problem statement is posted as a demand on the database of precedents. This means that the database manager has to find precedents that match the demand (pattern). If no suitable precedent is found, the program retrieves the causes of the problem. These causes are then posted as demands. This process of posting demands, retrieving causes of problems and posting the causes as new demands proceeds recursively till suitable precedents are found.

Subgoal-matching. A precedent is said to be suitable with respect to a posted demand (pattern) if it contains a design strategy which attains a goal (pattern) that matches the demand. If such a match is found, the design strategy in the precedent is transferred to the current design problem. If, on the other hand, a match between the demand and the goal of the precedent is not found, the program retrieves an explanation (pre-coded) of how the design strategy in the precedent attains the precedent's goal. An explanation is usually a trace of how the different parts of the overall strategy address the subgoals of the overall goal. After retrieving an explanation, the program matches the demand (pattern) against the subgoals of the precedent strategy in order to find a match. If no match is found, failure is announced. The advantage of using this technique is that it takes advantage of the fact that: even if the overall goal or the surface features of a precedent is radically different from the current problem it is possible that they might have common subgoals. It is for this reason that the program appears to reason analogically.

The process terminates when all the causes, goals and subgoals have been satisfied by precedents. A trace of the solution takes the form of an AND-OR tree with precedents or parts of precedents attached to the branches.

Other important work in innovation by analysis has been reported by Cagan [Cagan 88] and Mittal [Wang, Mittal & Leifer 89].

Conclusion. In this subsection we reviewed three strategies for debugging designs. All three methods are different from mutation processes in that, they perform bug analysis using domain knowledge and/or dependency links. In general, the process of analysis is that of finding the causes of a problem. Whenever a direct solution to a bug is not found, the causes of the problem are determined. Causes are found through dependency links or by qualitative reasoning. The ability to analyze and to retrieve or generate new modifications is an important ingredient of design systems.

CYCLOPS and EDISON are two systems that use both relaxation/mutation techniques coupled with knowledge based reasoning (analogy and modifications). At this stage it is still hard to tell what is the right way to mix mutation processes and knowledge based processes in design automation.

5.4. Criteria Emergence and Serendipity Recognition

In order to innovate, it is important not only to be able to explore new alternatives but to also be able to recognize innovative alternatives when they are generated.

The BACON program, for example, discovered new laws by detecting numerical trends. Using a depth first control strategy, the program generated data that was checked for patterns of relationships among variables. The AM program, like BACON, had no explicit goal to discover. It used heuristics of interestingness to move from one interesting concept to another.

This section discusses how a reminding can be used as a way of recognizing and measuring interestingness. Alternatives generated through criteria relaxation or mutation can sometimes cause the designer to be reminded of some previous episode. It is through such a reminding that the designer may: recognize an opportunity, serendipitously solve some previous goal, or emerge new design criteria.

Here is a partial list of the types of remindings a new alternative can generate:

- ***Emergent Criteria.*** Consider the following scenario: a landscape designer is working on the layout of suburban neighborhoods. Using constraints about acceptable slopes, soil conditions,

aspect etc. he delineates suitable regions of the landscape. He then starts locating various housing units on the suitable sites. After completing a preliminary layout he sits back to inspect his work. He suddenly realizes that a swampy area on the landscape provides by-far the best possible view of a picturesque mountain range. He had overlooked this site because his original set of goals and objectives did not take the mountain range into account. Further, there was a constraint that all swamps are unacceptable. This scenario, shows how a designer can emerge new criteria as he goes through the design process. (CYCLOPS emulates this behavior).

- ***I will know what I want, when I see it!*** This is equivalent to searching for alternatives hoping a positive reminding occurs. For example, you walk into a departmental store with the idea of buying a birthday gift for a friend. You are not sure what you want, but you walk around the store hoping to find something suitable.

- ***A new episode solves a previous problem.*** It's a lazy sunday morning, you pick up the newspaper and start scanning it not looking for any article in particular. You suddenly find something interesting and read the full column. The reason this happens is that, the article answers some dormant question or problem or curiosity. While working on one design problem, it is possible that one might find a solution to some other problem. In order to support such behavior, we need a mechanism that matches the current input episode to some precedent in memory, where, the precedent is some previously unanswered question.

- ***A new state leads to the serendipitous solution of a given problem.*** The DAYDREAMER program is one which displays creative behavior through the use of a mutation mechanism and a serendipity recognition mechanism. The program takes descriptions of events in the world as input and produces a sequence of events it will perform in an imaginary world. The output represents the program's "stream of thought", a daydream.

5.4.1. Discovery Systems

Discovery systems are programs that search the state space not with the purpose of finding a solution but to discover some regularity, a pattern or a concordance in the provided data. Discovery programs generally use rather simple statistical or numerical analysis techniques to discover new theories, relationships etc. These systems rely on the computer's ability to search tirelessly and are not based on any cognitive model of human thought processes.

Even though discovery systems have a very different purpose than design innovation systems, there are several techniques that we can borrow from the discovery literature. Techniques include the use of exploration heuristics, operators for exploration, and testing techniques that recognize a new discovery. The last issue is of great importance. How does a discovery system know it has made a discovery? There are essentially three methods:

1. The system can have a global goal to find some pattern. For example, to find a relation among attributes in an amorphous dataset.

