COGNITIVE SCIENCE 9, 417–430 (1985)

Comparing Natural and Abstract Categories: A Case Study from Computer Science

BETH ADELSON
Yale University

In order to address the issue of whether abstract and concrete objects are categorized similarly, this paper looks at how computer scientists categorize the common concepts of their field. Using the methodology of Rosch, Mervis, Gray, Johnson, and Boyes-Braem (1976) and of Rosch and Mervis (1975) to identify the categorizations used by computer scientists, this study found that the categorization phenomena reported for concrete objects are quite general, operating over a wider range than was previously thought.

Are abstract and concrete objects categorized similarly? In studies of psychological categories of concrete objects, Rosch and her colleagues have documented two phenomena (Rosch, Mervis, Gray, Johnson, & Boyes-Braem, 1976). They found a basic level of conceptual abstraction and a perception of categorical structure based on family resemblance. To parallel Rosch's experiments, this article reports three studies of categories of abstract objects. Working within a Roschian experimental paradigm, we ask whether computer scientists identify a natural, basic level of abstraction similar to that found for concrete objects? We also ask whether these categories exhibit prototypicality and family resemblance characteristics?

The question of whether abstract and concrete objects are similarly categorized is motivated by the results of Adelson (1981) which suggested that expert programmers chunk single lines of code into conceptual objects. Adelson asked novice and expert programmers to recall a set of 16 lines of programming code which had been presented in random order. Although

Thanks to E. E. Smith, Elliot Soloway, R. Duncan Luce, Roger Brown, Miriam Schustack, and Stephen Kosslyn for their advice and guidance. Special thanks to Jeannine Pinto for her assistance.

Correspondence and requests for reprints should be sent to the National Science Foundation, Room 336, 1800 G. Street, N.W., Washington, DC 20050 where the author is on leave from the Yale Artificial Intelligence Laboratory.

the subjects had not been told that the 16 lines could be organized either conceptually into 3 programs or syntactically into 5 categories according to the control words that they contained, analyses of the order of recall for each group showed that the experts had clustered the lines into complete programs and the novices had clustered the lines according to syntactic categories. This finding suggests that expert programmers regard familiar concepts as abstract objects,[1] and it leads us to ask two questions: (1) Are abstract objects, like concrete objects, classified at several levels of abstraction and are those different levels used in different ways? (2) For a given familiar category of concepts, at a given level of detail, do we find that some members of the category are more representative of the category than others? These questions are relevant to our understanding of categorization, as well as to our understanding of programming, since theories of categorization need to account for the categorization of abstract as well as concrete entities.

PART I: LEVELS OF ABSTRACTION

In the first part of this article we look at abstraction, the "vertical dimension" of classification, to see whether there is a basic level of abstraction at which concepts are classified. The background for this issue is provided by the substantial evidence that a basic level of classification exists for concrete objects. Murphy and Smith (1982) and Rosch et al. (1976) define this level in the following way: when subjects are asked to list the features of an object at each of three increasingly abstract levels of classification, the basic level of taxonomy is the most inclusive level at which there is the greatest increase in the number of features listed. For example, when subjects were asked to list the features of members of the categories FRUIT, APPLES and GRANNY SMITH APPLES, if on the average, they were to list 7 features for FRUIT, 12 features for APPLES, and 15 features for GRANNY SMITH APPLES, then APPLE would be considered the basic level of conceptualization for that familiar object. This means that there is one level of classifying objects which has a moderately, but not overwhelmingly, rich representation. This moderately rich representation is termed basic because it is found to be the most available representation (i.e., first accessed) when recognizing an object (Rosch et al., 1976). It is also the first learned and most likely to be coded linguistically (i.e., represented as a one-word label as in the case of APPLES rather than GRANNY SMITH APPLES).

The first experiment presented here used Rosch et al.'s (1976) feature listing criterion and methodology in an attempt to replicate those results for abstract, computational concepts.

[1] These objects are abstract in the sense that they do not refer to physical objects.

EXPERIMENT I

Method

Subjects. Twenty-four expert subjects: Teaching fellows and advanced graduate students in Computer Science.

