Technical Report 853

Learning Shape Descriptions: Generating and Generalizing Models of Visual Objects

Jonathan Hudson Connell

MIT Artificial Intelligence Laboratory

This blank page was inserted to preserve pagination.

Learning Shape Descriptions:

Generating and Generalizing Models of Visual Objects

by

Jonathan Hudson Connell

Abstract:

We present the results of an implemented system for learning structural prototypes from gray-scale images. We show how to divide an object into subparts and how to encode the properties of these subparts and the relations between them. We discuss the importance of hierarchy and grouping in representing objects and show how a notion of visual similarity can be embedded in the description language. Finally we exhibit a learning algorithm that forms class models from the descriptions produced and uses these models to recognize new members of the class.

Revised version of a thesis submitted to the Department of Electrical Engineering and Computer Science on August 27, 1985 in partial fulfillment of the requirements for the degree of Master of Science.

Acknowledgements

Many people contributed to this work by reading drafts of various papers and by helping me prepare for my Oral Exam. In particular, thanks to Patrick Winston who got me started on this project, and to Mike Brady for seeing it through. Thanks also to Steve Bagley, Margaret Fleck, Anita Flynn, and Scott Heide for intellectual companionship and to Boris Katz for encouragement along the way. Special thanks to Sue Berets who patiently examined numerous forks over dinner.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's Artificial Intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505, the Office of Naval Research under contract number N00014-77-C-0389, and the System Development Foundation.

Table of Contents

1.1. Introduction

This research has two goals. The first is to generate symbolic descriptions of objects from video images. The second is to generalize these descriptions into models. The ability to generate and generalize descriptions is a prerequisite for recognition. Figure 1 shows how the system is used. We start by presenting the two examples at the left. The system forms semantic networks for each image and generalizes the descriptions into a model which covers both the examples. We then present the system with the test case at the bottom and ask whether it is an instance of the model. The system produces an answer by comparing the network for the test case with the model it created earlier.

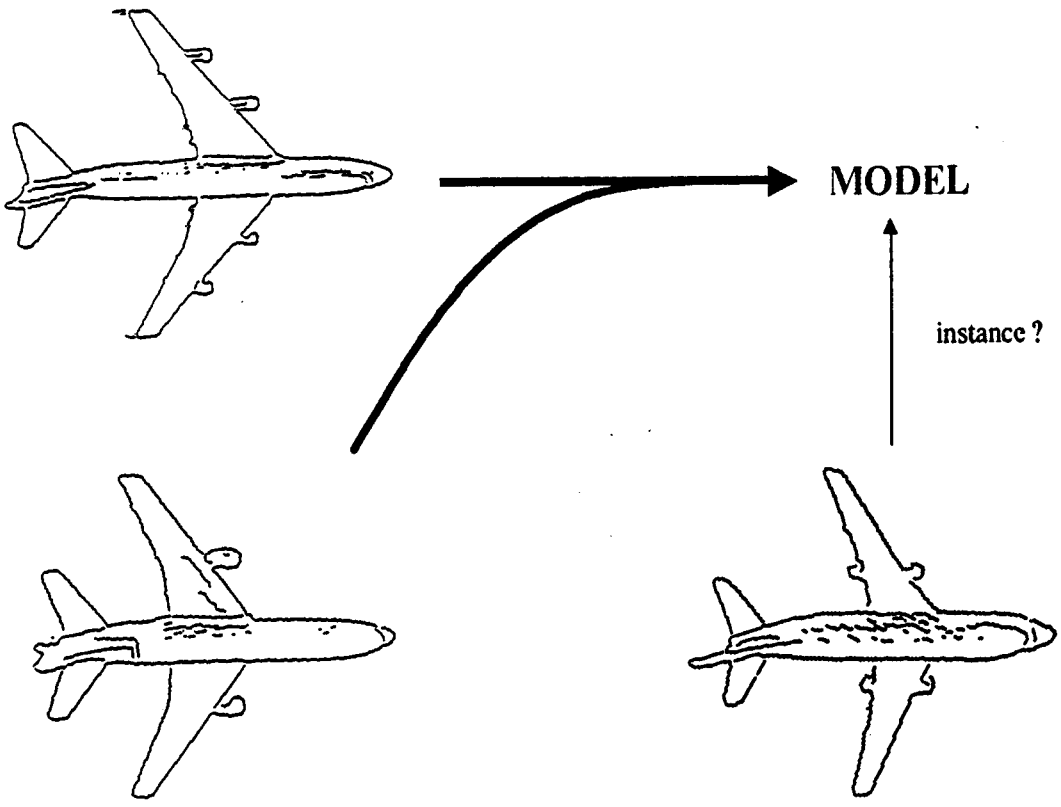


Figure 1-1. The networks for the two airplanes shown at the left are generalized to form a model. Using this model the system can determine whether an object is an airplane or not.

In accomplishing these goals, this thesis makes several interesting contributions to the fields of vision and learning. Among these are:

- a partial theory of what we see when we look at an object. Specifically we conjecture that objects are broken into parts which obey certain restrictions.
- a partial theory of visual similarity – why two objects look alike. This is based on several types of structural abstraction.
- an argument that the syntax of a representation should reflect its semantics. We also show how to represent visual objects in accordance with this principle.
- a technique called Gray coding that allows simple feature dropping to accomplish the effects of many traditional induction heuristics.
- results from a fully implemented system that works on many examples. Starting from gray-scale images, the system learns highly detailed class models and subsequently uses these models for recognition.

1.2. Symbolic Descriptions

The vision system is built on top of the Smoothed Local Symmetries program of Brady and colleagues [Brady and Asada 1984, Heide 1984]. A block diagram of the entire system is shown in Figure 2. Brady's program takes the gray-scale video image, finds local symmetries in it, and then feeds the digested version to our new segmentation and description modules. The segmentation program breaks the image into region primitives and then sends the result to the description module which forms an appropriate semantic network to describe the parts and their relations.

Brady's program, the basis for the vision system described here, has three internal stages. First, the program finds the edges in an image using an edge-finder developed by Canny [1983]. Next the bounding contour of the shape is extracted and approximated by circular arcs [Asada and Brady 1984]. Finally the program looks for local reflectional symmetries between these contour segments [Brady and Asada 1984]. The final output of Brady's program is shown in Figure 3a.

The construction of the next two portions of the vision system is described in detail in this thesis. The first of these parts, the segmentation program, computes parameters such as length, aspect ratio, area, and orientation for each of the symmetries found. It then joins symmetries which are different sections of the same primitive region. Finally, it chunks the image into a collection of non-overlapping

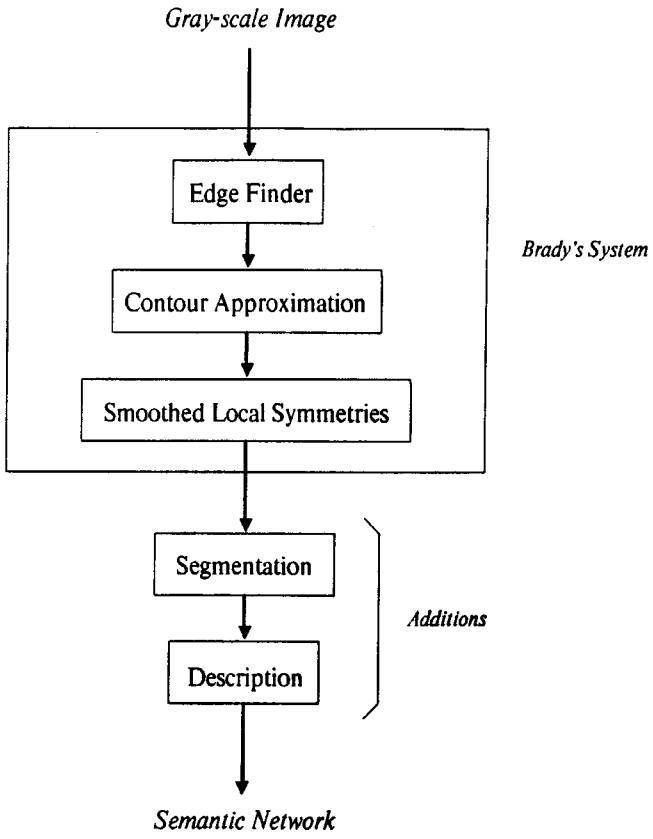


Figure 1-2. The vision system. The edge finder was built by Canny and the contour approximation and Smoothed Local Symmetries programs were written by Brady and Asada. This thesis describes the construction of the segmentation and description modules.

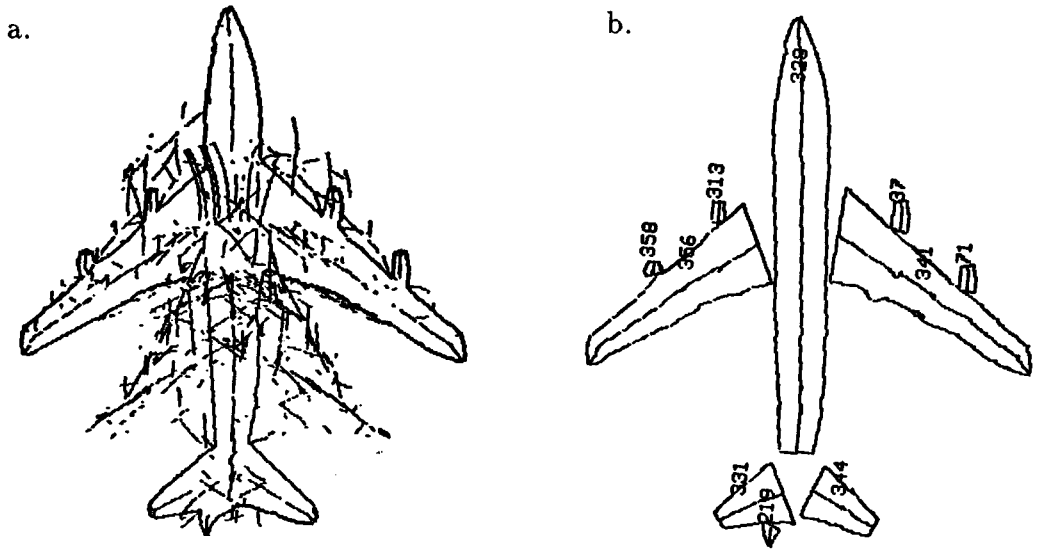


Figure 1-3. Segmenting an object. a. The Smoothed Local Symmetries found in an image of a 747. b. The subshapes of the 747.

pieces based on the extended symmetries found. A typical segmentation produced by the program is shown in Figure 3b.

The second portion, the description module, computes symbolic descriptors for the shapes of each of the pieces found by the segmentation program. It also determines which pieces are joined together and generates a summary of exactly how the pieces are joined. Finally these two types of information are combined in accordance with various rules of abstraction to generate a semantic network for the object. Figure 4 shows a simplified version of such a description.

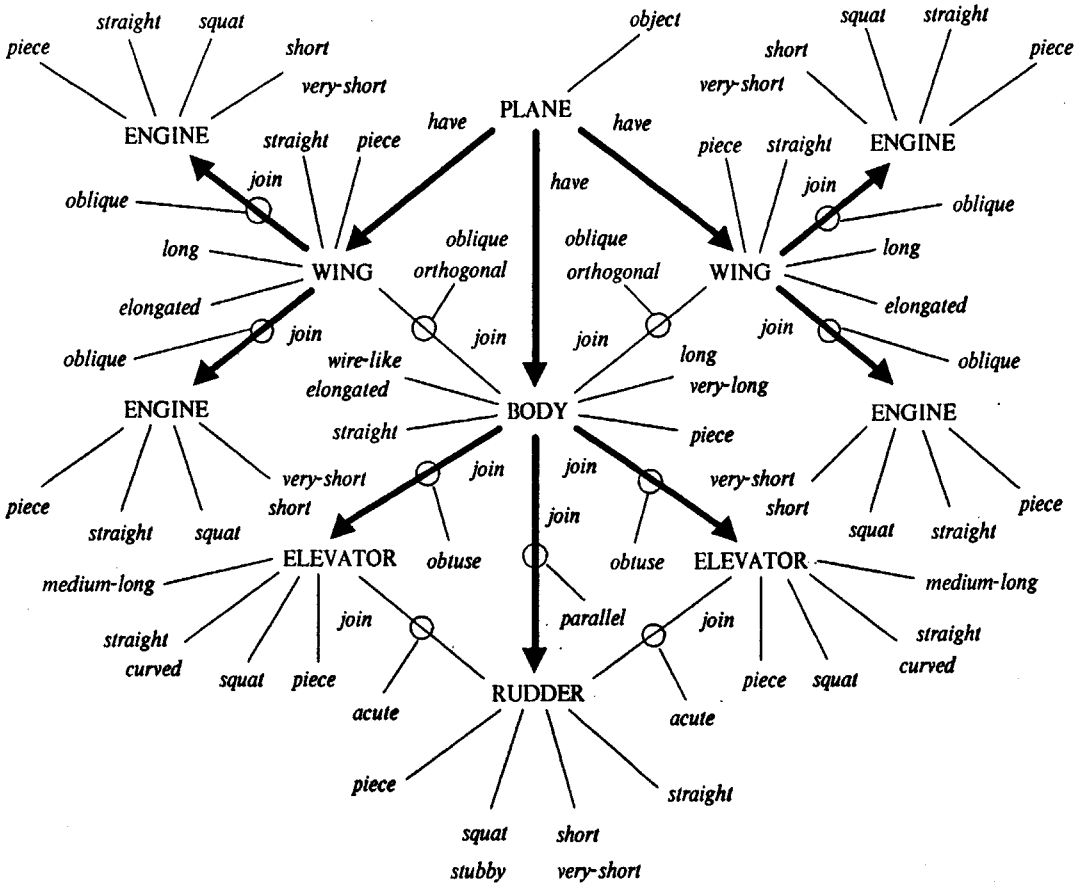


Figure 1-4. A portion of the hierarchical semantic network for the 747. Information about the edges of regions has been omitted.

The descriptions generated satisfy several properties required of a good representation [Marr and Nishihara 1978, Brady 1983]. First of all, a representation must be *efficiently computable* from the data available. The system presented here has been

implemented and generates descriptions in a reasonable amount of time (15 minutes start to finish on a 300 x 400 gray-scale image), hence it qualifies on this count. Second, a representation must be relatively *insensitive to noise.* Segmentation, the cornerstone of the system, is highly stable since it is based on Brady's SLS which was designed to tolerate relatively noisy object boundaries. The system is also insensitive to small variations in the parameters of regions, thanks to special techniques for representing ranges and geometric structures. Thus, similar images produce similar semantic nets. Finally, a representation must be *complete.* To demonstrate that the system captures all the relevant information in an image we have built a program which takes a semantic net and draws the corresponding object. This programs produced the pictures in Figure 5 from the descriptions in Figure 4 and Figure 6.

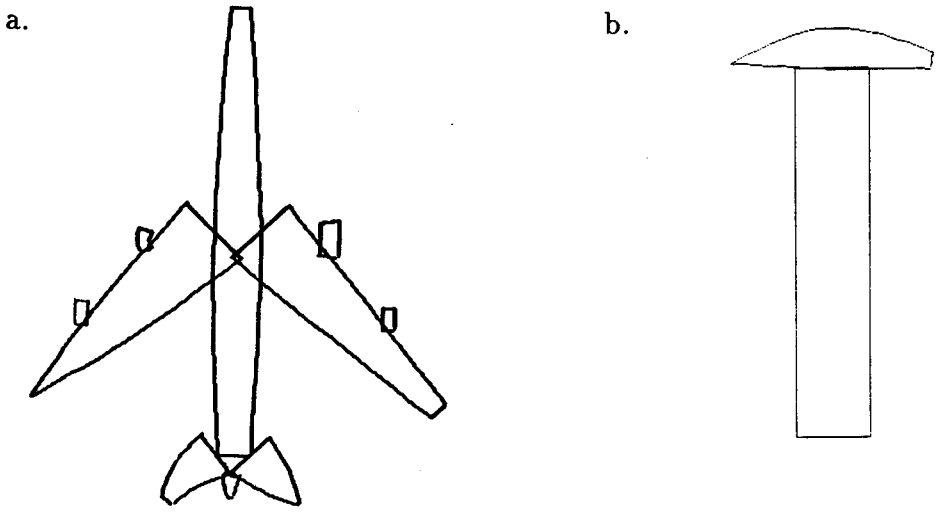


Figure 1-5. Objects reconstructed from their semantic nets. a. The 747. b. A tack hammer.

1.3. Template Formation

The learning program takes the semantic networks produced by the vision system and generalizes them to form models which can be used as recognition templates. The initial template is made by copying the first example's network. Then, as new images are presented, this template is modified to cover all the images which are members of the class which it specifies, and to exclude all the non-members.

The algorithm for modifying the template is similar to the method used by Winston [1970, 1982]. Suppose a new example is presented which almost matches the current template. Since the template should have *exactly* matched this example, the

differences between the template and the example are deemed acceptable variations. Therefore, the template is modified by removing the conditions that were not exactly matched by the example. Now suppose, instead, that we present a *non-example* which almost matches the current template (a *near-miss*). In this case the differences between the non-example and the template are definitely important. Therefore, the template is modified by requiring that future instances MUST have any of the features of the template lacking in the non-example, but MUST NOT have the extra, unmatched features of this non-example.

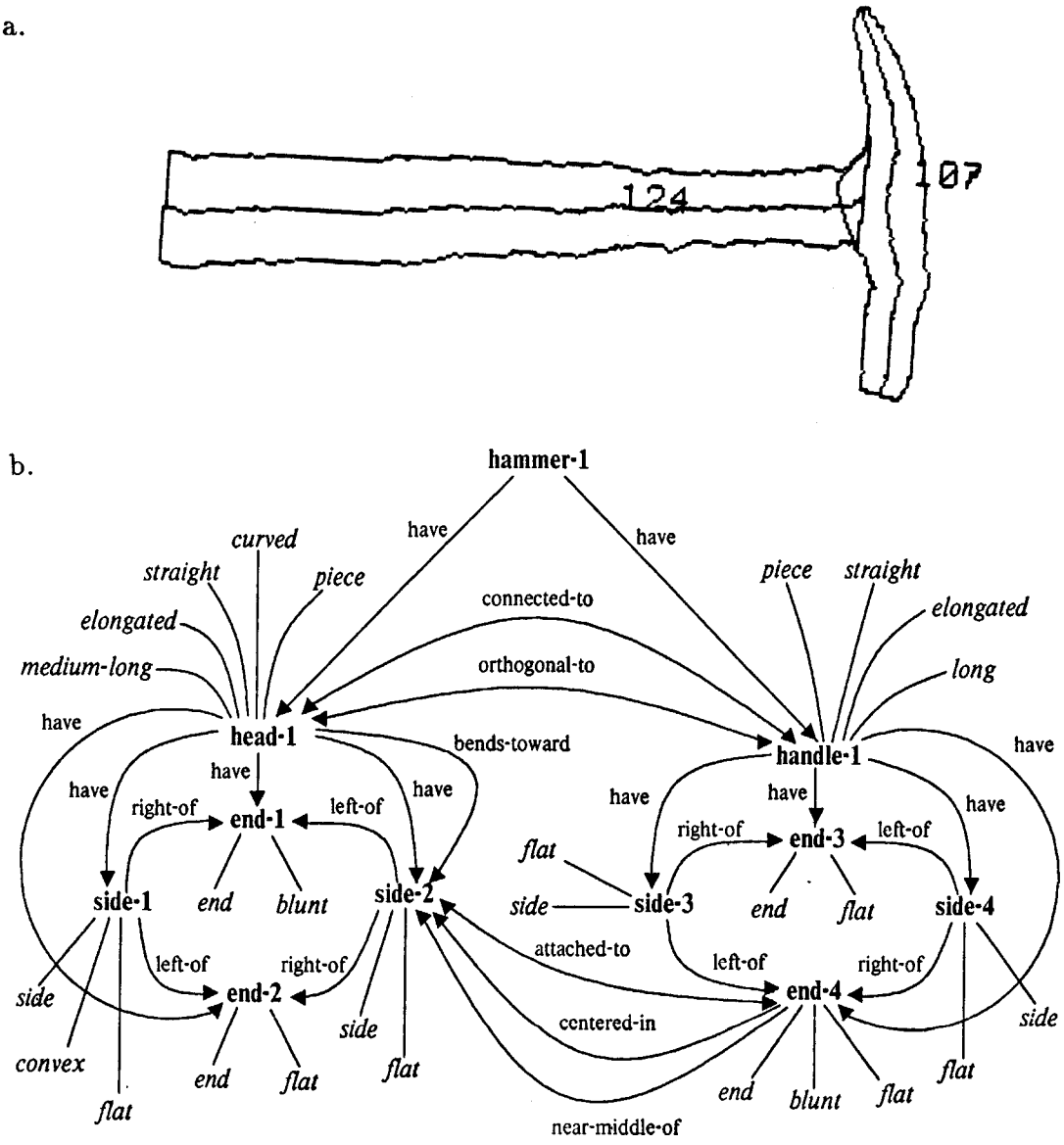
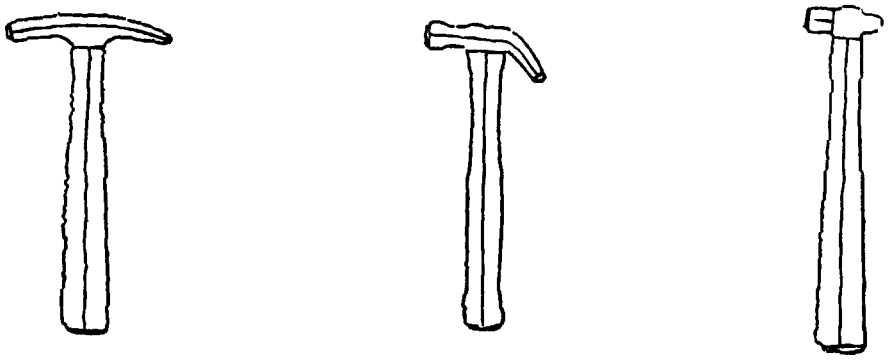


Figure 1-6. Generating descriptions. a. A segmented tack hammer. b. The full semantic net for the tack hammer which is used as the initial model of a hammer.

a.



b.

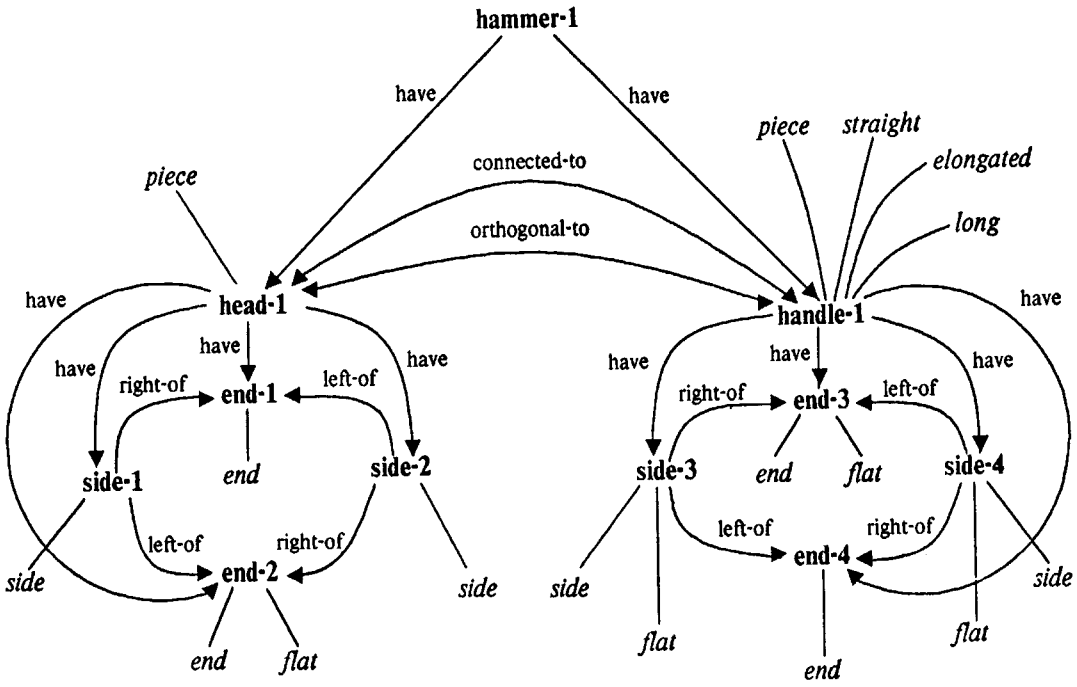


Figure 1-7. Generalizing descriptions. a. The hammers presented to the system as examples. b. The concept learned from the three examples.

Figure 6 shows the full semantic network generated by the vision system for a tack hammer. This becomes the initial template for the *hammer* concept. Figure 7 shows the template after two more examples have been presented. Note that the resulting description only partially specifies the size and shape of the head. This is because the head of the ballpein hammer is much shorter than the heads of the others

while the head of the claw hammer is much more curved than either of the other two. This leads to a template which makes no restriction on the length of the head or how curved it should be. Note also that the specification of how the head joins the handle has been relaxed. This is because the handles of both the claw hammer and the tack hammer are connected to the side of the head while the handle of the ballpein hammer is connected to the end of its head.

The learning algorithm presented goes beyond Winston's program in three major ways. First, it relies less on the teacher to choose appropriate examples and presentation sequences. Second, it handles fully disjunctive concepts through the use of *non-models*. Third, it incorporates domain-specific knowledge in its matching and generalization routines. These three points will be discussed in more detail later in the thesis.

1.4. Pointers

The rest of this thesis describes the theory behind our implemented system in more detail. In particular:

- Chapter 2 discusses why we segment objects into pieces and how this is done. This chapter also contains a section showing the results of the segmentation program.
- Chapter 3 explains the language used to represent shapes and the types of visual abstraction employed by the system. Some sample descriptions produced by the system are presented at the end.
- Chapter 4 describes how models are produced and refined, and how the system recovers from over-generalization.
- Chapter 5 shows the system in action and includes several worked examples of the learning procedure.
- Chapter 6 summarizes the abilities and limitations of the system and suggests some possible applications. It closes by proposing several interesting extensions that could be made to the current system.

2.1. Issues

For the system to *learn* about shapes it must first be able to *perceive* them. In this chapter we describe the vision system we have built and the descriptions it produces. Our approach is based on two principles:

- An object should be broken into regions.
- Each region should correspond to an intuitive piece of the object.

Segmenting an object into parts helps us satisfy several of the criteria for a good visual representation [Marr and Nishihara 1978, Brady 1983]. First, a visual representation must be *rich* in the sense that it preserves all the important facts about an object. By segmenting an object into parts we can obtain a description which is richer than, for example, feature sets. In a feature set shape representation, objects are described using sets of global parameters such as their bounding box, center of mass, number of holes, etc. [Perkins 1978, Ballard and Brown 1982]. The main advantage of the feature set approach is that it is relatively insensitive to noise in an image. However, not only are feature sets insensitive to noise, they are also insensitive to important detail in the image. As illustrated in Figure 1, many shapes which are perceptually quite different yield the same description. Conversely, the set of parameters representing an object does not contain enough information to allow us to reconstruct it. The description { *aspect-ratio = .33, area = 18, holes = 1, center = (0,0)* } could refer to any of the objects in Figure 1. Using parts we can create a better descriptions of these objects such as "a rectangle that has a hole in it and sprouts two bent arms opposite each other".

A good representation should also make important facts *explicit*. For instance, the relative length of a protruberance like the blade of a screwdriver is often important and therefore should be made explicit. Some schemes such as representing a shape by its bounding contour [Freeman 1974, Ballard 1981, Hoffman and Richards 1982], contain the required information but not in an explicit form. Figure 2a shows a screwdriver and Figure 2b shows the list of corner points and the chain code for the image. To determine that the blade is the same length as the handle requires quite a bit of computation in either contour representation. The curvature and elongation of a part are also hard to determine as is the distance between two points

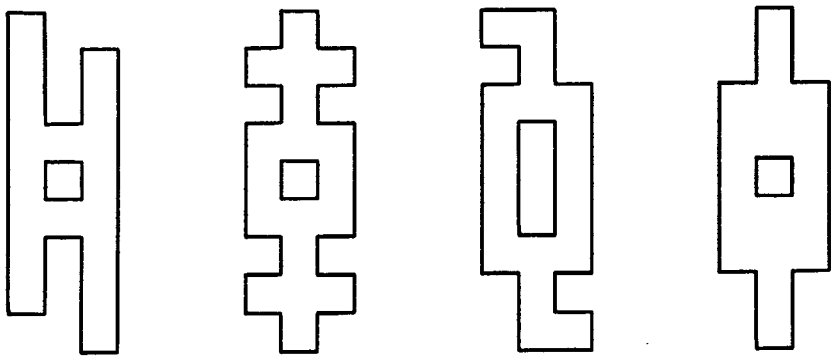
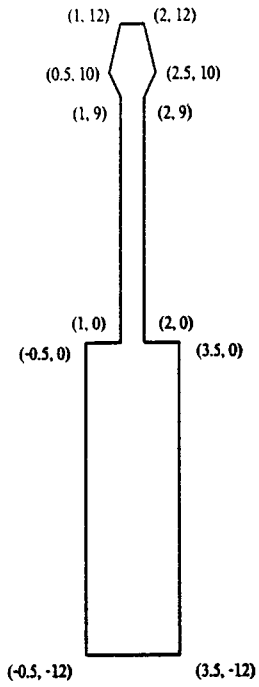


Figure 2-1. Parts provide richer descriptions than sets of global parameters. All the objects shown have the description: { *aspect-ratio = .33, area = 18, holes = 1, center = (0, 0)* }.



Point List:

(1, 12) (2, 12) (2.5, 10)
(2, 9) (2, 0) (3.5, 0)
(3.5, -12) (-0.5, -12) (-0.5, 0)
(1, 0) (1, 9) (-0.5, 10)

Chain Code:

00676656666666666666666600
06666666666666666666666444
44444222222222222222222222
20002222222222222222232212

Figure 2-2. Parts make lengths explicit. It is difficult to tell from either of the two contour representations that the screwdriver's blade is the same length as its handle.

on the contour. Entirely region-based representations [Nevatia and Binford 1977] have similar problems; they fail to make explicit contour information such as the curvature of the side of a part and the adjacency of edges. This suggests using a hybrid representation which captures the important facts about both the regions and their contours.

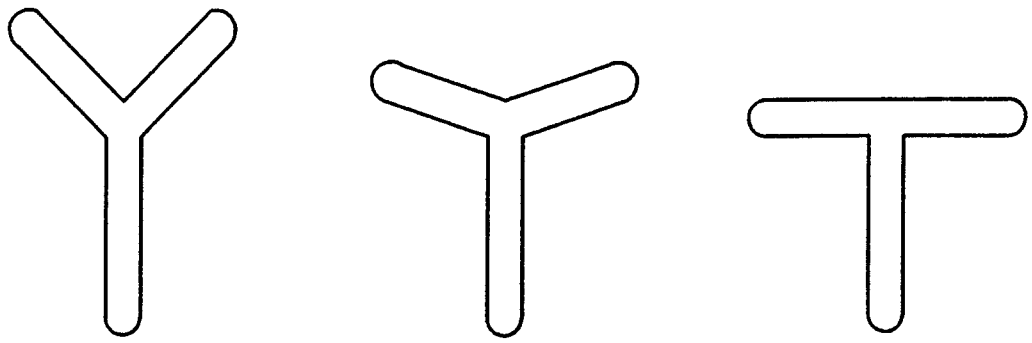


Figure 2-3. Parts help us achieve stability with respect to configuration. Although a "Y" and a "T" look different, once we realize that they both have "arms" we can transform one into the other.