2. The new alternatives may cause reminding of an episode from a database of past experiences. If this database has knowledge from a wide range of sources, then it is possible that a reminding will help recognize an interesting solution. A good example is the case of the cardiologist who, in the process of testing new medicines for hypertension found that one of the drugs, minoxidil, had an annoying side effect. It caused patients to grow excessive hair. If the doctor was only interested in cardiology he would not have have realized that he had accidentally discovered a wonder drug for baldness.

3. The new alternative is better than previous alternatives. CYCLOPS uses pareto-optimality to evaluate alternatives in relation to previous designs.

Discovery by Bacon. Bacon is a quantitative discovery system that finds mathematical relations between given data sets. For example, Bacon can discover Kepler's third law by working with a given dataset of distances (d) of the different plants to the sun with their corresponding time periods (p). The goal of the program is to find a relation f such that f(p,d) = constant. Let us assume the dataset has both p and d increasing monotonically (shown below, adapted from [Walker 87]):

Planet	p	d
Mercury	1	1
Venus	8	4
Earth	27	9

As the values increase monotonically, a certain heuristic suggests the term (d/p) be calculated. On seeing that d and (d/p) vary inversely, another heuristic suggests multiplying the two to give (d^2/p), by following such heuristics the program finally finds (d^3/p^2) to be constant. Which is Kepler's third law! Bacon's approach is interesting but it is limited in its inability to handle noisy data. Further, it cannot handle irrelevant data, and definitely not irrelevant noisy data.

Patterns in medical data. Another data driven discovery system is a medical data analyzer called RX ([Blum 82]). RX works from a database of patient information and tries to find medical relations between the attributes. It uses statistical techniques and medical domain knowledge in the process. The program uses cross-correlation to hypothesize relations. It then checks the hypothesis for statistical significance. RX, unlike BACON, has some medical domain knowledge that it uses to build new relations. Medical relations discovered by RX are added into the program's knowledge base and can be used in future iterations.

Some discovery systems use knowledge more extensively, they are not purely data-driven. For example AM/EURISKO [Lenat 76] and DAYDREAMER [Muller 87] are examples of systems that innovate by both syntactic searching and intelligent use of knowledge and heuristics. These systems are discussed in the following subsections.

CYCLOPS uses a database of precedents to recognize designs as being interesting. After a design is found to be pareto optimal it is passed onto the precedent's database for further scrutiny. If a design has certain characteristics that match those of a precedent, then the precedent is retrieved. If the precedent has favorable or unfavorable effects associated with it, these effects are passed on to the design alternative in the form of a new criterion. This is how CYCLOPS emerges new criteria during it's exploration of the design state space.

5.4.2. Serendipty Recognition

Serendipity recognition in DAYDREAMER. The serendipity mechanism enables the accidental achievement of goals or subgoals. The mechanism responds to states, actions, physical objects and retrieved episodes. Consider the following scenario of DAYDREAMER operating in the domain of social relations: DAYDREAMER is female and is a great fan of the movie star Harrison Ford. One of the goals of DAYDREAMER is to date Mr Ford. The program is then given an input state: "Harrison Ford is at the Nuart Theater". In response to this input, the serendipity mechanism recognizes that the episode is relevant to its goal to date Mr Ford. It then starts working through the subgoals of the top level goal. A subgoal of the top level goal is meeting Mr Ford; a subgoal of meeting Mr. Ford is having a conversation with him; a subgoal of having a conversation is being in proximity to Mr. Ford so that it is possible to talk; a subgoal of being in proximity to a person is being at the same location as the person; a subgoal of being at the same location is knowing where the person currently is; and as Mr. Ford is at the Nuart theater, DAYDREAMER decides to go there!

DAYDREAMER plans its dreams with the use of rules that are derived from other episodes. The program uses a pre-compiled network of rules. The network is essentially a graph where any two rules are connected if the goal of one rule matches the subgoal of another. The program starts with the new state pattern and the goal pattern. It then identifies two rules, one which produces the goal state and another which corresponds to the input state. It then, finds a path in the rule graph between the two rules. This kind of search is called ***intersection search*** [Quillian 68] and is, in essence, a proof mechanism. In order to complete the serendipity, DAYDREAMER verifies the path by checking that each subgoal unifies with the antecedent goal of the rule that it is connected to. All unbound variables (if any) are collected and their values can be retrieved from the associated episodes. In this way, new objects or situations from other episodes get used in the dreams the program generates.

The serendipity mechanism in DAYDREAMER provides us with a method for finding a plan for solving a design problem by using a retrieved precedent/episode. The process of finding the connection will lead to the use of several other episodes. One problem with this approach is the need for a good verification technique. Sometimes, the connection between a problem and its solution might follow a path that is bizarre and impractical. For example, a design system that is trying to place a house on a swamp might decide to suspend the house with balloons. An innovative idea, but impractical. One way to check for impracticality is to search the episodic database to find any connection between the generated plan and an unfavorable condition. For example, after generating the balloon solution, the program starts searching the rule graph till it finds a path to a problem. For example, it might find that balloons are unstable in wind, and that the site is windy and that unstable houses are not acceptable. In this way we make the "closed-world" assumption that, if none of the rules in the graph can find a problem with the generated design, then it is acceptable. CYCLOPS uses this "closed-world" assumption to verify analogies that it draws. CYCLOPS also reasons by matching subgoals to base and target. It's subgoal matching mechanism is given below:

Subgoal-matching. A precedent is said to be suitable with respect to a posted demand (pattern) if it contains a design strategy which attains a goal (pattern) that matches the demand. If such a match is found, the design strategy in the precedent is transferred to the current design problem. If, on the other hand, a match between the demand and the goal of the precedent is not found, the program retrieves an explanation (pre-coded) of how the design strategy in the precedent attains the precedent's goal. An explanation is usually a trace of how the different parts of the overall strategy address the subgoals of the overall goal. After retrieving an explanation, the program matches the demand (pattern) against the subgoals of the precedent strategy in order to find a match. If no match is found, failure is announced. The advantage of using this technique is that it takes advantage of the fact that: even if the overall goal or the surface features of a precedent is radically different from the current problem it is possible that they might have common subgoals. It is for this reason that the program appears to reason analogically.

5.4.3. Learning from Experience

This section has been about emergent criteria and serendipity recognition. The approaches discussed are based on a cognitive model of behavior . The idea of storing experiences in the form of episodes, rather than abstract operators, is closer to a cognitive model than an operator based algorithm.

We talked about design systems discovering new ideas/artifacts and finding innovative solutions serendipitously. A question that one may ask about design systems that have this kind of behavior is: "If the program already has the episode in memory, then how is it's use of the episode is an accident. If the program has access to it, then why can't it just use it?" There are two responses to this question:

Database size. Oftentimes the episodic memory is so large that testing all episodes against the current problem is impossible. The main task of controlling programs is to generate the right index to retrieve precedents that may be useful. Sometimes a problem just cannot be solved by episodes that first come to mind. One might solve such problems after thinking about it for some time, or just taking a walk in the park. "Consider an inventor who is blocked in the task of creating a new kind of door, one that allows people through either side simultaneously. The inventor decides to 'quit thinking' about the problem and take a walk. While walking along a river, the inventor sees a water wheel. Suddenly a solution comes to mind: the revolving door..... The water could not be accessed directly..." [Dyer et.al. 86]. If the designer had tried to scan his memory for all the episodes he probably would have never found the right episode.

Learning to use new criteria. Before starting a design process a designer might list down a set of criteria. These criteria represent what the designer thinks is important, prior to the design process. During the process of design new criteria might emerge through a reminding process. One might recognize problems in the design which had gone un-noticed earlier. After a set of criteria are emerged and used, they become part of the designer's experience. The next time the designer comes around to a similar problem his list of prior criteria will be longer and he will appear to have more expertise. The prior criteria for a design problem is dictated by the design culture one is operating in. As one gains experience in designing and one solves problems by analogy to episodes outside the current design culture, the new episodes become part of the design culture. For example, the design of a structural member by analogy to the structure of honeycombs was a very innovative idea when it was first proposed. Today however, the honeycomb structure is a part of the mechanical engineer's design culture and is mentioned in introductory engineering design text books. Let us try an experiment to show how criteria emergence is helpful. Imagine yourself to be a designer who has the task of designing a hot/cold water tap. Think up some criteria that you might impose on the design. Write them down on a piece of paper before you continue. ....Pause..... I am sure you came up with a few good criteria, here is an excerpt from the list of an expert designer:

- Maximum throughput 20 l/min at 20 bar pressure
- to fit household basins, convertible for wall fitting
- light operation (for children)
- self flushing (no deposits)
- trade mark prominently displayed
- handle/knob not heat over 35C, even under prolonged use

- no sharp edges
- easy maintenance, can use standard spare parts
- ... and so on ....

The expert's list is longer and has some concerns that you probably missed. Interestingly however, many of the expert's criteria seem to be fairly obvious: surely you want the tap to be operable by children; surely no sharp edges; surely it should be maintainable.... These requirements are **not** alien to use but we did not invoke them because they just were not indexed right. However, they are now!

6. Creativity

Creativity can be defined as the production of something - a theory, an artifact, or a painting - which in new and is useful in some way. In the previous sections of this report we examined a definition of innovation that was based on the notion of a design culture. Creativity can be viewed as including all innovation is and something more. A design seems creative if it is radically different from past designs, is highly imaginative, or artistic. Philosophers and psychologists have studied creativity for many years. Much of this work has been aimed at describing the symptoms or end products of a hitherto unseen process which has come to be called ***illumination***. Illumination is the sudden idea formation that appears to take place during a creative spell. Namely, the "eureka" phenomenon.

6.1. Background

The notion of illumination was introduced by Wallas [Wallas 26]. His model of creativity had the following steps:

1. preparation (preliminary work)

2. incubation (allowing some time for the ideas to sink in)

3. illumination

4. verification (testing)

This basic model was expanded and modified over the years [Rossman 31, Osborn 53].

During the 1950's and 1960's a lot of interest was generated in using models of creativity to develop methods for improving creativity of people in the workplace. Two popular techniques Brainstorming and Synectics were developed.

Osborn [Osborn 53] extended Wallas' model of creativity to a seven step process:

1. Orientation: Pointing up the problem

2. Preparation: Gathering pertinent data

3. Analysis: Breaking down the relevant material

4. Ideation: Piling up alternatives by way of ideas

5. Incubation: Letting up, to invite illumination

6. Synthesis: Putting the pieces together

7. Evaluation: Judging the resulting ideas

Emerging from such models of creativity, are techniques to aid in human creative processes. Much of the work in AI approaches to innovation and creativity are based, in principle, on such techniques.