Stimuli. According to Knuth (1973), two classes of concepts can be distinguished in traditional structured programming: algorithms and data structures. Within each of these classes there exist two subclasses on which research has focussed (with regard to such issues as best and worst case behavior). The subclasses of algorithms are *SORTING* and *SEARCHING;* the subclasses of data structures are lists and trees. *SORTING, SEARCHING, LISTS,* and *TREES* were chosen as the four middle-level categories since they were well known and they possessed superordinate and subordinate classifications, which are necessary for the procedure used here. Three common members of each subclass were used as stimuli at the subordinate level of classification. *ALGORITHM* and *DATA STRUCTURE* were used as the superordinates (Knuth, 1973). The set of materials and their relationships are listed as follows.

Design and Procedure. On three separate days, subjects were asked to list the features of either one superordinate, two middle (one from each superordinate) or four subordinate items (one from each middle category). In the explanation of each day's task, subjects were given examples, at the appropriate level, of the kind of features that might be listed for *VEHICLE,* for *BICYCLE* or for *TEN-SPEED BICYCLE.* They were told to use this as an analogy to the current stimulus items.

On the day before seeing the subordinate level names, subjects completed a paired-associate learning task in order to ensure that all subjects knew which name referred to each concept. In the learning task, names were paired with *C* language implementations of the concepts. Subjects were shown each of the names with its implementation. After seeing the full set they were then shown each of the implementations and asked to give its name. Subjects were corrected if they responded incorrectly. Naming continued until subjects could name the whole set twice through, without error. The implementations were designed to be natural and clear exemplars of the concepts. Subjects were tested individually via computer terminals. The items used at each level were counterbalanced over subjects so that items were presented equally often across subjects. A partial latin square design was used to order the level of abstraction to which the subjects responded.[2]

[2] The details of the design differ from those used by Rosch et al. (1976) since large numbers of subjects who are experts in computer science are difficult to come by.

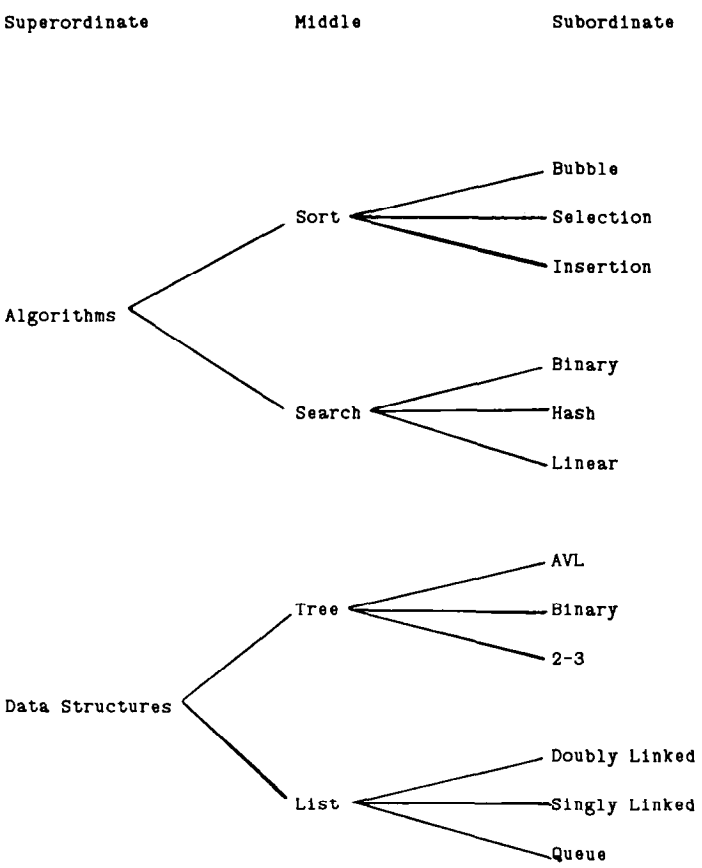

Figure 1. The 18 Stimulus Items and Their Relationships

Results and Discussions

The hypothesis being investigated here is whether computational concepts have a basic level of abstraction. This hypothesis will be supported if we replicate the results of Rosch et al. (1976), finding that a level of classification exists such that more inclusive classifications elicit few attributes and less inclusive classifications elicit few additional attributes.

