Metaphors and Models

by

Michael R. Genesereth
Computer Science Department
Stanford University
Stanford, California 94305

1. Introduction

Much of one's knowledge of a task domain is in the form of simple facts and procedures. While these facts and procedures may vary from domain to domain, there is often substantial similarity in the "abstract structure" of the knowledge. For example, the notion of a hierarchy is found in biological taxonomy, the geological classification of time, and the organization chart of a corporation. One advantage of recognizing such abstractions is that they can be used in selecting metaphors and models that are computationally very powerful and efficient. This power and efficiency can be used in evaluating plausible hypotheses about new domains and can thereby motivate the induction of abstractions even in the face of partial or inconsistent data. Furthermore, there is a seductive argument for how such information processing criteria can be used in characterizing "intuitive" thought and in explaining the cogency of causal arguments. The idea of large-scale, unified knowledge structures like abstractions is not a new one. The gestalt psychologists (e.g. [Kohler]) had the intuition decades ago, and recently Kuhn [Kuhn], Minsky [Minsky], and Schank [Schank & Abelson] have embodied similar intuitions in their notions of paradigms, frames, and scripts. (See also [Bobrow & Norman] and [Moore & Newell] for related ideas.) The novelty here lies in the use of such structures to select good metaphors and models and in the effects of the resulting power and efficiency on cognitive behavior.

This paper describes a particular formalization of abstractions in a knowledge representation system called ANALOG and shows how abstractions can be used in model building, understanding and generating analogies, and theory formation. The presentation here is necessarily brief and mentions only the highlights. The next section defines the notions of abstraction and simulation structure. Section 3 describes the use of abstractions in building computational models, and section 4 shows how abstractions can be used to gain power as well as efficiency.

2. Abstractions and Simulation Structures

Formally, an *abstraction* is a set of symbols for relations, functions, constants, and actions together with a set of axioms relating these symbols to each other. Abstractions include not only small, simple concepts like hierarchies but also more complex notions like concavity and convexity or particles and waves. A *model* for an abstraction is essentially an interpretation for the symbols that satisfies the associated axioms. Different task domains can be models of the same abstraction (as biological taxonomy, geological time, and organization charts are instances of hierarchies); or, said the other way around, each abstraction can have a number of different models. Importantly, there are multiple computational models for most abstractions. In order to distinguish computer models from the task domains they are designed to mimic, they are hereafter termed *simulation structures*, following Weyhrauch [Weyhrauch].

There is a strong relationship between abstractions and metaphors, or analogies. Many analogies are best understood as statements that the situations being compared share a common abstraction. For example, when one asserts that the organization chart of a corporation is like a tree or like the taxonomy of animals in biology, what he is saying is that they are all hierarchies. With this view, the problem of understanding an analogy becomes one of recognizing the shared abstraction.

Of course there are an infinite number of abstractions. What gives the idea force is that the simulation structures for certain abstractions have representations that are particularly economical, algorithms that are particularly efficient, or theorems that are particularly powerful, e.g. hierarchies, grids, partial orders, rings, groups, monoids. Consequently, there is advantage to be gained from recognizing the applicability of one of these special abstractions rather than synthesizing a new one.

Even when the applicability of such special abstractions and simulation structures cannot be determined with certainty (say, in the face of incomplete or faulty information), there is advantage in hypothesizing them. Until one is forced to switch abstractions due to incontrovertible data, one has an economical representation and powerful problem solving methods. By biasing the early choice of abstractions in this way, these criteria can have qualitative effects on theory formation.

3. Models

The importance of abstractions and their associated measures of economy, efficiency, and power is clearest in the context of a concrete implementation like the ANALOG knowledge representation system. The interesting feature of ANALOG is that it utilizes a variety of simulation structures for representing different portions of the knowledge of a task domain. This setup is graphically illustrated in figure 1. The user asserts facts in the system's uniform, domain-independent formalism, and the system stores them by modifying the appropriate simulation structure. Facts for which no simulation structure is appropriate are simply filed away in the uniform representation. (ANALOG currently uses a semantic network representation called DB [Genesereth 76]. The formalism allows one to encode sentences in the predicate calculus of any order and provides a rich meta-level vocabulary.) Descriptions of each of ANALOG's abstractions and simulation structures are also encoded within the DB representation.