Finally, a good visual representation must be *stable*. There are three major problems in recognizing a previously seen object: differences in illumination, presentation, and configuration. A stable representation is one which can tolerate reasonable variations in each of these conditions. Stability with respect to illumination suggests removing illumination-dependent information before attempting recognition. This is usually achieved by extracting the edges from an image. This does not mean illumination information is unimportant; it is essential for other processes such as shape-from-shading [Horn 1977]. Stability with respect to presentation means tolerating differences in the size, position, and orientation of an object within an image. This is usually achieved, as in feature sets, by building a representation based on operators which are invariant with respect to a suitable set of affine transformations. This is one area in which classic perceptrons fall short [Minsky and Papert 1969]. The third type of stability, stability with respect to configuration, is where the division of an object into parts becomes crucial. The idea is that an object with moveable parts should be recognizable independent of the relative orientations of the parts. For instance, the overall shapes of the letters "Y" and "T" are very different, yet, as

shown in Figure 3, we can imagine a Y being transformed into a T by bending down its "arms". Such a transformation can not be imagined without realizing that both the Y and the T are composed of a vertical stroke and two arms.

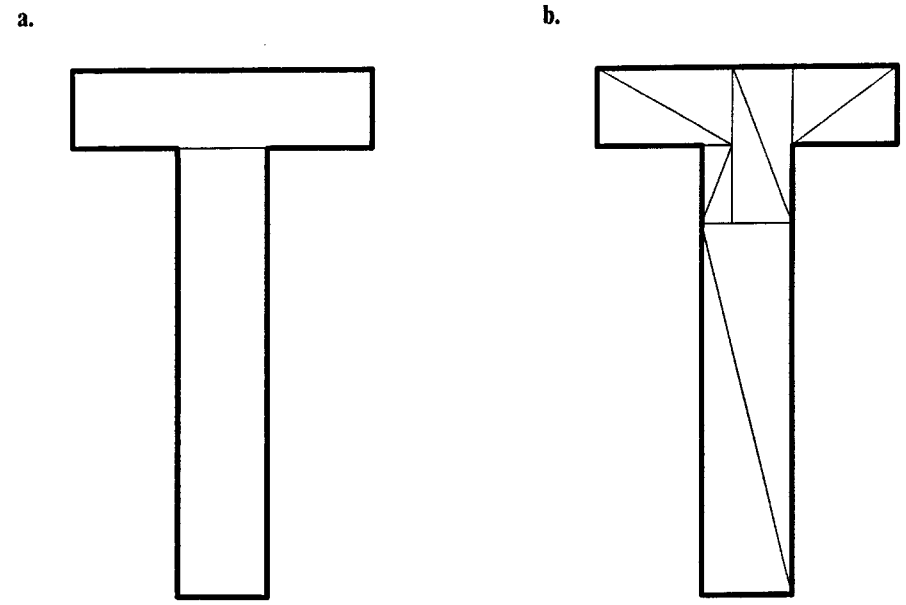


Figure 2-4. A segmentation should make boundaries explicit without introducing extra boundaries. a. A good segmentation. b. A bad segmentation.

Not all segmentations are good, however; the regions of an object must have *intuitive appeal.* For instance, the segmentation in Figure 4a is more intuitive than the decomposition in Figure 4b. Both segmentations yield a rich description yet there seem to be far too many pieces in the second one. It is also irksome that some pieces terminate without any special reason while other pieces cross what seem to be natural boundaries. The reason that the segmentation in Figure 4b is bad is because it does not conform to our intuitions about boundaries. Specifically, it does not mention some of the prominent borders while explicitly suggesting breaks that have no justification. When two pieces are joined the contour of the object usually contains evidence of this join [Marr 1977, Bagley 1985]. Many of the proposed joins in Figure 4b leave no mark on the contour, while none of the joins explain such obvious contour features as the corners formed by the head and the handle.

2.2. Smoothed Local Symmetries

To find parts we employ Brady's *smoothed local symmetries* [Brady and Asada 1984]. SLSs pick out reflectional symmetries in an object - symmetries which suggest plau-

sible axes for the parts of the object. The geometry of a local symmetry is shown in Figure 5. Imagine a circle which is tangent to the edge of the object at two points. The midpoint of the chord connecting the two points is defined as the symmetry point. The smoothly joined loci of such points are called the symmetries of the object.

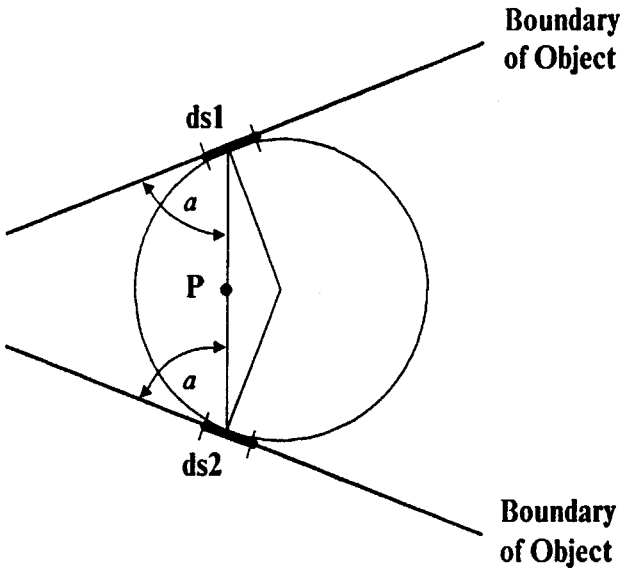


Figure 2-5. The definition of a *smoothed local symmetry*. The piece of contour ds1 becomes ds2 when reflected through point P. An SLS is a smooth collection of such symmetry points, P.

The SLSs of an object describe regions of the object. Figure 6a shows the SLSs for a claw hammer while Figure 6b shows the regions described by two of the SLSs. The lines in Figure 6a are the reflectional symmetry axes, reflecting one piece of contour through this line gives us the piece of contour on the other side. These axes are called *spines* while the perpendicular rulings are called *ribs*. Ribs join corresponding points on the two pieces of contour. For instance, consider the spine between the top of the pein and the handle. The last rib of this spine joins the tip of the claw, point A, with a point a third of the way down the handle, point B. This means that if we reflect a short piece of contour near A across the spine, it will lie exactly on top of the corresponding short piece of contour near B.

Not all symmetries between contour segments are allowed. We can assign a direction to each contour segment based on which side of the segment corresponds to the inside of the object. Since we use SLSs to find axes of plausible parts, only symmetries which agree on where the inside of the part is located are allowed. Figure

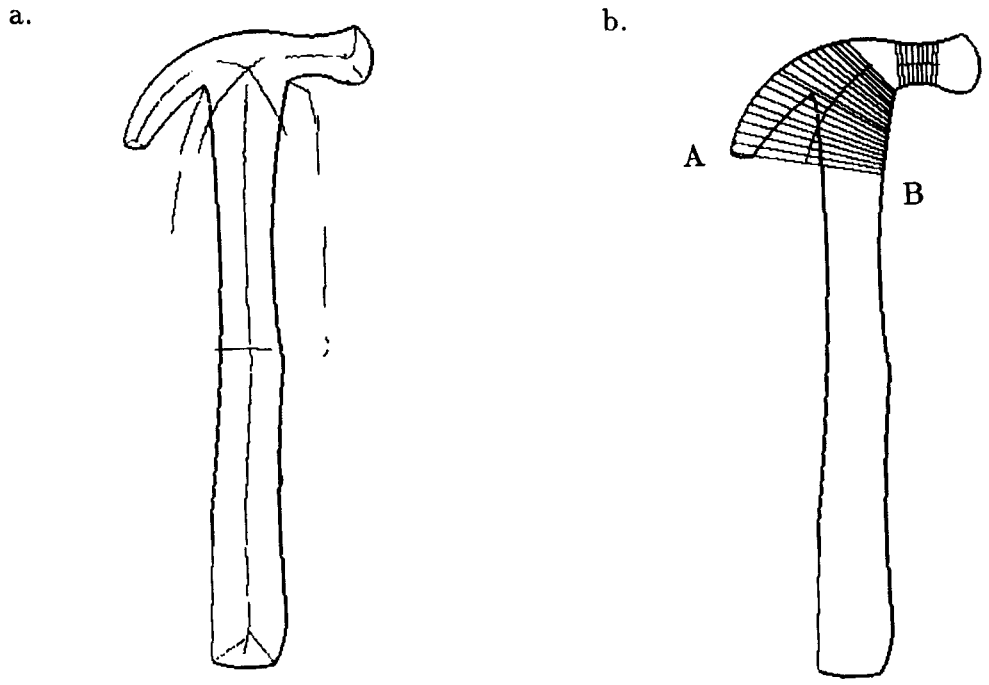


Figure 2-6. SLSs describe regions. a. The SLSs of a claw hammer. b. The regions described by two of the spines shown in a.

7a shows an *internal* symmetry which suggests a solid part of the object. We can also have *external* symmetries, as in Figure 7b, which describe empty, space-filled regions. Symmetries such as in Figure 7c, however, do not have any consistent physical interpretation.

The SLS was developed by Brady as an improvement on the Symmetric Axis Transform [Blum 1973, Blum and Nagel 1978]. The SAT is also known by many others names, including the Blum transform, the grassfire transform, and the Voronoi diagram. The SAT and the SLS of a rectangle are shown for comparison in Figure 8a. While the SLS, like the SAT, can be defined in terms of inscribed circles, there are two differences. First of all, the symmetry axes found by the SLS are built from the midpoints of chords, rather than from the centers of circles as in the SAT. More important, however, the circles in the case of the SAT must be *entirely contained* in the object; in the SLS they merely have to be tangent at two points.

The SLS has three main advantages over the SAT. First of all, it allows for *multiple interpretations* of a shape. As shown in Figure 8a, the SLS picks up both of the axes of the rectangle while the SAT picks up only part of the horizontal axis. Depending on which SLS we choose to describe the figure, the rectangle can be interpreted as

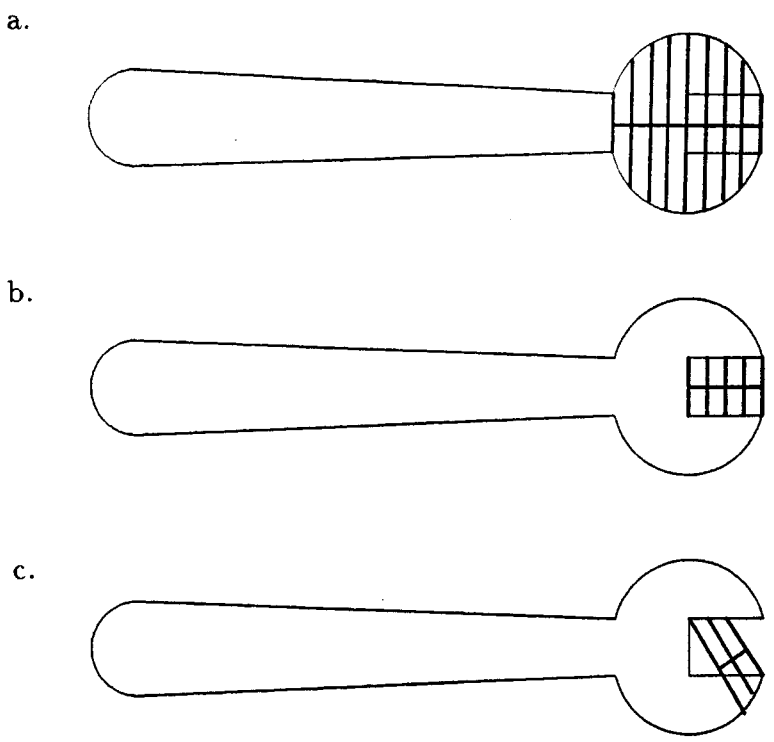


Figure 2-7. Symmetries must agree on where the inside of the region is. a. An internal symmetry. b. An external symmetry. c. A symmetry which is not allowed because it has no physical interpretation.

either a squat vertical shape or an elongated, horizontally oriented shape. The second advantage of the SLS is that it supports *cut-out* interpretations. Figure 8b shows the SLS and SAT of a rectangle with a nick in it. The SAT tries to describe the shape as a collection of disjoint solid and space-filled regions. The SLS, on the other hand, allows negative regions to overlap positive regions. Using the SLS we can interpret the shape as a rectangle, described by a horizontal interior spine, with a piece cut out of it, described by the external symmetry in the nick. Finally, the third advantage of the SLS is that it is *local.* Consider, again, the SAT of the rectangle with a nick in it shown in Figure 8b. The locus of circle centers originating in the corner of the nick extends far beyond the pieces of contour which are the borders of the region it describes. The SLS of the corner of the nick, on the other hand, does not extend past the projection of the two contour segments. A even more blatant example of the SAT's non-locality is the transform of a human profile shown in [Blum and Nagel 1978].

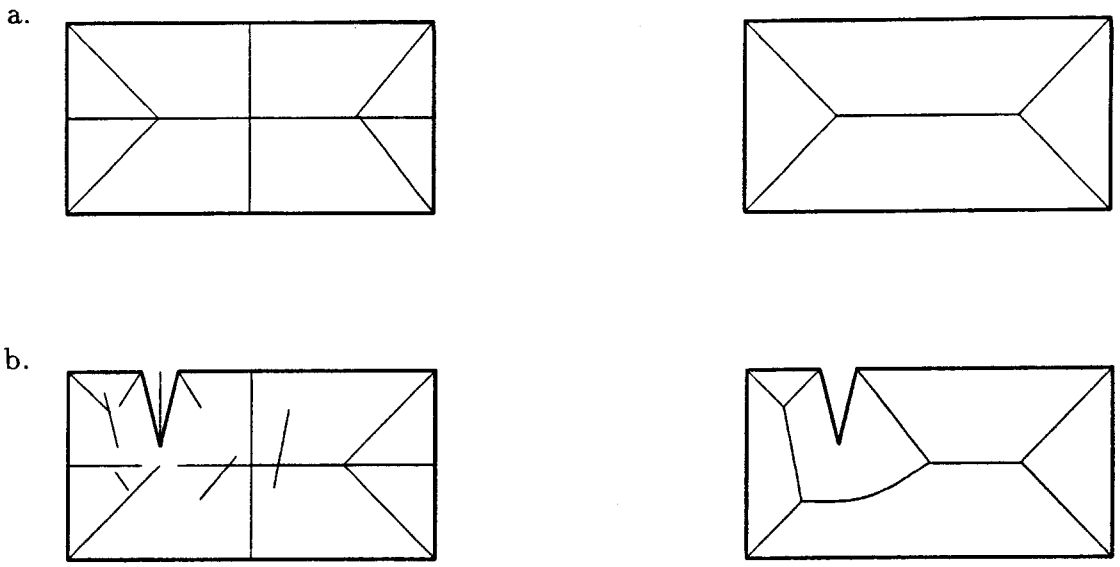


Figure 2-8. Comparing the SLS and the SAT. a. The SLS (left) picks up both axes of the rectangle while the SAT (right) does not. b. The SLS (left) suggests a cut-out interpretation for the nick.

2.3. Finding Plausible Parts

Part of the definition of an SLS is that it is the *maximal* smooth locus of symmetry. Currently Brady's system finds SLSs by approximating the boundary of an object with a series of lines and circular arcs, and then computing, analytically, the symmetries between all these sections of contour. The SLS between any two such smooth pieces of contour is guaranteed to be smooth but it may not be maximal; often a symmetry can be smoothly extended beyond the original pieces of contour. One such case is shown in Figure 9a, where breaks in a symmetry have been introduced by the contour approximation. At the bend in the top figure, for instance, the central symmetry of the "arm" is disrupted because new linear segments start at the "elbow". Symmetries can also be broken at joins between subparts as shown in Figure 9b. Here the break occurs because one or both sides of the symmetry are missing. The point is that, since symmetries form the axes of plausible parts, we need to extend the symmetries in these cases.

There are several important restrictions on when two spines can be joined. The first of these is that the ends of the spines must be *reasonably close together.* This is usually not a problem for breaks caused by the contour approximation. However, as shown in Figure 10, it can become significant when there are subshape joins. The issue is more than just finding a way to heuristically reduce the potentially vast number

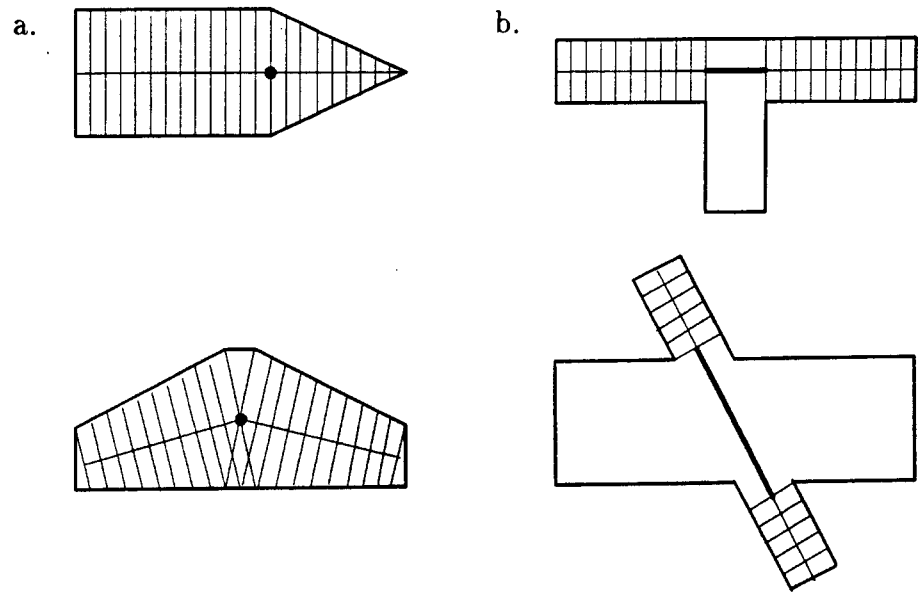


Figure 2-9. Sometimes SLSs need to be extended across gaps. a. Breaks can be introduced by the contour approximation. b. Breaks also occur at subpart joins.

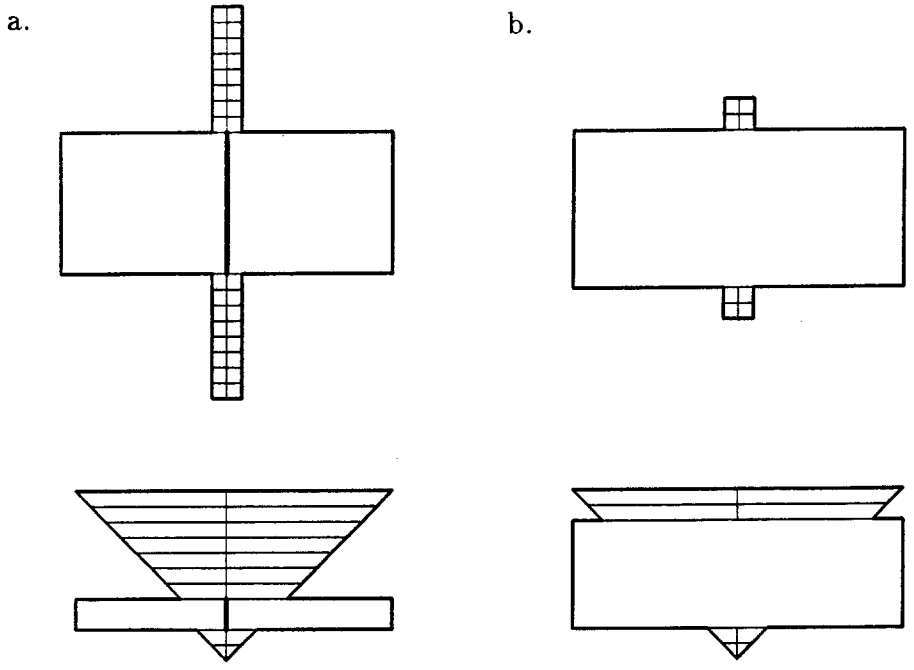


Figure 2-10. Spines which are far apart cannot be joined. The spines in a are close enough but the spines in b are not.

of possible extensions, although this is important. The point is that if the spines are too far away, they are not perceived (by humans) as parts of the same region. This restriction reflects a fidelity versus smoothness trade-off like those discussed in [Poggio and Torre 1984]. While the two spines may smoothly extend, there simply is not enough evidence to support the joined interpretation when the gap is large. The program actually measures the size of a gap by comparing its area to the area of the spines to be joined. Comparing area, rather than length, is important in the case of spines which are tapered, as shown in the bottom half of Figure 10.

Another requirement for combining two symmetries is that their *axes smoothly extend.* For two symmetries to extend their axes must be pointing in more or less the same direction. Figure 11a shows several cases in which spines can not be joined because they are going in different directions. When two spines are close together but oriented differently it is often the case that the regions described by the two spines have been physically joined end-to-end. Knowing that the two sections were originally distinct can be important. Thus, since such joins seldom produce perfectly aligned sections, we disallow large changes in orientation. The orientation restriction alone is not strong enough, however, as shown in Figure 11b. Here the spines are pointing in exactly the same direction but when joined they have a kink in the middle. In the real world, kinks such as these are often created by a shear force applied to an object. This suggest that we treat the two sides as different parts. We can strengthen the axis condition by requiring the lateral offset to be small compared to their widths. These two conditions can be summed up by restating the axis criterion in terms of the joined spine: the joined spine should have *no major bends.* This obviously works for the case where the axes are oriented differently. It also works for the offset case if we consider drawing in the section of spine joining the two spines on either side. The combined spine will then be "Z" shaped with two distinct bends. In practice, however, spines never extend perfectly, we need to tolerate small offsets. This is why we say no "major" bends. A major bend is a change in the orientation of the spine over a length comparable to the width of the two spines at the join. The width of the spines provide us with a natural scale over which to find discontinuities. The no major bends criteria is similar to Bagley's [1985] collinear extension heuristic for polygons. It is also similar to the spline method proposed by Heide [1984], except that our version takes scale into account.

The last requirement for joining two spines is that the region which results must be *locally convex.* Examples of regions which are and are not locally convex are shown

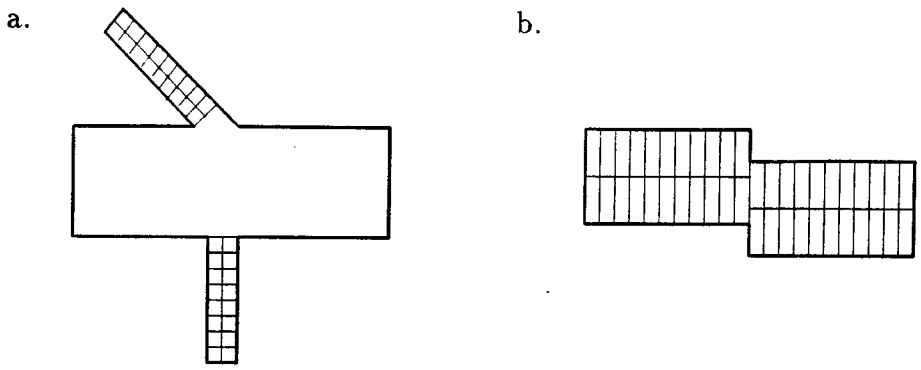


Figure 2-11. Spines can only be joined if their axes smoothly extend. The spines shown cannot be joined because a. they are pointing in different directions. b. they are offset from each other.

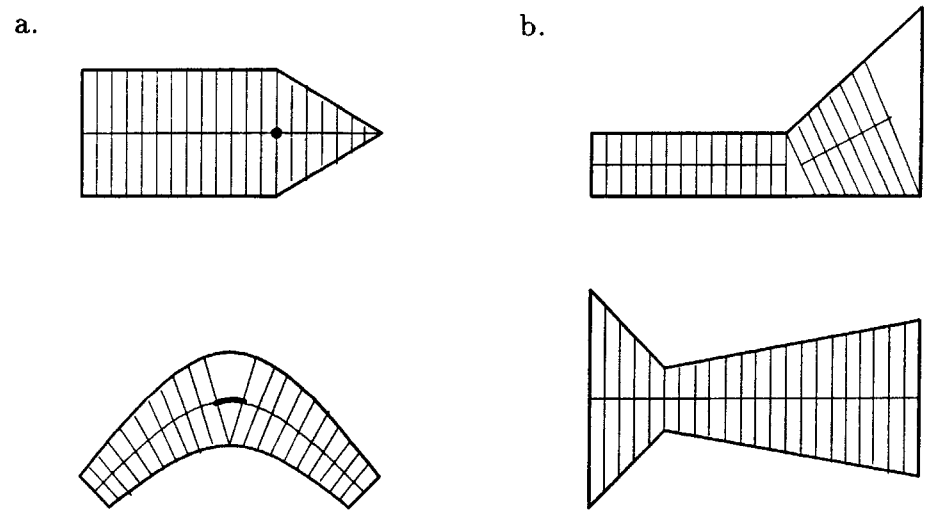


Figure 2-12. Parts are by their nature convex with respect to their axes. The spines in a can be joined but the spines in b would yield non-convex regions if joined.

in Figure 12. The "local" in the term local convexity means local with respect to the axis of the region. Imagine pulling on the two ends of a spine until the spine was straight, if the deformed region is convex then the original region was locally

convex. This means that regions which do not look globally convex, such as the bottom example in Figure 12a, can still be locally convex. The rationale behind the local convexity criterion is that the concavities of an object correspond to the physically weakest places in an object. If we were to drop the object and let it shatter, these are the places at which we would expect breaks to occur. The local convexity requirement covers earlier suggestions that regions should be segmented at minima of width [Fairfield 1983], at matched concavities [Marr 1977], or minima of curvature [Hoffman and Richards 1982].

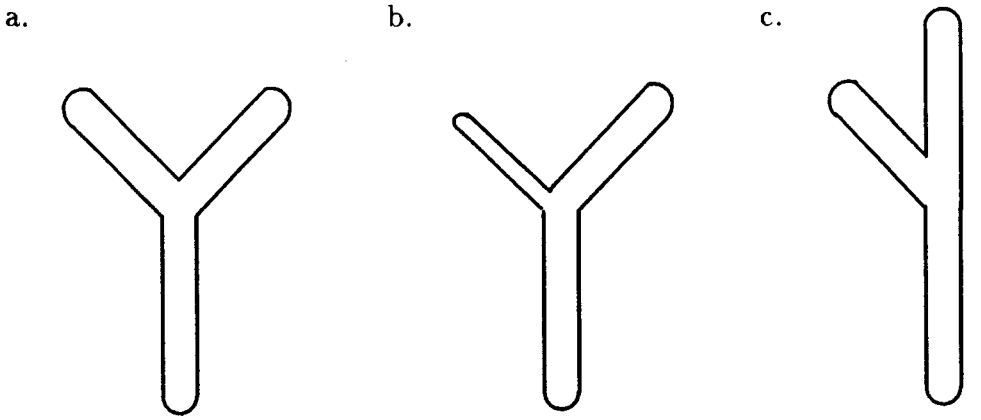


Figure 2-13. Sometimes there is no clear choice of what to join to what, as in a. Varying the b. width or c. angle of one of the pieces can break the tie.

Sometimes we leave two spines unjoined even if they satisfy the gap, axis, and convexity conditions. Such a case is shown in Figure 13a. The upright of the "Y" is reasonably close to both of the two arms and could be joined to either. However, since we have no grounds for choosing one arm over the other, we follow the *principle of least commitment* [Marr 1982] and leave the upright unjoined altogether. This is important because, under the no-overlap condition, the upright can be joined to only one of the two arms, not both. Often such a cautious approach is not necessary because a decision can be made based on differences in width or angle. Figure 13b

and Figure 13c show two cases in which, although either of the two joins is plausible, joining the upright to the more similar of the two arms is clearly favored.

To summarize the previous discussion, segmenting an object into parts requires first finding all the plausible parts. This is a two step process: first, we determine the smoothed local symmetries of the object, then we try to link symmetries that are reasonably close together. To be linked, two symmetries must match in angle and the combined region must be locally convex. Each of these maximally extended symmetries specifies the axis of a plausible part of the object. These rules are not intended as a complete theory of shape perception, rather they are merely heuristics that have been found to work well in practice. They serve to prohibit some of the truly egregious joins while picking up most of the obvious cases.

2.4. Choosing a Decomposition

The SLSs of an object suggest parts but there are far more symmetries than there are intuitive pieces. We must choose some subset of the SLSs to describe the object. The idea is to explain all the edges seen as the borders of some number of regions. This set of regions should be *non-overlapping* and *connected.* The regions must be largely disjoint since in the real world two things can not occupy the same space. The regions must be connected otherwise we could not pick up the object as a unit. Figure 14 shows examples of sets of regions that violate these conditions.

At present our vision system does not have a good understanding of regions. The non-overlap and connectivity conditions are enforced using contour information only. The non-overlap condition is implemented by assuming that each section of contour is the border of exactly one region. Thus if two regions claim the same section of contour it assumes that they overlap. Connectivity is also viewed in terms of the contour of the object. The system decides that two pieces are connected if their borders are a short distance away along the contour. By "short" we mean small as compared to the width of the pieces.

Constraining the parts of the object to be connected and non-overlapping still leaves us with the problem of deciding which of the plausible parts gave rise to this border. For instance, a common situation is illustrated in Figure 15. Not only do we have two obvious parts but we also have the envelope of the pair. This leads to two possible interpretations of this shape each of which satisfies the non-overlap condition. The object could either be two pieces joined at their ends, or a wedge shaped piece with a smaller wedge cut out of it [Bagley 1985]. To solve this problem

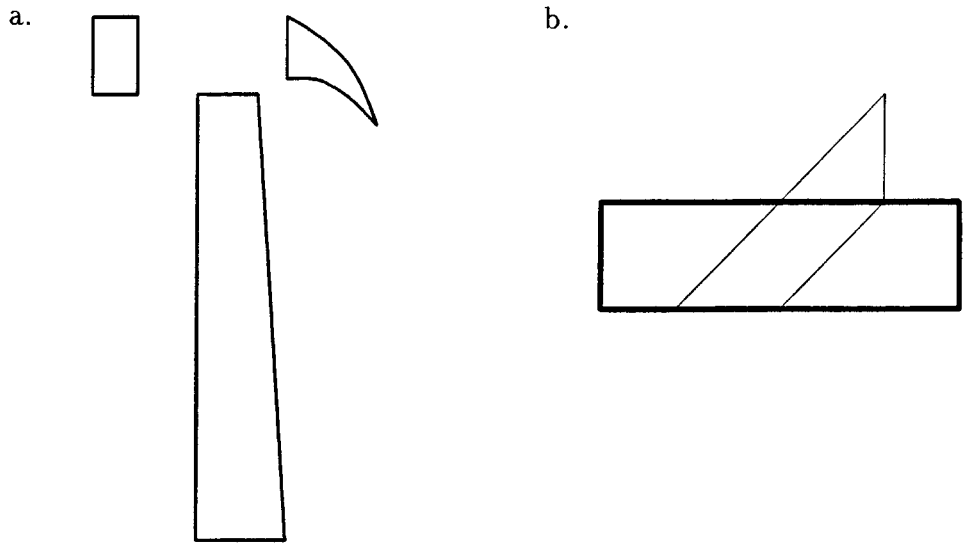


Figure 2-14. The sets of regions shown are not good objects because a. they are not connected and b. they overlap.

we make the observation that long skinny regions tend to be more salient than short fat regions. Although roughly circular regions also stand out [Fleck 1985], the SLS is best at detecting elongated regions. This suggests that the two piece interpretation is better than the wedge interpretation because the two side pieces are more elongated than than their envelope. If we measure the elongation of a piece in terms of its aspect ratio (the average width divided by the length) we can restate the "long and skinny" heuristic more precisely. When two regions are competing to explain the same section of contour, *the region with the lower aspect ratio wins.*

This heuristic sometimes generates an interpretation which fails to satisfy the connectivity condition. This is the case with the fork shown in Figure 16a. The two side tines have much lower aspects ratios than the head of the fork and therefore claim the outer edges of the fork's contour. The region corresponding to the head is suppressed because part of its border has already been claimed by a supposedly better description. This, however, leaves the tines unconnected to the handle of the fork. One solution is to backtrack by allowing the head of the fork to claim the two sides even though the tines have better aspect ratios. This leads to the cut-out interpretation shown in Figure 16b. Another solution is to *relax the overlap condition* by including the required connecting region in the final segmentation even though part of its contour has already been claimed. This method generates the

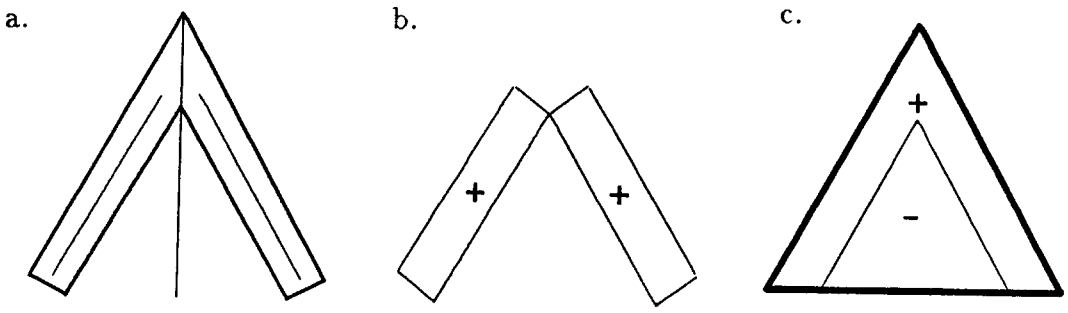


Figure 2-15. There is more than one decomposition for most objects. a. The symmetry axes of the object. b. A two part interpretation. c. A wedge interpretation. We favor the decomposition in b because the parts have low aspect ratios.