The mean numbers of features listed by our subjects at each level of classification was: Superordinate, 4.9; Middle, 6.9; and Subordinate, 7.1. To test the basic level hypothesis, Rosch et al. (1976) used two *t* tests. First, they determined that the number of features listed at the middle level was significantly greater than the number of features listed at the superordinate level. Next, they determined that the difference between the number of fea-

tures listed at the subordinate versus the middle. We found, as did Rosch: (a) the number of features listed at the middle level was significantly greater than the number of features listed at the superordinate level ($t(41) = 6.25$, $p < .001$), and (b) the difference between number of features listed at the subordinate and middle levels was found to be significantly less than the difference between middle and superordinate ($t(37) = 1.98$ $p < .05$). Both results support the hypothesis that abstract concepts show the same differences in richness of representation as concrete objects and that this difference can be used to identify one level as basic. Thus, when the basic level is defined in the same way for both concepts and objects, a basic level of representation can be identified for computer scientists' concepts.

As mentioned before, one property of the basic level of classification (which suggests the name "basic") is that during recognition, items are first thought of in terms of their basic-level classification. That is, Rosch et al. (1976) found subjects were able to verify that objects were members of their basic-level categories faster than they could verify members of their subordinate and superordinate categories. For example, when shown a picture, they were quicker to affirm that the picture was of an apple than of a fruit or a Granny Smith apple. The next experiment looks at this characteristic of the basic level for programming concepts.

EXPERIMENT II

To further support the existence of a basic level for programming concepts we asked whether this level of classification has the same behavioral characteristics as it has for concrete objects?

Rosch et al.'s (1976) verification task, described above, provided a test of the hypothesis that the basic level classification was the one which was used first to recognize familiar items. Another, more direct, test of this hypothesis can be provided simply by asking subjects to classify programming concepts in the way that they feel is most natural. This was done in Experiment II.

Method

Subjects. Six experts, different from those in Experiment I, served as paid volunteers.

Stimuli. The 12 implementations of the concepts used in the first experiment were used here as stimulus materials. The relationships among the set of concepts are the same as in Experiment I. Each concept can be classified

in three ways: as a member of a basic-level category, as a member of its subordinate category, and as a member of its superordinate category.

Design and Procedure. Subjects were given 12 sheets of paper, each sheet having one of the set of 12 *C* language implementations on it. They were asked to sort the set of 12 items into the organization that they felt was most natural for the set. The experimenter recorded these classifications and then returned the sheets to the subjects. They were then asked to re-sort their first classifications; either into more or less inclusive categories. This sorting was also recorded. The subjects were once again given the set, but in the grouping that they had first constructed. The process was then repeated for the level of inclusivity not yet tested.

The order of sorting was: natural, more inclusive, less inclusive for half of the subjects and natural, less inclusive, more inclusive for the other half of the subjects. The materials were presented in a random order before the first sorting and returned to subjects in the order established by the first sorting for subsequent sortings. Subjects were told that the groupings they made could contain from 1 to 12 items, that there was no "correct" size for any particular level, and that the size of the groupings should only reflect what they thought was a natural organization at each level.

Results and Discussion

The hypothesis of this experiment, that subjects most naturally organize programming concepts in terms of their basic level of classification, would be supported if subjects chose the classification identified as basic in the last experiment as the first, most natural way to classify or sort items.

The results of the sorting task used here suggest that subjects do find that the basic level categories form the most natural classification for these concepts. Considering the basic level to be the level suggested by the results of the first experiment, the set of twelve programs form four basic categories: *SORTS, SEARCHES, LISTS,* and *TREES.* Therefore, our hypothesis predicts that, since six subjects performed this task, 24 basic level categorizations would be formed, across subjects. The data supported our hypothesis; 92% of the categorizations were basic categories (using a sign test, $p < .001$). In the more and less incusive sortings, we again find categories which correspond to the classifications used here and established by theoretical computer scientists. However, these classifications are more variable and less cohesive than the basic level sortings. Subjects listed 75% of the 12 anticipated superordinate categories and 31% of the 72 anticipated subordinate categories ($p > .10$ by a sign test in both cases).

Summary & Conclusions for Part I

To summarize, the results of the first two experiments suggest that a basic level of conceptualization exists for the concepts frequently used in computing. We find, as did Rosch et al. (1976), that a basic level can be identified by looking at differences in the number of features associated with varying labels for a given item and that this level can be used to predict the way in which a concept will most readily be categorized.

