CUCS-124-84

Understanding Hierarchically Structured Objects

Kenneth Wasserman

May 1984

This research was supported in part by the Defense Advanced Research Projects Agency under contract N00039-82-C-0427.

Table of Contents

Abstract

Natural and artificial hierarchical systems are pervasive. There is a strong need on the part of researchers in artificial intelligence and other areas of cognitive science to have mechanisms for "understanding" such systems. Furthermore, computer programs often represent data in a hierarchical form and they would greatly benefit from a technique that would allow them to dynamically build such classification systems from this data. This paper presents a formalism for describing hierarchies and then uses this formalism to explore the issue of inheritance from multiple sources which is of particular importance in hierarchy understanding.

1. Introduction

Much of what we perceive in the world around us is hierarchical. Physical objects, river and road systems, library systems, family relations and all kinds of taxonomies are but a few examples of common hierarchical phenomena (see [Simon 81] for more examples). Humans are apparently adept at understanding information presented as hierarchies. We even create abstract hierarchically structured systems when entertaining ourselves; most western music fits this form. It seems logical that artificially intelligent beings (computers) should have a similar capacity to comprehend hierarchies.

Artificial Intelligence researchers have used hierarchies for representing knowledge almost since its beginning (for example, NOAH [Sacerdoti 75] stored its knowledge as a hierarchy of plans and SCHOLAR [Carbonell 70] used a semantic net-based hierarchy to encode facts about South America). Usually these knowledge structures are built by a human expert and used by a computer program to either solve a problem or store more information according to the previously established classification categories. Furthermore, the data put into these hierarchical classification systems are sometimes hierarchically structured themselves. For example, animal body parts form a hierarchy that is classified in a taxonomy that distinguishes creatures based upon their physical attributes [Hayes 77].

There is an obvious need for computer systems to automatically classify their own data (for example, [Michalski and Stepp 83]. In our opinion this can only be achieved by building systems that can "understand" the hierarchies that fall within their domain of expertise. In addition, there remains a great deal to be learned about the fundamental nature of hierarchies. The goal of our research is to yield interesting results in both of these areas.

The word ***understand*** is used in many senses both in ordinary conversational language and in Artificial Intelligence. The main thrust of all of these is the incorporation of an idea or experience into memory in relation to other, pre-existing ideas or experiences. Incorporating a piece of knowledge into memory is usually termed ***learning***. One form of learning, ***generalization***, is the process by which similarities among various pieces of information are ferreted out and a structure is created that embodies the facts common to these instances. In some sense all

learning might be considered to be generalization. Thus, "understand" will refer to the generalization-type learning process for the purposes of this paper.

We will present a formalism for exploring issues relating to hierarchy understanding and use this formalism to explore one of the many important problems. The problem of building a generalization hierarchy using data that can be inherited from multiple sources will be analyzed within the context of hierarchy understanding. The paper concludes by identifying several other issues of importance that need to be addressed before a complete hierarchy understanding system can be developed.

2. Definitions and Formalism

In order to discuss the nature of hierarchies some terminology must be presented. A *hierarchy* is a group of objects that exist in some partially ordered state such that a sub-group of objects that are all ***subservient*** to another object form a logical class. In this way, a hierarchy serves as a classification system for the objects that comprise it.

What makes one object, or node, in a hierarchy subservient to another depends on the information that the hierarchy is attempting to capture. For example, the often-used IS-A link (see [Quillian 68] for early work using IS-A links in semantic nets) serves to indicate that one node is an instance of another node. It has application to almost any domain with a classification scheme based upon group membership. PART-OF is another commonly used inter-node relation. (See [Winograd 72] for similar examples in the blocks world.) Here the link denotes that one object is physically included within another. A third example, from the business community, is the idea of *chain-of-command*. For practical purposes this concept can be expressed as a binary relation between two nodes and might be termed REPORTS-TO (e.g., the president REPORTS-TO the chairman of the board). This relation is used by corporations to build organizational hierarchies.