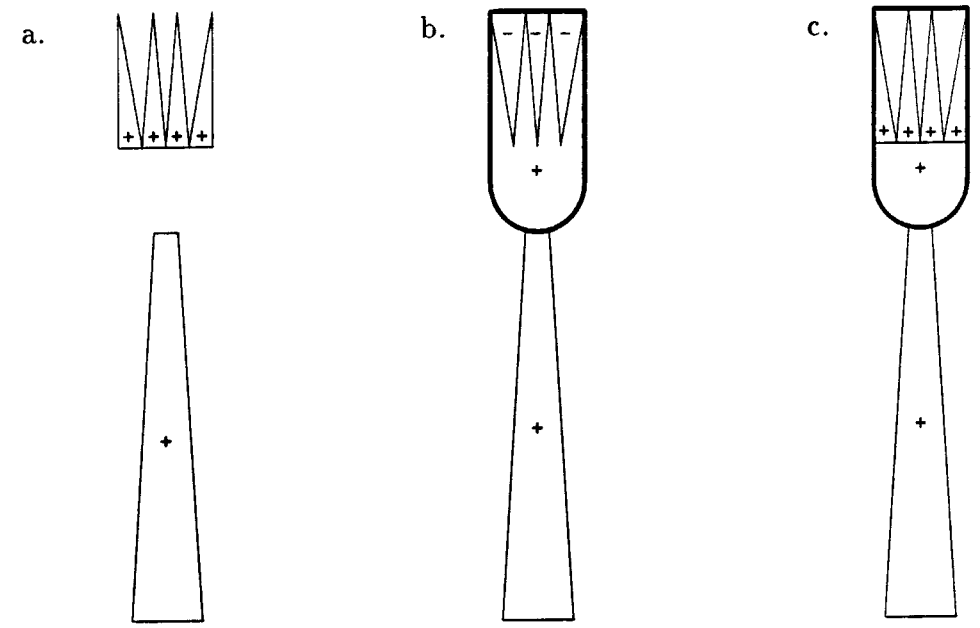


Figure 2-16. Sometimes the overlap condition must be relaxed. a. The head of the fork has unfortunately been omitted because the tines had better aspect ratios. b. An alternative cut-out interpretation. c. An envelope interpretation.

interpretation shown in Figure 16c and is the method used by the program. The head of the fork is viewed as a rectangle which has substructure; the tines only become evident when the head is examined more closely.

We have now presented the complete segmentation scheme used by our program.

After finding the plausible parts of an object, the program chooses a connected, disjoint set of these parts to describe the object. The interpretations produced favor long skinny pieces over shorter squatter ones.

2.5. Parameters of Parts and Joins

After segmentation there is still more work to be done: the pieces found must be described. We know that the pieces found all have smooth axes and that they are convex but this leaves their shape still largely unspecified. By the definition of a SLS, the shapes of pieces we have found can be described by two functions: the *orientation* of the axis and the *width* of the cross-section. These are both functions of length along the spine. However, instead of recording the orientation and width at every point, we summarize the functions using several shape parameters. Earlier we criticized such parameters but this was because they failed to accurately represent the structure of the shapes from which they were computed. We claim that six numbers suffice to give a good approximation of the shape of the pieces found by our segmentation algorithm. Furthermore, all of these parameters are dimensionless and invariant with respect to scaling, rotation, and translation. The required parameters are:

- the aspect ratio of the region
- the overall curve of the spine
- the relative widths of the two ends
- the tapers of the two ends

Each of the parameters has a formal definition as shown in Figure 17. The *aspect ratio* of a region is its average width divided by its length. As mentioned before, this measure plays an important role in selecting a segmentation for an object. The second parameter, the *overall curve* of the region, corresponds to the "dented-ness" of the spine. It is defined as the average deviation of the axis from the line connecting its two ends normalized by the length of the axis. These two parameters were also used by Heide [1984], although he computed the overall curve by taking the average curvature of the spine times its length. Our method is less sensitive to the often noisy orientations of the ends and thus gives an intuitively better measure of the piece's bend. Heide's representation also did not include the four end parameters. The *end widths* are the widths of the two ends divided by the average width of the region

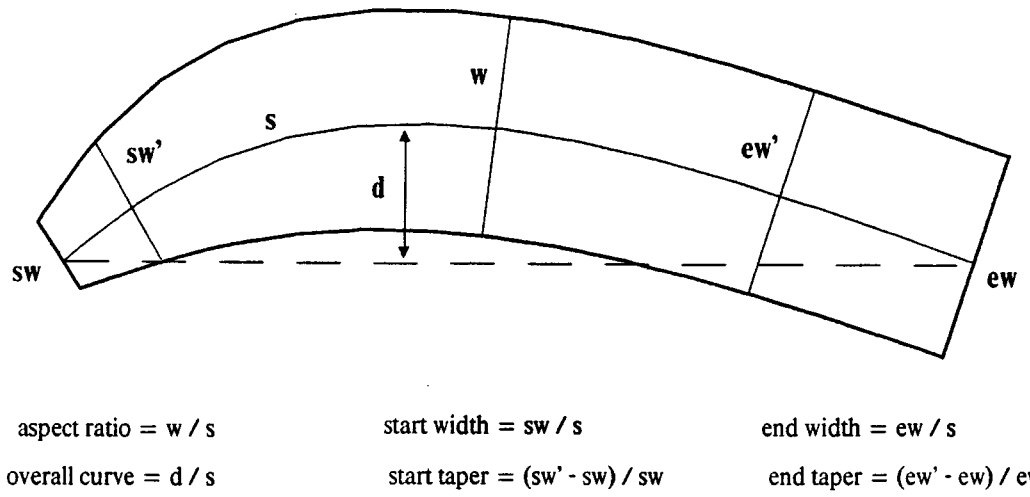


Figure 2-17. The shape of a piece is described by six parameters. These parameters are sufficient because each piece is locally convex and has no major bends.

while the *end tapers* are the local derivatives of width at the two ends. The taper of a end is measured over a scale comparable to its width.

These parameters accurately represent the shape of a piece due to the restrictions we placed on the extension algorithm. First, a piece is guaranteed to have *no major bends*. This means the most it can do is either curve gently or oscillate slowly. Neglecting the second case (wiggly things are relatively rare), spines are either straight or they bend gradually to one side. The overall curve parameter is meant to capture the degree and direction of this bending. There are cases, such as spirals, in which the overall curve of the spine is not meaningful, but for the typically short spines in most segmentations this parameter is adequate. Second, because a piece is *locally convex*, we know that there are no constrictions in the middle of the piece; the only places at which the region can get smaller are the two ends. Thus, if we know the average width of the piece (obtainable from the aspect ratio) and the width at the two ends, we have a rough sketch of the width function. Experience has shown, however, that this is not quite enough; we often need to know how pointed an end is as well. For

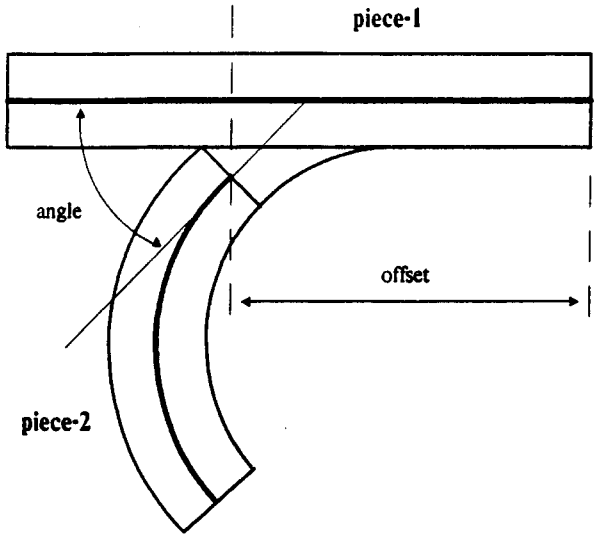


Figure 2-18. Joins are described in terms of the angles of the pieces and the location of the join with respect to each piece.

this reason, we also the record the rate of change of width at each end.

Not only must we describe the shapes of the objects, we must also describe how they are combined. We do this by finding and describing the *joins* between pieces. Finding joins is easy since joins are connections between pieces and we have already discussed how connectivity is determined. We must do more, however, than just say that a certain piece is joined to a certain other piece. There are two other important things we must determine:

- the relative angle of the pieces joined
- the location of the join with respect to each piece

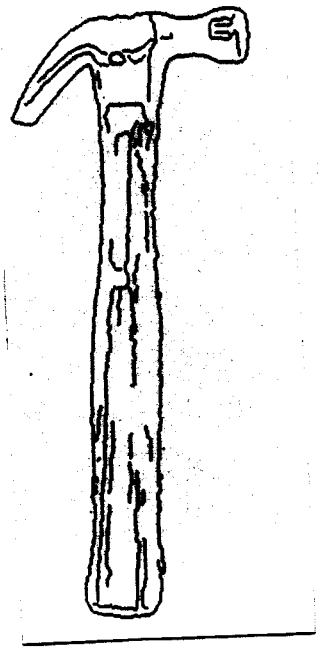
The relative angle between two pieces is calculated by taking the difference between the local angles of each spine. Here "local" means over an interval comparable to the width of the piece at the join. For instance, the angle of the join shown in Figure 18 is 45 degrees with piece-2 leaning toward the left end of piece-1. The location of a join is specified with respect to the coordinate frame defined by the axis of the piece. The join shown occurs halfway along piece-1 and near the end of piece-2. We also record, as suggested by Marr [Marr and Nishihara 1977], whether the join occurs at the end of the piece or along its side. Thus, the position of the join in Figure 18 is further specified by noting that the top end of piece-1 is connected to the lower side

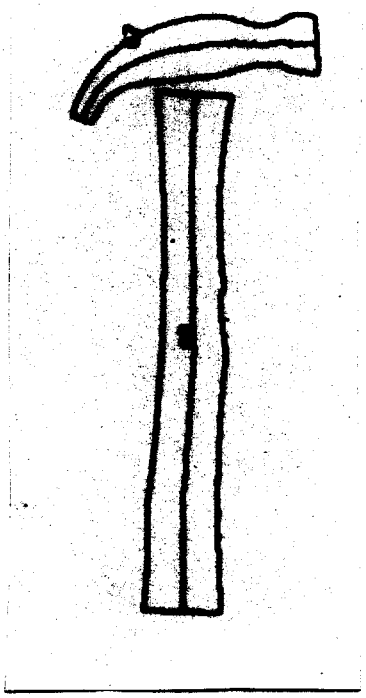
of piece-2.

2.6. Segmentation Results

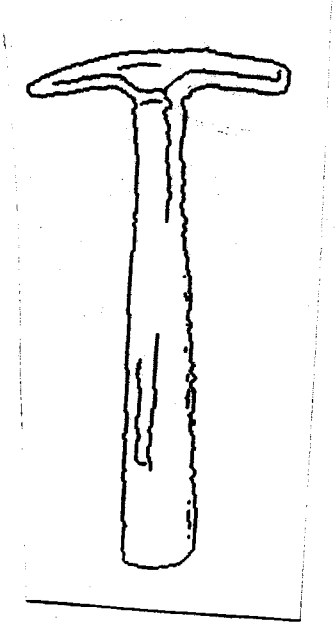
The following pictures were generated by our implemented segmentation program. We show the edges extracted from the original gray-scale image and, next to it, the output of the segmentation program. For each part of the object we show the part's spine, the sections of contour described by the part, and its first and last ribs.

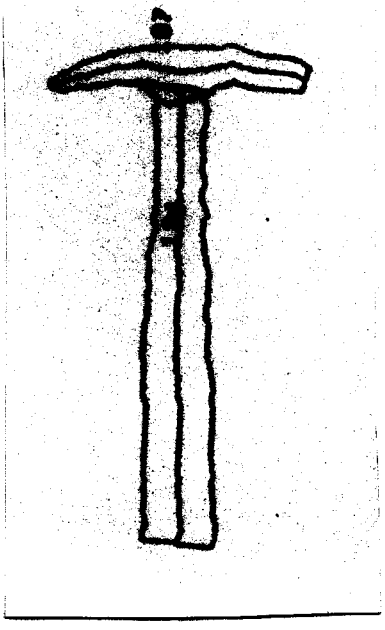
Claw Hammer



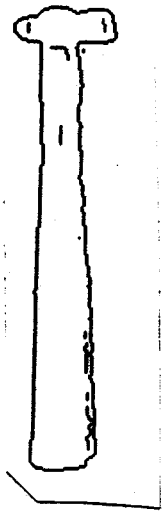


Tack Hammer

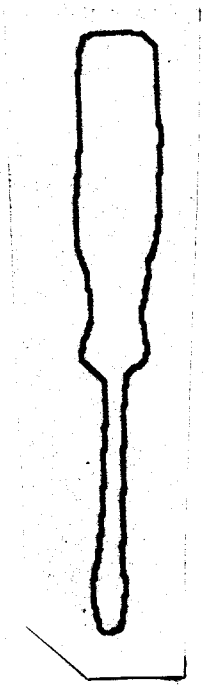


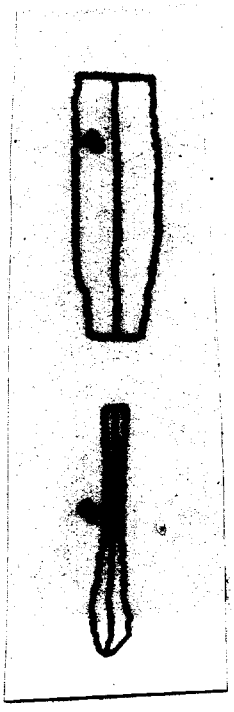


Ballpein Hammer

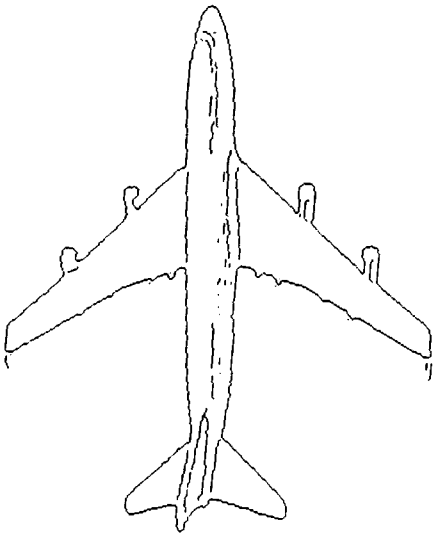


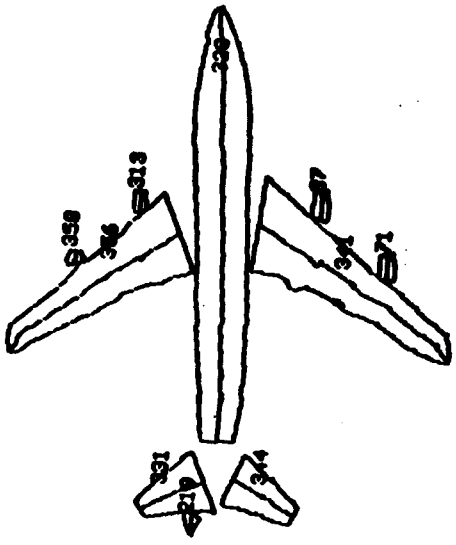
Screwdriver



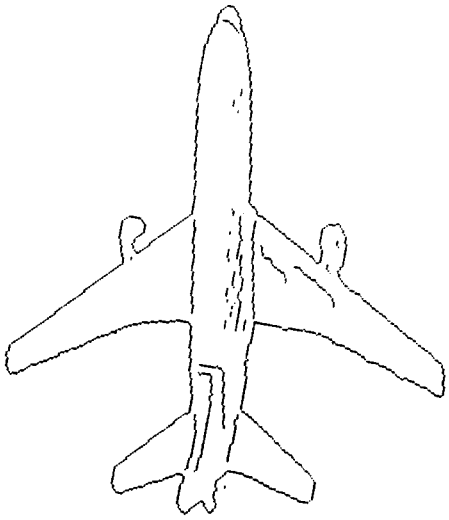


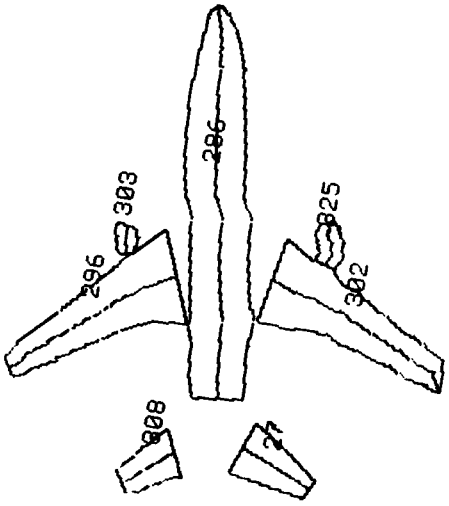
Boeing 747



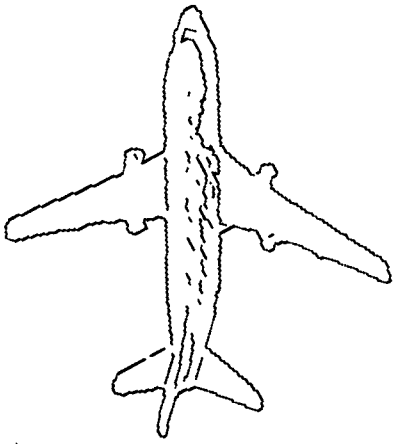


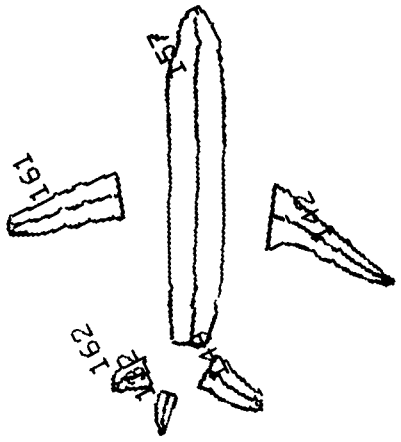
Lockheed 1011



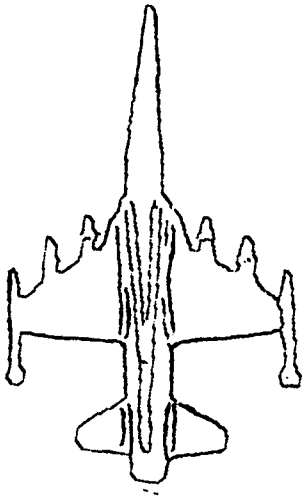


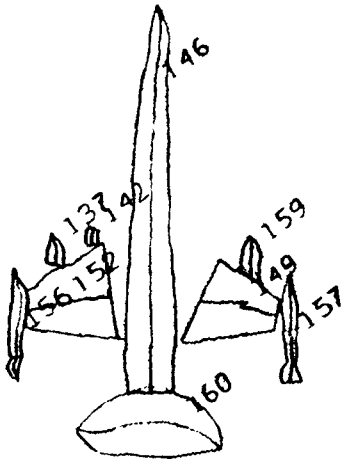
Douglas DC-9





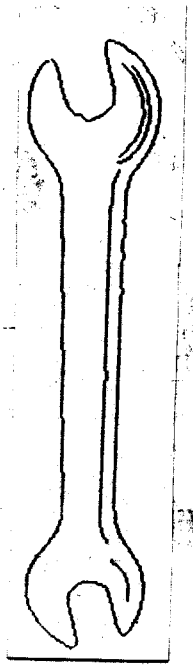
Northrop F-5

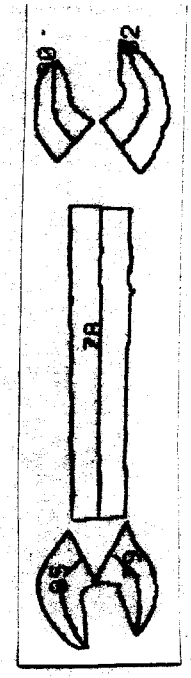




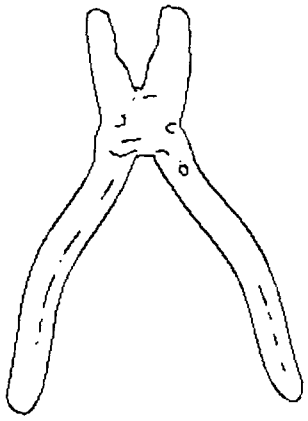
Fork

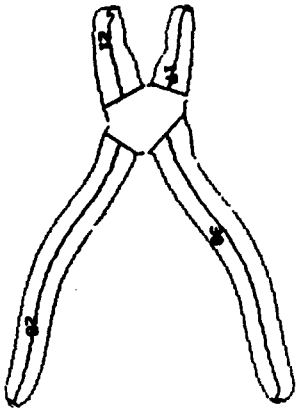
Crescent Wrench



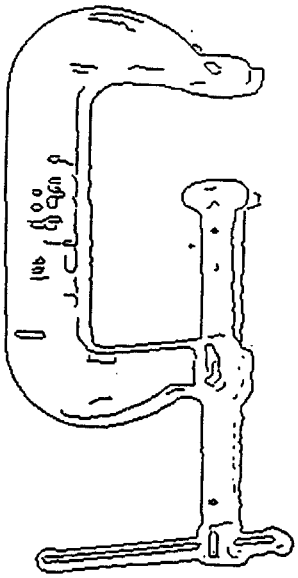


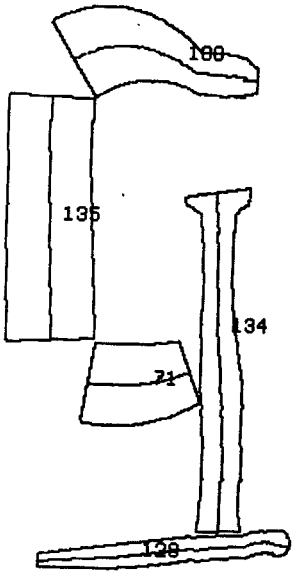
Pliers



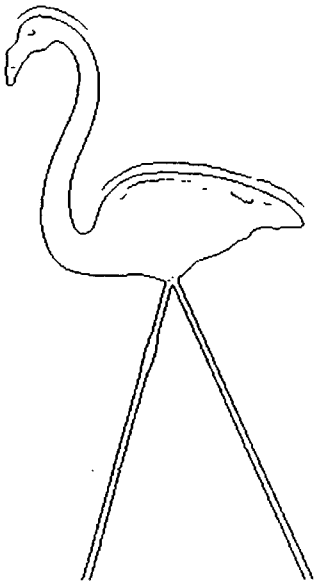


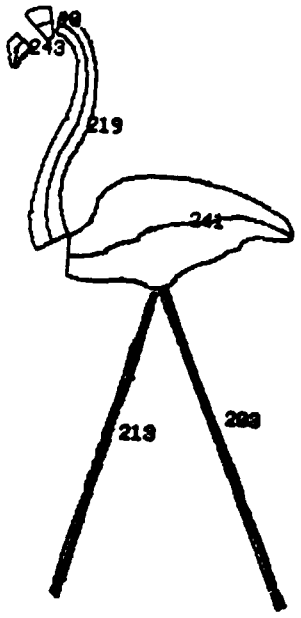
C-Clamp



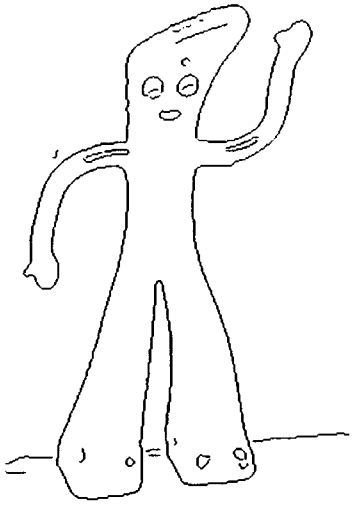


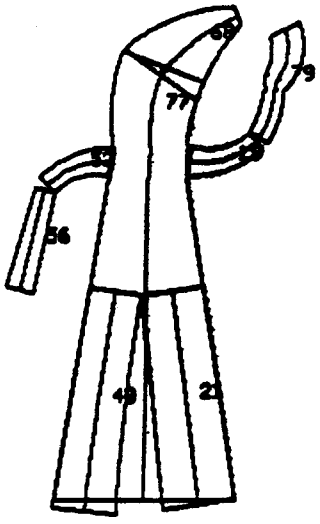
Flamingo



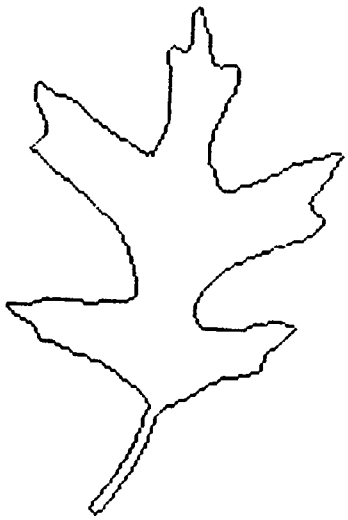


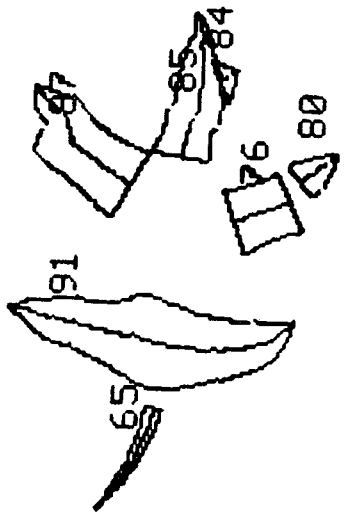
Gumby



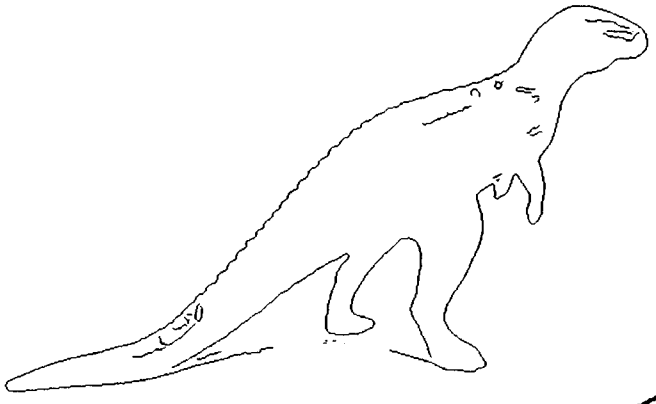


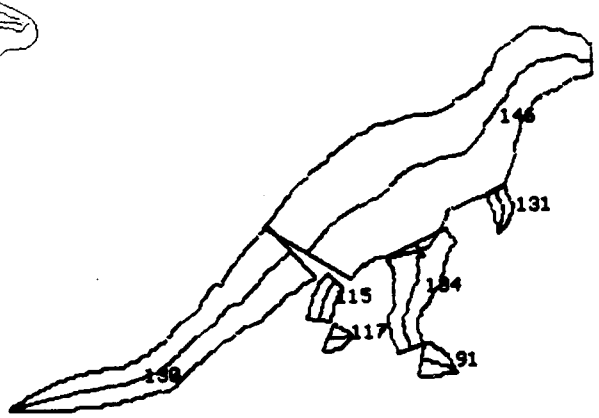
Oak Leaf





Megalosaurus





3.1. Syntax as Semantics

In order for the learning program to produce meaningful results it must have knowledge of what makes things visually similar. Since determining the similarity of two objects is obviously very important for learning, the representation should make such similarities *explicit*. We accomplish this by embedding our knowledge of visual similarity directly in the representation. The embedding is based on the principle that *syntactic distance should reflect semantic distance*. Specifically:

- Similar things should give rise to similar descriptions
- Dissimilar things should yield manifestly different descriptions

As shown in Figure 1, not all representations follow this principle. Everyone has had the experience of getting in the family car and driving around the block to watch the odometer change from 99,999 to 100,000 miles. There is nothing particularly special about 100,000 that makes it extraordinarily different from 99,999, yet the representation leads us to believe that there is. The mile between the two is in fact

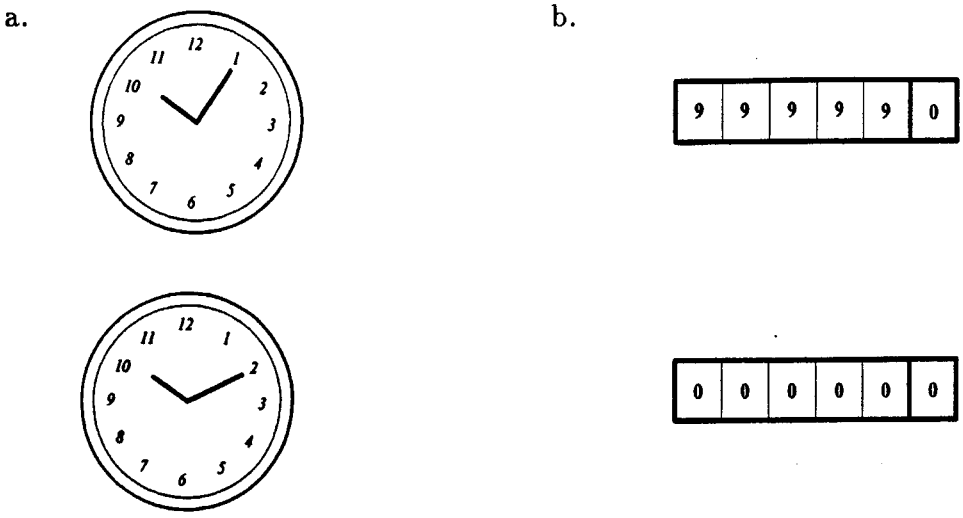


Figure 3-1. The syntax of a representation should reflect its semantics. Analog clocks conform to this principle while car odometers do not. a. 10:05 looks very much like 10:10 b. 99,999 looks very different from 100,000 even though there is nothing special about the mile in between.

no different than the mile between 76,982 and 76,981, two mileages which look very similar. The odometer fails to make similar mileages look similar. This is a general problem with digital readouts: the notion of similarity needed to compare two numbers is specified by the subtraction *procedure* rather than by the representation of the numbers. Analog readouts, on the other hand, do not have this complication. Consider the times 10:05 and 10:10. In digital form, half the digits in the representation of the time change which suggests that the two are rather different. The difference between 9:55 and 10:00 is, apparently, even worse. On an analog clock, however, the representations of the two times are very similar; the only difference is that the minute hand has moved slightly. This is what we mean when we say the syntax of the representation should reflect the semantics underlying it.