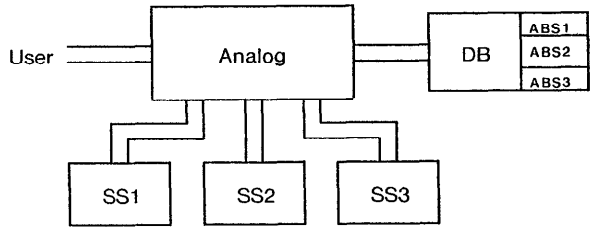

Figure 1 - An Overview of ANALOG

This approach departs from the custom in knowledge representation systems of using uniform, domain independent formalisms. While there are advantages to uniformity, in many cases the representations are less economical than specialized data structures, and the associated general procedures (like resolution) are less efficient or less powerful than specialized algorithms. For

example, a set in a small universe can be efficiently represented as a bit vector in which the setting of each bit determines whether the corresponding object is in the set. Union and intersection computations in this representation can be done in a single machine cycle by hardware or microcoded boolean operations. By contrast, a frame-like representation of sets would consume more space, and the union and intersection algorithms would have running times linear or quadratic in the sizes of the sets. The distinction here is essentially that between "Fregean" and "analogical" representations, as described by Balzer [Balzer]. Note that ANALOG's approach is perfectly compatible with uniform knowledge representation systems like DB and RLL [Greiner & Lenat]. The addition of abstractions and simulation structures can be viewed as an incremental improvement to such systems, and their absence or inapplicability can be handled gracefully by using the uniform representation.

It's important to realize that ANALOG is not necessarily charged with inventing these clever representations and algorithms, only recognizing their applicability and applying them. The approach is very much in the spirit of the work done by Green, Barstow, Kant, and Low in that there is a knowledge base describing some of the best data representations and algorithms known to computer science. This knowledge base is used in selecting good data representations and efficient algorithms for computing in a new task domain. One difference with their approach is that in ANALOG there is a catchall representation for encoding assertions when no simulation structure is applicable. Other differences include an emerging theory of representation necessary in designing new simulation structures (see section 3.2) and the use of the criteria of economy, efficiency, and power in theory formation.

ANALOG's use of simulation structures is in a very real sense an instance of model building. Architects and ship designers use physical models to get answers that would be too difficult or too expensive to obtain using purely formal methods. ANALOG uses simulation structures in much the same way. In fact, there is no essential reason why the simulation structures it uses couldn't be physical models. Furthermore, as VLSI dissolves what **John Backus** calls the vonNeumann bottleneck, the number of abstractions with especially efficient simulation structures should grow dramatically.

3.1 Building a Model

As an example of modeling, consider the problem of encoding the organization chart of a corporation. The first step in building a model for a new task domain is finding an appropriate abstraction and simulation structure. The knowledge engineer may directly name the abstraction or identify it with an analogy, or the system may be able to infer it from an examination of the data. In this case, the hierarchy abstraction is appropriate, and there are several appropriate simulation structures. One of these is shown in figure 2. Each object in the universe is represented as a "cons" cell in which the "car" points to the object's parent. The relation (here called **Rel**) is just the transitive closure of the **Car** relation, and **Nil** is the root. For the purposes of this example, the "cdr" of each cell may be ignored.

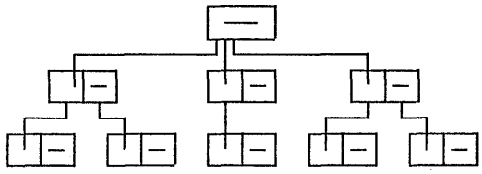

Figure 2 - A Simulation Structure for the Hierarchy Abstraction

An important requirement for a simulation structure is that it be modifiable. Therefore, it must include actions that the model builder can use in encoding knowledge of the task domain. Usually, this requires the ability to create new objects and to achieve relations among them. In this case, the **Ncons** subroutine creates a new object, and **Rplaca** changes an object's parent.

Part of the task of finding an appropriate abstraction and simulation structure is setting it up for use in encoding knowledge of the task domain. This includes three kinds of information. The first is an index so that the system can determine the simulation structure appropriate to a new assertion. (This index is necessary since several domains and simulation structures may be in use simultaneously). Secondly, there must be a procedure for mapping each assertion into its corresponding assertion about the simulation structure. And, finally, the system must have information about how to achieve the new assertion.