These examples illustrate that there is a single *fundamental relation* that serves as the backbone of any hierarchical structure. IS-A, PART-OF and REPORTS-TO are the links that give structure to the classification, component and chain-of-command hierarchies. They are all examples of fundamental relations, which we

will term F-RELs for convenience. To be sure, other relations can and do exist between nodes in a hierarchy. For example, channels of communication between members of a corporation can be represented as relations between nodes in the organizational chart. However, the F-REL of REPORTS-TO is the most significant relation and provides the framework for the hierarchy to be assembled on. Therefore, it is the F-REL of the hierarchy that determines the partial ordering of its objects.[1]

As stated earlier, generalization is what is implied by the word "understand", in our usage of it. When two or more objects are compared, a generalization about them can be made. Furthermore, when two or more generalizations (or an object and a generalization) are compared, a higher level generalization can often be made. This process will produce a *hierarchy of generalizations*. The F-REL of this hierarchy will be called VARIANT-OF.

The objects that are the nodes of this generalization hierarchy need not be unitary; they can be hierarchically structured entities themselves. Herein lies the notion of "understanding hierarchies". It is precisely this: hierarchical systems can be understood by building a generalization hierarchy in which each object in it either represents a single instance of a hierarchy in the domain whose knowledge is being encoded, or is itself another generalization. For example, to understand automobiles one could build individual hierarchies for each car based upon the PART-OF F-REL (e.g., the cylinders are PART-OF the motor which is PART-OF the car). These would then become objects in a generalization hierarchy that might represent information such as: A Ferrari is a VARIANT-OF a sports car which is a VARIANT-OF an expensive automobile.

Understanding hierarchical systems requires the use of at least two *orthogonal* representation hierarchies. One is the structure hierarchy based upon the F-REL of the domain, which we will call the F-TREE, "Tree" being easier to read than "hierarchy" and these structures are usually tree-like in appearance. An F-TREE

---

[1]It should be pointed out that a particular domain may have several equally valid F-RELs. For example, although must corporate hierarchies are built on the REPORTS-TO F-REL there might be a need to design a structure chart based upon physical location. In this case the proper F-REL might be IN-SAME-REGION. However, for most domains there is only one obvious F-REL to use.

provides a description of the structure of a single hierarchy in the domain under study. The other representation structure is the generalization hierarchy, which we will term the G-TREE for similar reasons. The G-TREE describes a whole set of instance hierarchies within one classification hierarchy. The two trees are orthogonal in the sense that they represent different ways of looking at the same set of objects. The F-TREE of an object symbolizes the way that a particular object is structured (e.g. physical structure, chain-of-command, etc.), while the G-TREE represents how the objects can be classified according to their F-TREE encodings.

In order to facilitate our presentation of F-TREEs, G-TREEs and their interrelation we need a concise notation. The essential facts that must be made apparent by a good notational scheme are: given any node, what are its immediate descendants in the F-TREE (i.e., subservient nodes) and what are its immediate G-TREE descendants (i.e., instances and/or other generalizations). Furthermore, such a scheme should be flexible enough to allow us to add new operators into it. In particular, we will need to add operators that tie together the F-TREEs and G-TREEs. Although these two trees were described as orthogonal, the understanding process is rooted in the interaction between the two trees.

Encoding	Meaning
A: B, D	node A has F-TREE subservient nodes B and D
B: C, >F	node B has F-TREE subservient node C as well as G-TREE variant F
C: E	node C has F-TREE subservient node E
D:	node D has no F-TREE or G-TREE descendants
F: +G	node F has an F-TREE subservient node, G in addition to the ones it inherits from B

Each node in the F-TREE and G-TREE that has children is represented by its name followed by a list of its descendants, a colon delineates the parent from the children. G-TREE variants are prefixed by a ">" to distinguish them from the F-TREE descendants in the list of children. A "+" symbol prefixed before an F-TREE descendant indicates that it is an *added* F-REL link (in addition to those that it inherits from its parent(s)). The F-REL of the hierarchy is implicit here and must be stated outside the context of the encoding.

Figure 2-1: Notation for F-TREEs and G-TREEs.