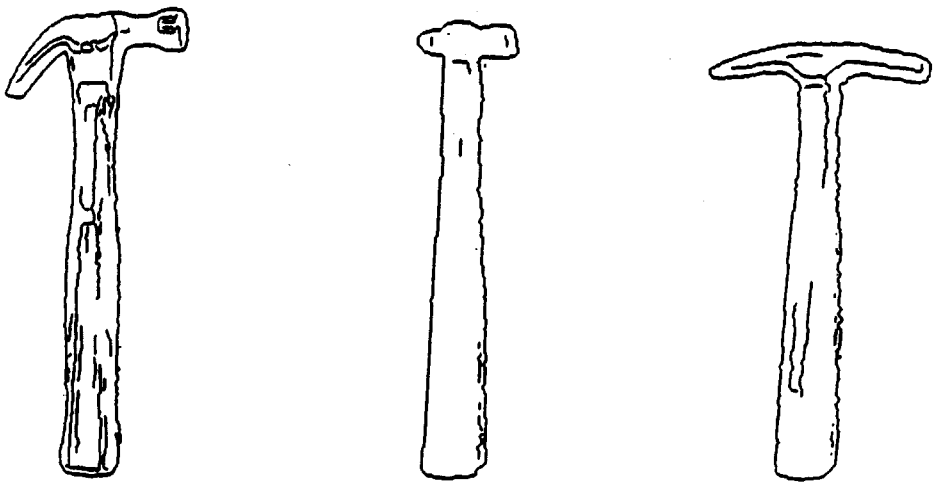


Figure 3-2. The prototype of a hammer is a special kind of signal. This signal is present in the examples shown but it has been corrupted by the details of the objects. A good representation should be insensitive to such noise.

This principle is related to the *stability* of a representation. A good representation should be insensitive to noise; if the same signal is present the same description should result. The "syntax as semantics" principle extends this idea to things which are merely similar, not identical, to give us immunity to a different kind of noise. The signals we are interested in are class prototypes. The noise mixed with this signal are the irrelevant, and sometimes conflicting, details of examples from which the concept is built. Figure 2 shows the basic hammer "signal" corrupted by various "noise"

functions. The examples shown differ from each other primarily in the type of pein the particular hammer has: be it curved and pointed, short and rounded, or straight and square. Our extended notion of stability requires that these examples give rise to highly similar descriptions since they are all instances of the same concept.

As stated, we are concerned primarily with the similarity of objects. In this respect, our approach differs from that of classic representation languages such as KRL [Bobrow and Winograd 1977], KLONE [Bobrow and Webber 1980], and KRYPTON [Brachman, Fikes, and Levesque 1983]. These systems were concerned with managing taxonomies and making simple deductions from them. The representations developed were carefully tailored to make the required operations fast and easy. Since we are interested in similarity, we have instead designed our representation to make the comparison of objects easy. The exact manner in which this was done will be explained in the following sections.

3.2. The Network Formalism

The representation language in which we embed our knowledge of similarity is based on the idea of semantic nets. A semantic net is a collection of nodes with labelled links between them. However, a semantic net is not just a graph; it must have meanings attached to each link and node [Woods 1975, Brachman 1983]. Our networks have two types of nodes. One type, called *things*, stand for typically noun-like entities such as objects, regions, and ends. These are represented by dark boxes in Figure 3. The other type of node are called *predicates* and are drawn as light colored boxes. Each predicate is connected to its arguments by a number of labelled arcs which form the links in our semantic net. We make a further distinction between predicates based on their arity. Unary predicates are called *properties* and are used to represent intrinsic qualities of a thing. For instance, the handle of the hammer shown in Figure 3 has the properties of being STRAIGHT and being a REGION. N-ary predicates are called *relations* and are used to represent facts about groups of things. Figure 3 shows two relations involving the head of the hammer: it is ATTACHED to the handle and it HAS an end.

Each of the things in a relation plays a particular, usually different, role in the situation described by the relation. For instance, saying that "A is-bigger-than B" is certainly different from saying that "B is-bigger-than A". To make clear their respective roles, we assign a *label* to each of the participants in a relation. Figure 4 shows several examples of this. The IS-BIGGER-THAN relation has the labels

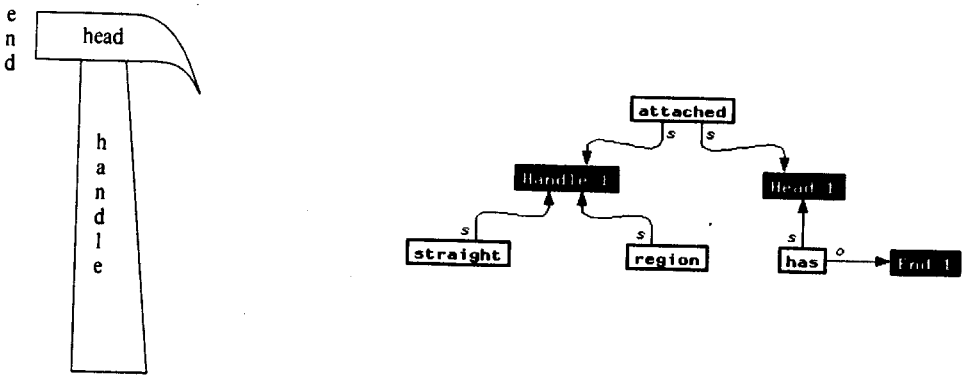


Figure 3-3. Descriptions are collections of things and predicates. Things are represented by black boxes while properties and relations are shown as white boxes.

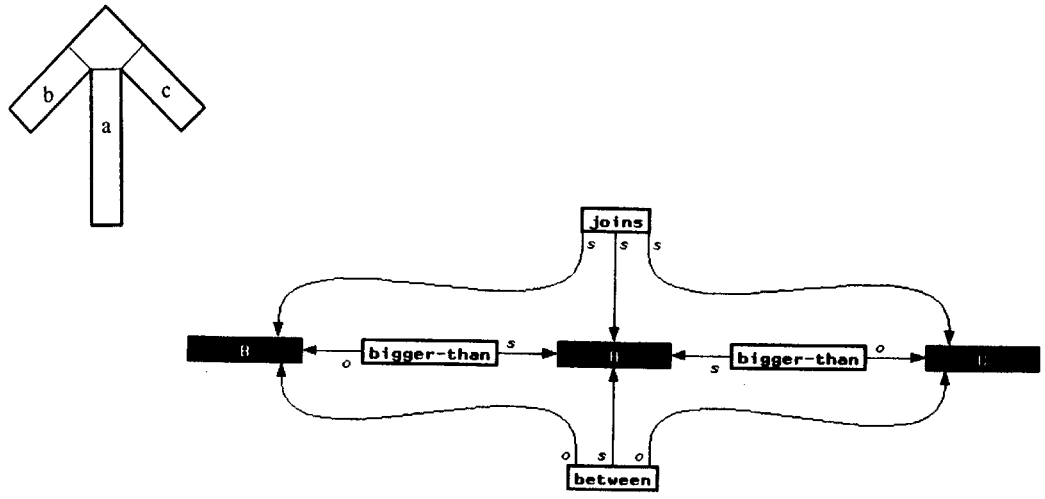


Figure 3-4. The arguments of predicates are labelled so they can be distinguished. Sometimes several arguments play the same role as in the JOIN and BETWEEN relations.

"s" for subject and "o" for object which allow us to tell which part is the big one. Sometimes, however, several arguments play the same role in a relation. It does not matter whether we say "A is-connected-to B" or "B is-connected-to A", the point is

that the two parts are joined together. Since both parts play the same role we assign them the same label. Figure 4 shows an example of this in which three pieces all meet at a common place. Since all the pieces play the same role, all the arcs coming out from the JOIN relation have the same label (predicates are not required to have a fixed arity). The BETWEEN relation is similar except that the thing in the middle needs to be distinguished from the things on the ends. This is done by labelling the ends as "o" and the center as "s".

Our representation is similar to the representation of circuits used by Davis [1984] in this respect. In his representation each module has a number of labelled ports. Modules are connected by running a wire from a port of one device to some port of another. The predicates in our representation correspond to the modules in Davis's representation. Each of our predicates has a set of labelled arguments which are used to "connect" it to other predicates: two predicates are connected if they have some argument in common. Thus, Davis's wires correspond to things in our networks.

Predicates serve to *modify* their arguments. A thing has no intrinsic meaning, it is just a nameless cipher until it participates in some predication. Since a thing is no more than a collection of features, adding a predicate changes the import of the thing. Much as things can be modified by predicates, complete predications can also be modified by other predicates. Figure 5 shows two examples of this. Here A is joined to B and the join occurs at the end of B. The AT relation involving the JOIN relation further specifies the join by telling its location. The OF relation involving the END property says that E is an end with respect to B. This is what Hendrix [1978] calls a "partitioned" semantic network. A similar technique is used by Davis [1984] for representing the inner structure of an electronic module.

Modification is an important feature of the language for two reasons. First, it allows us to make a distinction between the main point and the details. Since the AT relation modifies the JOIN relation, it is less *important* than the JOIN fact. This would not be the case had we concocted a new three-place predicate JOINED-TO-AT and applied it to A, B, and E. The JOINED-TO-AT predicate makes the fact that the pieces are connected and the fact that connection occurs at E equally important. Even worse, however, it makes the two facts inseparable. This is the second advantage to having predicates with modifiers: it allows us to express *partial information.* For instance, we may know that two pieces are joined but not know where. We can not use the JOINED-TO-AT predicate in this case because we do not know its third argument. One solution would be to use both the JOIN and JOINED-

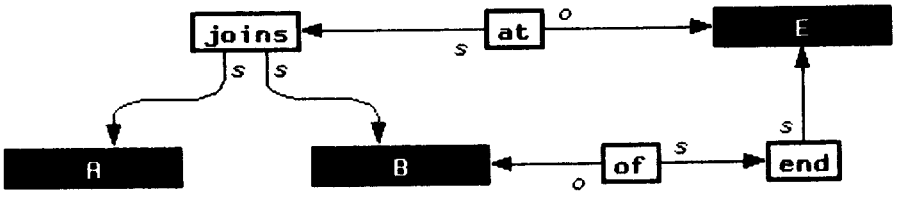


Figure 3-5. Predications can be modified by other predicates. This is important for specifying the relative importance of facts and for representing partial information.

TO-AT relations, the choice depending on whether we knew the location of the join or not. This, however, does not capture the fact that the first two arguments of the JOINED-TO-AT predicate also JOIN each other. While it seems obvious to us, if we changed the name of the JOINED-TO-AT relation to INTERSECTION-OF-IS, or even FOOZLE, it would be much less obvious. We stated earlier that we wished to put as much of the knowledge as possible into the structure of our representation. For this reason we allow predicates to be modified, instead of creating new compound predicates as KLONE [Brachman, Fikes, and Levesque 1983] and KRL [Bobrow and Winograd 1977] do.

The representation language described is an improved version of the semantic nets used by Winston [Winston 1981, Katz and Winston 1983]. Winston's networks suffered from several problems. First of all, they were restricted to binary relations which are insufficient for representing relations such as BETWEEN which naturally takes three arguments [Woods 1975]. To represent this fact in Winston's system one might use the statement:

((A between B) and (A between C))

or the two statements:

((A is between) to B)
((A is between) to C)

However, neither of these seems particularly palatable. We solve this problem in our representation by allowing predicates of any arity. Another problem with Winston's representation is that all the links in his network must be directed. This poses a problem for symmetric relations like JOIN which would have to be represented by two facts like:

(A join B) and *(B join A)*

While this captures the essential information, it fails to make it clear that the two facts are the same. This becomes important if we receive the additional information that this join is at E. To represent this it would be necessary to add a TO link to *both* of the JOIN relations.

3.3. Local Matching

We use a local matching scheme to compare two descriptions cast in this semantic network representation. The matcher is based on the principle that *distance corresponds to importance*; the further away something is in the network, the less influence it has. For instance, in Figure 6 nodes B and C are more important to A than D is. Since PART OF can be considered a compound predicate, both B and C are one predicate away from A. The D node, on the other hand, is two predicates away from A; we can not get from A to D without first passing through C. This does not mean that the importance of each node is fixed, rather the importance varies depending on our location in the network. From node D, C is the most important node, followed by A and B. From node C's point of view, all the nodes are equally important. For the local matching procedure to work it is essential that the links in the network be set up so that distance really does correspond to importance. This is one instance of the "syntax as semantics" principle embodied in our representation.

Two networks are compared by gradually extending the matching horizon in a manner much like spreading activation [Quillian 1968]. Suppose we were matching an object to the model of Object-1 shown in Figure 7. At the first level the program only considers those things which are at a distance of zero or less from Object-1; the part of the model it sees is totally contained within the horizon marked "Level 0". It then compares the example to the model on the basis of all the predicates whose arguments are known. In Figure 7 there is only one such predicate: OBJECT. Assuming that the example is a satisfactory match to the model at this level, we extend the horizon one predicate further to the horizon labelled "Level 1". This involves finding analogs for Region-1 and Region-2 in the example object which satisfy the predicates within the

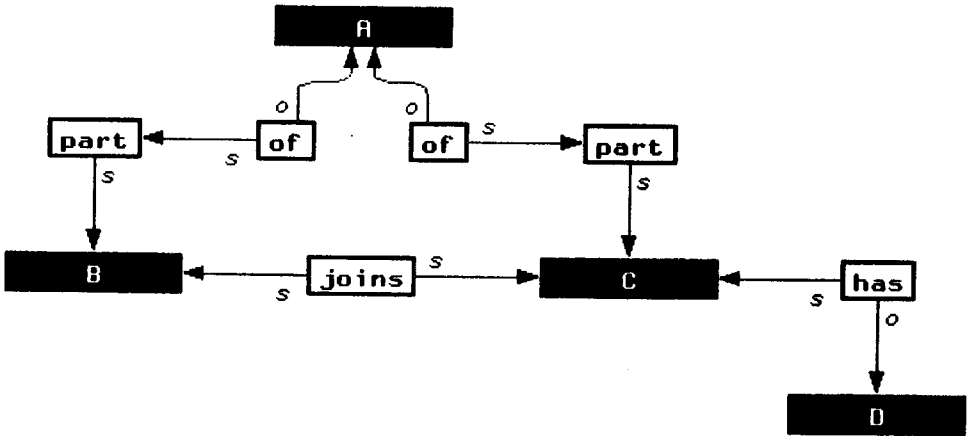


Figure 3-6. Distance corresponds to importance. B and C are more important then D from A's viewpoint. From C's viewpoint all the nodes are equally important.

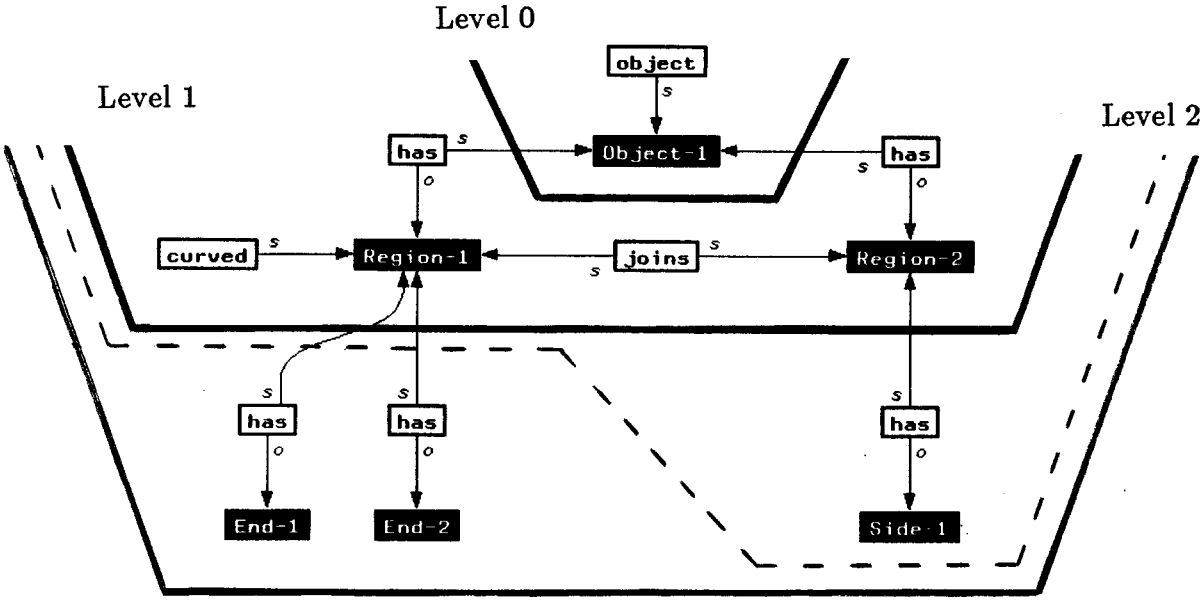


Figure 3-7. The local matching process gradually extends its matching radius to new predicates. If no match is found for Region-1 at level 1, the horizon would only be expanded to the dotted line at level 2.

current horizon. These two regions must each have a HAS link to the example object, the regions must JOIN each other, and the analog of Region-1 must be CURVED. Assuming suitable regions could be found, the horizon is then expanded to encompass

End-1, End-2, and Side-1 as well and the matching process continues.

Sometimes, however, it is not possible to find matches for certain things. Suppose that we were unable to find any region of the example object corresponding to Region-1 in the model. This might be enough to cause the match between the example and the model to fail at level 1. In a such a case the matching procedure *stops* and reports that the example matches the model down to level 0. That is, the example matches at an abstract level but not at a more detailed one. A similar thing could happen if, at level 2, the system found a binding for End-1 but not for End-2. This implies that the example object has a region which fits the coarse description of Region-1 but not the finer one. As before, the program stops trying to match Region-1 in more detail when this occurs. This means we must be careful to ensure that the matching levels truly correspond to levels of abstraction when we encode objects in this representation. Now, instead, suppose that, although no analog was found for Region-1, the program decides the example still adequately matches the model at level 1. In this case, the horizon is expanded to the dotted line shown in Figure 7. Since we can only compare predications in which all the arguments of the predicate are known, it makes no sense to expand the horizon out from Region-1 – any predications involving Region-1 would always have at least one argument that was unknown. Therefore the system effectively *cuts off* the whole branch involving Region-1 since it would be silly to look for the end of a region if the region itself could not be found. Once again, when encoding an object we must take this phenomenon into account so that these truncations accurately reflect the dependencies in the data.

To summarize, the local matching procedure progresses out from an initial set of correspondences as long as the example continues to match the model. As new correspondences are added, the system checks that they satisfy all the predicates applicable at that level. If, because of newly exposed details, a node no longer matches the model sufficiently well, search along the associated branch of the model is terminated. Otherwise, the system expands its horizon and attempts to find still more correspondences which are consistent with what it already knows. A worked example of this procedure can be found at the beginning of Chapter 5.

3.4. Gray Coding

Determining whether two regions look similar, a crucial part of the local matching procedure, involves comparing their shape parameters. The parameters of a shape are its length, aspect ratio, overall curve, and whether it is filled in or is a cut-out.

Conceptually, these properties fall into two classes: linear descriptors and Boolean descriptors. Comparing two Boolean descriptors is easy; if they match the objects are the same, if they do not match then the objects are different. Generalizing a description containing a Boolean descriptor is equally easy, we just omit the descriptor (this is the well known *drop link* heuristic). Linear descriptors, however, are not as easily compared and generalized. One approach would be to encode values using symbols that stood for small range of values. The problem with this is that in order to compare parameters the system has to understand the meaning of each symbol and then perform subtraction to get the answer. As mentioned before, we wish to encode the similarity metric more directly in the representation than this.

0	**0**	**0**	=	*0*
0	**0**	**1**	=	*1*
0	**1**	**1**	=	*2*
0	**1**	**0**	=	*3*
1	**1**	**0**	=	*4*
1	**1**	**1**	=	*5*
1	**0**	**1**	=	*6*
1	**0**	**0**	=	*7*

Figure 3-8. Gray numbers representing the values 0 to 8. Notice that only one bit changes as the values ascend.

We encode linear descriptors by a technique we call *Gray coding*. This technique was inspired by the Gray numbers [Hamming 1980, Davison and Gray 1976] used in digital communication. There, the idea is to represent each number such that toggling any one bit only changes the number represented by one. Figure 8 shows how the number one is encoded as 01, two as 11, and three as 10. In this way, the semantic distance between two numbers is reflected in the syntactic Hamming distance (how many bits are different) between their representations. A similar technique has been developed in phonology [Kenstowicz 1979, Chomsky 1968] for representing the features of vowels. This form of representation has two nice properties. First, to

compare two numbers all we have to do is *count* the number of bits that are different. Second, a good generalization can be made by simply *dropping* bits. For instance we could represent the interval [1..2], a generalization from the two values 1 and 2, by X1 where X stands for "do not care". This is the effect of the *close interval* heuristic proposed by Dietterich and Michalski [1981]. Unfortunately the encoding shown limits us to intervals of size 2 or 4 (one or two "do not cares"). We are also stuck with a fixed set of end points for intervals; we can not, for instance, represent the interval [0..2].

To encode intervals we take the table in Figure 8 and turn it on its side. As shown in Figure 9, this means we cover the number line with a series of overlapping ranges. Each range is associated with a Boolean variable which is true if the value being encoded falls in this range and false otherwise. A particular value is encoded by the set of intervals it is in: 5 is encoded as (e f g). Moving slightly to either direction gives us (d e f) for 4 and (f g h) for 6. As claimed, a small change in value leads to a small change in representation. We can also represent any generalization we want by using a smaller set. For instance, the interval [4..5] is represented by the set (e f), a set which subsumes the sets corresponding to 4 and 5. The partitioning of the number line shown in Figure 9 gives us greater flexibility than normal Gray numbers. We can use this partioning to represent intervals of width 1, 2, or 3 and these intervals can have any value we want as their end points.

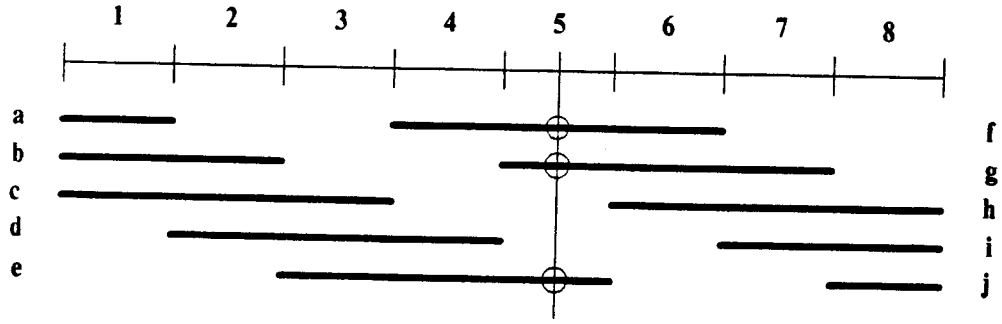


Figure 3-9. Gray coding intervals. The value 5 is represented by the set (e f g).

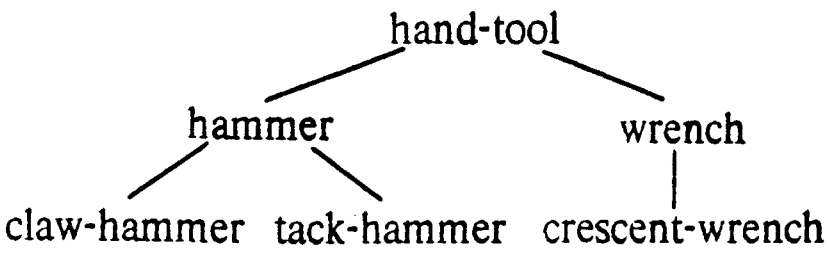


Figure 3-10. Gray coding hierarchies. A crescent wrench is represented by the set (tool wrench crescent).

Gray coding can also be used to represent hierarchies. As shown in Figure 10, we do this by associating a different Boolean variable with each leaf and branch-point in the tree. This variable is true if the item being encoded lies beneath it in the tree and false otherwise. For instance, a tack hammer is represented by the set (tool hammer tack). Using this representation, it is evident that a claw hammer, (tool hammer claw), is closer to a tack hammer than is a crescent wrench, (tool wrench crescent). Furthermore, we can achieve the effect of the *climb tree* generalization heuristic by once again simply leaving things out of the set. A generalization of (tool hammer claw) and (tool hammer tack) is (tool hammer) which represents the branch point directly above the two in the tree.

Gray coding has two important kinds of extensibility. First the resolution of a Gray code can be increased. Suppose we initially divide an interval into eight discrete ranges and create the corresponding Gray code predicates. Later we find that eight divisions are not enough, we really need more like sixteen divisions. In such a case the granularity of the representation can be made finer by simply adding more predicates. A particular number will now be represented by a set of four predicates instead of a set of three. One way to do this is by adding predicates corresponding to ranges which are same width as the original set but which are offset by half a unit. Another possibility is to add ranges which are smaller than the original. Both of these approaches are

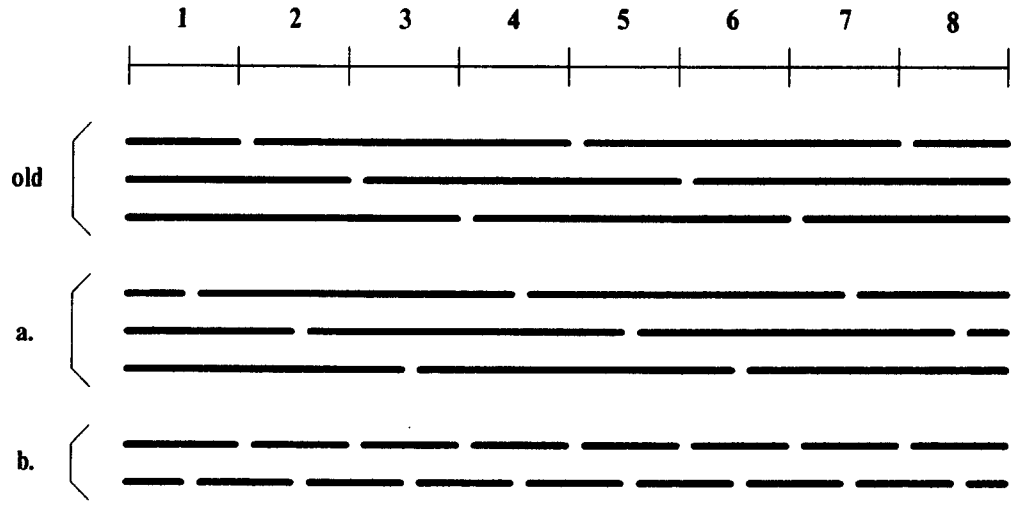


Figure 3-11. Increasing the resolution of a Gray coded interval. a. Adding more predicates of the same width as the originals. b. Adding new narrower predicates. We prefer the narrow predicates because they can be represented using the modifier technique.

shown in Figure 11. In our system we use the narrower predicates approach since it meshes neatly with the modifier technique to give us properties like VERY curved. Similar extensions can be made for hierarchies: not only is something an end, it is the end OF of a particular region; not only is some object a hammer, it is a CLAW type hammer. The key point is that the same mechanisms as described before will continue to work on these new expanded representations.

Gray coding also allows us to add new data types to the system without changing the similarity metric or generalization procedures. In our system not only do we have ranges of shape parameters and classification hierarchies, we also have other types such as edges, regions, and objects. Instead of creating new procedures for telling, say, how similar two regions are, we can use the mechanisms we already have provided that we encode the information about regions appropriately. Similarly, instead of defining a new generalization mechanism for every type introduced, we can use the generalization procedure outlined above. Thus, we can have an extensible type system without incurring the overhead of a plethora of mechanisms. The trick is properly encoding the new type so that the old procedures will do the right thing.

One example of a new type is a region. We have already shown how individual

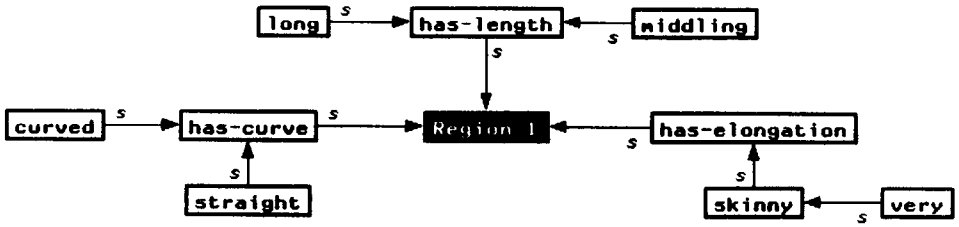


Figure 3-12. In the representation of a region the Gray code predicates for each parameter are grouped separately.

shape parameters are compared, we now need to compare whole regions as well. Although we have been discussing shape parameters as if they were predicated of the region itself, this is not true. We actually *group* the predicates for each parameter separately as shown in Figure 12. We can now look at a region as having its own set of Boolean predicates, LENGTH-X, ELONGATION-Y, and CURVATURE-Z, which can be used to compare the region to other regions. If we had attached all the properties directly to the region, the similarity metric for regions would be wrong. A small difference in each the length and the curvature would look as bad as a totally different length because the same number of properties would fail to match in each case. Grouping the related properties together allows us to *isolate* the small changes from each other. The changes are isolated because at each level of modification we determine whether the parameters are reasonable matches. In the case of the two small changes the system deems that the length and curvature are reasonably close. It then reports that the length and curvature really do match, hiding for the moment that there was a small discrepancy. The region sees that all three of its parameters match the parameters of the model and hence the whole region matches the model region. In the case of a total mismatch in length the region sees that its curvature and elongation match but that its length does not. Thus, the region is not an exact match to the region in the model; at best it is only close.

3.5. Structural Approximations

As mentioned before, incorporating abstraction into the representation language is important. First of all, abstraction is useful for speeding up the matching process. The problem of matching two graphs is difficult and in the worst case can grow exponentially with the size of the graphs to be matched. By matching an example to a model at the most abstract level and then progressively matching the details, the problem can be reduced to a manageable size. The second thing abstraction does is to specify how to make generalizations. For two objects to share the same coarse-level description they must be roughly similar. Thus the levels in the representation delineate acceptable partial matches, each having a different degree of specificity.