It is possible that *what* level is considered basic may change with context, but this may be true for concrete as well as abstract items. For example, a programmer who has been writing a portion of a program that will sort a data set and has been dealing with issues of worst versus average case behavior, may temporarily treat sorting as a superordinate level, treat a particular algorithm as the basic level, and treat alternative implementations as the subordinate level. It is also possible that a programmer who is trying to comprehend a large program that someone else has written might consider larger problems than *SORTING* or *SEARCHING* to be basic in that context. Although the basic level may shift, the experiments presented here still suggest that there is a similarity between the way in which abstract and concrete objectives are classified.

PART II: PROTOTYPICALITY

EXPERIMENT III

In the first part of this paper we looked at the vertical dimension of categorization. Here we look at prototypicality, the "horizontal dimension," in order to further explore the similarities between the classification of abstract and concrete objects. Towards this end we will ask, for a category of concepts, at a given level of detail, whether some members of the category are more representative than others? In looking at categories of concrete objects, Rosch and Mervis (1975) found that, for any given level of abstraction, the members of the category vary in the degree to which they are judged to be good exemplars of the category. They also found that the total frequency of a category member's features predicted how good an exemplar it would be judged to be. If we find that abstract categories have the same kind of variations in prototypicality as do concrete categories, and if we also find that in both cases subjects' judgements about these variations can be predicted by the features of the category members, then we have some evidence that both abstract and concrete objects are similarly categorized.

Method

Subjects. Ten expert programmers fluent in the programming language *C* were used as subjects in this experiment. The subjects were either professional programmers or advanced graduate students in Computer Science. All subjects had had several years of programming experience, as well as formal knowledge of computer science.

Stimuli. An initial group of expert subjects (different from the 10 mentioned above) was asked to list all of the sorting algorithms with which they were familiar. The eight most frequently listed were implemented in the programming language *C*. The algorithms used are listed here and are described in Appendix 1: Selection Sort, Insertion Sort, Binary Insertion Sort, Bubble Sort, Shell Sort, Merge Sort, Quick Sort, and Heap Sort.

Procedure. Three tasks were used in this experiment; the first two follow the procedure originally used by Rosch and Mervis (1975) to look at the structure of categories of artifacts. The tasks consisted of:

1. *Feature Listing Task.* Subjects were asked to list the features of each of the algorithms. They were given examples of the kind of features that might be listed for a bicycle, such as "has wheels," and "you can ride on it." They were then asked to perform the analogous task, by listing properties such as aspects of the behavior of the algorithm. Subjects were shown the implementations of the algorithms rather than just given the names to ensure that there was no confusion as to each name's referent.

   The implementations of the algorithms had been designed to be the most natural representations of each one so that subjects could actually list the features of the algorithms rather than of the particular implementation. This distinction was stressed in the instructions.
2. *Rating Task.* For each of the algorithms, subjects were asked to rate: (1) How prototypical it was. (2) How psychologically simple or complex it was.[3] (3) How frequently they had seen that algorithm. (4) How frequently they used that algorithm. (5) How well they understood it. (6) How recently they had learned it.
3. *Similarity Judgements.* Subjects were asked how similar each algorithm was to every other algorithm.

The feature listing and the rating tasks were counterbalanced for order and then given to all subjects. This was followed by the similarity judgement task. Subjects were tested individually. Stimuli were presented via booklets

[3] We made a distinction here between perceived or psychological simplicity and computational complexity.

in which subjects both saw the materials and the questions and wrote their responses.

Results and Discussion

The first goal of this experiment was to see if the processes that operate in the categorization of concrete objects also operate in the domain of abstract concepts. If we find that abstract categories have the same kind of structure as concrete categories, and if we also find that this structure leads to the same kind of judgements about category members in both cases, then we have some evidence that the same kind of processes operate in both cases.

Recall that when looking at concrete categories, Rosch and Mervis (1975) found that the members of the category varied in their degree of prototypicality. They also found that a category member's prototypicality was predicted by the total frequency of its features. An analysis of variance showed that here the category members also varied in their rated prototypicality ($F(7,56) = 6.01$, $p < .01$). Table I which follows shows the average prototypicality of each of the eight algorithms.

TABLE I
Results of the Prototypicality Rating

Algorithm	*Average Prototypicality*
Selection	6.0
Insertion	5.8
Bubble	5.7
Binary Insertion	4.7
Merge	4.0
Quick	3.8
Shell	3.6
Heap	3.1

An analysis of variance was done to determine the effect on the prototypicality rating of the order in which the rating task and features listings were performed. No significant difference was found between subjects who had done the rating before the feature listing and those who did it after ($F < 1$).