Here are some of the major characteristics of such techniques:

- Use of Analogy, the bringing together of seemingly disparate ideas.

- Asking the right questions and the use of techniques for transforming questions in interesting ways.

- Generating lots and lots of alternatives and throwing away the bad ones.

6.2. Analogy

The creativity literature places a lot of importance on the role of analogy and metaphor in problem solving. For example, Synectics [Gordon 61] uses analogies to solve problems. The Synectics process involves: making the strange familiar and making the familiar strange. Simply put, it involves making connections between the target problem and base concepts from conceptually distant parts of memory. To make the familiar strange is to distort, invert, or mutate the everyday ways of looking and responding to problems in an attempt to cause an useful reminding. Gordon has identified four mechanisms for making the familiar strange, each metaphorical in character:

- **Personal Analogy.** Personal identification with the elements of a problem releases the individual from viewing the problem in terms of its previously analyzed elements. For example, a chemist might imagine himself to be a molecule, permitting himself to be pushed and pulled by other molecules.

- **Direct Analogy.** This involves the direct comparison of parallel facts, knowledge, or technology. For example, Sir March Brunel solved the problem of underground construction by watching a shipworm tunneling into timber. The worm constructed a tube for itself as it moved forward, this led to the classical notion of caissons. To automate such an analogy, one needs to have a good way of representing processes qualitatively.

- **Symbolic Analogy.** Symbolic Analogy uses objective and impersonal images to describe a problem. It is not clear yet how to automate the use of imagery and visualization in problem solving.

- **Fantasy Analogy.** This is based on expressing one's wish about what one would like to have. A part of the Synectics Approach is to ask the question, "How do we in our wildest fantasies desire the artifact to do?"

Techniques such as Synectics are aimed at helping people be creative by inducing them to take different views of a problem and by drawing analogies. In order to make interesting analogies one needs to retrieve the right precedents. For which, one needs the right index into memory, such indices come from asking the right questions.

6.3. Asking the Right Questions

Questions are the cues into memory. They provide the index/pattern to search memory with. It is through the process of posing the right questions and redefining them that one can retrieve useful precedents.

Several techniques for posing questions have been developed. Here are a few:

Wishful Thinking. [Rickards 74] To fancifully think about some goal. The questions may be of the following forms:

"What I would really like to be able to do is ....."
"If I could break all the constraints ....."

Boundary Examinations. [Rickards 74] Starting with a statement of the problem, one picks up random phrases in the statement and asks why? For example, the statement:

> How to develop the motorway network to allow for gradual replacement of rail by road transport as a consequence of relative lack of flexibility of the former?

is converted into the following questions (with respect to the underlined phrases):

- develop: Why develop this at all? Are we solving the right problem?
- motorway replacement: Why motorways only? Why a Network?
- gradual replacement: Why only gradual? Why replace? Why not append?
- lack of flexibility: Why is rail not flexible? Can it be made flexible?

Questions such as these give us many new ideas as they produce indices to precedents not obvious with respect to the original statement of the problem.

Creativity by brainstorming. [Osborn 53] Osborn suggests that brainstorming is improved by asking questions about the problem. The purpose of questions is to spur ideation. Here is a generic list of questions one can ask about almost any problem:

- Can it be put to **other uses**? Are there new ways to use?
- Can I **adapt**? What else is like this?
- **Modify**? New twist?
- **Magnify**? **Minify**? Longer? Larger? Condensed?
- **Substitute**? Other ingredients? Other power source?
- **Redefine**? What are the causes and effects? Can rearrange?
- Can I **reverse**? Turn upside down?
- **Combine**? Blend? Alloy?

Divergent Thinking. [Guilford 59] Divergent thinking involves deliberate attempts to not follow the beaten path. It requires the ability to rapidly produce ideas. This is related to another strain of thinking called ***Ideonomy*** (Wall St. Journal, Jun1, 1987). Ideonomy is the science of laws of ideas and of the application of such laws to the generation of all possible ideas in connection with any subject, idea or thing. It is by mixing lists of natural phenomena and fallacies, for instance, that many questions can be generated. The inventor of Ideonomy, Mr. Gunkel, has developed a computer program that helps spew out combinations of ideas.

The underlying principle of all these techniques is that good ideas can be generated by having lots of ideas and throwing away the bad ones. Interestingly, many of the techniques use syntactic methods for generating ideas. For example, boundary examination and ideonomy use purely syntactic reformulation of the problem or the questions related to the problem.

This brings us back to our earlier discussion about Exploration and Discovery in design. Earlier in this report we discussed the role of artifact mutation as a means of discovery. Here we take this notion further by suggesting idea mutation and question mutation as methods for exploring ideas. In terms of design, idea mutation is equivalent to making mutations to the generic parts that are synthesized into complete designs, and question mutation is equivalent to mutating the specifications and design problem statement.

There are few systems that implement some techniques of question formulation and reformulation discussed above. Such systems, though successful, are just scratching the surface of the problem of

understanding human creativity.

Question Transformation in CYRUS. The program CYRUS [Kolodner 80] used interesting techniques of question reformulation to search memory with. It is a question answering program that draws on an episodic knowledge base.