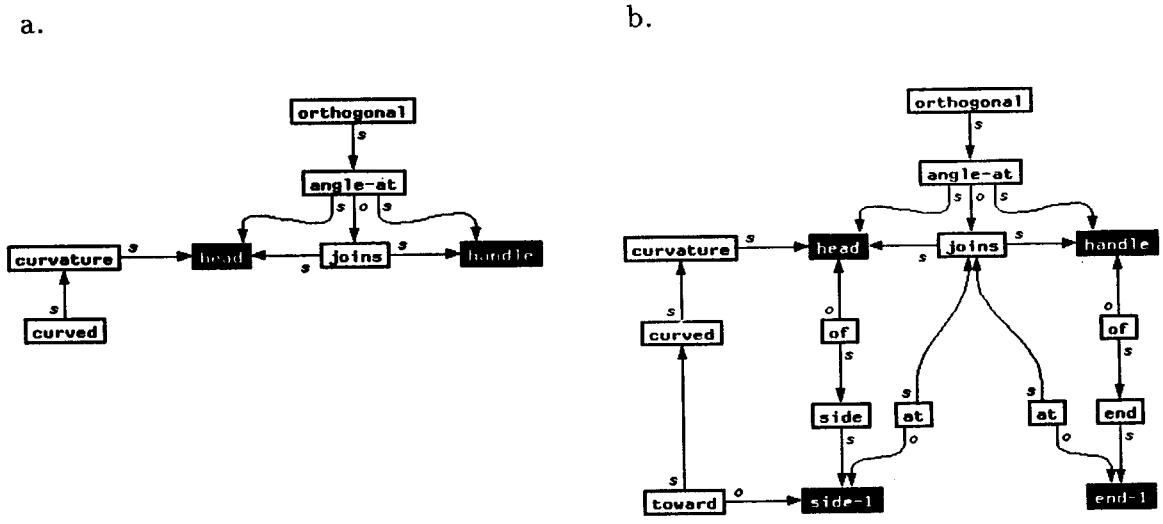


Figure 3-13. Edge abstraction. a. An abstract version of head to handle join. b. A more detailed version involving the edges of the regions.

An important facet of this work has been determining appropriate types of abstraction for shape descriptions. We have found several ways of approximating the structure of an object to be particularly useful in this respect. The first of these is *edge* abstraction which states that regions are more important than their boundaries. Figure 13 shows an example of edge abstraction. Using the local matching scheme, at the top level all we know is that the head of the hammer is elongated and curved

and that it is attached at a right angle to the handle. When we expand the horizon to include the edges as well, we can say which direction the head curves and more accurately specify its shape by examining the width and taper of the head at each end. Furthermore, by using the edges of the two parts, we can say exactly how and where the head joins the handle.

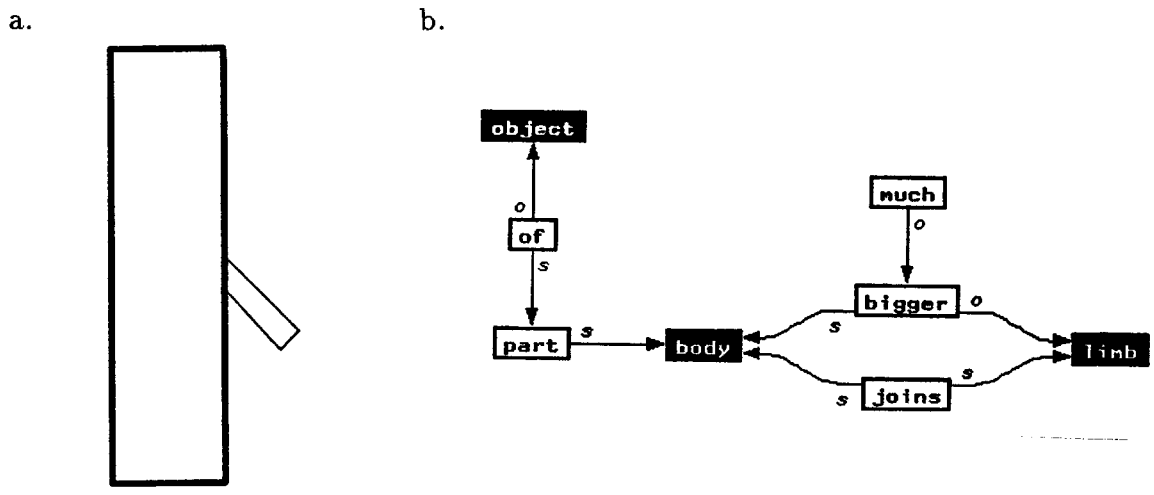


Figure 3-14. Size abstraction. a. The body is more important than the limbs. b. For this reason the body is closer to the object node than the limb is.

Another form of structural approximation is *size* abstraction which captures the fact that big pieces seem more "important" than small pieces. Small does not necessarily mean unimportant, however; the type of tip a screwdriver has is more important than the shape of its handle. Yet the importance of the tip of the screwdriver has nothing to do with its size, the tip is important because it determines the *function* of the tool. When we do not know anything of the function of some object, physical size is often a good indicator of perceptual salience. This is also known as the "body and limbs" heuristic and is shown in Figure 14a. The idea is to omit the small protrusion from the coarse level description of the object. This is done by putting the limb syntactically further than the body from the object node as shown in Figure 14b. Using the local matching scheme, the first horizon extends only as far as the node representing the body; the limb node is invisible. Later, if the body node is adequately matched, the horizon is expanded to encompass the limb node as well. This makes sense since the body provides a natural coordinate frame for describing

the limb and thus must be established before the limb can be matched. A good generalization of the objects structure can also be formed by simply forgetting about the small protrusion. This is exactly what the local matching scheme does when presented with two examples that have very different looking protrusions. While the shapes of the two example objects closely match at the first level, at the next level they are substantially different. Because of this the matcher stops the comparison at the first horizon and drops all mention of the protrusions from the model.

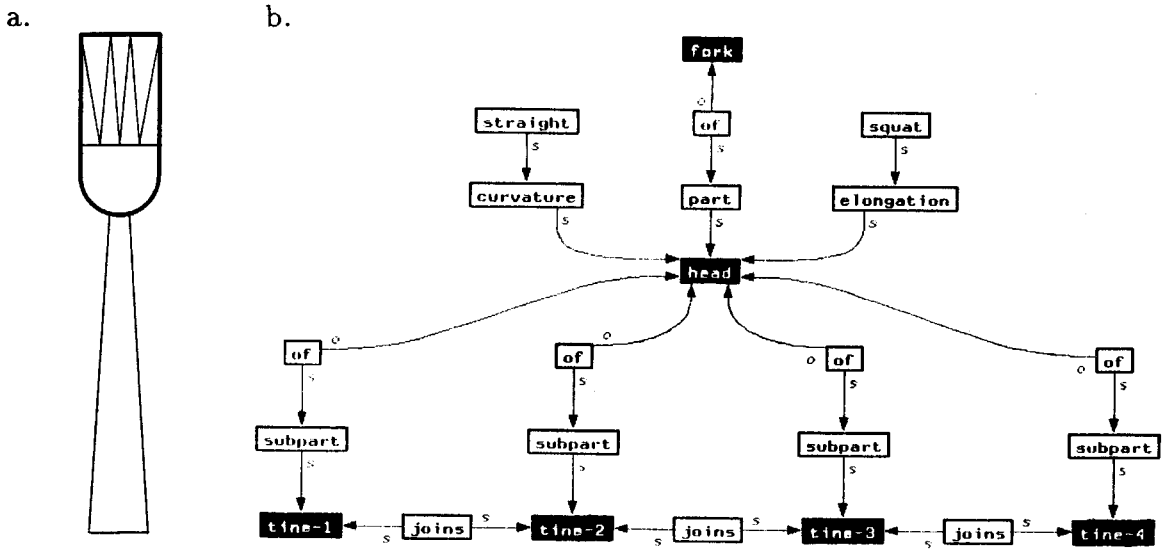


Figure 3-15. Envelope abstraction. a. The tines of the fork are subparts of its head. b. The tines are represented like limbs.

The third form of abstraction, *envelope* abstraction, is similar to size abstraction. In Chapter 2 we discussed relaxing the "non-overlap" condition for the head of the fork. As shown in Figure 15a, the head of the fork is defined largely by the envelope of the tines. The head is more important than the tines not only because it larger, but also because it completely *subsumes* the region described by the tines. For this reason, the head is represented in Figure 15b as a rectangle whose internal structure is further specified by the tines. The rectangle description helps support functional analogies such as seeing a fork as a shovel [Agre 1985]. It also lets us match a "spork", a combination of a spoon and a fork which has a much more prominent head, to the fork model.

The last type of structural approximation is *chain* abstraction. The idea is that several parts connected end-to-end, as in Figure 16a, form a special kind of larger

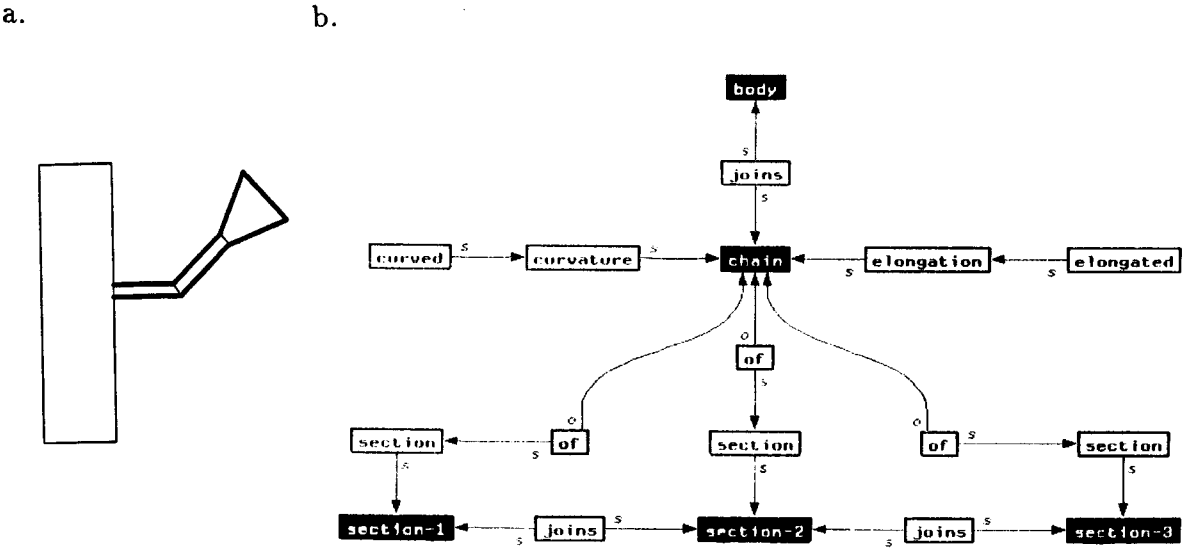


Figure 3-16. Chain abstraction. a. The limb is composed of three different parts. b. These parts are grouped into one unit in the representation.

part. This larger part has certain properties, such as size, elongation, and curvature, that represent the ensemble. The representation of a chain of parts is shown in Figure 16b. Chains are important because of the connectivity requirement discussed in Chapter 2. Removing any of the parts, other than the ends, causes the object to be broken into two different objects. When the chain is connected to a larger piece, as in Figure 16a, only the most distal section can be removed without fragmenting the object. Representing chains as shown in Figure 16b also helps the system tolerate the fixed thresholds employed in the symmetry extension algorithm. The last section of the chain in Figure 16a is a different piece from the middle section only because of the "local convexity" condition; if the piece had been slightly less tapered it would have been joined with the middle section to form a single part. Similarly, the middle section is distinct from the innermost section only because of the "no major bends" condition – decreasing the angle between the two sections would cause them to be joined. While the substructure of whole limb would be different in these cases, the overall properties of the chain are unchanged. A related topic, the use of chains to recognize different configurations of the same object, is discussed further in Chapter 5.

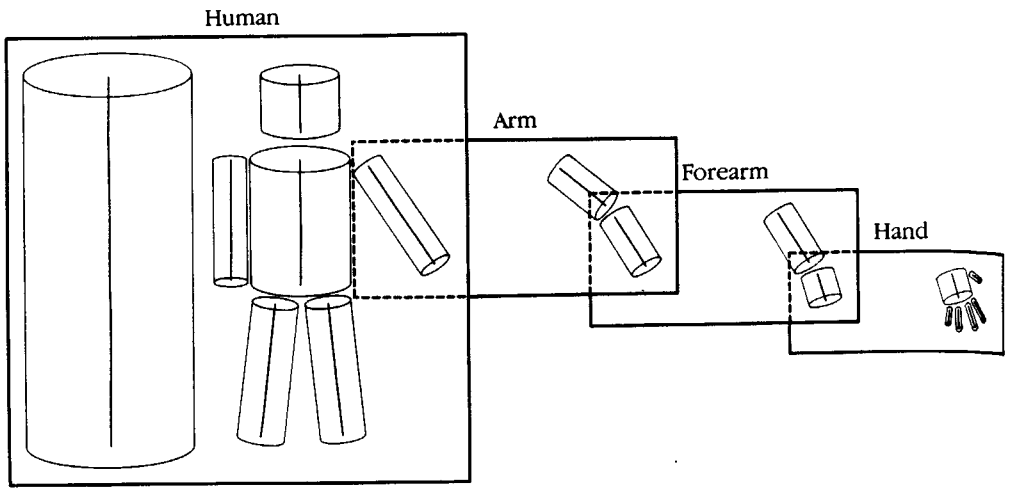


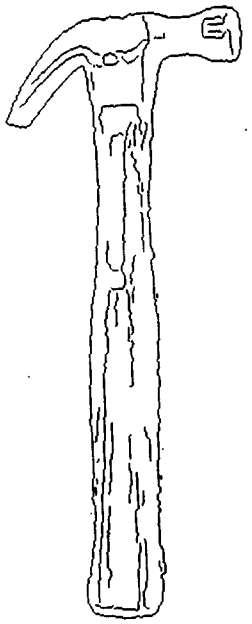
Figure 3-17. The decomposition of a human figure as suggested by Marr. Several examples of size, chain, and envelope abstraction are evident.

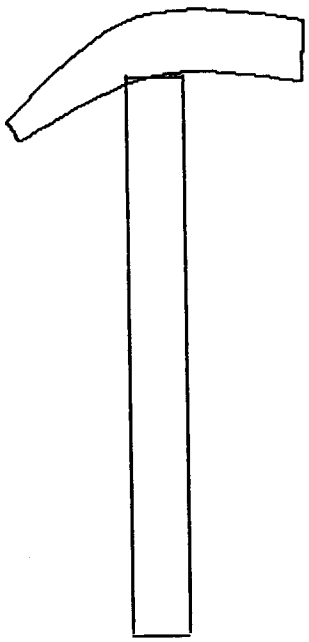
Many of these forms of abstraction were implicit in the work of Marr and Nishihara [1978]. Figure 17, for instance, contains examples of size, chain, and envelope abstraction. However, Marr never specified exactly why the object was broken down in the form shown nor how to accomplish this. We have answered not only these two questions but have also shown how to represent structural approximations and how to use them in forming generalizations.

3.6. Sample Descriptions

The following pages show the full semantic nets computed by our program for a claw hammer and a Lockheed 1011 airplane. These networks are very large and have been broken down into page-sized sections for easier viewing. We also show, for each object, a picture of the object drawn by the program based on the object's semantic network description. The edges of the original image are presented for comparison.

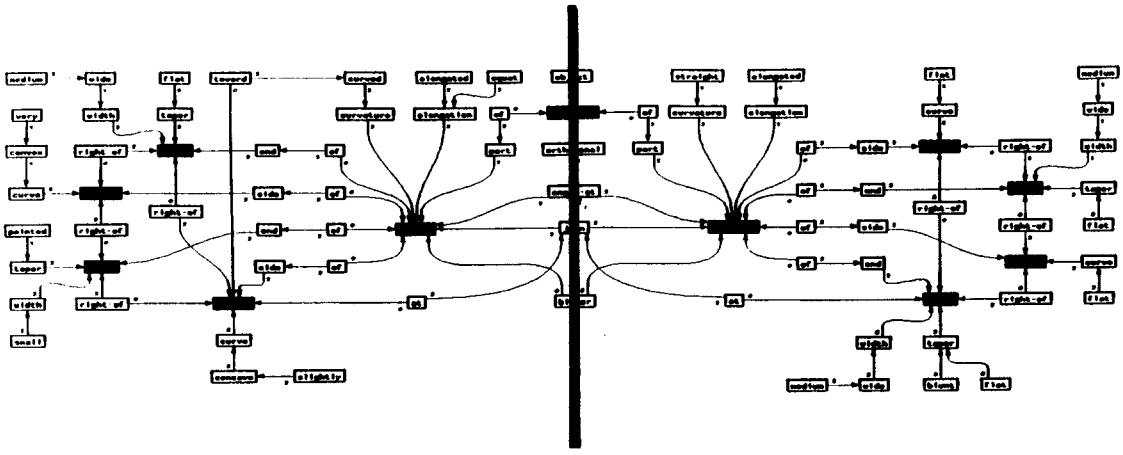
Claw Hammer



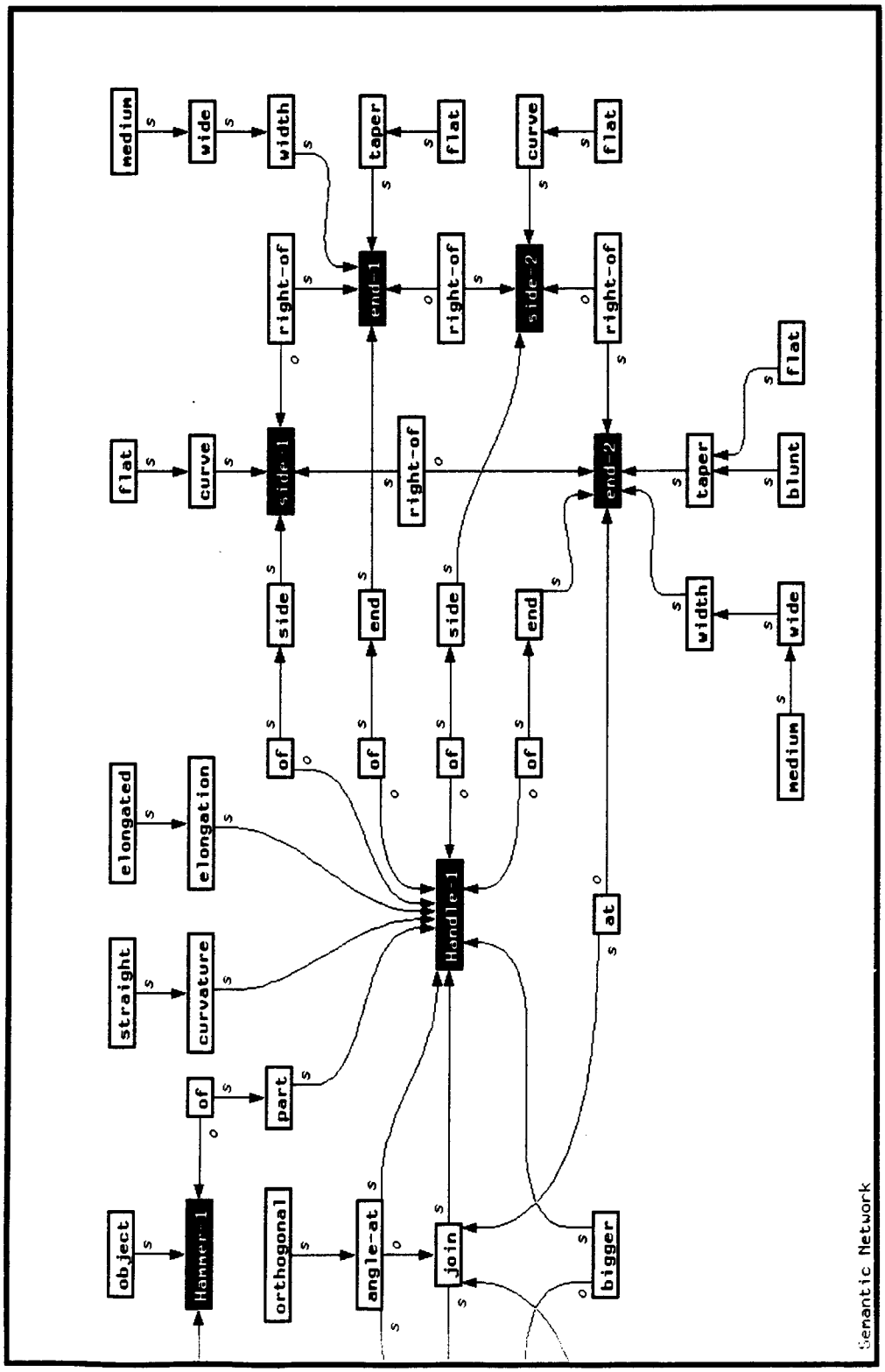


Head

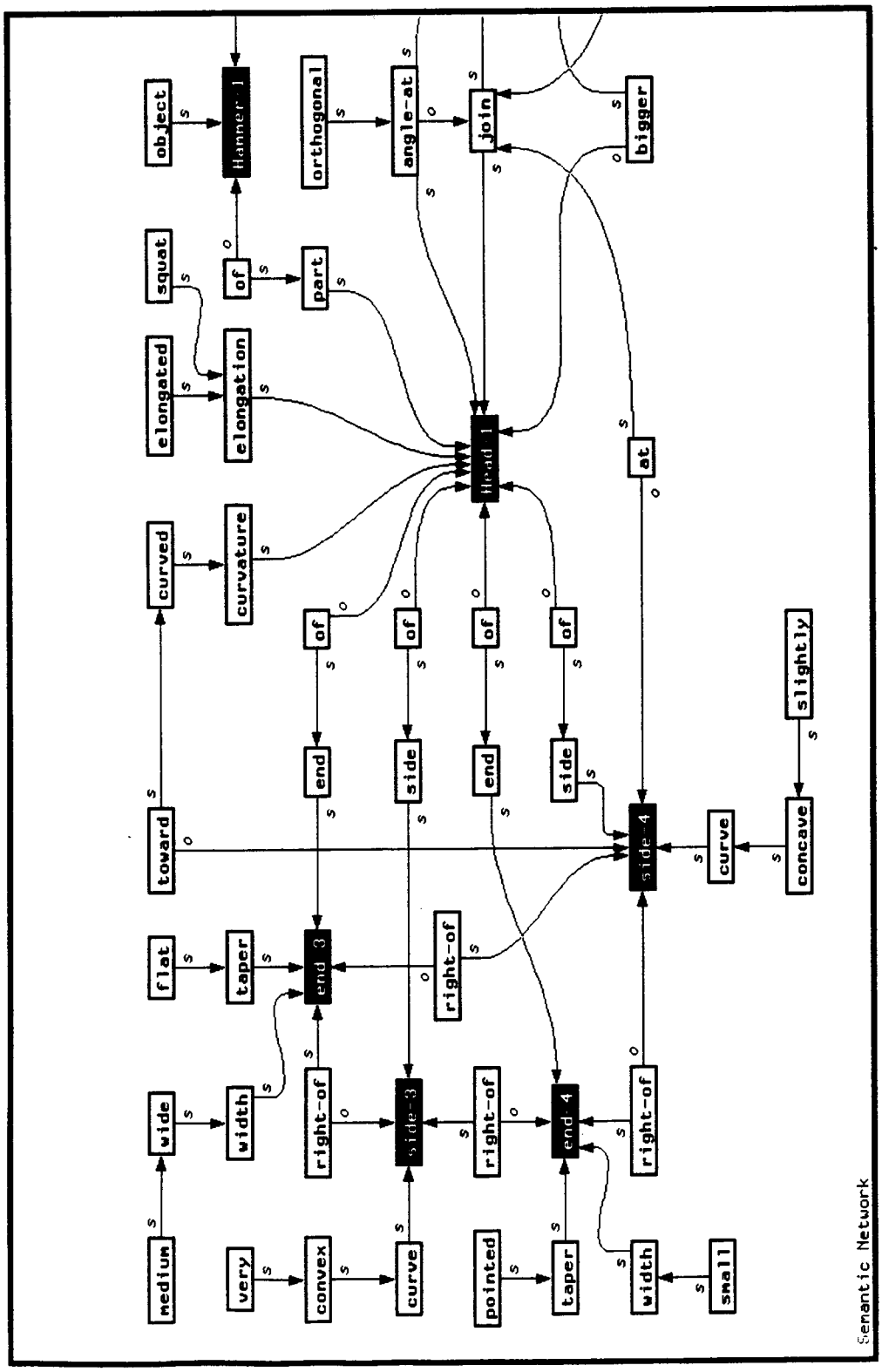
Handle



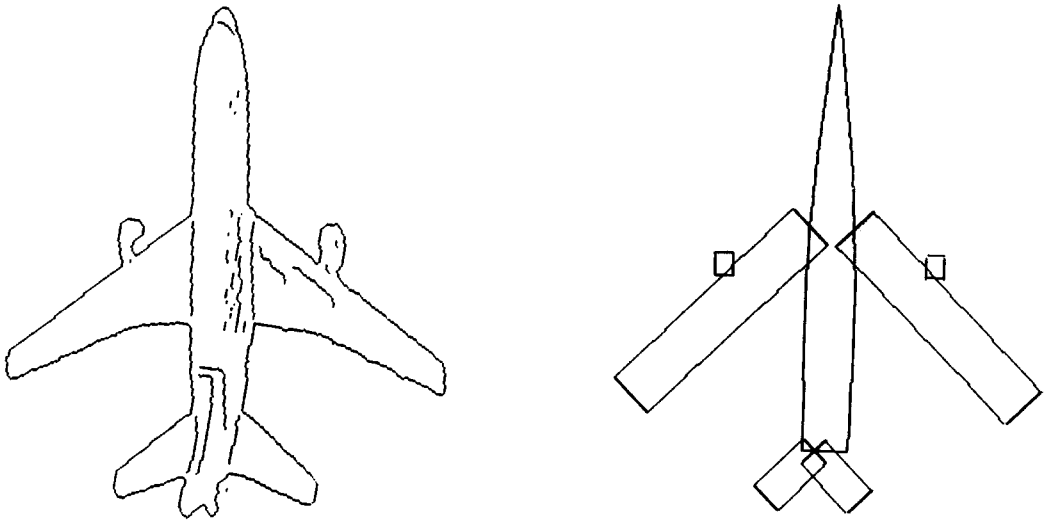
Claw Hammer (handle)

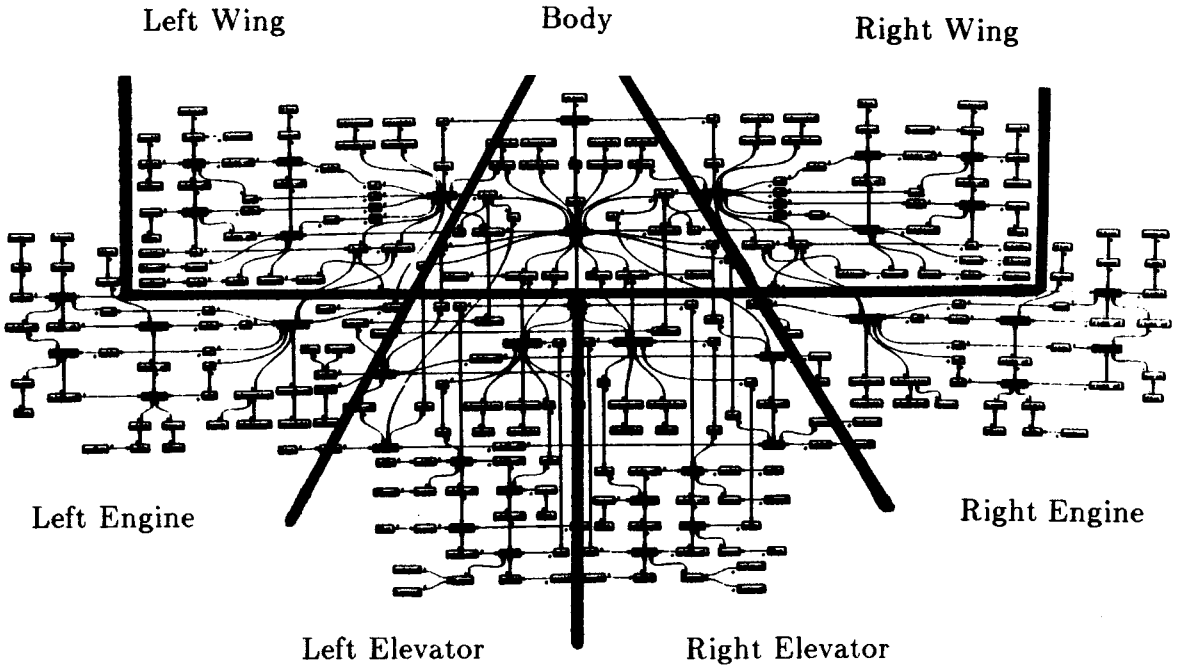


Claw Hammer (head)