A family resemblance score was computed for each algorithm. This score was the number of times each algorithm's features were mentioned averaged[4] across the sum of the number of features in the algorithm, the

[4] In calculating a family resemblance score Rosch and Mervis (1975) use the sum rather than the average. In their work, all items had the same number of features and so the sum and the average produce identical correlations. Here the number of features can vary from item to item and so using the sum could potentially have the effect of inflating family resemblance scores for atypical category members which may have many low-frequency features. As a result we present the family resemblance score calculated using the average although the pattern of results is unchanged here when the sum is used.

number of subjects, and the number of category members. The family resemblance score and the prototypicality rating of each algorithm were then correlated ($r = .92$, $p < .05$). So we find, as did Rosch and Mervis (1975), that the prototypicality of a category member is predicted by the total frequency of its features. To investigate the presence of other factors underlying the relationship between prototypicality and family resemblance a partial correlation was computed. Partialling out the effects of frequency of use, frequency of encounter, order of learning, and complexity increased the correlation to $r = .95$ ($p < .05$) suggesting that these factors were not artifactually inflating the correlation between prototypicality and family resemblance. An analysis of variance was done on the effect of the order of having done the feature listings on the family resemblance scores. No significant difference was found as to whether subjects had done the listings before the prototypicality rating as compared to after ($F < 1$).

Since this abstract category has the same variation in the judged prototypicality of the category members as did Rosch and Mervis' (1975) concrete categories and this variation is predicted by the same measure in both cases, we have further evidence that both abstract and concrete objects are similarly categorized.

In the feature rating task, subjects listed numerous features. However, this does not necessarily mean that the features actually underlie the subjects representations, or that they are used in concept identification. Unless we find that the features listed for the algorithms actually predict something about the algorithms, we would not want to say that this is the case. However, the subjects' judgements of the similarity among the algorithms provides us with evidence that the features which subjects listed have some reality. Additionally, the similarity judgements suggest that there is a clearly discernable structure to this categorization.

In Figure 2 shows the results of a two-dimensional scaling solution obtained from the subjects' judgements of the similarity among the algorithms. Algorithms rated as similar to one another cluster together; the distance between them is inversely related to their similarity. We see that the four algorithms with the lowest subject ratings on perceived simplicity cluster in the first quadrant with positive y-values.[5] In addition, the four algorithms with the highest-rated perceived simplicity cluster in the third and fourth quadrants with negative y-values.[6] This suggests that perceived simplicity, the most frequently mentioned of the abstract features, forms a basis for organizing the spatial model. It forms the vertical dimension in Figure 2. We also see that the two recursive algorithms cluster in the third

[5] The circles surrounding algorithms clustering together were obtained from a hierarchical clustering of the subjects' similarity ratings.

[6] There is an interesting inverse relationship here between perceived simplicity and computational complexity.

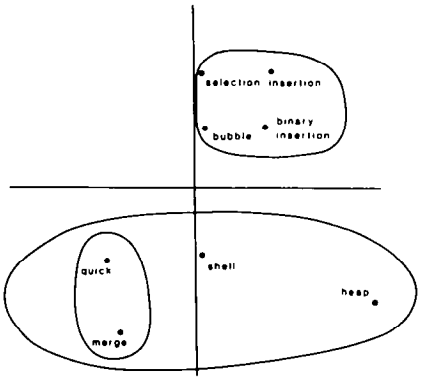

Figure 2. The hierarchical clustering embedded in the two-dimensional scaling solution for interitem similarity.

quadrant with negative x-values, while the six iterative algorithms have positive x-values. This suggests that whether a program was iterative or recursive, the two most frequently mentioned concrete features, form the second basis for organizing this space by providing a horizontal dimension. The spatial models of the relationships among the algorithms suggest that the features that were listed do play a role in the representation of both the individual algorithms and the category that they comprise. The results of the scaling also suggest that computer scientists form intuitively structured, orderly categories for familiar concepts.

Conclusions

The results of these studies suggest that both abstract and concrete concepts are similarly categorized. This does not weaken the findings of such studies as Murphy and Smith (1981) or Jolicoeur, Gluck, and Kosslyn (1984) which suggest that the categorical representations of concrete objects are perceptual. In fact it suggests that these classification phenomena are quite general, operating over a wider range of concepts than was previously thought.

