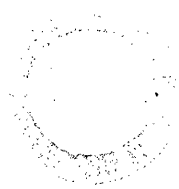
RELAXATION AND ITS ROLE IN VISION

by

Geoffrey E. Hinton

Ph.D. Thesis,
University of Edinburgh,
1977



The composition of this thesis and the research reported in it are entirely my own work, except where otherwise stated.

Geoffrey Hinton

CONTENTS

CONTENTS 3

ACKNOWLEDGEMENTS 7

ABSTRACT 8

OUTLINE 9

CHAPTER 1 : SEARCHING FOR OPTIMAL VISUAL INTERPRETATIONS. 12

1.1: Structure and process in visual perception 12
1.2: Why tentative hypotheses are necessary 14
1.3: Two ways of avoiding tentative hypotheses 15
  1.3.1: The principle of least commitment
  1.3.2: Feature semantics
1.4: Ways of finding consistent interpretations 18
  1.4.1: Huffman/Clowes labelling
  1.4.2: Growing alternative consistent contexts.
  1.4.3: Waltz filtering
1.5: The need for optimisation. 24
  1.5.1: Consistency versus goodness in the blocks world.
1.6: Ways of finding good interpretations. 27
  1.6.1: Guided depth-first search.
  1.6.2: Conniving.
  1.6.3: Assumptions and specialist error procedures.
  1.6.4: Bar-finding in Popeye.
  1.6.5: Marr and Poggio (1976).
  1.6.6: The breakdown of Waltz filtering.
1.7: Explicit numerical scores 36
  1.7.1: Probabilities and the costs of hypotheses.
  1.7.2: The advantages of a numerical definition of the optimum.
1.8: Pattern Recognition and the Misuse of Numbers 39
  1.8.1: The pattern recognition paradigm.
  1.8.2: Inadequacies of Pattern Recognition
1.9: Branch-and-Bound search. 43
1.10: The Relevance of Parallel Hardware. 44
1.11: Summary of Chapter 1. 47

CHAPTER 2: THE TASK OF SEEING SOME OVERLAPPING RECTANGLES AS A PUPPET. 50

2.1: The ease and purpose of the task. 50
2.2: Pictorial input. 51
2.2.1: The range of possible pictures.
2.3: Non-pictorial input. 52
2.4: Output of the best global interpretation. 54
2.5: The puppet model 54
2.5.1: Defining satisfactory joints.
2.6: Definition of the required output. 56
2.6.1: What pictures depict.
2.6.2: Basic definition of the best puppet.
2.6.3: Modification of the definition of best.
2.6.4: Equal rivals.

CHAPTER 3: THE PUPPET FINDING PROGRAM. 60

3.0: The two main stages : An overview 60
3.1: The main data-structures and their creation 61
3.1.1: Representing zones and computing their overlaps.
3.2: Creating the network of candidate hypotheses 63
3.2.1: Types of nuclei
3.3: Numerical constraints between supposition values. 67
3.3.1: The function of continuous supposition values.
3.3.2: States of supposition values: terminology.
3.3.3: Normalised linear combinations
3.4: Probabilities and supposition values 70
3.5: The hyperspace model 70
3.6: Representing arbitrary logical constraints 71
3.7: Non-integer optima
3.8: The numerical constraints in the puppet task 76
3.9: The simplex algorithm 78
3.10: Assigning preferences to hypotheses 80
3.11: The abstract optimization problem and the type of solution required. 80
3.12: Two types of relaxation 82
3.13: Two components of the relaxation operator 82
3.14: Achieving feasibility 83
3.15: The speed of convergence on a feasible state. 87
3.16: Achieving optimality 89
3.17: A method of increasing the convergence speed 90
3.18: The method of selecting the final set of hypotheses. 91
3.19: The final form of the relaxation operator 92