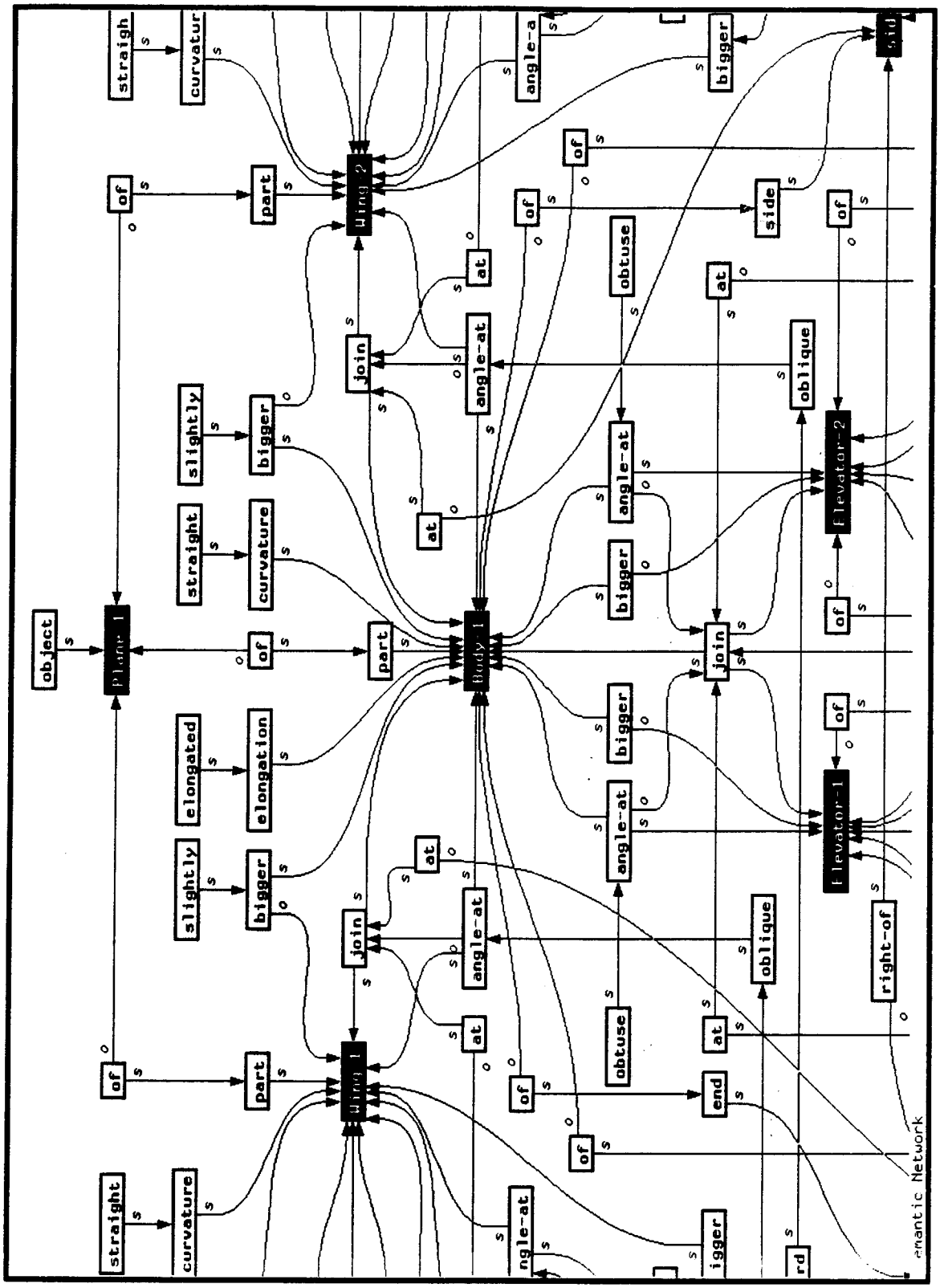


Lockheed 1011

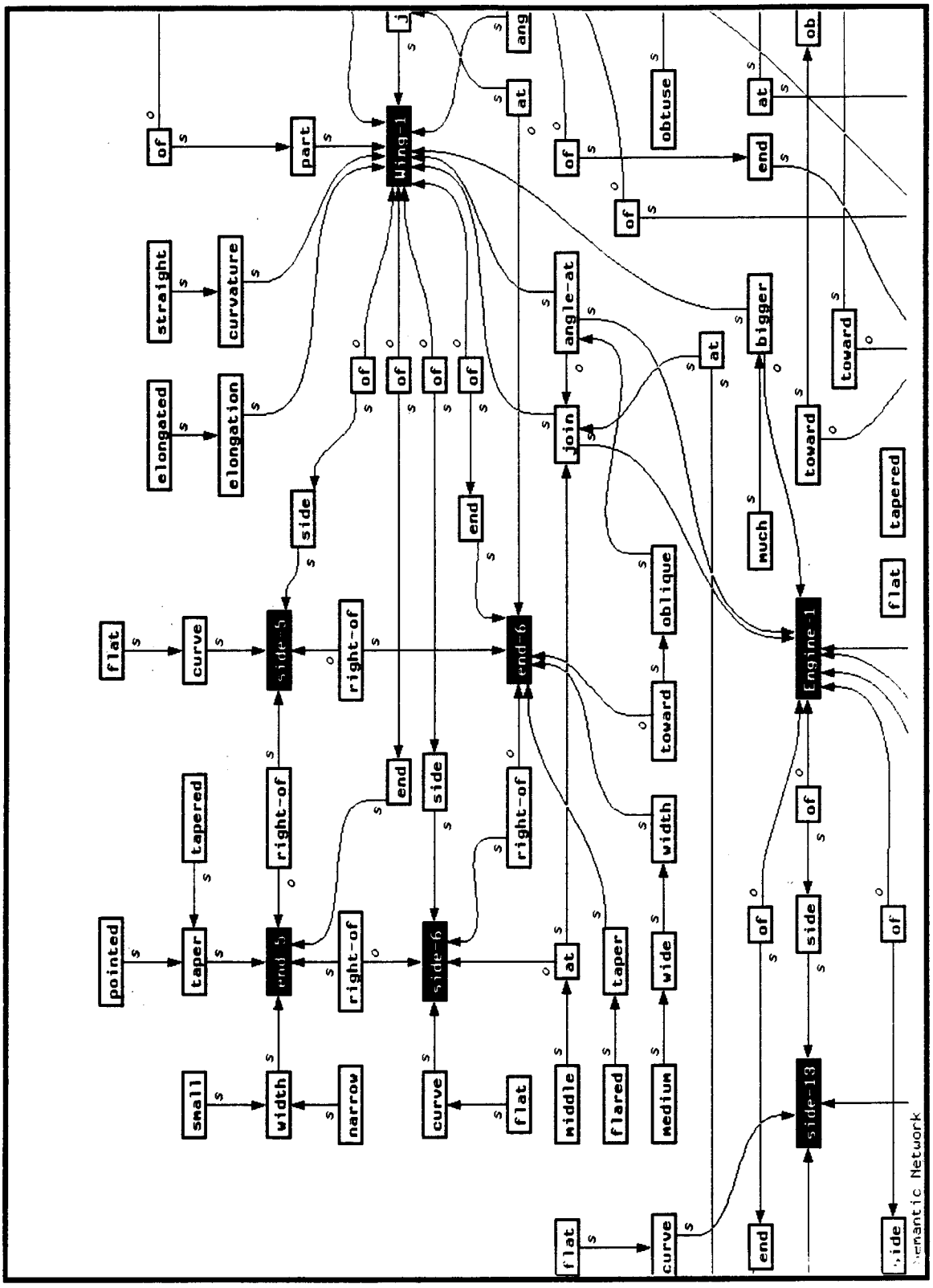




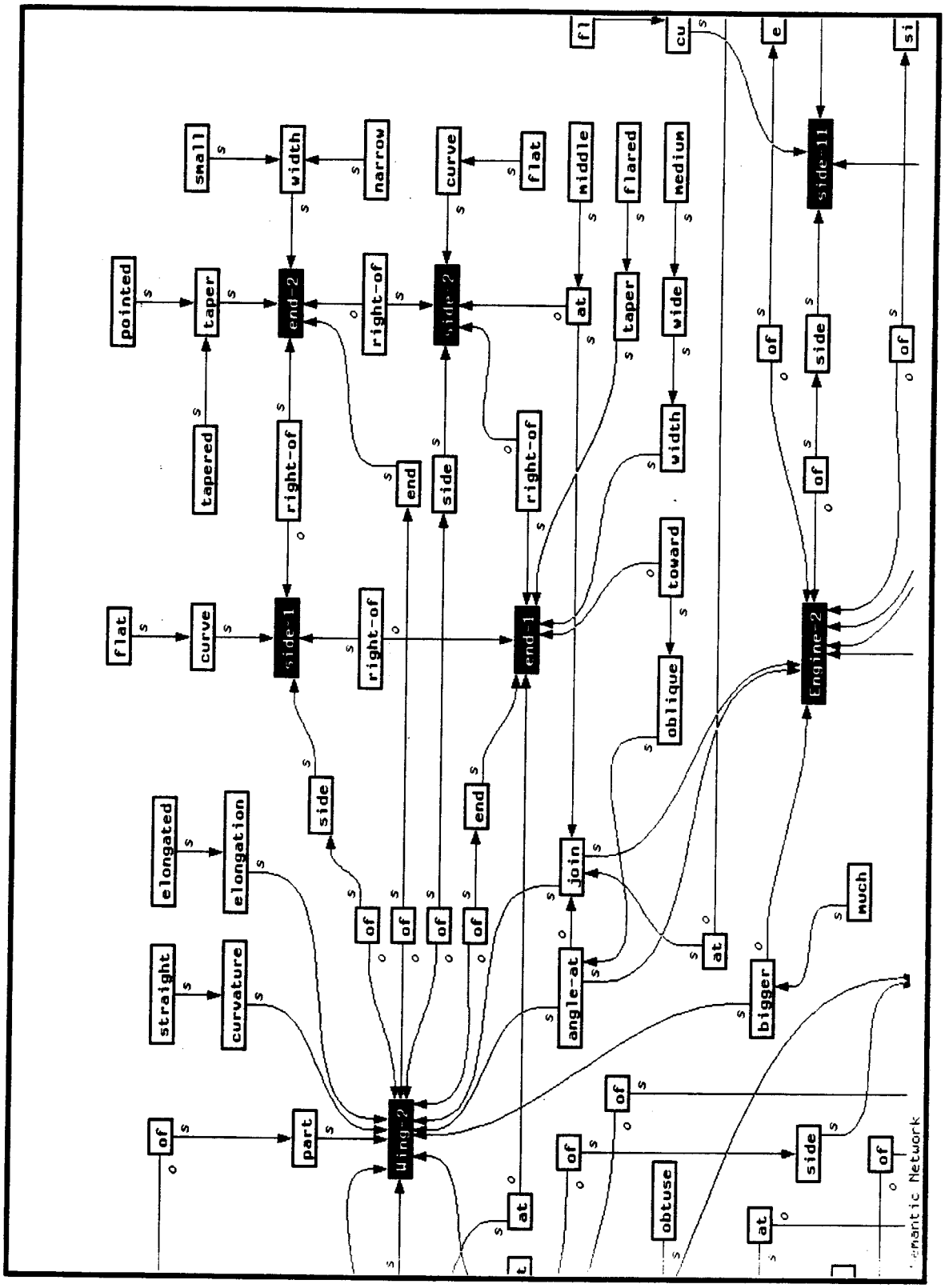
Lockheed 1011 (body)



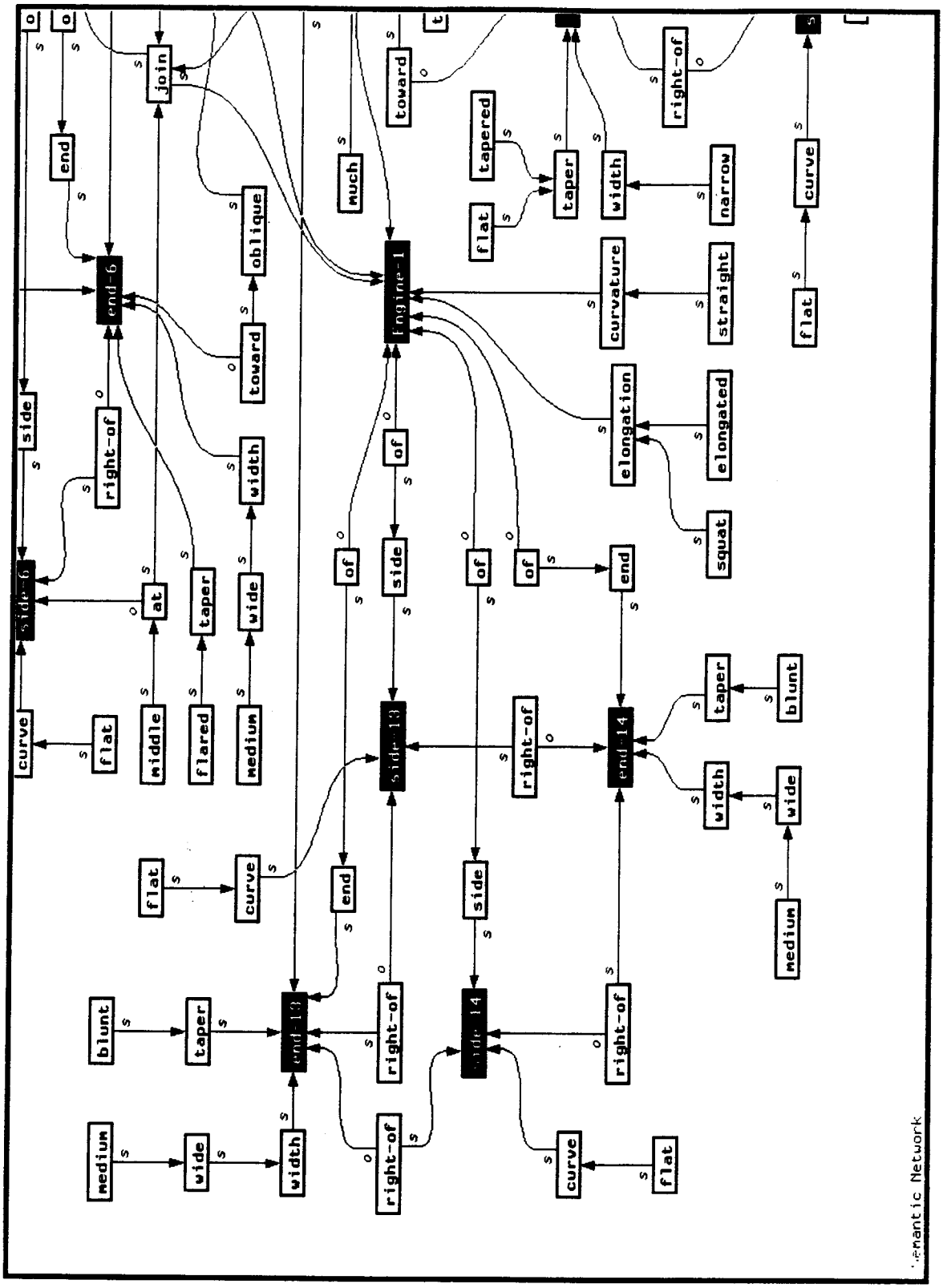
Lockheed 1011 (left wing)



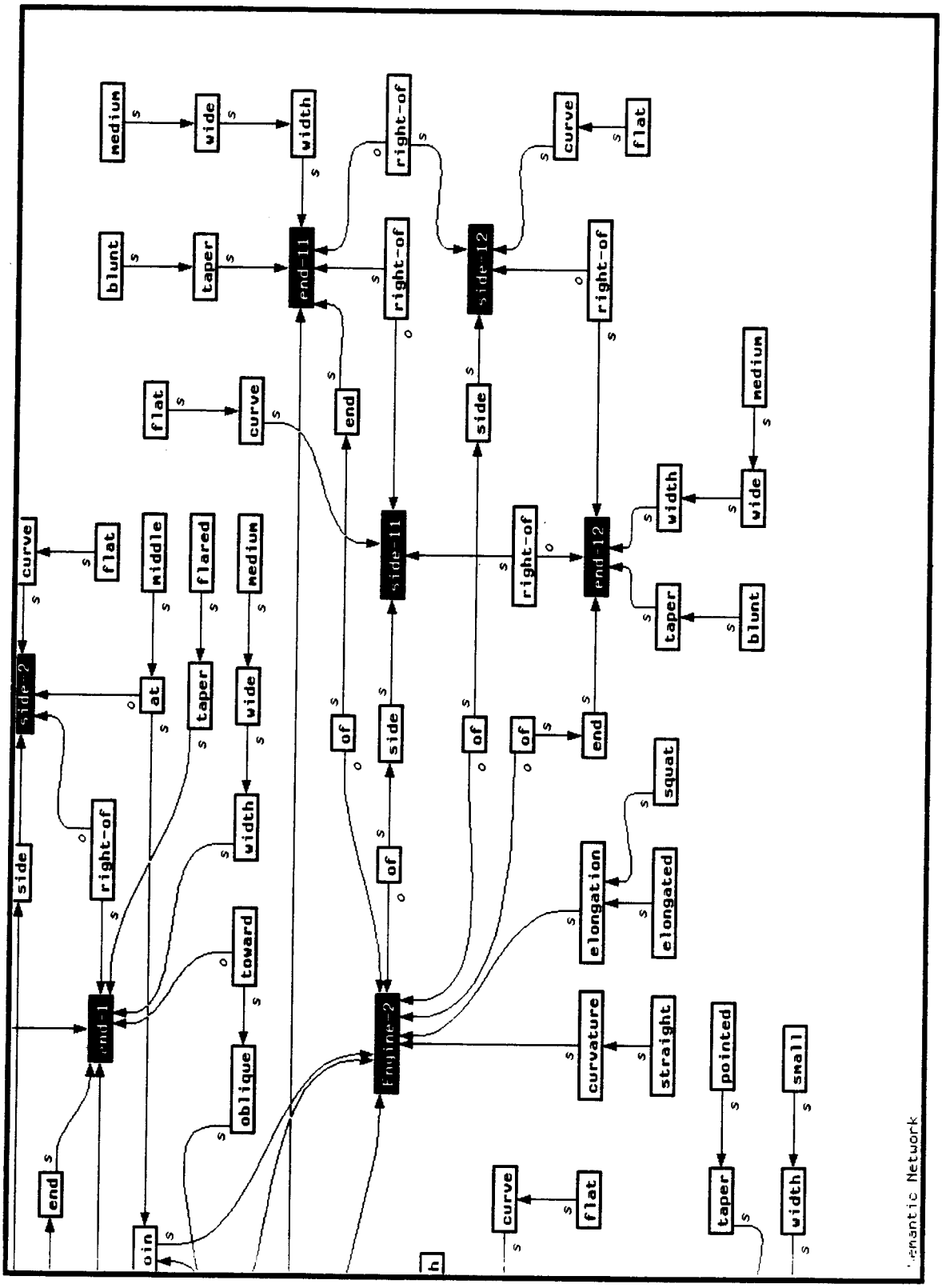
Lockheed 1011 (right wing)



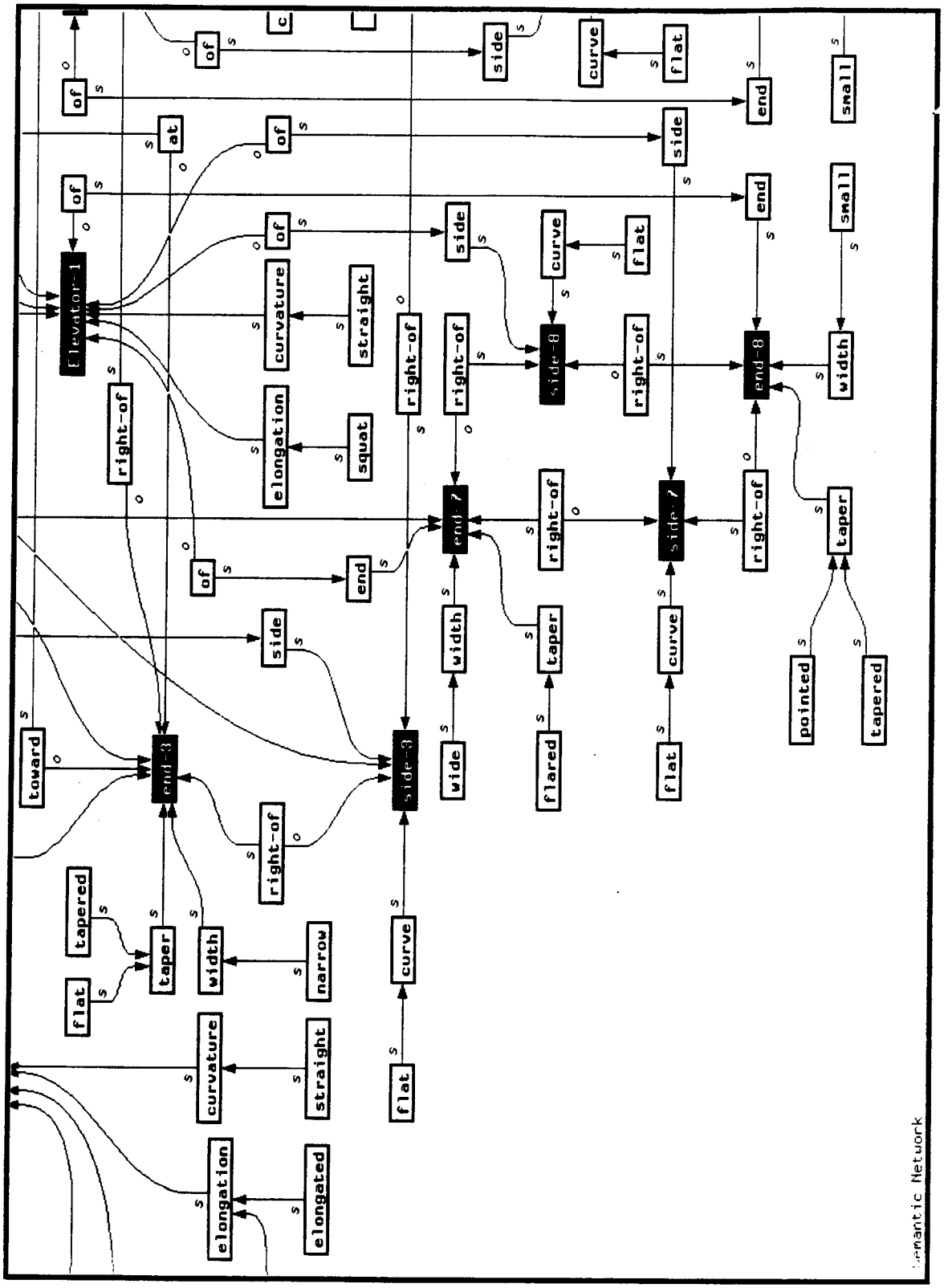
Lockheed 1011 (left engine)



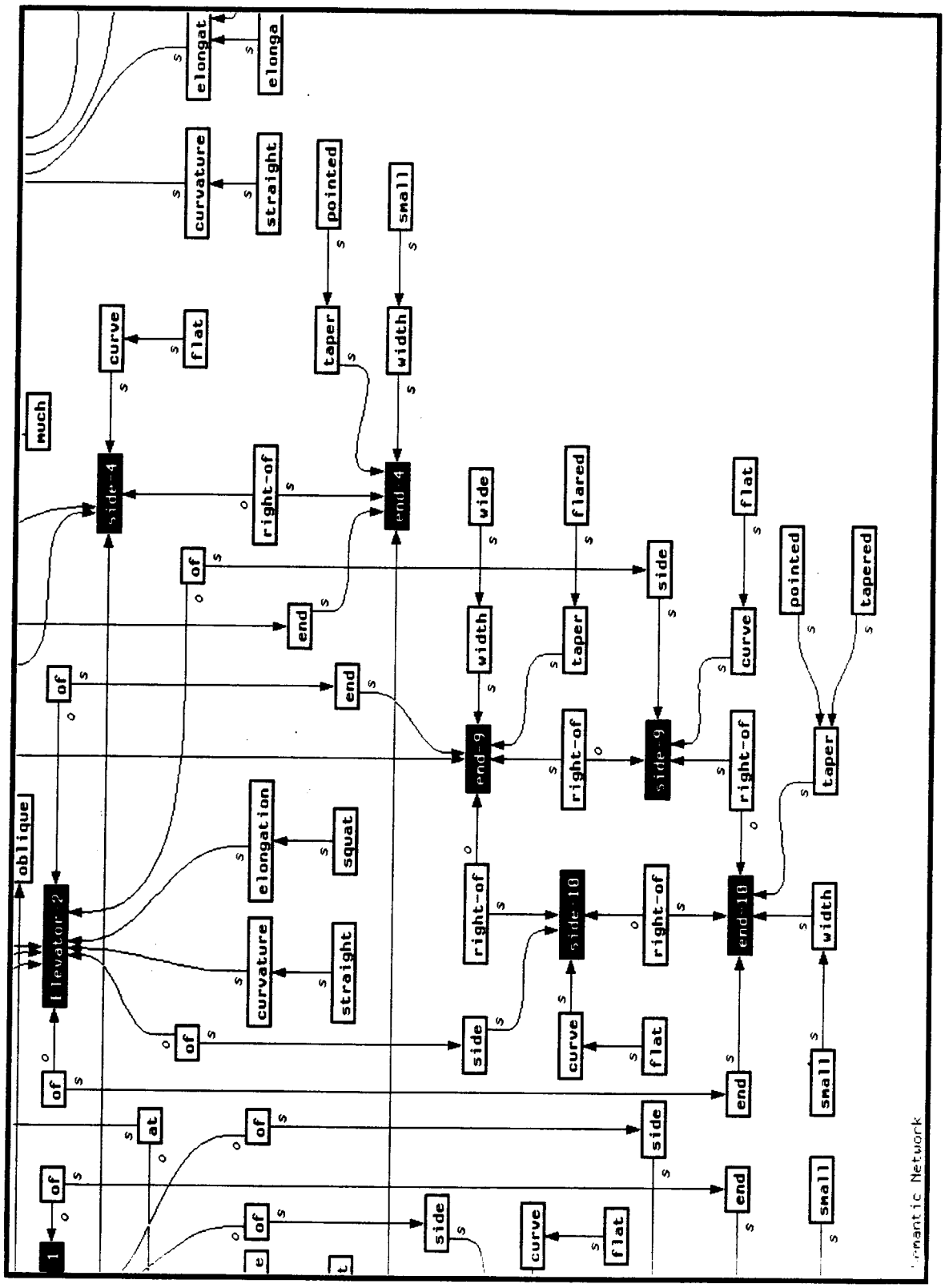
Lockheed 1011 (right engine)



Lockheed 1011 (left elevator)



Lockheed 1011 (right elevator)



4.1. General Principles

The primary goal of this thesis is to show how models of objects can be learned using visual data. This is accomplished in our system by presenting several example objects and saying whether they belong to a particular class or not. The learning program then creates a prototype description which can be used to recognize other members of the class. For the learning program to operate reasonably it needs to:

- be capable of learning from positive examples only.
- make an educated guess if it is not sure of its answer.
- not make any rash generalizations.
- be able to recover from over-generalizations.

There are various rationales behind these design criteria. The reason for the first of these, *positive examples only*, is that humans seem to be able to learn what things are without ever being shown counter-examples. Counter-examples are used to refine concepts and make fine distinctions; they are not necessary to learn the basic concept. This is not to say that the program can learn what something is without learning what it is not, rather we claim that this learning can occur without explicitly presented negative examples. This is possible because each model of a class contains an *implicit* description of its complement. That is, given a class model, there are things which are so obviously different that they can not possibly belong to the class. The set of all such grossly different things is the inverse of the class. Thus by knowing what is in a class and having a suitable similarity metric, we also know what is not in the class.

The second criterion, *educated guessing*, is largely a performance consideration as it deals with how the models are used rather than how we construct them. In the real world, not everything falls neatly into a class having a compact description, nor do we always completely learn a concept before being required to apply it. In both these cases it seems reasonable to attempt to answer rather than giving up in confusion. This is especially important when some definite answer is required, either a "yes" or a "no", and giving up is interpreted as "no". By making educated guesses

the system is more likely to give the correct answer in ambiguous cases and in the face of incomplete models.

The other two criteria constrain how models are modified. The *no rash generalizations* heuristic is meant to make the system less gullible. Suppose that the system only knows about the shapes of things and we tell it that both humans and whales are mammals. The shapes of these two objects are grossly different, so different that if we were to make a generalization based solely on shape information it would amount to "All things are mammals". This is clearly wrong. The system should be more skeptical than this; it should hestitate before discounting most of what it knows about the members of the class of mammals. However, no matter how careful the system is, it is bound to make mistakes like this every so often. This is the reason for the fourth desideratum, *recovery from over-generalization.* If mistakes are going to occur they should be correctable; they should not be allowed to cause permanent damage.

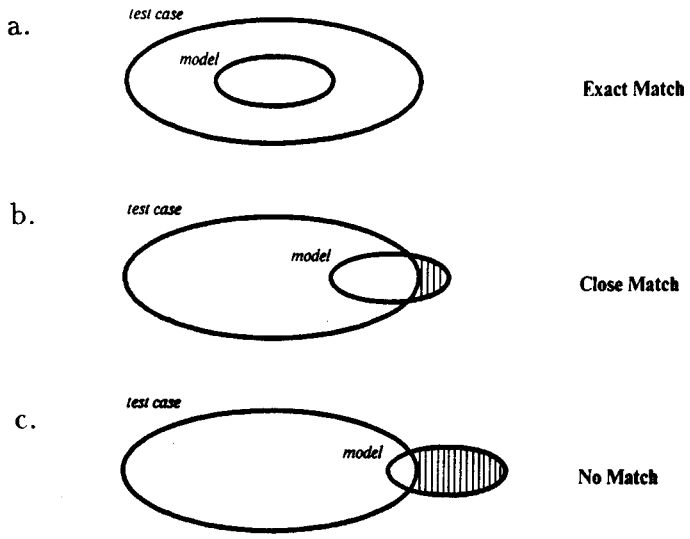


Figure 4-1. Possible outcomes of matching an example to the current model: a. exact match b. close match c. no match. The circles represent sets of features.

It is useful at this point to introduce some terminology involving the outcome of the matching process. This terminology will help explain the algorithm we have developed and further clarify several of the design criteria presented above. When comparing an example to an existing model, the example can either exactly match the model, closely match the model, or fail to match altogether. These three cases

are depicted in Figure 1. The circle labelled "model" represents the features required to be a member of the class. The circle labelled "test case" represents the features possessed by the example presented. Figure 1a shows an *exact match.* The test case has all the features required by the model and may have non-required features as well. In Figure 1b the test case is a *close match* to the model. The shaded area represents features which are required by the model but which are lacking in the example. Finally, Figure 1c shows the case of *no match.* Again the shaded area represents features which were required but which are not present. In this case we deem the discrepancy to be too large and thus the match fails.

These distinctions are intimately related to the functioning of the learning system. They help us define, for instance, what constitutes an "educated" guess. If an example is an exact or close match to a class model the system will say that the example really is a member of that class. This is part of the meaning of what "close" means; a close match is a reasonable guess to make. Close matches also help define what a "rash" generalization is. If the example is not reasonably close to the the current model then altering the model to cover the example is a mistake.

Obviously our algorithm relies heavily on this notion of closeness. This is a domain-dependent measure that accounts for a number of types of similarity between objects. For reasons previously cited, we have embedded this knowledge in our representation language. The learning system complements this metric much as the A-box cooperated with the T-box in KRYPTON [Brachman, Fikes, and Levesque 1983]. The job of the T-Box was to take care of all the special inheritance inferences made possible by the representation, while the A-box handled the other types of reasoning. In our system the similarity metric forms the equivalent of the T-box – the representation is specifically designed to support similarity comparisons – while the learning procedure fills the role of the more general A-box.

4.2. Creating and Generalizing Models

The response of the learning program depends on the outcome of the match between the example and our current model. First, suppose that we present the system with an example and tell it that the example is a member of the class being learned. If the example *exactly matches* the current model then the system will correctly state that the example is a member of this class. Since the system produces the right answer everything is fine; no further action is necessary as Figure 2a shows. Suppose, however, that there is *no existing model,* that this is the first example the system has

ever seen. In this case we create a model for the class based on the description of the example presented. This is the situation illustrated in Figure 2b. The rationale behind this action is that if we have only seen one example of a class, then that example is a good prototype for the class. Certainly all known members of the class (there is only one) will exactly match this model.

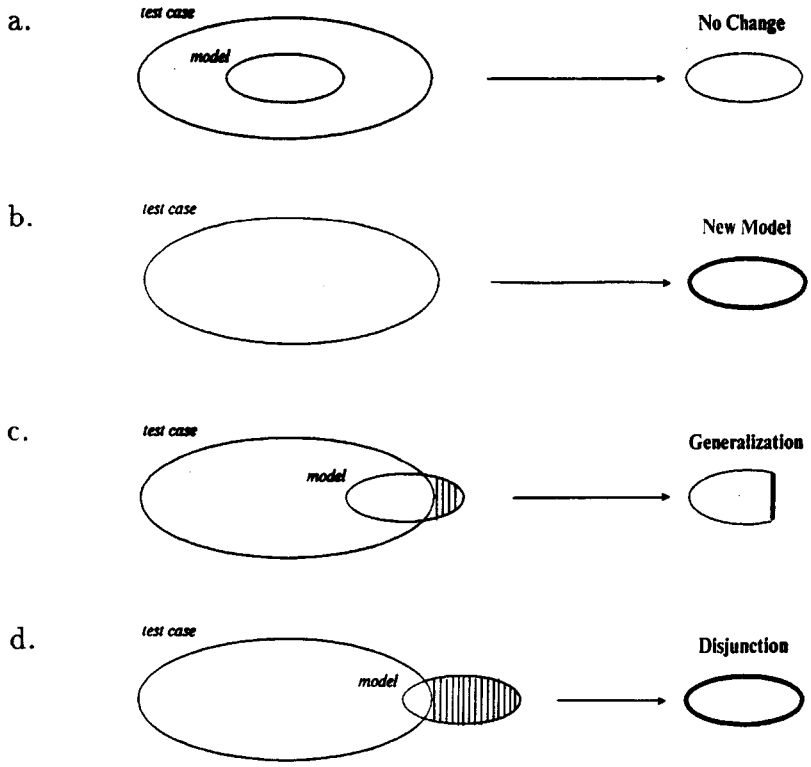


Figure 4-2. Learning from mistakes. The learning algorithm takes different actions depending on the outcome of the match.