The CYRUS program's memory consisted of all news stories about one person, Cyrus Vance. It could answer questions about the news stories. For example, the question (adapted from [Schank 86]):

Has your (Cyrus Vance) wife ever met Mrs Begin?

As CYRUS does not have information in memory to answer the above question, it has to transform the question using some domain knowledge.

Original Question (Q1): Has your wife ever met Mrs. Begin?

(Q2): Where would they have met?

(Q3): Under what circumstances do diplomat's wives meet?

(Q4): Under what circumstances do diplomats meet?

(A4): On state visits to each other's countries and at international conferences.

(A3): When they accompany their husbands on these visits.

(Q3a): When did Vance go to Israel?

(Q3b): When did Begin to to the U.S.?

(A3a/A3b): various dates can be retrieved from memory.

(Q3c): Did their wives accompany them on any of these trips?

(A3c): A few trips where this happened is found.

(Q2a): During what part of a trip would the wives have met?

(A2a): During a state dinner.

(A1): Probably on May 24, 1977, at a state dinner in Jerusalem. Both wives were present. They probably met.

The question reformulation process converts an un-answerable question into a series of questions that are more relevant to the data in memory.

CYRUS's memory, as we discussed earlier in this report, is in the form of a network of episodes discriminated by their differences. The difference is with respect to the direct attributes of the episodes. Attributes such as topic, participants, location etc. In the future we will see indexing schemes based on relationships among attributes. Such organizations will be able to answer indirect questions such as: "What are all the things in the room that can be used as an ice-cream-scoop?"

Demand Posting in CYCLOPS. CYCLOPS uses demand posting as a means of questioning its database of precedents. The program starts by posting the top-level goal, it then keeps reformulating the question by drawing upon the causes of the problem. For example, in its attempt to solve the problem of erosion caused by terracing a steep housing site, it uses the following questions:

(Q1): Can I think of some way of reducing the erosion?

(Q1a): What are the causes of erosion?

(A1a): Terracing a steep slope causes erosion.

(Q2): Can I think of some way of not having to terrace the ground?

(Q2a): Why is the ground terraced?

(A2a): Because a house cannot be put on steep ground.

(Q3): Can I think of some way of not having the house on the ground?

(A3): Try putting the house on stilts.

CYCLOPS uses a tree-like structure to keep track of the questions and precedents it uses along the way.

Question tweaking in SWALE. SWALE is a understanding system that creatively retrieves and uses precedents [Schank 86]. Given a story, the program starts by checking if the story fits memory, if not, it detects an anomaly, and then searches for and Explanation Pattern[2] (XP) that can explain the anomaly. Finally it attempts to apply the retrieved XPs. If the XP is not applicable, SWALE tweaks the XP to make it applicable. Finally, if an explanation is generated, it is integrated into memory and generalized.

Consider the following scenario:

Input Episode: *Swale[3], a successful 3-year old race horse, was found dead in his stall a week after winning the Belmont Stakes race.*

SWALE starts by trying to detect anomalies. It retrieves a RACE-HORSE script [Schank & Abelson 77] from memory which says that horses usually die many years after they retire from racing. SWALE finds a temporal anomaly. When the program detects and anomaly, it starts searching for an explanation pattern (XP) which can explain the anomaly. This is similar to how CYCLOPS uses precedents to recognize design problems.

In order to be creative, SWALE searches both **routinely** and **unusually**. An unusual search is based on using unusual features of the anomaly as indices into memory. For example, the fact that Swale the racehorse is successful and young is used to find XPs with the index: "death of the successful and young." The following XP is retrieved:

The **Jim Fixx** XP:

1. Joggers jog a lot.
2. Jogging results in physical exhaustion because jogging is a kind of exertion.
3. Exertion results in exhaustion.
4. Physical exhaustion coupled with a heart defect can cause a heart attack.
5. Heart attack can cause death.

The above statements make up the belief of the XP. These statements, like those in CYCLOPS's precedents, represent causation. SWALE however, uses many types of causation other than just physical-causation. For example: social-causation, emotional-causation and economic-causation.

After an XP is retrieved, certain applicability checks are made. These checks are in the XP. For

---

[2]A precedent

[3]The program is called SWALE and the horse it reasons about is named Swale

example, in trying to apply the Jim Fixx XP (above), the program asks itself the following question: "Could Swale be a Jogger?" In order to answer this question the program searches its database of XPs and scripts related to Swale, race-horses in general, and generalizations of the class: race-horses. If the program finds a matching XP, the XP is used directly. If, on the other hand, an XP is rejected due to a near-miss, the XP is then considered for tweaking. Several tweaking strategies are available. SWALE has heuristics for tweaking. FOr example, when trying to determine if Swale could be a jogger, the program finds a script which attributes jogging only to humans. The theme "jogging" is not appropriate for hoses. In order to tweak the Jim Fixx XP, SWALE uses the heuristic: "Substitute another theme which is more appropriate for the actor." Given the following faulty theme:

```
The faulty theme:[ ACTOR:      SWALE
                   SCRIPT:     JOGGING
                   ROLE:       ACTOR]
```

the program retrieves the belief labeled JOGGING, which is as follows:

```
JOGGING: [      ACTION: DESCRIPTION
                ACTOR:  SWALE
                ACTION: MOVE-BODY-PART
                PART:   FEET
                SPEED:  FAST]
```

Next, the program retrieves several themes Swale participates in:

THEME1: Swale often has ACTOR role in the HORSE-RACE theme.