CHAPTER 4: THEORETICAL ANALYSES OF RELAXATION, AND SOME POSSIBLE EXTENSIONS. 94

4.1: The avoidance of Explicit Enumeration 95
4.2: Decomposition into Interacting Sub-Systems 95
4.3: The Time Taken to Reach Equilibrium 98
4.4: Introducing non-linearity. 104
4.5: The Need for Intermediate Level Hypotheses. 105
4.6: Weak rules 107
4.7: Using relaxation to guide hypothesis creation 109
4.7.1: The extended puppet-finding task
4.7.2: Generators
4.8: Optimising real-valued parameters 114

CHAPTER 5: COMPARISONS BETWEEN L.P. RELAXATION AND ALTERNATIVE SYSTEMS. 119

5.1: Rosenfeld, Hummel and Zucker (1975). 119
5.1.2: The non-linear model.
5.2: Line Labelling using LP relaxation 126
5.3: Yakimovsky and Feldman (1973) 129
5.3.1: A relaxation formulation
5.4: Barrow and Tennenbaum (1976) 135
5.4.1: The task
5.4.2: The general strategy
5.4.3: Likelihoods and their modification.
5.4.4: An abstract example
5.4.5: Comparison of MSYS with LP relaxation.
5.5: Growing islands of consistent hypotheses. 140
5.6: Matching by Clique finding 144
5.7: Hierarchical synthesis 145

CHAPTER 6: PERCEPTUAL SCHEMAS AND THEIR RELATIONSHIP TO PERCEPTUAL AWARENESS. 149

6.1: Current awareness and stored knowledge 149
6.2: Frames 153
6.2.1: An example of a schema
6.2.2: Minsky's theory
6.2.3: Some Difficulties for Frames

CHAPTER 7: A SYSTEM WHICH USES RELAXATION TO COORDINATE NETWORK GROWING RULES. 161

7.1: Overview of the SETTLE system. 161
7.2: Schemas. 163
7.3: Slots. 164
7.4: Bonds 165
7.5: Specifying Rules. 166
7.6: Rule invocation. 168
7.7: Jobs 171
7.8: An example of the SETTLE system in action. 174
7.8.1: Specifying rules about family relationships.
7.8.2: Interpreting claims about specific people.
7.8.3: The effect of more, incompatible claims.

CHAPTER 8: SUMMARY. 179

8.1: Presuppositions of the relaxation approach 179
8.2: The choice of numerical scores 181
8.3: Details of the relaxation operator 181
8.4: The SETTLE system 182
8.5 Relaxation and human vision 182
8.5.1: The temporal structure of vision
8.6: What has been shown. 185

APPENDICES

Appendix 1: Computing whether convex polygons overlap. 187
Appendix 2: Using penumbras to aid line labelling. 189
Appendix 3: Code for the puppet-finding program 191
Appendix 4: How the supposition values change in examples of the puppet program. 210
Appendix 5: Code for the line labelling example. 214
Appendix 6: Code for the SETTLE system 215

BIBLIOGRAPHY 235

ACKNOWLEDGEMENTS

I would like to thank Christopher Longuet-Higgins, Frank O'Gorman, and Aaron Sloman who have proved invaluable. Christopher tirelessly supervised the work and helped me to clarify my ideas, Frank explained many difficult points about vision, and Aaron selflessly provided support, understanding and many helpful ideas, as well as reading much of the manuscript.

I have been greatly helped by discussions with many members of the A.I. communities at Edinburgh, Sussex and Essex. I would especially like to thank Harry Barrow, Richard Bornat, Mike Brady, Max Clowes, Roddy Cowie, Steve Draper, Richard Gregory, Jim Howe (who also helped administratively), Steve Isard, Christof von der Malsburg, Dave Owen, Larry Paul, Robin Popplestone, Richard Power, Naomi Roberts, Arnold Smith, Mark Steedman, Sylvia Weir, and David Willshaw.

Wendy Taylor kindly did the typing under difficult conditions, ably assisted by Judith Dennison, Jane Blackett and Pru Heron.