If a model of the class already exists, it might be that the new example only *closely matches* this model as in Figure 2c . The system will correctly state that the example is a member of the class due to the "educated guess" heuristic. However, the example presented was explicitly included in the class – the teacher said it was a member. Under what Winston has dubbed "felicity conditions" [Winston 1984, VanLehn 1983], we assume that the teacher intended this to be a good example of the concept rather than some fringe case. The fact that this example was not an exact match to the class model then implies that our current model is inadequate; it should be extended to cover this positive example.

By our definition of "close" we know that it is appropriate to generalize the model

in this situation - the only question is how to accomplish it. Our approach is very simple: we *ablate* the current model by removing all the features that were not present in the example (the shaded region in Figure 1b). In general, this is an atrocious way to create generalizations. The reason it works in our case is that *we have carefully constructed the representation such that ablation produces meaningful generalizations.* This is one effect of forcing the syntax of the representation reflect its semantics: we can have a very simple procedure for creating generalizations. Anyhow, since we have removed all the unfulfilled requirements from the model, the example will now exactly match the model. The next time the example is presented the system will have no doubts as to whether it belongs to the class or not.

The last case is when the example presented *fails to match* the current model as in Figure 2d. This does not mean that there was no overlap between the example and the model, merely that the overlap was insufficient. In this case we create a new, secondary model for the class based on the example presented. Had we generalized the old model by intersecting its feature set with that of the example, we would have thrown away a great deal of the useful detail in the original model. We believe such drastic revisions are dangerous and should be avoided. Our solution is to represent the class using a *set* of prototypes much like in Mitchell's version spaces [Mitchell 1978].

Employing a set of models rather than a single prototype is invaluable when dealing with disjunctive concepts. Seemingly disjunctive concepts often arise in reasoning between form to function. For instance there can be many structurally different types of "handles". Some handles are curved like those on a briefcase or a cup, others are knob-like as are the handles of a dresser, still others are elongated sticks such as the handle of a hammer or a baseball bat. The thing that unifies all these examples is their function not their form. Similarly, the representation space that unifies whales and other mammals is reproduction. Yet we are often able to recognize the function of some object without actually using it; we can infer its function merely by looking at it. This suggests that sometimes visual information is all that is needed in order to classify objects with regard to their function. Learning such a concept on the basis of entirely geometric information, however, is hopeless unless we allow disjunctions.

4.3. Specializing Models

We have discussed how to generalize models but it is also necessary to specialize them sometimes as well. No matter how careful we are about altering the current models,

for instance, *over-generalizations* are bound to occur. When a model has been over-generalized not only does it cover all the members of a class, it also inadvertently covers some items which are not members as well. This problem is corrected by specializing the model so that it excludes all the incorrectly labelled items. Another case in which specialization is required is when there are *exceptions* to the rule. Exception is different from disjunction: the reason for creating a disjunction is some new example that is not covered by the current model while the reason for making an exception is some new non-example that is incorrectly covered. Once again the way to remedy this is by specializing the model so that it no longer covers the exceptions.

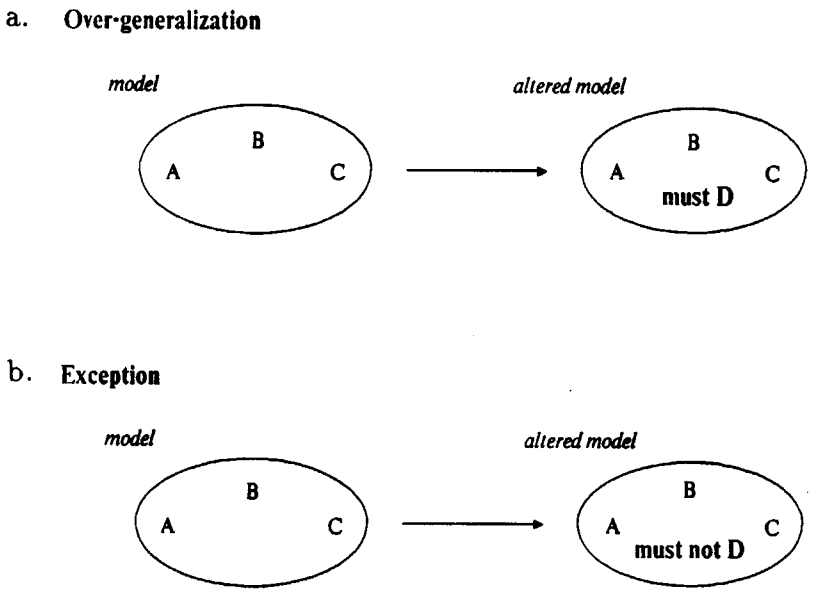


Figure 4-3. The use of two specialization heuristics. a. Over-generalization is countered by *requiring* a particular feature. b. Exceptions are rejected by *forbidding* some of their characteristic features.

To specialize a model we must present the system with negative examples, things which the teacher says are not members of the class. However, negative examples are only useful when they have been misclassified by the system. There is no problem if the system says that these items are not members of the class when they truly are not. Requiring the item to be misclassified is similar to Winston's claim that under "the no-guessing principle" the only useful non-examples are *near misses* [Winston 1984]. Remember that our system will not claim that a test case is a member of a class unless it is either an exact or close match to the current model of that class. Thus "far misses", in which there are many differences between the test case and the model, will never be deemed members of the class. The only kinds of non-example

that could be misclassified are near misses or exceptions.

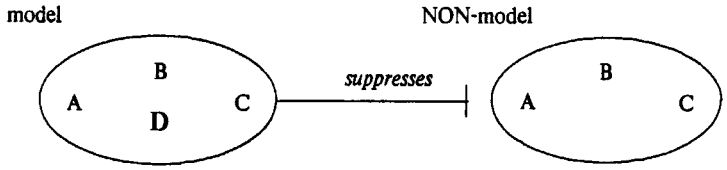
There are two heuristics which are especially useful for correcting over-generalizations and handling exceptions. They are known as *require link* and *forbid link* [Dietterich and Michalski 1981] or, equivalently MUST have and MUST NOT have [Winston 1970]. These two heuristics are used as shown in Figure 3. Over-generalization usually occurs because we have thrown away some useful feature of the model. The simple solution to this problem is to put the feature back by making it a required feature. Exceptions, on the other hand, are generally distinguished by some particular feature or set of features not present in the model. Thus, to recognize that an exception is not a member of the class we make all these features forbidden.

4.4. Non-Models

Instead of having features of different strengths as suggested in the last section, our learning program uses *combinations of models and non-models* to cope with over-generalization and exception. Whenever a non-example is misclassified, we create a new non-model based on the non-example much as we created a new, regular model for a positive example which was not covered by the existing model. This is again similar to the specialization technique used in version spaces [Mitchell 1978]. Non-models have the same status as regular models; they can participate in disjunctions, be generalized, and be specialized. They follow all the rules listed in Figure 2. The only difference is that they explicitly delineate what is *not* in the class.

Non-models allow us to achieve the effects of MUST and MUST NOT. As shown in Figure 4, *require link* is implemented by having a negative non-model which is identical to the positive model except that it lacks a single feature. If we tell the system that something which matches both the model and the non-model is not a member of the class, the system will never decide that a test case lacking this one feature is a member – the feature has become mandatory. The second heuristic, *forbid link*, is implemented in a similar fashion. This time we construct a non-model which is exactly the same as the original model except that it has the extra, forbidden feature as well. Now any test case which has all the features required by the model but also has the forbidden feature will match both the model and the non-model equally well. However, as before, we specifically instruct the system to favor the non-model in the case of such a tie. Thus the system will decide that any test case having the forbidden feature is not a member of the class.

a. **MUST have D**



b. **MUST NOT have D**

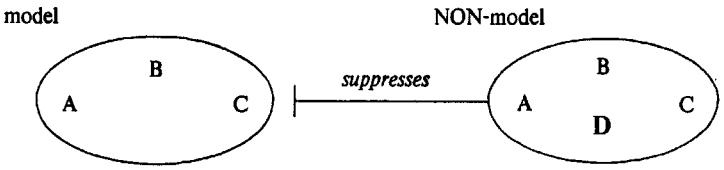


Figure 4-4. *Require link* and *forbid link* can be achieved using combinations of models and non-models. a. For *require link* the model has an extra feature. b. For *forbid link* only the non-model has the forbidden feature.

While we can explicitly instruct the system what to do in the case of ties, not all interactions between models and non-models lead to such conflicts. For instance, the situation in which a test case exactly matches a model and closely matches a non-model can be simply resolved without advice from the teacher. This is possible due to the conceptual difference between a close match and an exact match. If a test case exactly matches the concept then it is a bonafide example of the class; if it only closely matches then it is not a obviously a member of the class but we are willing to take a guess on it. Thus we have more reason to believe that the test case really is a member of the class than that it is not. In both this case and its converse, in which the non-model is exactly matched while the positive model is only closely matched, we can make an informed decision: the exactly matched model or non-model wins over the closely matched one. This is akin to the *maximum specificity* heuristic used in many production systems.

When there is a real conflict we await a verdict from the teacher and then change the priorities of the conflicting models to ensure that only the correct answer will be reported in the future. A real conflict is when a test case either exactly matches or closely matches both a model and a non-model. In such a case we have no basis for making a decision and hence must rely on the teacher for guidance. The way that the teacher's judgement is then incorporated depends on whether the model and

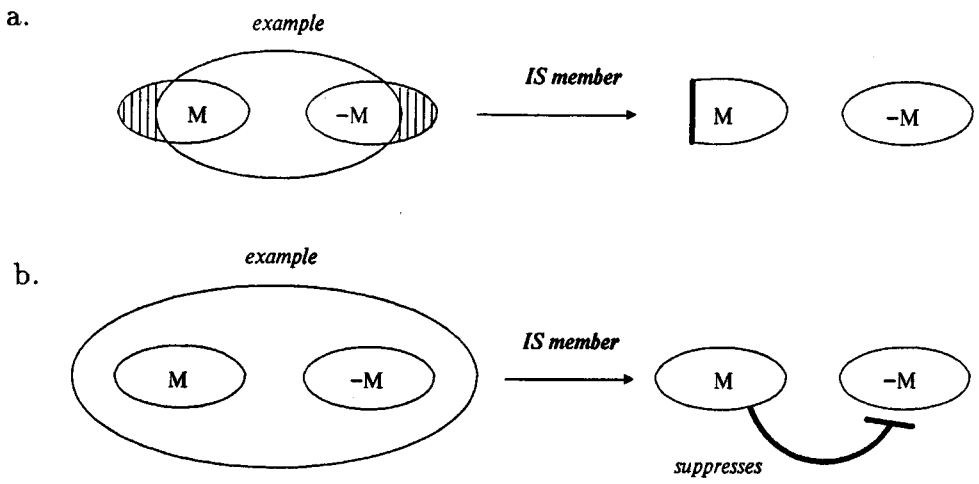


Figure 4-5. Non-models can conflict with models. Given the correct answer, the system resolves these conflicts in various ways.

non-model were exactly matched or only closely matched. If it is an exact conflict we explicitly rank the two opponents one above the other. The effect of this ranking is that the correct one will *suppress* the incorrect one if they are ever in exact conflict again. The other type of conflict is when a test case closely matches a model and a non-model. Here we do not need to impose a ranking. Once we determine, on the basis of the teacher's pronouncement, which was the correct guess to make, we can ensure that in the future we will make this guess by *generalizing* either the model or the non-model. The correct model or non-model will now exactly match the test case (it was generalized to cover this item) while only closely matching the other. Since exact matches always take precedence over close matches, the dispute is settled.

Non-models have two advantages over the MUST have and MUST NOT have links used by Winston. The biggest advantage is that they allow *conjunctive exceptions.* That is, it could be that an item possessing either of two features belonged to the class but that when both features were present the item was not a member of the class. Such an example is shown in Figure 6. Here we are trying to construct a

definition of "hammer" that includes both a and b but excludes c. It would be wrong to say that a hammer MUST NOT have a curved head since the head of hammer in Figure 6b is curved. We also can not say that the hammer MUST have a pointed pein since the hammer in Figure 6a has a flat-ended pein. While MUST and MUST NOT clauses can not cope with this example, it can be easily handled by a single non-model which has a curved head and a flat-ended pein.

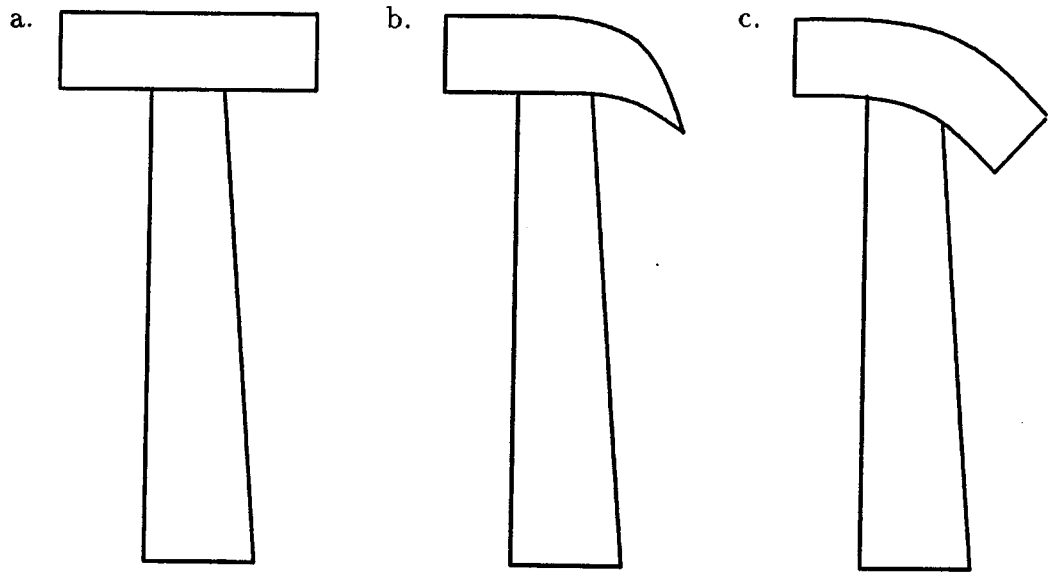


Figure 4-6. Conjunctive exceptions. A hammer can have a flat pein or a curved head but not both. Winston's MUST have and MUST NOT have links are inadequate for describing this; non-models must be used.

The second advantage to non-models is that since they share all the properties of regular models, they can be *refined* through specialization and generalization. In Winston's system [Winston 1970] making a clause into a MUST or MUST NOT clause was permanent; there was no way to turn a mandatory feature back into an optional one. To get around this his system was simply very conservative about creating MUST and MUST NOTs in the first place. It waited for a near miss in which it was possible to discern one key difference before altering its model. Yet disregarding non-examples when there is more than one possible discrepancy severely impairs the value of negative examples. Our system is more flexible and can work with any near miss since it has the ability to generalize non-models. When in doubt, it decides that all differences are important and incorporates all of them into a non-model. Later, more negative examples can be presented to progressively generalizes the non-

model until it differs from the positive model by only the one difference that all the non-examples had in common. This idea, similar to Winston's *near miss group*, is a natural outgrowth of using non-models.

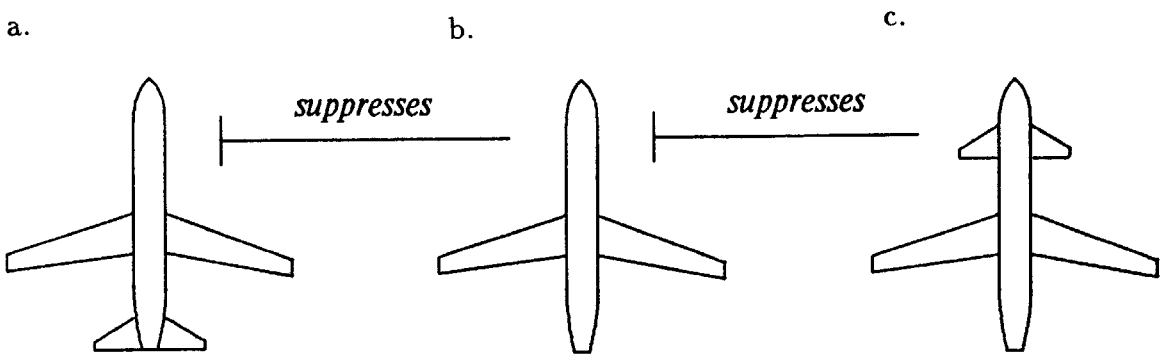


Figure 4-7. Exceptions can themselves have exceptions. a. The plane model. b. A non-model that suppresses the plane model. c. A model developed for planes with canards. This model suppress the non-model in b.

Non-models, just like regular models, are sometimes over-generalized or have exceptions. The technique for coping with this is the same as for regular models: a non-model is specialized by being suppressed by a positive model. Figure 7 shows the program's model of a typical jet airplane and a non-model which looks just like the model but lacks elevators in the back. This pair represents the fact that although something may look a great deal like an airplane, if it does not have elevators it can not possibly be one. While for the most part true, there are certain planes which have front-mounted canards instead of elevators. To correct the program's conception of a plane, we create a new model describing planes with canards and make this model suppress the previous non-model. Thus the exception itself has an exception. The suppression scheme we use to handle such situations is much like putting a *censor* on a rule in Winston's system. The major difference is that a censor requires a named intermediary in order to suppress a rule. That is, some rule concludes fact X which is one of the "unless" clauses of another rule and thus keeps this other rule from firing. Our non-models, on the other hand, directly suppress their associated models.

Obviously we have borrowed many ideas from Winston. One of the contributions of the learning algorithm presented here is that it incorporates all these ideas in a unified framework. Another improvement is that the learning algorithm has a good grasp of the types of generalization which are appropriate to its domain. Such knowledge of similarity is crucial for learning which involves complex data. Finally, our algorithm relies less on the teacher than does Winston's. Through the use of models and non-models it is not confused by inherently disjunctive concepts. Furthermore, because it can handle examples that differ in more than one way from its current model, it does not have to be spoon-fed near misses. The next chapter contains several examples which illustrate these points.

5.1. Structural Prototypes

The major focus of our research has been on learning structural descriptions of objects. The idea is to present the system with pictures of several different airplanes and have it construct the description of a prototypical plane. Figure 1 shows the edges of the three airplanes we used to form the "airplane" concept. The pictures are actually of model airplanes, not full-sized jets. They are (from left to right): Boeing 747, Lockheed L-1011, and Douglas DC-9. Geometrically, these airplane vary primarily in the aspect ratio of the body, the number and placement of engines, and the position of the wings along the fuselage. They were presented to the system in the order shown.

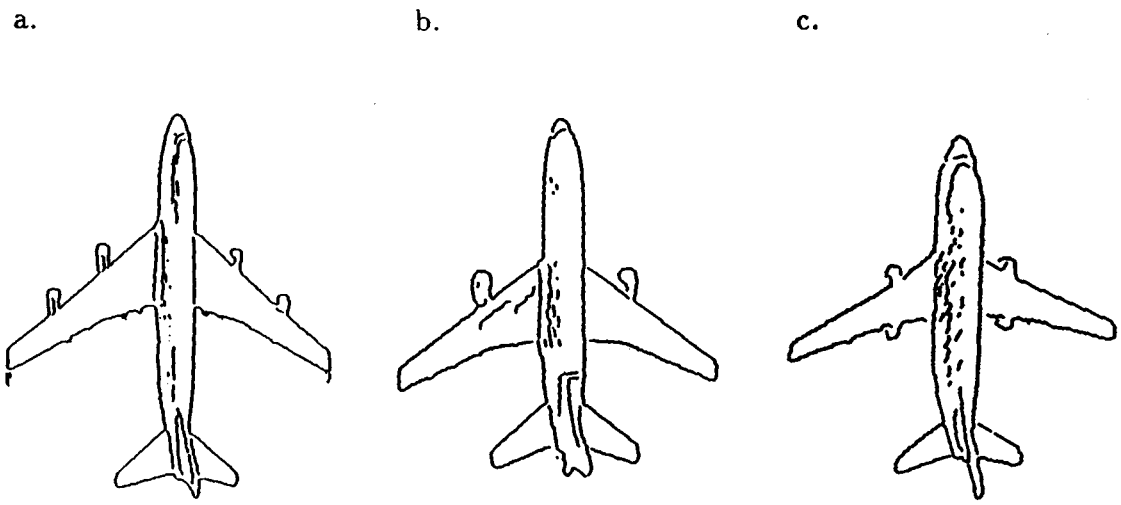


Figure 5-1. The edges of some airplanes (not to scale). a. A Boeing 747. b. A Lockheed 1011. c. A Douglas DC-9. These three examples were used to learn the "plane" concept.

We start by showing the system the gray-scale image of the 747 and telling it that the object in the picture is a "plane". This causes the initial model of the "plane" class to be a copy of the 747's description. In other words, the system believes that to be a plane an object must look like a 747. If we now ask whether the object in

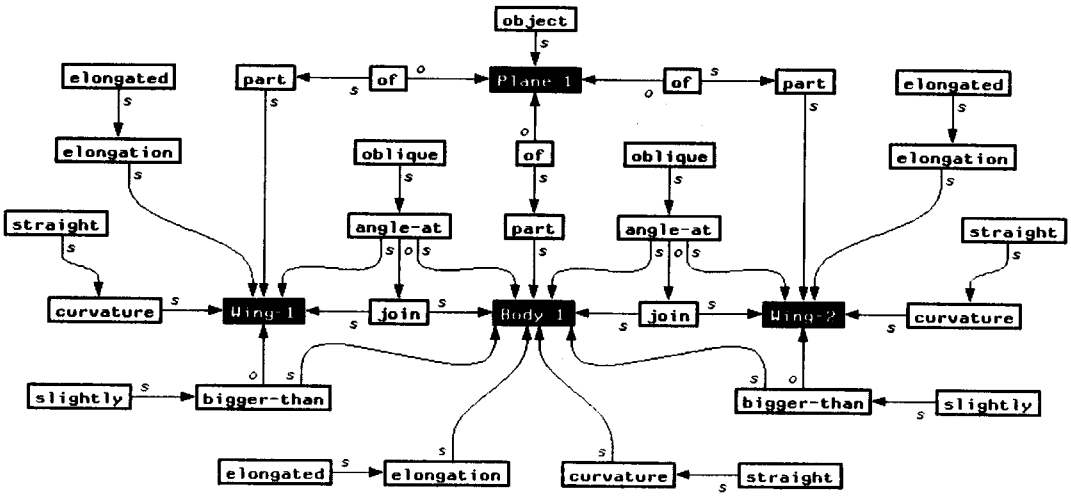


Figure 5-2. The semantic network description of the "plane" model at level 1.

the picture just shown is a plane, the system will answer YES because the object's description *exactly* matches the system's model of the class.

Next we present the picture of the L-1011 and ask whether this object is a plane. The matcher starts out by looking at only the top-level properties of the L-1011 since they are the only predicates that have all of their arguments bound. At this level the L-1011 exactly matches the plane model since, as can be seen in Figure 2, the only property the model requires is OBJECT. Next the program expands its matching radius by examining the relations emanating from the top node of the model. Figure 2 shows all the predicates which can be tested at the second level, level 1. The matcher starts by trying to find a binding for Wing-1, the left wing. The corresponding region in the L-1011, the test case, should be ELONGATED and STRAIGHT. It also must be PART OF Test-Object-1, the node representing the entire L-1011, otherwise it would fail to satisfy the dependency specified in the model. This description, however, matches both the body and the two wings of the test case so the system is unsure which binding to make. Since it does not mind backtracking, it tentatively decides that Test-Body-1 is the correct binding for Wing-1. It now goes on and looks for a binding for Body-1. The thing bound to Body-1 should also be ELONGATED and STRAIGHT and must be PART OF Test-Object-1. Additionally,

it should JOIN Test-Body-1 at an OBLIQUE angle. Test-Wing-1, the left wing of the L-1011, satisfies all these criteria so the system tentatively binds Test-Wing-1 to Body-1. So far we have:

Wing-1 = Test-Body-1
Body-1 = Test-Wing-1

At this point the matcher has confused the body and the wing of the test object but does not yet realize it.

The system discovers its mistake when it tries to find a binding for Wing-2. The part of the test case corresponding to Wing-2 should JOIN Test-Wing-1 at an oblique angle, yet the only thing which is PART OF Test-Object-1 and satisfies this requirement has already been bound to Wing-1. This tells the system that at least one of the two bindings it has made is wrong. Therefore the system scraps both bindings and tries a different set. Suppose, instead, it binds Test-Wing-2 to Wing-1 and Test-Body-1 to Body-1. This allows it to bind Test-Wing-1 to Wing-2 resulting in the bindings:

Wing-1 = Test-Wing-2
Body-1 = Test-Body-1
Wing-2 = Test-Wing-1

While this allows the test object to exactly match the model at this level, these bindings are still incorrect; the system has rotated the plane about the axis of the body such that the two wings are swapped. However, because all the predicates at this level have been satisfied, the system does not know it has made an error. The model of a plane at level 1 merely says to look for three long skinny things, two of which are joined to the third. As shown in Figure 3, many objects fit this description.

The next level adds additional constraints, enough to allow the system to correct the bindings it made. Figure 4 shows the relevant portions of the plane model at level 2. Each wing must be joined to a side of the body and the two wings must be on opposite sides. The left versus right ambiguity can now be resolved because both wings lean TOWARD END-2, the tail end of the body, but only the left side is RIGHT-OF END-2. In this way the system discovers that the bindings it made at level 1 are not the best ones. Eventually the system discovers the correct bindings and is able to compare the example to its model of a plane.

The system finds that L-1011 matches the current plane model, but not exactly. All the top level structure is the same – the L-1011 has parts which look like a body and two wings and these parts are in the proper relative orientations. However, at

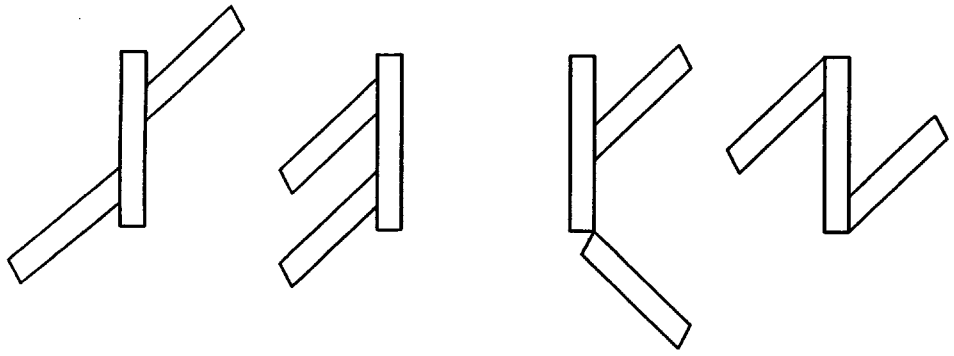


Figure 5-3. Some of the objects which fit the description of a plane at level 1.

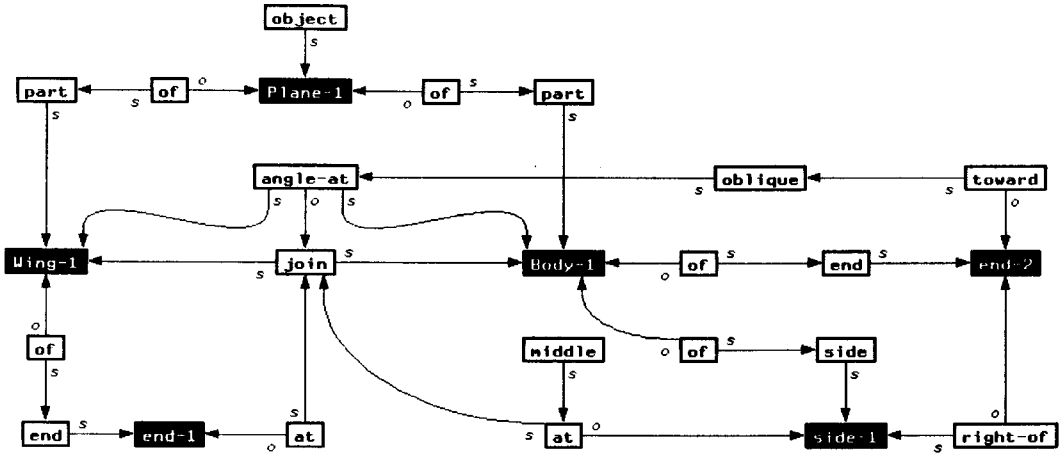


Figure 5-4. The description of the plane at level 2 specifies the joins precisely enough to disambiguate the two wings.

the level of the wing model the program notices that there is a discrepancy. The model, based on the 747, had two engines on each wing while the example, the L-1011, only has one. The system compares the wing of the L-1011 to the wing of the plane model using the Gray coding metric. Figure 5 shows the relations involving the wing which are visible after the horizon has been expanded. The fact that the wing

is missing an engine causes one of the JOIN relations to remain unmatched. This is a small difference since the other JOIN relation and the four SIDE OF and END OF relations are all matched. Therefore, the system judges that the piece is still close enough to the prototype description to be called a wing. When the top level of the matcher asks if the piece in question looks like a wing the answer is YES; no mention is made of the fact that the wing is missing an engine. While the difference only matters locally and is isolated from the higher levels of matching, this does not mean the program forgets about it: it needs to remember this at least to perform ablation. The point is that the program's judgement of the similarity of two objects is based on a number of very simple *local* decisions.

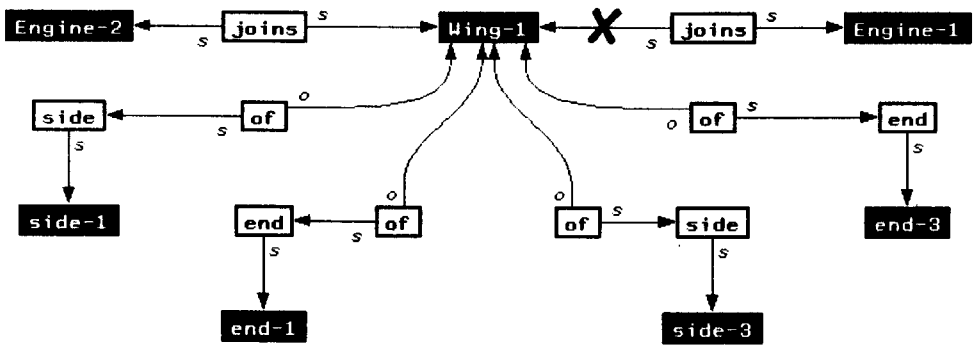


Figure 5-5. The relations involving the wing which must be matched. The missing engine only affects one of these relations.

The third example airplane refines the model in a similar way. The DC-9 differs from the other planes by having a squatter body and skinnier wings. The final model learned from all three examples describes a plane as being, first of all, something with a body and two wings, each of which must have a particular shape. The topology of the model says that the wings must be connected on opposite sides of the body and swept backward. When prompted for more detail, the model reveals that there are two elevators attached to the end of the fuselage. The model has been generalized by ablating the parameters of the wings and body to allow a fairly broad range of shapes for these parts. Furthermore, the model now specifies that each wing need

only have one engine on it, although more are permitted.