Once the simulation structure is chosen and set up, the system builder can begin to assert facts about the task domain, and the system will automatically modify the simulation structure. As an example of this procedure, consider how the system would handle the assertion of the fact **(Boss-of Carleton Bertram)**. First, it would use its index to determine that the simulation structure of figure 2 is being used and to recover the mapping information. Then it would map the assertion into the simulation domain. In this case, let's say that **Arthur** is the boss of **Bertram** and **Beatrice** while **Carleton** has been installed in the model as **Beatrice**'s employee. Then the new assertion would be **(Rel ((Nil)) (Nil))**, where the first argument is the object representing **Carleton** and the second represents **Bertram**. By examining the meta-level information about the simulation structure, the system retrieves a canned procedure **(Rplaca)** for achieving this fact and executes it, with the result that **Carleton**'s "car" is redirected from **Beatrice** to **Bertram**.

An interesting aspect of model building is that complete information is often required. For example, in adding a node to the simulation structure of figure 2, the system must know the object's parent in order to succeed. (It has to put something in the "car" of the cell.) This problem can sometimes be handled by the addition of new objects and relations that capture the ambiguity. For example, one could add the special token **Unknown** as a place filler in the simulation structure above. (Of course, the resulting structure would no longer be a hierarchy.) Another example is using the concept of uncle as a union of father's brother and mother's brother. Unfortunately, this approach increases the size of the model and makes deductions more difficult. Unless there are strong properties associated with such disjunctive concepts, it is usually better to carry the ambiguity at the meta-level (i.e. outside the model, in the neutral language of the knowledge representation system) until the uncertainty is resolved.

Another interesting aspect of the use of simulation structures is the automatic enforcement of the axioms of the abstraction. For example, in the simulation structure of figure 2, it is impossible to assert two parents for any node simply because a "cons" cell has one and only one "car". Where this is not the case (as when a simulation structure is drawn from a more general abstraction), the axioms can still be used to check the consistency and completeness of the assertions a system builder makes in describing his task domain. For example, if the system knew that a group of assertions was intended to describe a hierarchy, it could detect inconsistent data such as cycles and incomplete data such as nodes without parents.

3.2 Designing a Simulation Structure for an Abstraction

The only essential criteria for simulation structures are representational adequacy and structure appropriate to their abstractions. For every assertion about the task domain in the language of the abstraction, there must be an assertion about the simulation structure; and the structure must satisfy the axioms of the abstraction.

In creating a simulation structure, one good heuristic is to try to set up a homomorphism. Sometimes, the objects of the simulation structure can be used directly, as in the case of using "cons" cells to represent nodes in a hierarchy. In the example above, the mapping of objects from the corporation domain into the domain of list structure was one-to-one, i.e. the corporate objects were all represented by distinct pieces of list structure, and

the relations and actions all mapped nicely into one another. Of course, this need not always be the case. Consider, for example, the state vector representation of the Blocks World proposed by McCarthy, in which the **Supports** relation between each pair of blocks is represented by a distinct bit in a bit vector. (Think of the vector as a matrix in which the $\langle i, j \rangle$th bit is on if and only if block i is on block j). In this representation the fact **(Supports A B)** would translate into something like **(On Bit-AB Vector-1)**, and there would be no distinct representations of the blocks A and **B**.

In other cases, more complex objects may be necessary in order to provide enough relations. When a domain does not provide an adequate set of relations, it's a good idea to synthesize complex structures from simpler ones. For example, a simulation structure for an abstraction with three binary relations could be built in the list world by representing objects as pairs of "cons" cells in which the "car" represents the first relation, the "cdr" points to the second cell, the "cadr" represents the second relation, and the "cddr" represents the third. This approach is facilitated by programming languages with extensible data structures.

Obviously, it pays to economize by using the predefined relations of the simulation domain where possible. For example, a good representation for a univariate polynomial is a list of its coefficients, and one gets the degree of the polynomial for free (the length of the list minus 1). One advantage is representational economy; another is automatic enforcement of the abstraction's axioms, as described in the last section.