THEME2: Swale often has ACTOR role in the EAT-OATS theme.

THEME3: Swale often has ACTOR role in the SLEEP theme.

etc..

The program then tries to substitute themes and asks questions of the new themes. For example, it finds a match between the RUN belief (above) and the racing belief in the HORSE-RACE theme. This is because it finds a match between the JOGGING script and the RUN script. Finally, the program conjectures that Swale probably died of exhaustion as described in the retrieved Jim Fixx XP. In order to complete the train of reasoning, SWALE goes on to asking the next question: "Could Swale have had a heart defect?" This question, in turn, is answered by finding XPs as described above.

It is through the process of asking direct and indirect questions, followed by tweaking, that SWALE can creatively explain situations. SWALE is the only system that explicitly uses heuristics to adapt precedents to problems at hand.

Schank suggests several heuristics for question transformation and XP tweaking. Here are some examples (adapted from [Schank 86]):

Question Transformation

Transform the original question about why X occurred into: How can I correctly predict the outcome of situations such as X, next time?

To find the answer to a question, transform it by relaxing the set membership constraints on the objects in the question.

To determine if a goal is anomalous, change the initial question into one whose intent is to find out if the goal is governed by a belief that the actor might have.

XP Tweaking

If a condition is clearly false, ignore it temporarily, make it true later if the rest works out.

If one is reminded, while tweaking, of a related fact, suspend tweaking and apply the fact to the original question.

If a rule applies in a given situation, try reversing its actors and objects and see what remindings occur.

6.4. Conclusions on Creativity

What do we have to learn form the creativity literature? Here are a few observations:

- "Creativity is not such a mysterious process. It depends upon having a stock set of explanations and some heuristics for finding them at the right time, and for tweaking them after they have been found... Search and adaptation of patterns are two of the biggest problems facing AI" [Schank 86].
- Asking questions as indices into memory is crucial.
- The use of syntactic methods to transform questions can lead to remindings of episodes that solve problems or explain anomalies in creative ways.
- The indexing mechanism used determines how easy or how difficult it is to retrieve a precedent at the right time.
- Indexing depends on the representation used for precedents. We have yet to come up with a good representation scheme that will allow easy cross-contextual remindings.
- Is it possible to come up with heuristics which help adapt precedents? How can one learn such heuristics?

References

[Addanki & Davis 85]
Addanki, S. & Davis, Ernst S.
A Representation for Complex Domains.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. 1985.

[Blum 82] Blum, R.L.
Discovery and Representation of Causal Relationships from a Large Time-Oriented Clinical Database: The RX Project.
In Lindberg, D.A.B. (editor), *Medical Informatics, Volume 19*. Springer-Verlag, New York, 1982.

[Bourne et.al. 89] Bourne, D., D. Navinchandra, R. Ramaswamy.
Relating Tolerances and Kinematic Behavior.
In J. Gero (editor), *AI in Design*. Computational Mechanics, U.K., 1989.

[Buchanan & Feigenbaum 78]
Buchanan, B.G., E.A. Fiegenbaum.
Dendral and Meta-Dendral: Their Applications Dimension.
Artificial Intelligence 11(1):5-24, 1978.

[Cagan 88] Cagan, J. and Agogino, A. M.
1stPRINCE: Innovative Design from First Principles.
In *7th National Conference on Artificial Intelligence*. AAAI-88, Minneapolis, MN, August 21-26, 1988.

[Carbonell 83a] Carbonell, J. G.
Learning by Analogy: Formulating and Generalizing Plans from Past Experience.
In Michalski, R. S., J. G. Carbonell, T. M. Mitchell (editor), *Machine Learning: An Artificial Intelligence Approach*. Tioga Press, Palo Alto, CA, 1983.

[Carbonell 83b] Carbonell, J. G.
Derivational Analogy and its role in Problem Solving.
In *Proceedings of AAAI-83*, pages 64-69. 1983.

[Carbonell 86] Carbonell, J. G.
Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition.
In Michalski, R. S., J. G. Carbonell, T. M. Mitchell (editor), *Machine Learning: An Artificial Intelligence Approach Vol 2*. Morgan Kaufman, 1986.

[Cheyayeb 87] Cheyayeb, F.
A Framework for Engineering Knowledge Representation and Problem Solving.
PhD thesis, Dept. of Civil Engineering, M.I.T., Cambridge, MA, May, 1987.

[DeJong 81] DeJong, G.
Generalizations based on explanations.
In *Proceedings of the Seventh IJCAI*, pages 67-69. Morgan Kaufmann, 1981.

[Dixon 65] Dixon, J.R.
Design Engineering: Inventiveness, Analysis and Decision Making.
McGraw Hill, 1965.

[Dyer et.al. 86] Dyer M.G., M. Flowers, J. Hodges.
EDISON: An Engineering Design Invention System Operating Naively.
In *Proceedings of the First International Conference on Applicatoins of AI to Engineering*. April, 1986.

[Evans 68] Evans, T. G.
A Program for the Solution of a Class of Geometric Analogy Intelligence Test Questions.
In Minsky, M. (editor), *Semantic Information Processing*. MIT Press, Cambridge, 1968.