The work was funded by research studentships from the Science Research Council and the Royal Society. Later extensions were made as part of a project on "Computational Flexibility in Visual Perception" (S.R.C. Grant BRG 8688-7).

ABSTRACT

It is argued that a visual system, especially one which handles imperfect data, needs a way of selecting the best consistent combination from among the many interrelated, locally plausible hypotheses about how parts or aspects of the visual input may be interpreted. A method is presented in which each hypothesis is given a supposition value between 0 and 1. A parallel relaxation operator, based on the plausibilities of hypotheses and the logical relations between them, is then used to modify the supposition values, and the process is repeated until the best consistent set of hypotheses have supposition values of approximately 1, and the rest have values of approximately 0.

The method is incorporated in a program which can interpret configurations of overlapping rectangles as puppets. For this task it is possible to formulate all the potentially relevant hypotheses before using relaxation to select the best consistent set. For more complex tasks, it is necessary to use relaxation on the locally plausible interpretations to guide the search for locally less obvious ones. Ways of doing this are discussed.

Finally, an implemented system is presented which allows the user to specify schemas and inference rules, and uses relaxation to control the building of a network of instances of the schemas, when presented with data about some instances and relations between them.

OUTLINE

This thesis explores the idea that relaxation may be a good way of organising the interactions between different hypotheses during the process of constructing the internal representation of a scene.

Chapter 1 argues for some of the presuppositions behind the use of relaxation: that a visual system needs to formulate tentative hypotheses; that it needs to be able to find a good consistent set of these hypotheses; that the best set may be defined in terms of numerical scores for the individual hypotheses; that the constraints between hypotheses need to be explicitly represented; and that a method which can use constraint propagation and can take advantage of parallel hardware is desirable.

Chapter 2 defines a task designed to test and illustrate the use of relaxation. The task is to perceive a collection of overlapping transparent rectangles as a puppet. Many of the problems that arise in vision (e.g. parts missing due to occlusion) are deliberately avoided in this task.

Chapter 3 explains the puppet-finding program. First, it explains how the program discovers and represents the various possible hypotheses about the in-

terpretation of rectangles as puppet parts, and about the interpretations of overlaps between rectangles as joints between puppet parts. Then it shows how logical constraints between hypotheses give rise to numerical constraints between their supposition values. Finally, it introduces and analyses a relaxation operator which manipulates the supposition values on the basis of the constraints and the preferences for individual hypotheses. The operator picks out the best consistent set of hypotheses. Various aspects of the relaxation process are illustrated with examples produced by the program.

Chapter 4 discusses various theoretical issues about relaxation that arise from the puppet-finding program. It attempts to analyse the relaxation process, particularly the time it requires. It also points out some of the strengths and weaknesses of relaxation, and discusses some ways of extending it to cope with specific theoretical difficulties.

Chapter 5 compares my relaxation system with other systems which were selected for comment either because they used a form of relaxation, or because they used explicit numerical scores in defining the best interpretation, or because they dealt with the problem of finding the best instantiation of a model. To aid comparison with another system, there is a section on the use of relaxation for Huffman/Clowes line labelling, which shows clearly the similarities and differences between relaxa-

tion and Waltz filtering.

Chapter 6 is a theoretical interlude from the details of relaxation. It discusses the relationship between stored knowledge and the representations created during perception. The function of the chapter is to argue against the idea that perception is merely a process of matching the data to stored models, and thus to prepare the ground for the rule-based SETTLE system presented in Chapter 7. The issues are extremely complex and so only a rather superficial treatment is possible, but it may be sufficient to explain the approach adopted in the SETTLE system.

Chapter 7 describes and illustrates an implemented system which allows the user to define schemas and inference rules which can be applied to combinations of instances of the schemas. When given some assertions about related instances, the system notices which rules apply, and it uses relaxation to find the best consistent network of instances, given the input assertions. The processes of relaxation and of making inferences are integrated so as to avoid forward chaining based on premises that are rejected by relaxation.