In order to use a simulation structure, it may be necessary to transform objects into a canonical form. For example, one can represent a univariate polynomial as a list of coefficients, but the polynomial must be in expanded form. There is a large body of literature on canonical forms for specific algebraic structures, while [Genesereth 79] gives a general but weak technique for inventing such forms directly from an abstraction's axioms.

In using a simulation structure, there is a tradeoff between the amount of work done by the model and the amount done by the knowledge representation system. For example, in the simulation structure of figure 2, one must loop over the parent relation to determine whether two objects are related. This can be done either by the knowledge representation system or by a LISP procedure in the simulation structure. Obviously, it's a good idea to have the simulation structure do as much work as possible.

3.3 Interfacing Simulation Structures

For most interesting task domains, the chances are that a single simulation structure is not sufficient. In such cases, it is sometimes possible to piece together several different simulation structures. The simplest situation arises when the objects of the task domain form a hierarchy under the "part" relation. Then one can choose one representation for the "topmost" objects and a different representation for the parts. The spirit of this approach is very similar to that of "object-oriented" programming in which each object retains information about how it is to be processed. One disadvantage of this approach is that each object must have explicit "type" information stored with it. Barton, Genesereth, Moses, and Zippel have recently developed a scheme that eliminates this need by separating the processing information from each object and passing it around in a separate "tree" of operations. ANALOG uses this scheme for encoding the operations associated with each simulation structure.

Task domains with several relations are sometimes decomposable into several abstractions, and these relations can then be represented independently. More often the relations are interdependent; and, when this is the case, the interdependence must be dealt with in the uniform representation.

Even when a single abstraction would fit the task domain, it may be advisable to use several. Consider, for example, a partial order that nicely decomposes into two trees. Furthermore, there are often advantages to multiple representations of objects, as argued by Moses [Moses].

4. Thinking With Abstractions and Simulation Structures

The use of specialized simulation structures gives ANALOG an economy and efficiency not possible with a uniform representation. The economy can be expressed in terms of the space saved by representing assertions in the simulation structure rather than the uniform representation. This economy derives from the elimination of the overhead inherent in uniform formalisms and the use of relations implicit in the simulation structure (as the length of a list reflects the degree of the polynomial it represents). The efficiency refers to the time involved in doing deductions and solving problems. This efficiency may be attributable to clever algorithms, or it may be the result of long familiarity with the domain from which the abstraction evolved (due to the memory of many special case heuristics). Lenat [Lenat et. al.] discusses how a computer might improve its own performance by self-monitoring.

An interesting possibility suggested by this economy and efficiency is for the program to use these criteria in evaluating plausible hypotheses about a new domain. In the face of incomplete or contradictory data, the program should favor the more economical abstraction. Clearly, there is some evidence for this sort of behavior in human cognition. Consider, for example, Mendeleev's invention of the periodic table of the elements. He was convinced of the correctness of the format in spite of contradictory data, for reasons that can only be identified as simplicity.

These criteria of economy and efficiency are also of use in characterizing why it is easier to solve problems from one point of view than another, e.g. proving a theorem using automata theory rather than formal grammars. Part of what makes causal arguments (see [deKleer] for example) so compelling is that they are easy to compute with. The reason for this is that a causal argument is an instance of a cognitively efficient abstraction, namely a directed graph. One is tempted, therefore, to generalize deKleer's notion of causal envisionment as finding economical and efficient abstractions (perhaps identified with analogies) in which the desired conclusions are reached via simple computations.

The idea can be carried a bit further and generalized to include the criterion of problem solving power. In particular, one should favor an abstraction for its ability to solve a pending problem despite insufficient data. The obvious difficulty is that the assumption may be wrong or there may be several abstractions that are equally probable and useful. Consider, for example, the following arguments for determining the distance between the observer and the middle vertex of a Necker cube. "Well, the lines form a cube, and so the middle vertex must be closer to me than the top edge." "No, not at all, the figure is concave, and so the middle vertex must be further away." Both arguments are consistent with the data and refer to a single abstraction, and in each case the conclusion is deductively related to that view. A second example is evident in the particulate-wave controversy. The particulate view is a simple abstraction that accounts for much of the data and allows one to solve outstanding problems. Of course, the same can be said for the wave view. Unfortunately, the predictions don't agree. A similar argument explains the inferential leap a child makes in declaring that the wind is caused by the trees waving their leaves. When the child waves his hand, it makes a breeze; the trees wave when the wind blows; so they must have volition and motive power; and that would account for the wind.