[Faltings 89] Faltings, B.
Qualitative Kinematics in Mechanisms.
Artificial Intelligence , Expected in 1989.

[Fikes & Nilsson 71]
Fikes R.E., N.J. Nilsson.
STRIPS: A new approach to the application of theorem proving to problem solving.
Artificial Intelligence 2:189-208, 1971.

[Forbus 83] Forbus, K.
Qualitative Process Theory.
Artificial Intelligence 24:85-168, 1983.

[Forbus 84] Forbus, K.
Qualitative Process Theory.
Artificial Intelligence 24, 1984.

[Forbus & Gentner 86]
Forbus, K.D., D. Gentner.
Learning Physical Domains: Toward a Theoretical Framework.
In Michalski, R.S., J.G. Carbonell, T.M. Mitchell (editor), *Machine Learning: An Artificial Intelligence Approach, Vol 2*. Morgan Kaufmann, 1986.

[Gentner 83] Gentner, D.
Structure Mapping: A Theoretical Framework for Analogy.
Cognitive Science 7, 1983.

[Gentner 85] Gentner, D. and Landers, R.
Analogical reminding: A good match is hard to find.
In *Proceedings of the International Conference on Systems, Man and Cybernetics*, pages 607-613. Tucson, AZ., 1985.

[Gentner & Toupin 86]
Gentner, D., C. Toupin.
Systematicity and Surface Similarity in the Development of Analogy.
Cognitive Science 10:277-300, 1986.

[Gordon 61] Gordon W.J.
Synectics: The development of Creative Capacity.
Harper & Row, Publishers, NY, 1961.

[Gross 86] Gross, M.D.
Design as Exploring Constraints.
PhD thesis, M.I.T., 1986.

[Guilford 59] Guilford, J.P.
Creativity.
American Psychologist (5):444-454, 1959.

[Hammond 86] Hammond, K.J.
CHEF: A model of case-based planning.
In *Proceedings of AAAI-86*, pages 267-271. Philadelphia, PA, 1986.

[Holyoak 87] Holyoak, K.J., and Koh, K.
Surface and structural similarity in analogical transfer.
Memory and Cognition 15:332-340, 1987.

[Huhns 87] Huhns M.H., R.D. Acosta.
Argo: An Analogical Reasoning System for Solving Design Problems.
Technical Report AI/CAD-092-87, Microelectronic and Computer Technology Corporation, March, 1987.

[Joskowicz & Addanki 88]
Joskowicz, L., S. Addanki.
From Kinematics to Shape: An Approach to Innovative Design.
In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 347-352. 1988.

[Kedar-Cabelli 85a]
Kedar-Cabelli S. T.
Purpose-Directed Analogy.
In *Proceedings of the Cognitive Science Society Conference.* August, 1985.

[Kedar-Cabelli 85b]
Kedar-Cabelli, S.T.
Analogy - From a unified perspective.
Technical Report ML-TR-3, Department of Computer Science, Rutgers University, December, 1985.

[Kedar-Cabelli 85c]
Kedar-Cabelli, S.T.
Purpose-Directed Analogy.
In *Proceedings of the Cognitive Science Society Conference.* 1985.

[Kedar-Cabelli 85d]
Kedar-Cabelli, S.T.
Analogy - From a Unified Perspective.
Technical Report ML-TR-3, Rutgers Universtiy, 1985.

[Kling 71] Kling, R. E.
A Paradigm for Reasoning by Analogy.
Artificial Intelligence 2(2), 1971.

[Kolodner 80] Kolodner, J.L.
Retrieval and organizational strategies in conceptual memory: A computer model.
PhD thesis, Yale University, 1980.

[Kolodner 81] Kolodner, J.L.
Organization and retrieval in a conceptual memory for events.
In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.* 1981.

[Kolodner 88] Kolodner, J.L.
Retrieving Events from a Case Memory: A Parallel Implementation.
In *Proceedings of the 1988 Case-Based Reasoning Workshop*, pages 233-249. Clearwater, Fla., 1988.

[Kuipers 86] Kuipers, B.J.
Qualitative Simulation.
Aritifial Intelligence 29:289-338, 1986.

[Langley et.al. 87] Langley, P., H.A. Simon, G.L. Bradshaw, J.M. Zytkow.
Scientific Discovery - Computational Explorations of the Creative Processes.
The MIT Press, Cambridge, MA, 1987.

[Lenat 76] Lenat, D.B.
AM: An artificial intelligence approach to discovery in mathematics as heuristic search.
PhD thesis, Stanford University, STAN-CS-76-570, 1976.

[Lenat 83] Lenat, D.B.
EURISKO: a program that learns new heuristics and domain concepts, The nature of Heuristics III: program design and results.
Aritificial Intelligence 21(1,2), 1983.

[Lenat 84] Lenat, D.B.
Why AM and EURISKO Appear to Work.
Artificial Intelligence 24:269-294, 1984.

[Lozano-Perez 83] Lozano-Perez, T.
Spatial Planning: A Configuration Space Approach.
IEEE Transactions on Computers C-32(2), 1983.

[Mitchell et. al. 86]
Mitchell, T.M., R. M. Keller, S. T. Kedar-Cabelli.
Explanation-Based Generalization: A Unifying View.
Machine Learning 1(1), 1986.