Finally, there is a brief summary.

CHAPTER 1

SEARCHING FOR OPTIMAL VISUAL INTERPRETATIONS.

1.1: Structure and process in visual perception

Consider the pictures in figure 1.1. When we look at them it seems that we form a clear idea of what they depict. In understanding how this idea is formed, there are two sets of issues:

1. What is the nature of the idea once it has been formed? That is: What is the form of the representations produced by the process of perception?

2. What is the nature of the processes that generate the representations?

Understanding the nature of the representations used is probably the major part of understanding perception. It is hard to say any thing about perceptual processes without making some assumptions about what the processes are producing. However, it does not seem to be necessary to complete the investigation of the representations before starting the investigation of the processes. Indeed, any simulated perceptual system needs both representations and processes. Artificial Intelligence research (Minsky and Papert 1972, Clowes 1971, Winston 1970) has already shown that perception of a picture involves more than simply activating a number of feature analysers and

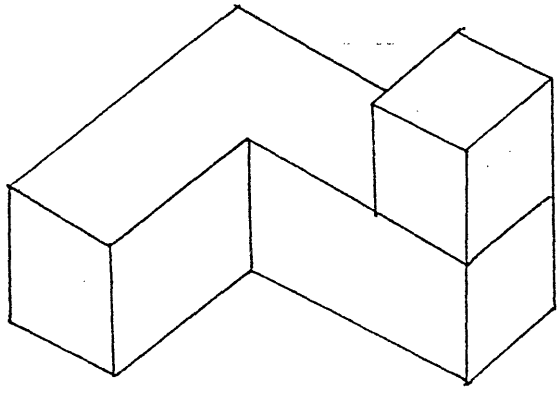


FIGURE 1.1a: A blocks world picture.

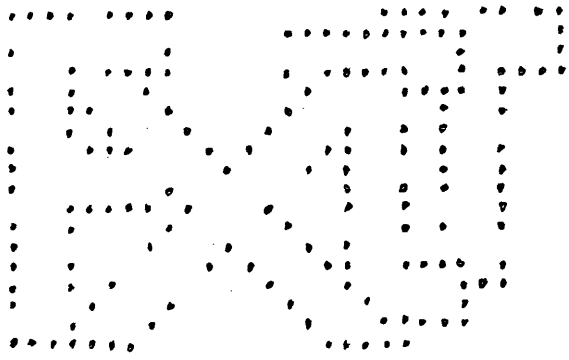


FIGURE 1.1b: A Popeye picture.

using them to put the picture into one of a fixed number of categories. The product of perception is not just a categorization. It is a complex description which has the following important properties:

1. A scene is articulated into a hierarchy of objects and parts of objects so that its description involves specifying the relationships between the objects and object parts. For example, in figure 1.1a, the description that constitutes the interpretation of the picture must somehow explicitly represent the fact that there is a cube resting on one end of an ell-beam.

2. As well as a hierarchy of objects within a domain there are also many different domains. For example, the lines in a picture and the edges of objects which they depict are quite different entities and need different representations. Similarly, in figure 1.1b the lines of dots, the bars whose edges these lines depict, and the letters whose strokes are depicted by these bars are all entities in different domains.

These considerations show that the representations produced by looking at a picture must be at least as rich as a relational network containing a great variety of different types of node, and many diverse relations (e.g. support, depiction, connection). The way in which nodes and relations of various types may be combined constitutes a kind of grammatical knowledge. It determines

which particular networks are possible given the initial picture structure. If we assume that perception involves building some kind of relational network which satisfies certain grammatical constraints, then it is possible to focus on some of the important issues about the way in which the network is constructed.