Figure 2-1 introduces the basic notational scheme we have chosen to accomplish the goals outlined above. The name given to identifiers can be abstract single

letter codes, abbreviations or more descriptive names. To put the abstract structure described in this figure into perspective, assume that the F-REL of this example is PART-OF. Then, this structure defines two objects, **A** and **F**. Object **A** has parts **B** and **D**; object **B** in turn has a part, **C** while object **D** has no parts. Furthermore, object **C** has **E** as a part. The second object, **F**, is actually a VARIANT-OF object **B**. Thus, it *inherits* all the parts (F-REL links) that **B** contains. It also has an *additional* part that **B** does not have, namely **G**. In total, **F** has **C** and **G** as immediate parts.

The concepts introduced by this example are: *inheritance* - nodes that are variants of other nodes in the G-TREE inherit all the F-REL links (parts in this case) that the parent node has; *addition* - the child node can have additional F-REL links that the parent does not have (this is denoted by the "+" symbol).

Inheritance is certainly not a new technique in AI research and is often the main motivation behind using hierarchical systems (see [Brachman 79a, Minsky 75] for examples). It has the very desirable benefits of saving on memory usage (in that data can be shared among similar nodes) and organizing information for easy retrieval (usually only a few pointers need be followed for a node to acquire its inherited data). We have specified that only F-REL links are inherited in our representation scheme. In fact, one often wants to allow for other information to be inherited, but since the examples in this paper deal only with F-RELs we will not consider inheritance of other data. It should be emphasized that variants are specifically *not* inherited.

A node that is a VARIANT-OF another node in the G-TREE can have F-REL links (parts in this case) added to its F-TREE that are not inherited from the parent node. Thus, *addition* is a means for specifying that a node is "just like its parent only it has added F-REL links". Note that addition of F-REL links is only possible if the node that is being added to is a VARIANT-OF some other node. Other concepts than inheritance and addition are easily added to this formalism. For example, *subtraction* is an operation similar to addition but specifies that a node is "just like its parent only its missing an F-REL link." Another operation, *substitution*, allows for the replacement of an F-REL link, that would be inherited from the parent node, by a different F-REL link in the child.

With this formalism in hand, we will explore what happens when a node inherits data from more than one parent in the G-TREE. In particular, we want to incrementally build unambiguous G-TREEs that allow for inheritance of data from multiple sources. It turns out that this process presents some difficult and interesting problems. We term this phenomena the *multi-source inheritance* problem. It is distinct from the problem of multiple inheritance often encountered in semantic net representations. We are concerned with creating correct generalizations while allowing for possibly conflicting data to be inherited from multiple sources as opposed to merging non-conflicting data from multiple parents into an existing node.

3. Multi-source Inheritance

To look at the multi-source inheritance problem we will use an example from the corporate world. Assume that a company has a vice-president of marketing (VP-1) who controls the activities of the director of public relations (DPR). A second company's marketing vice-president (VP-2) has a director of advertising (DA) under him and a third company's vice-president of marketing (VP-3) controls both the public relations director and the director of advertising. In addition, all three of these marketing vice-presidents also control the activities of a director of sales (DS) and a director of market research (DMR). Furthermore, we will consider a fourth company whose vice-president of marketing (VP-4) controls a director of market research, an advertising director and a director of public relations. Figure 3-1 summarizes what we know about these four company's vice-presidents. Because we are about to get somewhat technical and all of these executive titles are a bit lengthy to keep reading, Figure 3-1 shows assignments of short mnemonics to each job title. Although only these corporate hierarchies will be analyzed here, it should be noted that the problem we will discuss exists in *any* domain where multi-source inheritance of generalizations based on a single F-REL are used.