The results of both the feature listing and the multidimensional scaling suggest that expert programmers have representations of common concepts which contain both abstract and concrete features. Unfortunately, we have no clue as to how the abstract features are derived from the concrete features present in the code. For example, although there may be obvious concrete evidence that selection sort uses an array, there will not be direct perceptual evidence that the procedure divides the array into sorted and unsorted parts and then proceeds to extend the sorted part by picking elements from the unsorted part. Nevertheless, there is this pervasive tendency for experts in a domain to use concrete information in order to form a more abstract con-

ceptual representation for themselves (Adelson, 1981, 1984; Chi, Glaser, & Rees, 1981).

Novice programmers were not studied because they were not sufficiently familiar with the classes of concepts investigated. This is not to say that similar categorization processes might not exist for novices even if the categories and their exemplars did differ. Chi, Glaser, and Rees (1981) suggest that novice as well as expert physicists also have a basic level of conceptualization, but that the basic level is less abstract for novices than for experts. This point touches on the interesting but unresolved issue of the interaction between general and domain specific problem-solving skills.

APPENDIX 1

Selection Sort

Selection Sort first finds the smallest element and moves it to the front of the list by swapping it with the first element. Then the smallest of the remaining elements is moved to the second position. This process continues until the entire list is in order.

Insertion Sort

Insertion sort creates an ever longer sorted sublist at the beginning of the original list. At each iteration one new element is added to the sorted sublist by selecting out the first element of those not yet sorted and then moving all larger elements in the sorted section of the list one position to the right to make room for the new element in its correct place.

Binary Insertion Sort

Employs the same method as insertion sort except that the position for inserting each new element into the sorted part of the list is found by a binary rather than a linear search.

Bubble Sort

On each iteration bubble sort creates an ever larger sorted sublist at the end of the list. This is done by moving the element that at the beginning of iteration is the largest in the unsorted part of the list to the beginning of the sorted sublist. This is achieved by comparing each successive element in the

unsorted part of the list to each of the ones following it and exchanging them if the former is larger than the latter.

Shell Sort

Shell sort is similar to bubble sort except that the early iterations compare and exchange widely separated, rather than adjacent elements.

Merge Sort

Merge sort splits the list into two equal sublists and then recursively splits, sorts, and merges each sublist by picking the elements in each split sublist in order.

Quick Sort

Quick sort begins by choosing an arbitrary "pivot element" and then separating the list into two sublists, one containing elements which are smaller than the pivot and one containing elements that are larger. The left and right sublists are themselves sorted recursively in this way.

Heap Sort

Heap sort treats the list as if it were a balanced binary tree. First, by repeatedly exchanging, if necessary, the largest child of each node with the parent, the tree is given the "heap" property; each node is greater than its two children. This has the effect of bringing the largest element to the root of the tree. This element is then exchanged with the last element of the list, and the heap property is then restored for the tree which now consists of all the elements except the last. Now the largest remaining element is at the root of the tree. It is moved into place and the process is repeated until the list is in order.

REFERENCES

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory and Cognition, 9,* 422–433.

Adelson, B. (1984, July). When novices surpass experts: The difficulty of a task can increase with expertise. *Journal of Experimental Psychology: Learning, Memory and Cognition.*

Chi, M., Glaser, R., & Rees, E. (1981). Expertise in problem solving. In *Advances in the psychology of human intelligence* (Vol. 1). Hillsdale, NJ: Erlbaum.

Jolicoeur, P., Gluck, M., & Kosslyn, S. M. (1984). Pictures and names: Making the connection. *Cognitive Psychology, 16,* 243-275.

Knuth, D. E. (1973). *The art of computer programming.* Reading, MA: Addison-Wesley.

Murphy, G., & Smith, E. E. (1982). Basic-level superiority in picture categorization. *Journal of Verbal Learning and Verbal Behavior, 21,* 1-20.

Rosch, E., & Mervis, C. B. (1975). Family resemblances: Studies in the internal structure of categories. *Cognitive Psychology, 7,* 573-605.

Rosch, E., Mervis, C. B., Gray, W. D., Johnson, D. M., & Boyes-Braem, P. (1976). Basic objects in natural categories. *Cognitive Psychology, 8,* 382-439.