1.2: Why tentative hypotheses are necessary

The hypothesize-and-test paradigm is often used in Artificial Intelligence programs (Roberts 1965, Grape 1973) as a way of deciding how to interpret part of a picture. An important assumption of the paradigm is that once a specific hypothesis has been formulated on the basis of cues, it is possible to make a definite decision about whether the hypothesis fits the data, so that a hypothesis can be accepted or rejected immediately after it has been formulated, and it is not necessary to manipulate a number of tentative, interdependent hypotheses simultaneously. Unfortunately for the hypothesize-and-test method, there are many cases where no definite decision about a hypothesis can be made on the basis of the local data. The context may be necessary for disambiguation (Guzman 1971, Clowes 1971) as the E in Figure 1.1b shows. The context in which some local data is interpreted must itself be represented as a set of hypotheses about the interpretation of other data, so hypotheses about local interpretations may be mutually dependent,

and some kind of search mechanism is needed for selecting a consistent set of them.

1.3: Two ways of avoiding tentative hypotheses

Before discussing ways of handling interdependent, tentative hypotheses, two methods of eliminating the need for tentative hypotheses will be examined and rejected.

1.3.1: The principle of least commitment

This method, advocated by Marr (1976) and Sloman (1976) amongst others, involves never being more specific than the local data and the context warrant, so that hypotheses do not commit themselves to details that are, as yet, undecidable. This requires that a rich set of not-too-specific concepts be available. For example, in the early stages of perceiving a human form, a visual system may notice a part which is definitely either a leg or an arm, but which needs contextual disambiguation. If the system has the concept of a limb available, it can represent what it can safely conclude, without creating any tentative hypotheses about arms or legs. Then, when the context becomes clearer, the limb hypothesis can be refined appropriately (The clearer contect may involve non-committal limbs).

In practice, there are several difficulties in applying the principle of least commitment. First, an enor-

mous number of concepts of varying degrees of specificity may be needed to ensure that is possible to represent just what can be definitely inferred in a given situation and no more. Secondly, if hypotheses are to interact and progressively refine one another until they are all perfectly specific, then complex transition tables may be required to say how one hypothesis should be refined in the context of others. Finally, when the data is imperfect and the aim is not to find just any consistent global interpretation, but to find the best one, (see section 1.5) then it may be impossible to arrive at any definite conclusions about optimal interpretations on the basis of local evidence.

The principle of least commitment may be useful in avoiding unnecessarily large numbers of alternative hypotheses, but there is no reason to suppose that it can eliminate the use of alternatives altogether. I know of no system which does this, when interpreting complex imperfect data.

1.3.2: Feature semantics

The problem of choosing between alternative hypotheses arises because nodes in the network representing a scene are taken to imply the existence of entities in the scene, so nodes corresponding to non-existent entities are incorrect and must be rejected. Nodes can however be given a different semantics in which they only

imply things about the appearance of the scene. In the relational net built to represent figure 1.1b, for example, there could be two different nodes corresponding to the first letter. One node could represent the fact that it is somewhat E-shaped and the other that it is somewhat F-shaped. These two nodes are quite compatible, provided they are not taken to imply anything about which letter is really there in the optimal interpretation, so there is no need to reject one of the nodes as incorrect.

The reason for using the term "feature semantics" is that the output of feature analysers in pattern recognition programs is often given just this semantics. Consider for example, an analyser which looks for "bays" on the right of a figure (as in C and K). If the analyser responds positively to a particular figure, then the figure has the feature, since the precise definition of the feature is simply what the analyser responds to.

Marr's primal sketch (Marr 1975) also uses feature semantics. Symbolic descriptions in the primal sketch represent aspects of the grey-level data, rather than of the scene causing that data. These representations may nevertheless be expressed in terms of the scene elements which they appear to depict. (Section 1.6.4 discusses this difficult point in more detail). This is not intended as a criticism of the primal sketch. It is sensible to analyse the raw data and redescribe it in a more convenient form before trying to decide what it depicts.