The reasoning in these examples is usually termed "analogical". The key is the recognition of a known abstraction common to the situations being compared. This conception of analogy differs markedly from that of Hayes-Roth and Winston. In their view two situations are analogous if there is any match between the two that satisfies the facts of both worlds. If the match is good, the facts or heuristics of one world may be transferred to the other. The problem is that these facts may have nothing to do with the analogy. Just because two balls are big and plastic, one can't infer because one ball is red that the other is also red. Abstractions are ways of capturing the necessary interdependence of facts. For example, the size and material of a ball do affect its mechanical behavior, and so the skills useful for

bouncing one should be of value in bouncing the other. Also note that the match need not be close in order for there to be a useful analogy. Linnaean taxonomy and organization charts have few superficial details in common, but the analogy is nonetheless compelling, and as a result the algorithms for reasoning about one can be transferred to the other. The work of Hayes-Roth and Winston is, however, applicable where no abstractions exist yet. Their matching algorithms and the techniques of Buchanan, Mitchell, Dietterich and Michalski, and Lenat should be important in inducing new abstractions.

An important consumer for these ideas is the field of computer-aided instruction. There is a current surge of interest in producing a "generative theory of cognitive bugs" (see [Brown], [Genesereth 80a], and [Matz]). The use of abstractions and the criteria of economy, efficiency, and power in theory formation is very seductive in this regard. Unfortunately, there is no reason to believe that the hardware of a vonNeumann computer in any way resembles the specialized capabilities of the human brain. (Indeed, psychologists are still debating whether there are any analogical processes in the brain at all. See, for example, [Kosslyn & Pomerantz], [Kosslyn & Schwartz], [Pylyshyn], and [Shepard & Metzler].) Thus, the idea at present is not so much a model for human cognitive behavior as a metaphor.

5. Conclusion

The ANALOG system was developed over a period of time to test the ideas presented here. One program accepts an analogy and infers the appropriate abstraction; another builds a model of the task domain as assertions are entered; and a third uses the model to answer questions. There is a sketchy implementation of the simulation structure designer, but no effort has been made to build the theory formation program.

In summary, the key ideas are (1) the role of abstractions in understanding metaphors and selecting good models for task domains, (2) the use of models to acquire economy, efficiency, and problem solving power, and (3) the importance of these criteria in theory formation. Abstractions and simulation structures make for a knowledge representation discipline that facilitates the construction of powerful, efficient AI programs. The approach suggests a program for much future work in AI and Computer Science, viz. the identification of useful abstractions and the implementation of corresponding simulation structures that take advantage of the special computational characteristics of the vonNeumann machine and its successors.

Acknowledgements

The content of this paper was substantially influenced by the author's discussions with Bruce Buchanan, Rick Hayes-Roth, Doug Lenat, Earl Sacerdoti, and Mark Stefik, though they may no longer recognize the ideas. Jim Bennett, Paul Cohen, Russ Greiner, and Dave Smith read early drafts and made significant suggestions to improve the presentation. The work was supported in part by grants from ARPA, NLM, and ONR.

References

Balzer, R. Automatic Programming, Institute Technical Memo, University of Southern California/ Information Sciences Institute, 1973.

Barstow, D. R. *Knowledge Based Program Construction*, Elsevier North-Holland, 1979.

Brown, J. S. & vanLehn, K. forthcoming paper on learning.

Buchanan, B. & Feigenbaum, E. A. Dendral and Meta-Dendral: Their Applications Dimension, *Artificial Intelligence*, Vol. 11, 1978, pp 5-24.

deKleer, J. The Origin and Resolution of Ambiguities in Causal Arguments, Proc. of the Sixth International Joint Conference on Artificial Intelligence, 1979, pp 197-203.

Dietterich, T. G. & Michalski, R. S. Learning and Generalization of Characteristic Descriptions: Evaluation Criteria and Comparative Review of Selected Methods, Proc. of the Sixth International Joint Conference on Artificial Intelligence, 1979, pp 223-231.

Hayes-Roth, F. & McDermott An Interference Matching Technique for Inducing Abstractions, *Comm. of the ACM*, Vol. 21 No. 5, May 1978, pp 401-411.