5.2. Context Dependent Models

Our system is not restricted to creating prototypes for whole objects, it can also form prototypes for parts of them. This is possible because the representation has been carefully designed so that the view from *any* node in the network yields an appropriate similarity metric. The prototypes created typically specify not only the shape of the part but also its relation to the object as a whole. One example of such a context dependent description is the concept of a "wing". Here we mean the geometrical form and relations of a wing rather than its aerodynamic or functional properties. A wing has an intrinsic shape, basically straight and skinny, as well as being located at a particular place with respect to the rest of the plane.

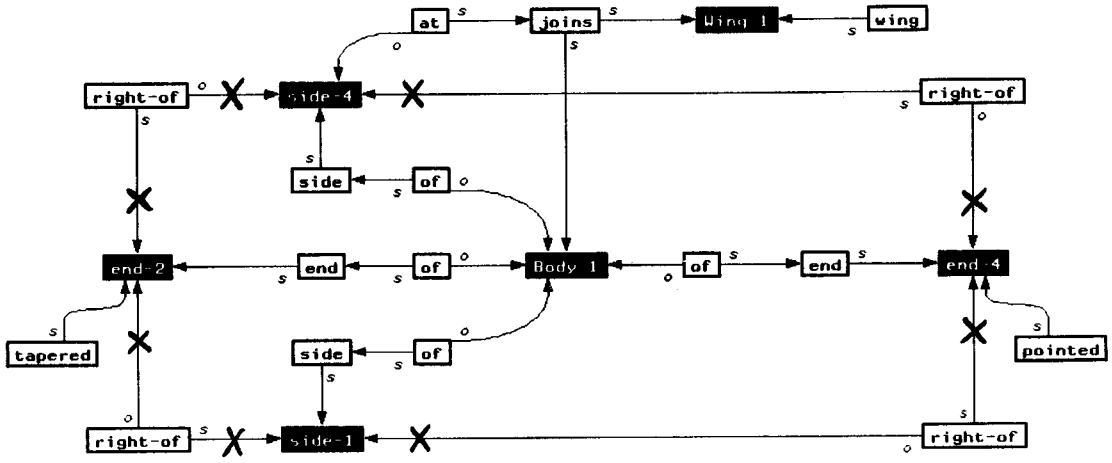


Figure 5-6. A wing can be joined to either side of the body. Dropping the four RIGHT-OF links makes the two sides indistinguishable.

We have taught the system the basic shape and position of a wing using the wings of several airplanes as examples. The shape of these wings differ slightly in terms of aspect ratio and size relative to the body. Such variations are easily handled by applying ablation to the shape parameters of the wing model to arrive at appropriate ranges of values. The wings of different planes also join the body at various angles and distances along the fuselage. Once again, ablating the properties of the wing-body

join allows the model to match all these examples. A more important difference is that a wing can be joined to either side of the plane's body: there are right wings and left wings. The two sides of the body are distinguished by their relation to the body's ends. As shown in Figure 6, one is RIGHT-OF the nose while the other is RIGHT-OF the tail end of the fuselage. To make the two sides of the body equivalent, we drop these RIGHT-OF links from the model. The model arrived at now recognizes both left and right wings and, thanks to ablation, tolerates a variety of wing shapes.

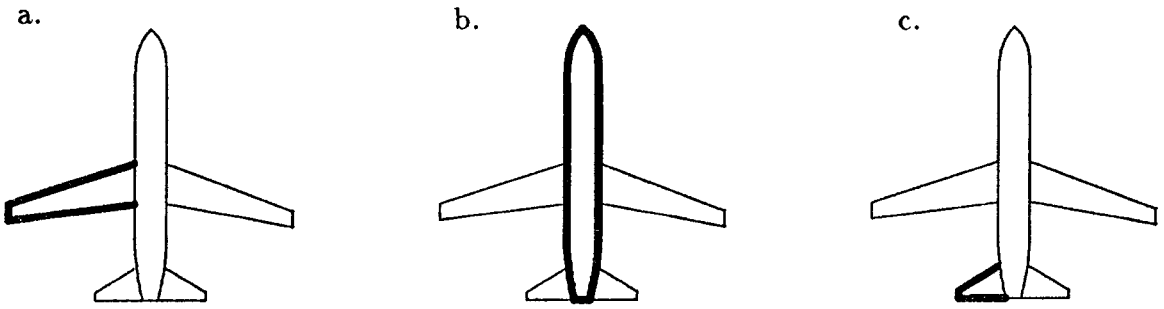


Figure 5-7. The wings of a plane have a particular relation to the rest of the object. a. A wing. b. A non-wing. c. Another non-wing.

Unfortunately, this model also recognizes things which are not wings such as those in Figure 7. Because the body has a shape which is roughly similar, it, too, is a close match to the wing model. The context which serves to distinguish the body from a wing is eliminated by dropping one JOIN link, a small change. To correct this misclassification we explicitly tell the system that the body is not a wing. In responce, the system forms a non-model based on this particular body. The body is not the only piece which might be mistaken for a wing, the elevators in the back of the airplane also closely match the wing model. Because they do not match the preliminary non-model constructed from the body's description, the system incorrectly classifies them as wings. When informed of its mistake the system creates a brand new non-model based on the elevator it was shown. As more elevators are presented, this non-model is gradually generalized. The same thing happens to the non-model derived from

the description of the body when we show the system the bodies of other planes and classify them as non-wings. At some point, the bodies start to match not only this non-model but also the non-model based on the elevators. This causes the two non-models to converge to the same description: a wing can not obliquely join the end of a straight piece. An elevator is not a wing because it is joined to the end of the body. Similarly, the body is not a wing because its end is joined to the end of an elevator. A real wing, however, is not joined to the end of anything; it is joined to the side of the body. Thus, the system has learned to correctly classify wings as members of the class while rejecting both bodies and elevators.

5.3. Articulated Objects

Many objects have moveable parts. We argued before that one reason for segmenting an object into parts was to allow the recognition of different configurations of the same object. To study this we have used a flexible model of the cartoon character Gumby. The task is to recognize Gumby independent of the position of his arms.

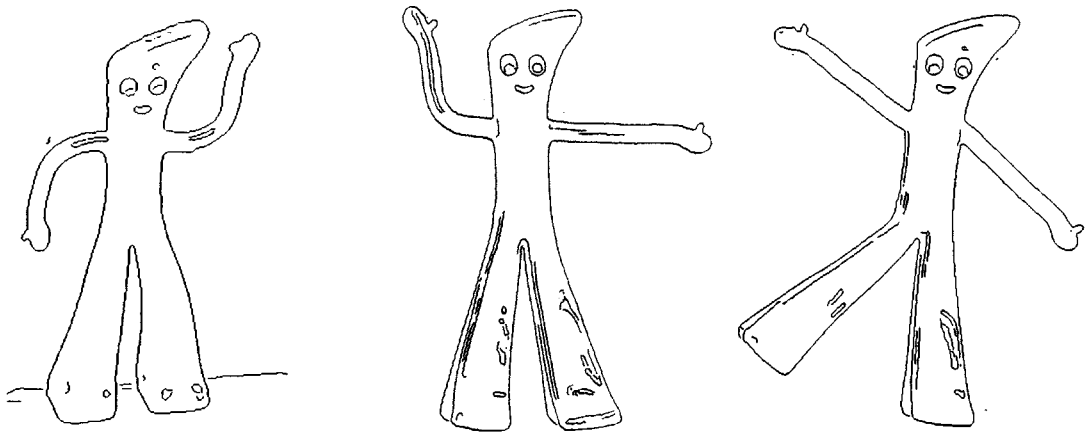


Figure 5-8. Several different poses of Gumby. The task is to recognize Gumby whether his arms are raised, straight out, or lowered. Note that a fully extended arm appears as a single piece, not two.

Gumby's arms vary in two ways in the pictures shown in Figure 8. The first difference is the angle that Gumby's arm makes with his body. His arms can be raised, extended horizontally, or lowered. The second difference involves the substructure

of his arms. In some pictures two sections are clearly visible while in other pictures there is no indication of Gumby's elbow whatsoever.

The first type of variation is easily handled by weakening the specification of the shoulder angle using ablation. The angle of Gumby's arm varies from acute with respect to his head, to orthogonal, to acute with respect to his waist. The position of the join, however, does not vary; the left arm is always joined to the torso half way up the left side. Thus, the final model specifies the location of the join precisely but puts no restriction on the relative angle of the pieces.

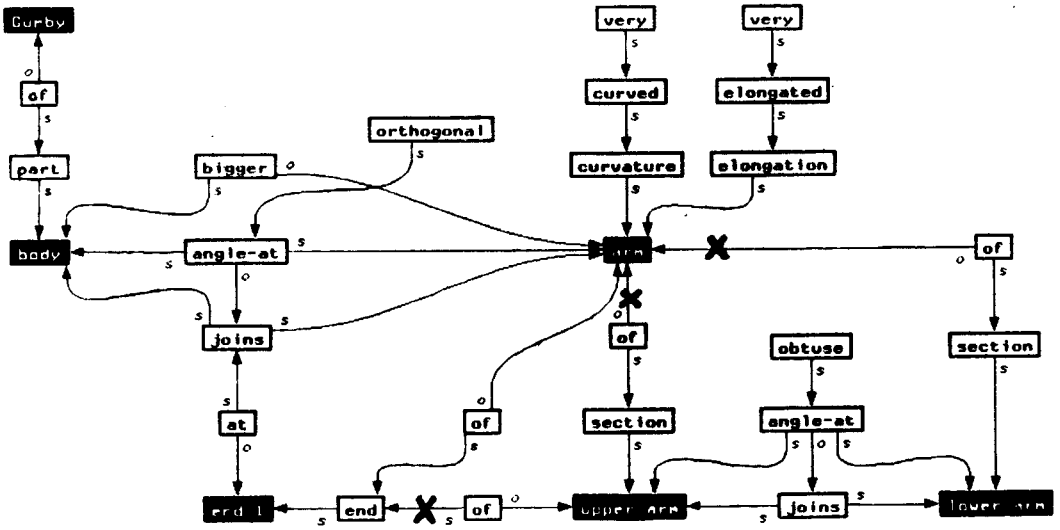


Figure 5-9. The two sections of Gumby's arm are parts of a *chain* which represents the whole arm. To match the picture with outstretched arms, certain links must be dropped from the model.

The second type of variation is handled using the *chain* heuristic described in Chapter 3. As Figure 9 shows, Gumby's upper arm and lower arm are both parts of a chain which has certain shape properties derived from the pieces of the chain. Only the chain, representing the entire arm, is joined to the body. In the pictures where Gumby's arms are extended each arm is a single piece with no substructure. However, this single piece has the same length and elongation as the chain although these two views of the arm obviously have different curvatures. Part of being able to recognize Gumby, therefore, is removing any restriction on the curvature of his arms. The description of Gumby must also be further modified by removing the details of the chain's substructure from the model. These links are marked by X's in Figure 9.

The chain heuristic lets us recognize articulated objects in arbitrary configurations but it does this by discarding the information about the moveable portions. A better solution would be to recognize that Gumby's elbow is a *joint* not a join. A joint is a connection between two or more pieces where the angle between the constituents is allowed to vary over some specified range. This approach would help us segment the image of a pair of pliers in which four pieces meet at the same place. From a single view it is unclear what should be joined to what. However, from two different views we can discover that this intersection is a joint. By observing that some of the pieces of the joint move together, we can determine that each handle should be joined to the opposite jaw. The joint approach is good for this purpose but the singularity resulting from the alignment of two sections would now have to be handled as a special case.

5.4. Functional Improvisation

We have shown how structural descriptions of things can be learned, yet certain things, like hammers, are more naturally defined by their function. One way to do this is to isolate the structural features of the hammer that enable it to perform its function. The functional description of a hammer can then be learned in terms of these structures and their interrelations. While this has been suggested before [Winston, Binford, Katz, and Lowry 1984], it has never been implemented.

The function of a hammer can be broken into subfunctions by examining the tool's interfaces to the user and to the workpiece. The first requirement is that the hammer must be able to contact the nail without slipping. The flat end of the head is ideally suited to this task. Second, since the hammer is powered by the action the the user's arm, it must be firmly graspable. There are several methods of prehension [Schlesinger 1919] each suited to some particular shape. Because the handle is relatively long and fairly wide, it can be gripped by wrapping the fingers around in one direction and the thumb in the other. Finally, given the dynamics of swinging a hammer, the head must be appropriately oriented with respect to the handle. The fact that front of the head is perpendicular to the direction of the hammer's motion allows this kinematic requirement to be met. Therefore, our functional description of a hammer is a graspable object that has a flat striking surface orthogonal to the direction of motion.

We have taught the system this definition by telling it straight-out the requirements and then showing it examples of various structures which are graspable, flat,

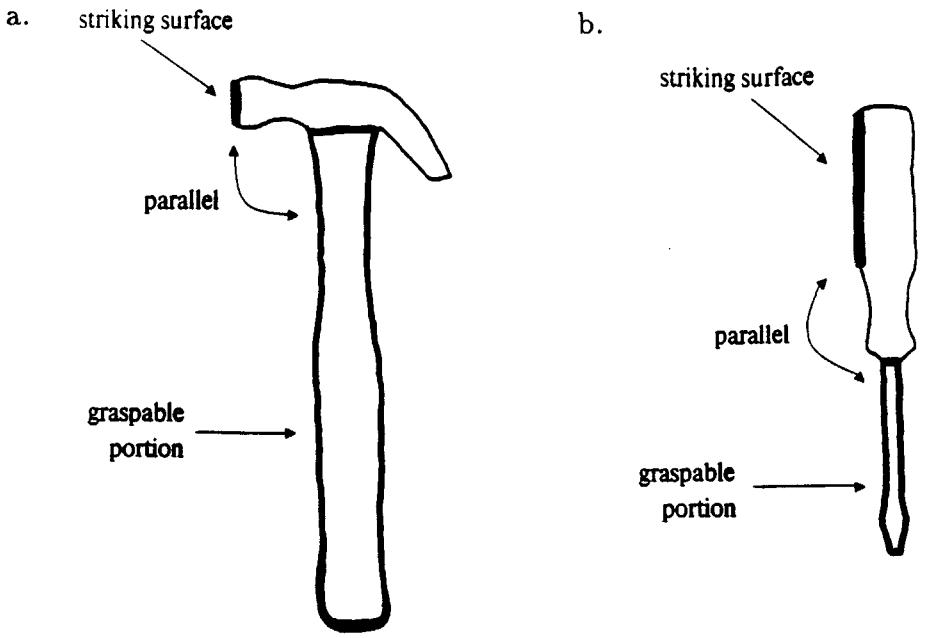


Figure 5-10. Function from structure. a. The structures of a hammer that contribute to its function. b. The functional description of a hammer mapped onto the structure of a screwdriver.

orthogonal, etc. The advantage to having a functional description of a hammer is that it allows us to improvise [Brady, Agre, Braunegg, and Connell 1984]. Suppose we needed to drive some tacks but did not have a hammer. One solution would be to use a screwdriver to pound them in. Our system suggests this using the functional definition we taught it. First, it notices that the blade and shaft of the screwdriver fit its structural description of a graspable part. Then it realizes that the side of the screwdriver's handle can be considered a striking surface. Since it is appropriately oriented with respect to the prospective handle, the screwdriver successfully matches the system's description of a hammer.

Conclusion

6.1. Summary

We feel that this research has been successful. The system described is fully implemented and generates shape descriptions from gray-scale images. Furthermore, it is capable of generalizing these descriptions in a reasonable manner. The system has been used to form structural prototypes and is now being applied in learning functional concepts. However, even if you disbelieve all our theories of shape and abstraction there are two things which should be clear:

- Images are not just collections of bits or even edges. It is entirely possible to extract a much higher-level representation from them which is intuitively satisfying.
- Each domain, especially the visual one, has its own types of similarity. A program can not learn about a domain without understanding this similarity.

The system as described has several known limitations. First of all, it is restricted to two-dimensional shapes. Work is being done on segmenting three-dimensional objects into parts [Ponce and Brady 1985], yet figuring out how to describe joins and measure visual similarity in 3-D are still tough unanswered questions. A second limitation of the current system is that the SLS also only works well if the object is composed of elongated parts. A companion technique known as *local rotational symmetries* [Fleck 1985] is being studied for detecting round regions. The most important deficiency, however, is that there is no provision for recognizing partially occluded objects using the models generated. These three topics are areas for future research.

Even with its limitations, there are several applications for our system. One of these is *generating geometric models* for a smarter recognition system such as ACRONYM [Brooks 1981]. ACRONYM knows about three-dimensional transformations like rotation out of the image plane, and can handle partially occluded objects. Currently it works using models from a CAD /CAM data base. While this is fine for many objects, especially machined parts, sometimes the required information is not readily available. In such situations our system could be used to complement ACRONYM by generating models directly from images of the objects.

Another applicable is *parts indexing.* When designing a new machine it is often possible to employ parts used in other machines also and thus avoid stocking new

items. In the old days it was enough to bring the specification of a part down to the stock room and ask the clerk if he knew of anything like it. Modern inventories, however, often contain millions of different parts – far too many to be kept track of by any individual. One approach to solving this problem is "group technology" [Boothroyd, Poli, and Murch 1982]. In this approach each part is assigned a three digit code describing whether it is cylindrical or prismatic, what sorts of symmetry it has, and the aspect ratio of its bounding polytope. The indexing problem is solved by mapping similar parts to the same code number. Our system could be used the same way, but with far richer descriptions. Each part would have its semantic network computed and stored away in a large database. To find items similar to some new part, we merely show the system a picture of the new part and then ask it to match this against the stored descriptions.

6.2. Future Work

Aside from fixing the short-comings listed above, there are several interesting extensions that might be made to the system:

- **Integrating visual and verbal knowledge**

Learning the class of some object can be viewed as determining which word corresponds to some visual stimulus. English descriptions of shapes are also filled with references to particular items: "pear-shaped", "hook-like", "V-necked", etc. Understanding these terms requires understanding the appearance of the objects they are derived from. We could also learn the meaning of non-class words such as "bent" and "parallel" much as we learned the functional properties of a hammer. Currently, we have a program which will draw pictures based on the semantic networks produced by the vision system. Since the output of the parser developed by Katz [Katz and Winston 1983] is compatible with our representation language, it would be a relatively simple matter to integrate these two programs. The result would be a system that could give an English descriptions of the shapes it saw as well as draw pictures based on a written descriptions.

- **Enhancing the contour representation**

The descriptions produced by our system are largely region-based. The only contour information they contain is the bentness of the spines, the curvature of the

sides of the symmetries, and the adjacency of the edges of a region. Even so, region-based descriptions often provide a good first-pass approximation to the shape of an object. Consider the running shoe shown in Figure 1. It has a fairly irregular shape but can be considered to be roughly ellipsoidal. Sometimes, just knowing that a shoe is ellipsoidal is all the information we need. If required, more detail can be added by noting certain contour features: the shoe is pointed in front, square at the back, and has a concavity surrounded by two points at the top. This information could be computed using visual routines [Ullman 1983] or a system such as that described in [Asada and Brady 1984].

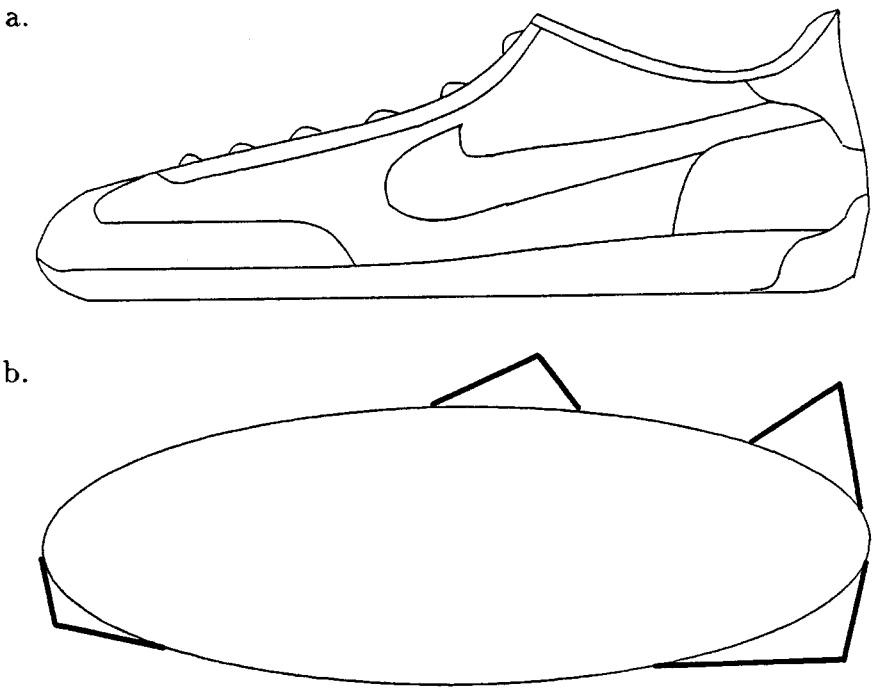


Figure 6-1. A running shoe is basically ellipsoidal as shown in a. More detail can be added to this by examining the contour as in b.

• Using partial information

Sometimes only a small portion of an object is visible. While not enough to allow recognition based purely on the visual data, it may be sufficient if we have other sources of information. For instance, if we know that the occluding object is the top of a hangar, then the symmetric pointed object protruding beneath it is liable to be

the nose of an airplane. Complete models of a shape can also be constructed using partial information in a manner similar to the way free-space maps for robots are built up from multiple views [Chatila and Laumond 1985]. Imagine building a description of a real, full-sized jet airliner by walking around the plane. It is impossible to see all the important features simultaneously because the object is self-occluding. Nevertheless, it is relatively easy for a human to get a comprehensive model of the plane from this exercise.

- **Determining function from form**

The approach to improvisation we presented in Chapter 5 only works if we already know the function of a tool. It would be interesting to go the other way, guessing the function of an object from its form. To do this we need to determine which parts of a shape have functional significance and how the functions of these subparts interrelate to yield the overall function of the object. This is part of the motivation behind the *Mechanic's Mate* project [Brady, Agre, Braunegg, and Connell 1984] which has taken the first step in this direction by compiling a partial list of the subshapes found in tools and the functions they are useful for. There is still much work to be done, however, before full form-to-function reasoning is realized.

Bibliography

Agre, Phil, [1985], "Routines", MIT Artificial Intelligence Laboratory, AIM-828.

Asada, Haruo and Michael Brady, [1984], "The curvature primal sketch", MIT Artificial Intelligence Laboratory, AIM-758.

Bagley, Steven C., [1984], "Using models and axes of symmetry to describe two-dimensional polygonal shapes" (SM Thesis), MIT Dept. Elec. Eng. and Comp. Sci.

Ballard, D. H., [1981], ""Strip trees: a hierarchical representation for curves"," *Communicaions of the ACM*, **24**, 111 - 122.

Ballard, D. H. and C. M. Brown, [1982], *Computer Vision*, Prentice–Hall, Englewood Cliffs, New Jersey.

Blum, H., [1973], "Biological shape and visual science (part 1)," *Journal of Theoretical Biology*, **38**, 205 - 287.

Blum, H. and Nagel, R. N., [1978], "Shape description using weighted symmetric axis features," *Pattern Recognition*, **10**, 167 - 180.

Bobrow, D. G., and T. Winograd, [1977], "An overview of KRL, a knowledge representation language," *Cognitive Science*, **1**, 3 - 46.

Bobrow, D. G., and B. L. Webber, [1980], "PSI–KLONE: Parsing and semantic interpretation in the BBN natural language understanding system", *Proc. of the 3rd CSCSI /SCEIO Conference*, Victoria, B. C., 131 - 142.

Boothroyd, G., C. Poli, and L. E. Murch, [1982], *Automatic Assembly*, Marcel Dekker, New York, Appendix 3.

Brachman, R. J., [1983], "What *ISA* is and isn't: An analysis of taxonomic links in semantic networks," *IEEE Computer, Special Issue on Knowledge Representation.*

Brachman, R. J., R. E. Fikes, and H. J. Levesque, [1983], "KRYPTON: A functional approach to knowledge representation," *IEEE Computer, Special Issue on Knowledge Representation.*

Brady, Michael, [1983], "Criteria for representations of shape", *Human and machine vision*, A. Rosenfeld, B. Hope, and J. Beck (eds.), Academic Press.

Brady, Michael, and Haruo Asada, [1984], "Smoothed local symmetries and their implementation," *Int. J. Robotics Research*, **3 (3)**.

Brady, Michael, Philip Agre, David Braunegg, and Jonathan Connell, [1984], "The Mechanic's Mate", *ECAI 84: Advances in Artificial Intelligence*, T. O'Shea (ed.), Elsevier Science Publishers B.V., North-Holland, Amsterdam.

Brooks, Rodney A., [1981], "Symbolic Reasoning Among 3-D Models and 2-D Shapes," *Artificial Intelligence*, **17**, 285 - 348.

Canny, John Francis, [1983], Finding Edges and Lines in Images, MIT Artificial Intelligence Laboratory, Cambridge Mass., AI-TR-720.

Chatila, R. and J. Laumond, [1985], "Position referencing and consistent modeling for mobile robots," *Proceedings of the IEEE Robotics Conference.*

Chomsky, Noam, and Morris Halle, [1968], *The Sound Pattern of English*, Harper and Row, New York.

Davis, Randall, [1984], "Diagnostic reasoning based on structure and behavior", MIT Artificial Intelligence Laboratory, AIM-739 (also in Artificial Intelligence (1985)).

Davison, L. D., and R. M. Gray, [1976], *Data Compression*, Dowden, Hutchinson and Ross.

Dietterich, T. G., and R. S. Michalski, [1981], "Inductive learning of structural descriptions," *Artif. Intell.*, **16**.

Fairfield, J., [1983], "Segmenting blobs into subregions," *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC–13**, 363 - 384.

Fleck, Margaret, [1985], "Local rotational symmetries" (SM Thesis), MIT Dept. Elec. Eng. and Comp. Sci.

Freeman, H., [1974], "Computer processing of line drawing images," *ACM Computing Surveys*, **6**, 57 - 98.

Hamming, Richard W., [1980], *Coding and Information Theory*, Prentice-Hall, Englewood Cliffs, NJ.

Heide, S., [1984], "A hierarchical representation of shape" (SM Thesis), MIT Department of Mechanical Engineering.

Hendrix, G. G., [1979], "Encoding knowledge in partioned networks", in *Associative Networks: Representation and Use of Knowledge by Computers*, N. V. Findler (ed.), New York, 51 - 92.

Hoffman, D. D. and W. A. Richards, [1982], "Representing smooth plane curves for recognition: implications for figure–ground reversal," *Proccedings of the 2nd National Conference on Artificial Intelligence*, 5 - 8.

Horn, B. K. P., [1977], "Understanding image intensities," *Artificial Intelligence*, 8 (2), 201 - 231.

Katz, Boris, and Patrick H. Winston, [1983], "A two-way natural language interface", *Integrated Interactive Computing Systems* P. Degano and Erik Sandewall (eds.), North-Holland, Amsterdam.

Kenstowicz, Michael, and Charles Kisseberth, [1979], *Generative Phonology*, Academic Press, New York.

Marr, D., [1977], "Analysis of occluding contour," *Proc. Roy. Soc. Lond. B*, **275**, 483 - 524.

Marr, D., [1982], *Vision*, W. H. Freeman, San Francisco.

Marr, D., and H. K. Nishihara, [1978], "Representation and recognition of the spatial organisation of three dimensional shapes," *Proc. Roy. Soc. Lond. B*, **200**, 269 - 294.

Minsky, Marvin and Seymour Papert, [1969], *Perceptrons*, MIT Press, Cambridge, Ma.

Mitchell, T. M., [1978], "Version spaces: A candidiate elimination approach to concept learning" (PhD Thesis), Stanford University.

Nevatia, R. and Binford, T., [1977], "Description and recognition of curved objects," *Artificial Intelligence*, **8**, 77 - 98.

Perkins, W. A., [1978], "A Model-Based Vision System for Industrial Parts," *IEEE Transactions on Computing*, **C-27**, 126 - 143.

Ponce, Jean, and Michael Brady, [1985], "Toward a surface primal sketch", *Three Dimensional Vision*, Takeo Kanade (ed.), Academic Press.

Poggio, T. and V. Torre, [1984], "Ill-posed problems and regularization analysis in early vision, MIT Artficial Intelligence Laboratory, AIM-773.

Quillian, M. Ross, [1968], "Semantic Memory" (PhD Thesis), in *Semantic Information Processing*, M. Minsky (ed.), MIT Press, Cambridge MA.

Schlesinger, G., [1919], Der mechanische Aufbau der kuntstlichen Glieder, *Glieder und Arbeitschilfen*, Berlin: Springer.

Ullman, Shimon, [1983], "Visual routines", MIT Artificial Intelligence Laboratory, AIM-723.

VanLehn, K., [1983], "Felicity conditions for human skill acquisition: Validating an AI-based theory" (PhD Thesis), MIT Department of Elect. Eng. and Comp. Sci..

Winston, Patrick H., [1970], "Learning structural descriptions from examples" (PhD Thesis), MIT Department of Elect. Eng. and Comp. Sci..

Winston, Patrick H., [1981], "Learning new principles from precedents and exercises," *Artif. Intell.*, **19**, 321 - 350.

Winston, Patrick H., [1982], "Learning by augmenting rules and accumulating censors", MIT Artificial Intelligence Laboratory, AIM-678.

Winston, Patrick H., [1984], *Artificial Intelligence, 2nd. Ed.*, Addison-Wesley, Reading, Ma..

Winston, Patrick H., Thomas O. Binford, Boris Katz, and Michael Lowry, [1984], "Learning physical descriptions from functional definitions, examples, and precedents", *Robotics Research*, Michael Brady and Richard Paul (eds.), MIT Press, Cambridge, 117 - 135.

Woods, W. A., [1975], "What's in a link", in *Representation and Understanding*, D. G. Bobrow and A. Collins (eds.), Academic Press, New York, 35 - 82.

This blank page was inserted to preserve pagination.

CS-TR Scanning Project
Document Control Form

Date : 1 / 25 / 96

Report # AI-TR-853

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 99(107-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☒ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:
☒ DOD Form (2) ☐ Funding Agent Form ☒ Cover Page
☒ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages (by page number): _____

Photographs/Tonal Material (by page number): _____

Other (note description/page number):

Description : Page Number:
(A) IMAGE MAP: (1-99) UNNUMBERED TITLE PAGE, 2-99
(100-107) SCANCONTROL, COVER, SPINE, DOD(2)
TRGT'S (3)
(B) CUT+PASTE F.C.'s ON PAGES 4, 6-10, 13-20, 22-23, 25-26, 28-29, 31-39, 42, 44, 46, 48-68, 70, 72, 74, 76-79, 81-82, 84-89, 91, 94

Scanning Agent Signoff:
Date Received: 1 / 25 / 96 Date Scanned: 2 / 1 / 96 Date Returned: 2 / 1 / 96

Scanning Agent Signature: Michael N. Cook

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AI-TR-853	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Learning Shape Descriptions: Generating and Generalizing Models of Visual Objects		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Jonathan Hudson Connell		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0505 N00014-77-C-0389
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, Massachusetts 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, Virginia 22209		12. REPORT DATE September 1985 13. NUMBER OF PAGES 102
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, Virginia 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

learning	shape description	Smoothed Local Symmetries
concept learning	machine vision	
SLS	high-level vision	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We present the results of an implemented system for learning structural prototypes from gray-scale images. We show how to divide an object into subparts and how to encode the properties of these subparts and the relations between them. We discuss the importance of hierarchy and grouping in representing objects and show how a

(over)

(block 20 continued)

notion of visual similarity can be embedded in the description language. Finally we exhibit a learning algorithm that forms classifiers from descriptions produced and uses these models to recognize

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**