We will start with the generalization structure built by comparing VP-1 with VP-2. Next, VP-3 will be added into the generalization hierarchy. Finally, the full magnitude of the problem will surface when VP-4 is taken into account. In this sense, we can call each company a *training instance* that generalizations are built from [Mitchell 77, Winston 72].

job title	symbolic F-TREE
v.p.-company-1	VP-1: DPR, DS, DMR
v.p.-company-2	VP-2: DA, DS, DMR
v.p.-company-3	VP-3: DPR, DA, DS, DMR
v.p.-company-4	VP-4: DPR, DA, DMR
director-public-relations	DPR:
director-advertising	DA:
director-sales	DS:
director-market-research	DMR:

Four F-TREEs are represented here, one for each marketing vice-president that will be used in our examples. Short mnemonics are assigned to each job title to condense the representations. Nodes VP-1, VP-2, VP-3 and VP-4 comprise the training instances for the example followed in the next 4 figures.

Figure 3-1: Summarized data on four companies.

VP-X: DS, DMR, >VP-1, >VP-2
VP-1: +DPR
VP-2: +DA

Company-1's vice-president of marketing (VP-1) and company-2's vice-president (VP-2) have been generalized, creating a new G-TREE node (VP-X) which contains the information that VP-1 and VP-2 have in common.

Figure 3-2: Generalization of company-1 and company-2.

Figure 3-2 shows how both VP-1 and VP-2 can be represented as variants of some hypothetical object (VP-X) which has the subservient objects DS and DMR associated with it. The only issue to take note of here is that a new G-TREE node has been created that captures the data that its two variant nodes have in common. We could have represented VP-1 as a VARIANT-OF VP-2 or vice-versa. But this representation would have lost the concept that VP-1 and VP-2 have equal status (i.e., it would have given one of them preferential status). Also it would hide the fact that they have both DS and DMR in common. It is helpful to think of the representation shown as *factoring out* what VP-1 and VP-2 have in common and storing it in node VP-X. With this in mind consider what happens when company-3's F-TREE is added into the G-TREE.

The new G-TREE node, VP-3 shown in Figure 3-3 demonstrates how its data can

VP-X: DS, DMR, >VP-1, >VP-2
VP-1: +DPR, >VP-3
VP-2: +DA, >VP-3
VP-3:

This G-TREE representation now includes company-3's vice-president. Since node VP-3 is just the union of the data in nodes VP-1 and VP-2 no new F-REL links have been added to the tree. This is an example of inheritance from two sources that causes no problems.

Figure 3-3: Addition of company-3 into the G-TREE.

be encoded by using inheritance from two sources. It is worth noting that only two variant links need be added to the representation in Figure 3-2 in order to capture all the data for node VP-3. The only common factor among company-1, company-2 and company-3's vice-presidents is that they all have the pair of employees DS, DMR. This has already been accounted for in node VP-X so no extra work (factoring) is needed here. There is also no uncertainty as to what node VP-3 should inherit. Obviously it should only have one F-REL link to DS and one F-REL link to DMR. Since it inherits these links from the same ***ultimate*** source (namely VP-X), there is no problem in determining if it should have two copies of DS, DMR or one copy. Unfortunately this simplicity does not usually prevail.

VP-X: DS, DMR, >VP-1, >VP-2
VP-Y: DPR, DMR, >VP-1, >VP-4
VP-Z: DA, DMR, >VP-2, >VP-4
VP-1: >VP-3
VP-2: >VP-3
VP-3:
VP-4:

A first attempt at incorporating VP-4 into the G-TREE shown in figure 3-3. It fails because of multi-source inheritance ambiguities. For example, it is unclear if VP-4 has 1 or 2 F-REL links to DMR.

Figure 3-4: A first try at incorporating company-4 into the G-TREE.

In Figure 3-4 we have attempted to incorporate VP-4 into the G-TREE. Two new G-TREE nodes have been created which represent different classifications of objects (vice-presidents) than node VP-X does. VP-Y and VP-Z represent objects that have subservient F-TREE nodes DPR, DMR and DA, DMR, respectively. These were built for the same reasons that node VP-X was formed. Node VP-4 is

a VARIANT-OF both VP-Y and VP-Z in that it has F-REL links to DPR, DA and DMR. The problem that arises here is: what F-REL links do nodes VP-1, VP-2, VP-3 and VP-4 really contain?