Genesereth, M. R. A Fast Inference Algorithm for Semantic Networks, Memo 5, Mass. Inst. of Tech. Mathlab Group, 1976.

Genesereth, M. R. The Canonicality of Rule Systems, Proc. of the 1979 Symposium on Symbolic and Algebraic Manipulation, Springer Verlag, 1979.

Genesereth, M. R. The Role of Plans in Intelligent Teaching Systems, in *Intelligent Teaching Systems*, D. Sleeman, ed. 1980.

Genesereth, M. R. & Lenat, D. B. Self-Description and Self-Modification in a Knowledge Representation System, HPP-880-10, Stanford University Computer Science Dept., 1980.

Green, C. C. The Design of the PSI Program Synthesis System, Proc. of the Second International Conference on Software Engineering, Oct. 1976, pp 4-18.

Greiner, R. D. & Lenat, D. B. A Representation Language Language, submitted for inclusion in the Proc. of the First Conference of the American Association for Artificial Intelligence, Aug. 1979.

Kant, E. Efficiency Considerations in Program Synthesis: A Knowledge Based Approach, doctoral dissertation, Stanford Univ. Computer Science Dept., 1979.

Kohler, W. *Gestalt Psychology: An Introduction to New Concepts in Modern Psychology*, Liveright, 1947.

Kosslyn, S. M. & Pomerantz, J. R. Imagery, Propositions, and the Form of Internal Representations, *Cognitive Psychology*, Vol. 9 No. 1, 1977, pp 52-76.

Kosslyn, S. M. & Schwartz, S. P. A Simulation of Visual Imagery, *Cognitive Science*, Vol. 1, 1977, pp 265-295.

Kuhn, T. *The Structure of Scientific Revolutions*, Univ. of Chicago Press, 1962.

Lenat, D. B. Automated Theory Formation in Mathematics, Proc. of the Fifth International Joint Conference on Artificial Intelligence, 1977, pp 833-841.

Lenat, D. B., Hayes-Roth, F., Klahr, P. Cognitive Economy in Artificial Intelligence Systems, Proc. of the Sixth International Joint Conference on Artificial Intelligence, 1979, pp 531-536.

Low, J. R. Automatic Coding: Choice of Data Structures, ISR 16, Birkheuser Verlag, 1976.

Matz, M. A Generative Theory of High School Algebra Errors, in *Intelligent Teaching Systems*, D. Sleeman, ed. 1980.

McCarthy, J. Finite State Search Problems, unpublished paper.

Minsky, M. A Framework for Representing Knowledge, in *The Psychology of Computer Vision*, P. H. Winston, ed., McGraw-Hill, 1975.

Mitchell, T. Version Spaces: A Candidate Elimination Approach to Rule Learning, Proc. of the Fifth International Joint Conference on Artificial Intelligence, 1977.

Moore, J. & Newell, A. How can MERLIN Understand, in L. W. Gregg, ed. *Knowledge and Cognition*, Lawrence Erlbaum, 1974.

Moses, J. Algebraic Simplification: A Guide for the Perplexed, *Comm. of the ACM*, Vol. 14 No. 8, 1971, pp 527-537.

Pylyshyn, Z. W. What the Mind's Eye Tells the Mind's Brain: A Critique of Mental Imagery, *Psychological Bulletin*, Vol. 80, 1973, pp 1-24.

Schank, R. & Abelson, R. Scripts, Plans, and Knowledge, Proc. of the Fourth Internatyional Joint Conference on Artificial Intelligence, 1975, pp 151-157.

Shepard, R. N. & Metzler, J. Mental Rotation of Three-dimensional Objects, *Science*, Vol. 171, 1977, pp 701-703.

Thorndyke, P. W. & Hayes-Roth, B. The Use of Schemata in the Acquisition and Transfer of Knowledge, *Cognitive Psychology* Vol. 11, 1979, pp 82-106.

Weyhrauch, R. Prolegomena to a Theory of Formal Reasoning, STAN-CS-78-687, Stanford Univ. Computer Science Dept., Dec. 1978.

Winston, P. H., Understanding Analogies, Mass. Inst. of Tech. Artificial Intelligence Laboratory, Apr. 1979.