However, the primary purpose of perception is to enable us to act in the world, and so perception must tell us what's there, not just how it appears. Sooner or later decisions have to be made between conflicting hypotheses (except when interpreting very simple data).

1.4: Ways of finding consistent interpretations

Given a number of interrelated tentative hypotheses, one problem is to find a consistent set of them. This section describes some of the known ways of achieving consistency, and then sections 1.6 and 1.9 discuss how these methods can be extended to the more difficult problem of finding interpretations which are good or optimal rather than just consistent. The Huffman/Clowes line-labelling task will be used to illustrate some of the methods and so it is defined below.

1.4.1: Huffman/Clowes labelling

Detailed discussions of line-labelling occur in several places (Huffman 1971, Clowes 1971, Waltz 1972, Winston 1977) so only a brief description is given here.

The input consists of perfect line-drawings of scenes composed of polyhedra. There are never more than three surfaces at a point in the scene, and the viewpoint is chosen so that vertices or edges are never on exactly

the same line of sight as a nearer vertex. Given these restrictions, the topology of the junctions in the picture provides good evidence about what kinds of edge are depicted by the lines (see figure 1.2). In the case of a tee-junction, the evidence has an unambiguous implication. The crossbar must depict an occluding edge belonging to the surface on the opposite side to the stem. Other junction types, however, provide ambiguous evidence about line labels. A globally consistent set of line labels can only be found by considering how the local evidence interacts. The interactions are based on the fact that a line must have the same labels at both ends.

1.4.2: Growing alternative consistent contexts.

Techniques such as depth-first and breadth-first search (see Nilsson 1971, Winston 1977) involve considering all the alternative ways in which a context (a consistent partial solution) can be extended. For each such extension, a new context is spawned, and ways of extending it are considered. All consistent solutions can be found in this way. For the line-labelling task, the contexts could consist of assignments of particular labels to some of the lines, and contexts could be extended by considering all possible labels for a previously unlabelled line. A context is consistent if the combination of line labels at each junction is one of the combinations allowed for a junction with that topology.