So far, there is no unique answer to this question within the realm of our formalism. The problem is in determining how many copies of a particular F-REL link should be inherited if these sources do not in turn inherit it from the same source. In the case we have illustrated, VP-4 could have, for example, two copies of DMR or just one copy. We could, of course, add a definition to our formalism to force a unique answer. One possibility is to simply state that inheritance of the same F-REL link from multiple sources results in the *union* of the sets of F-REL links from all of the parent nodes. (We will discuss this possibility later). However, such an arbitrary rule restricts the types of generalizations which are possible and detracts from possibly interesting studies of this problem. Therefore, we will look further.

Keeping in mind that we would like to factor out the common elements in G-TREE nodes, we try fixing the structure shown in Figure 3-4. To accomplish this, we realize that if we factor out the DMR node from VP-Y and VP-Z then node VP-4 will not have any ambiguities (i.e., VP-4 will inherit only one F-REL link to node DMR). Recognizing that node VP-X contains DMR as a factor, we also take it out by making VP-X a VARIANT-OF a new node, VP-T, which contains only DMR as an F-REL link. Next we make VP-Y and VP-Z variants of VP-T, as well. This will result in a structure that has only a single node that contains DMR, wherein all other nodes that need this as a subservient node will inherit DMR from this ultimate source. Figure 3-5 demonstrates exactly this.

Left to explain is how we determined that DMR had to be factored out from VP-X to fix the problem? Furthermore, how did we even know that we had to check nodes VP-1, VP-2 and VP-3 for potential problems? The answer is we didn't. With a little bit of thought one can reach the conclusion that each time a new generalization is made it is possible that this factoring problem might cause some previously represented node to become mis-represented (or at least ambiguous in meaning). To state this more precisely: if a new generalization is built that breaks up some previously existing group of factors, then it is possible that one or more

VP-T: DMR, >VP-X, >VP-Y, >VP-Z
VP-X: +DS, >VP-1, >VP-2
VP-Y: +DPR, >VP-1, >VP-4
VP-Z: +DA, >VP-2, >VP-4
VP-1: >VP-3
VP-2: >VP-3
VP-3:
VP-4:

Finally all the ambiguities have been eliminated. By factoring out DMR from nodes VP-X, VP-Y and VP-Z, node VP-T has become the *only* source of DMR for all of the nodes (vice-presidents) that have DMR in their F-TREE (employ). The key concept is: all common factors must be singled out to form multi-source inheritance hierarchies that are ambiguity-free.

Figure 3-5: Final G-TREE representation of all four companies.

nodes in the representation will inherit the same F-REL links from more than one, ultimate source.

Before describing each of the possible solutions we would like to make clear that the example just presented is quite simple. In fact it is *the* simplest example that demonstrates the full effect of the multi-source inheritance problem while incrementally incorporating training instances into a generalization hierarchy. In any real-life use of a generalization hierarchy a much deeper G-TREE would be involved. This, unfortunately, makes the factoring out process much more difficult; virtually the entire G-TREE would have to be searched for common factors whenever a new generalization is created. Five possible solutions to this problem are outlined in Figure 3-6.

The first possible method for dealing with the multi-source inheritance problem is the one pursued in our example. That is, after each new training instance (F-TREE) is incorporated into the G-TREE, the remainder of the tree must be searched for factors in common with those of any newly created generalizations. If there are any common factors, they must be singled out in order to avoid possible inheritance ambiguities. The resulting representation is both compact and contains only correct generalizations (those that mirror the F-TREE data). Unfortunately the factoring out process is a lengthy one which increases with the size of the G-TREE. For domains with a significant number of training instances, this method is impractical.

Method of implementation	Time	Space	Comments
1-Find common factors incrementally.	slow	efficient	the obvious approach
2-Factor out all primitives, even if not necessarily needed.	average	average	unnecessarily creates G-TREE nodes, obscures some generalizations
3-Use some heuristic to solve ambiguities.	fast	efficient	may produce wrong generalizations and limit possible generalizations
4-Don't allow multi-source inheritance at all.	very fast	efficient	severely restricts possible generalizations
5-Use separate G-TREE nodes for inheritance from different parents.	fast	poor	not too elegant, but practical

This table briefly describes several possible methods for solving the multi-source inheritance problem. The **Time** column indicates the time performance of each method relative to the others. Similarly the **Space** column shows the relative amount of storage space needed for the equivalent amount of information. The text describes each method in more detail.