[Mittal 85] Mittal, S., Dym, C. and Morjaria, M.
PRIDE: An Expert System for the Design of Paper Handling Systems.
In Dym, C. (editor), *Applications of Knowledge-Based Systems to Engineering Analysis and Design*, pages 99-116. American Society of Mechanical Engineers, 1985.

[Mostow 85] Mostow, J.
Toward Better Models Of The Design Process.
The AI Magazine , Spring, 1985.

[Mostow 86] Mostow, J.
Why are design derivations hard to replay?
In Mitchell T.M., J.G. Carbonell, R.S. Michalski (editor), *Machine Learning - A guide to current research.* Kluwer Publishing, 1986.

[Muller 87] Muller, E.T.
Daydreaming and Computation: A Computer Model of Everyday Creativity, Learning, and Emotions in the Human Stream of Thought.
PhD thesis, Univ. of California, Los Angeles, UCLA-AI-87-8, February, 1987.

[Murthy & Addanki 87]
Murthy, S.S., S. Addanki.
PROMPT: An Innovative Design Tool.
In *Proceedings of the sixth national conference on artificial intelligence*, pages 637-642. 1987.

[Navinchandra 89]
Navinchandra, D.
Exploration and Innovation in Design.
Springer Verlag, Expected in 1989.

[Nilsson 80] Nilsson, N.J.
Principles of Artificial Intelligence.
Tioga, Palo Alto, CA, 1980.

[Osborn 53] Osborn, A. F.
Applied Imagination.
Charles Scribner's Sons, New York, 1953.

[Pahl & Beitz 84] Pahl, G., W. Beitz.
Engineering Design.
The Design Council, Springer-Verlag, 1984.

[Quillian 68] Quillian, R.M.
Semantic Memory.
In Minsky, M. (editor), *Semantic Information Processing*. M.I.T. Press, Cambridge, 1968.

[Rickards 74] Rickards, T.
Problem Solving through Creative Analysis.
Wiley, NY, 1974.

[Rossman 31] Rossman, J.
The Psychology of the Inventor.
Inventor's Publishing, Washington, 1931.

[Schank 82] Schank, R.C.
Dynamic Memory: A Theory of reminding and learning in computers and people.
Cambridge University Press, 1982.

[Schank 86] Schank, R.C.
Explanation Patterns: Understanding Mechanically and Creatively.
Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[Schank & Abelson 77]
Schank, R.C., R.P. Abelson.
Scripts, Plans, Goals, and Understanding.
Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.

[Sriram 86] Sriram, D.
Knowledge-Based Approaches for Structural Design.
PhD thesis, Carnegie Mellon University, 1986.

[Stefik 80] Stefik, M.
Planning with Constraints.
PhD thesis, Stanford University, STAN-CS-80-784, 1980.

[Steinberg et.al. 86]
Steinberg L., N. Langrana, T. Mitchell, J. Mostow, C. Tong.
A Domain Independent Model of Knowledge-Based Design.
Technical Report AI/VLSI Project Working Paper No. 33, Rutgers Universtiy, March, 1986.

[Sycara 87] Sycara, K.
Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytic Methods.
PhD thesis, School of Information and Computer Science Georgia Institute of Technology, 1987.

[Sycara & Navinchandra 89]
Sycara, K., D. Navinchandra.
Integrating Case-Based Reasoning and Qualitative Reasoning in Design.
In J. Gero (editor), *AI in Design.* Computational Mechanics, U.K., 1989.

[Tong 86a] Tong, C.
Knowledge-Based Circuit Design.
PhD thesis, Stanford University, 1986.

[Tong 86b] Tong, C.
A framework for organizing and evaluating knowledge-based models of the design process.
Technical Report AI/VLSI Project Working Paper No. 21, Rutgers University, 1986.

[Tversky 77] Tversky, A.
Features of Similarity.
Psychology Review 84(4), July, 1977.

[Ullman & Dietterich 87]
Ullman, D.G., T.A. Dietterich.
Mechanical Design Methodology: Implications on Future Developments of Computer-Aided Design and Knowledge-Based Systems.
Engineering with Computers 2:21-29, 1987.

[Walker 87] Walker, M.G.
How Feasible is Automated Discovery.
IEEE Expert :69-82, Spring, 1987.

[Wallas 26] Wallas, G.
The Art of Thought.
Harcourt, NY, 1926.

[Wang, Mittal & Leifer 89]
Wang, W., S. Mittal, L. Leifer.
An Innovative Alternatives Generator for Force Transducer Conceptual Design.
In Navinchandra, D., M. S. Fox (editor), *Research Issues in AI and Design: Proceedings of the second AAAI Workshop on Design.* Morgan Kaufman Publishers (Forthcoming), 1989.

[Winston 80] Winston, P. H.
Learning and Reasonig by Analogy.
Communications of the ACM 23(12), December, 1980.

[Winston 81] Winston, P. H.
Learning New Principles from Precedents and Exercises: The Details.
Technical Report AI Lab Memo 632, MIT, A.I. Lab, 1981.

[Winston et.al. 83]
Winston, P. H., T.O. Binford, B. Katz, M. Lowry.
Learning Physical Descriptions from Functional Definitions, Examples and Precedents.
In *Proceedings of AAAI-83*. August, 1983.