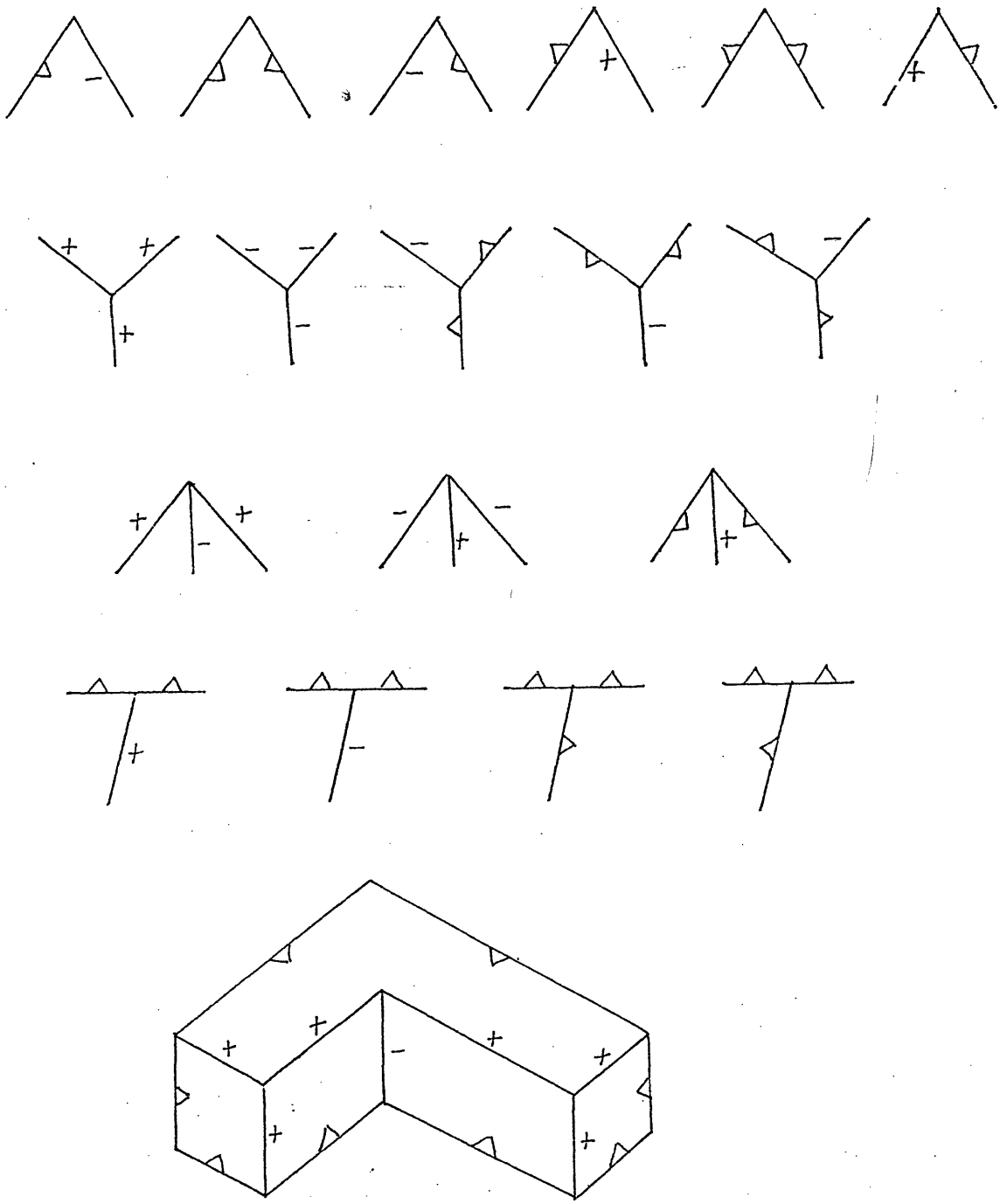


FIGURE 1.2: Showing all the possible junction labels, given the Huffman/Clowes restrictions, and an ell-beam illustrating them. "+" means a convex edge, "-" means a concave one, and " ⊦ " means an occluding edge with the attached *vee* lying in the nearer surface.

The contexts form an inverted tree, with complete labellings at its lowest tips. Depth-first and breadth-first search differ, as their names suggest, in the order in which this tree is investigated.

A major criticism of both these search techniques is that they perform a lot of unnecessary work because they do not make use of the fact that many suppositions are independent of one another. They do not keep track of which of the suppositions in a context were used in deriving a conclusion, and so they cannot use the conclusion in rival contexts in which it is also valid. They have to re-establish it each time. In figure 1.3, for example, the triangle has many alternative labellings. It seems silly to rediscover the possible labellings of the cube for each labelling of the triangle, but this is what is done.

The Conniver programming language (Sussman and McDermott 1972) embodies, among other things, one particular approach to this problem. It involves providing a hierarchy of contexts which are accessible to the user. A fact asserted in one context is available in all the descendents of that context. When a new fact is established, the user can ensure that it is asserted in a higher context than the current one if he is sure it is also valid there. This makes the fact available in rival contexts to the one in which it was discovered.