Figure 3-6: Five possible solutions to the multi-source inheritance problem.

The next method enumerated in Figure 3-6 is a modification of the first approach. Instead of searching the G-TREE for common factors each time a new generalization is created, this solution suggests that all possible factors (F-REL links) be built into separate G-TREE nodes. The use of one node per F-REL link would guarantee a non-ambiguous inheritance process. This method would almost certainly build unnecessary generalization nodes. It would not recognize, for example, that two F-REL links that occur together in all training instances should be represented as a single unit because they would be split up into separate generalization nodes. Although this approach would be substantially faster than the first one mentioned, it would require more memory space for the unnecessary generalizations.

Solution 3 has been mentioned before. It was said that the *union* operation could possibly solve the inheritance ambiguity problem. Using such a technique would

mean that regardless of the number of times a particular F-REL link occurs in the composite of all parent nodes, only one copy of it would be inherited. Union is just one of an infinity of possible methods that could be used. Unfortunately, any heuristic method leaves itself open to the possibility of creating wrong generalizations. Furthermore, some generalizations would be impossible to make using a given inheritance method. For example, if union was used, it would not be possible to represent an F-TREE that inherited one part from parent A and another, identical part from parent B, even though it really does have two of the same part.

For certain domains, solution 4, not allowing multi-source inheritance may be an acceptable alternative, but for most this will not do. However it does suggest a possible pragmatic solution that we have listed as method number 5.

The idea behind this method is to decompose each G-TREE node that is a VARIANT-OF more than one parent into separate nodes that are only a VARIANT-OF one parent each. Then a cluster of these decomposed nodes can be built into one logical unit. Thus, all of the information needed to reconstruct an F-TREE (from a given G-TREE node) is available in each of the nodes in the cluster. Furthermore, all possible paths of inheritance are available by looking at the cluster as a whole (this allows the G-TREE to form the same classification scheme as in method 1). The obvious drawback to this approach is that it is wasteful of memory in that it re-represents some data. However, only the data that is involved in nodes that have multiple parents are duplicated, so this scheme is often a useful approach to the problem.

In summary, the multi-source inheritance problem is a significant one that deserves serious consideration when large G-TREEs are involved. We have listed only a few possible solutions. Other researchers, notably Brachman (see [Brachman 79b] for an introduction, [Brachman 78] for more specifics), have addressed similar problems and devised various ways around it.[2] Nevertheless, it must be kept in mind when deciding on how a hierarchy understanding system should be constructed.

---

[2]Brachman's scheme allows inheritance links to be modified if they would cause a conflict of inherited data. However, his scheme is for non-hierarchically structured objects and is oriented toward generalizing an object's properties not its structure.

4. Conclusions

Many issues that arise in the context of hierarchy understanding are rooted in the somewhat broader arena of learning (generalization). The following are some of the more important questions in generalization (aside from multi-source inheritance) that bear on hierarchy understanding.

What are the differences between incremental and all-at-once type generalizations - the distinction here is between whether the training instances are available for analysis all at one time or whether they trickle into the generalizer one instance at a time. Should all possible generalizations always be made? How can instances with incomplete F-TREEs be incorporated into a G-TREE so that the information already present is not disturbed? (This is a standard occurrence in natural situations.) Can the information present in the G-TREE be used to fill in missing data in incomplete F-TREEs? How can erroneous generalizations be corrected? (See [Lebowitz 82], for example.) How can generalization systems be made robust (in the sense that they don't fail when given input not specific to their domain)?

In addition to these generalization problems, hierarchy understanding requires the solutions to several other ones: what is the best way to incorporate other, non-fundamental, relations into F-TREEs, is it useful to attempt to represent domains with multiple F-RELs within a single G-TREE? Work on RESEARCHER [Lebowitz 83], a program that reads and understands patent abstracts, has given us the need to explore all of the issues (and more) mentioned here. We are developing a representation/generalization system that automatically (and dynamically) classifies representations of the complex physical objects described in disk drive patents.

The formalism presented in this paper (when augmented with the subtraction and substitution operations) is useful for describing a wide variety of problems that arise in hierarchy understanding. In addition, it is easily expanded to take into account new operations that effect the interaction of the F-TREEs with the G-TREE. For example, non-fundamental relations that greatly enhance the meaning of a hierarchy can be added for a more complete picture of an individual F-TREE. Furthermore, generalizations about these relations can be incorporated into the G-TREE alongside the generalizations based upon the F-REL links.

Nevertheless, there is still much work to be done in understanding how learning about hierarchical systems takes place. There are undoubtedly many general principles concerning hierarchies that are yet to be discovered. This knowledge will surely lead to more intelligent computer systems and hopefully to a better understanding of how the human mind works.

5. Acknowledgments

I would like to thank Michael Lebowitz for reading over earlier versions of this paper and for many helpful discussions on hierarchy understanding. Much of the work on generalization presented in this paper was done jointly with Tom Ellman and Larry Hirsch. I would also like to thank John Kender for suggesting the idea of investigating hierarchies in general.

References

[Brachman 78] Brachman, R.J. A structural paradigm for representing knowledge. Tech. Rept. 3605, Bolt Beranek and Newman, Inc., 1978.

[Brachman 79a] Brachman, R.J. Taxonomy, descriptions and individuals in natural language processing. Proceedings of the 17th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, La Jolla, California, 1979, pp. 33 - 38.

[Brachman 79b] Brachman, R.J. On the epistemological status of semantic networks. In N.V. Findler, Ed., *Associative Networks*, Academic Press, New York, N.Y., 1979.

[Carbonell 70] Carbonell, J.R. "AI in CAI: An artificial intelligence approach to computer-aided instruction." *IEEE Transactions on Man-Machine Systems 11*, 4 (1970), 190 - 202.

[Hayes 77] Hayes, P.J. On semantic nets, frames and associations. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, International Joint Conference on Artificial Intelligence, 1977.

[Lebowitz 82] Lebowitz, M. "Correcting Erroneous Generalizations." *Cognition and Brain Theory 5*, 4 (1982), 367 - 381.

[Lebowitz 83] Lebowitz, M. RESEARCHER: An overview. Proceedings of the Third National Conference on Artificial Intelligence, American Association for Artificial Intelligence, Washington, DC, 1983.

[Michalski and Stepp 83] Michalski, R.S. and Stepp, R.E. "Automated Construction of Classifications: Conceptual clustering versus numerical taxonomy." *Pattern Analysis and Machine Intelligence 5*, 4 (1983), 396 - 409.

[Minsky 75] Minsky, M. A framework for representing knowledge. In P.H. Winston, Ed., *The Psychology of Computer Vision*, McGraw-Hill, New York, 1975.

[Mitchell 77] Mitchell, T.M. Version spaces: A candidate elimination approach to rule learning. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, International Joint Conference on Artificial Intelligence, 1977.

[Quillian 68] Quillian, M.R. Semantic memory. In M. Minsky, Ed., *Semantic Information Processing*, MIT Press, Cambridge, Mass., 1968.

[Sacerdoti 75] Sacerdoti, E.D. A structure for plans and behavior. Tech. Rept. 109, SRI International, 1975.

[Simon 81] Simon, H.A. The architecture of complexity. In H.A. Simon, Ed., *The Sciences of the Artificial*, MIT Press, Cambridge, Mass., 1981.

[**Wilensky 83**] Wilensky, R. *Planning and Understanding.* Addison-Wesley, Reading, MA, 1983.

[**Winston 72**] Winston, P. H. Learning structural descriptions from examples. In P. H. Winston, Ed., *The Psychology of Computer Vision*, McGraw-Hill, New York, 1972.

[**Winston 80**] Winston, P. H. "Learning and reasoning by analogy." *Communications of the ACM 23*, 1980, pp. 689 - 702.