An alternative to the Conniver policy of leaving the

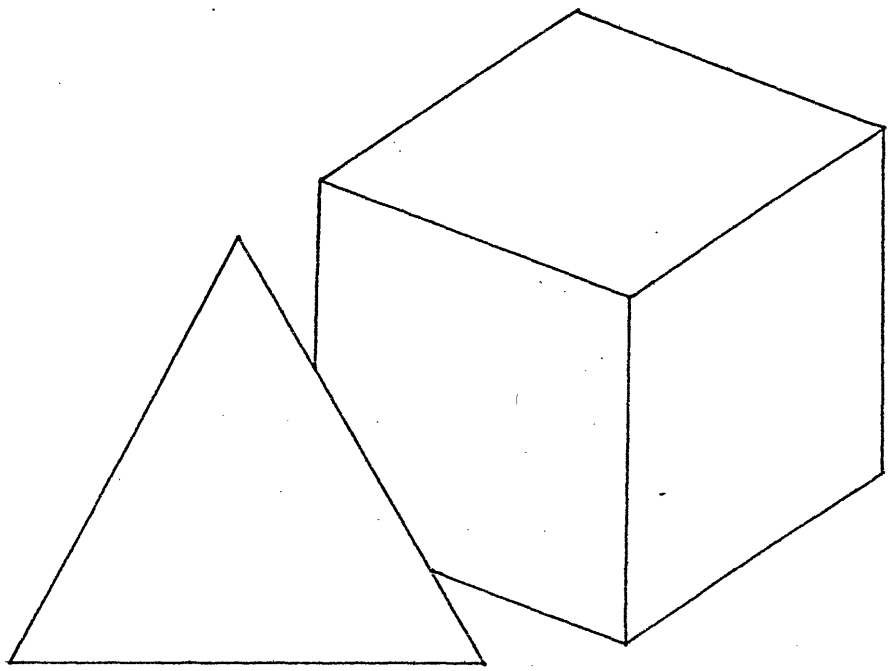


FIGURE 1.3: The "cube" can have many different labellings corresponding to different ways of being stuck to the background plane. Many of these choices are independent of the line labels chosen for the triangle .

problem to the user is to systematically record all the suppositions that are used in deriving each fact. The system can then automatically assert a fact in the highest context containing all the suppositions used to derive it. Alternatively, the system can set up demons which ensure that whenever a context contains all the suppositions previously used to derive a particular fact, that fact is automatically asserted. The latter method has the advantage that it may make the fact available in more contexts. Suppose, for example, that there is an ordered set of choice points A, B, C, D.... and that the choices are A1 or A2, B1 or B2, C1 or C2, D1 or D2... If it is discovered, whilst exploring the A1 branch of the search tree, that B1 and C1 imply D1, then the highest available context in which to assert D1 is (A1, B1, C1). This does not capture the fact that D1 must also be true in (A2, B1, C1). Because A comes above B and C in the search tree, there is no single place in the hierarchy of contexts where the assertion of D1 would make it available in just those contexts containing B1 and C1.

Stallman and Sussman (1976) describe a system for analysing electrical circuits containing non-linear components (e.g. transistors). Each such component can be in one of a number of roughly linear operating regions, and the system has to search for a consistent combination of regions for the different components. It searches by growing a number of contexts and it notices which suppo-

sitions about operating regions are used to derive the operating regions of other components. It uses these relationships to avoid having to rediscover the consequences of sets of suppositions. It also notices which suppositions are involved whenever a contradiction is derived, so that it can immediately reject any other context containing that combination of suppositions.

Stallman and Sussman's work has been mentioned because it implies that it is worth explicitly representing the logical relations between hypotheses (suppositions), rather than simply building up consistent sets of them. This policy is an important aspect of the relaxation method to be described later.

1.4.3: Waltz filtering

Waltz (1972) showed that Huffman/Clowes labelling could be extended to deal with line drawings containing shadow edges and also certain coincidences. This gives many more legal labellings for each junction type, which greatly increases the search space. However, Waltz showed that a filtering process can quickly eliminate most of the junction labels and often leaves a single consistent labelling. The process depends on keeping, for each junction, a list of all the labellings which are compatible with its topology. Each junction labelling must then find a "sponsor" at the other end of each of

the lines forming the junction. A sponsor is a labelling of the other junction which agrees on the labelling of the common line. If there is any line along which no sponsor can be found for a particular junction labelling, that labelling is removed from the list of possible labellings for that junction. This may well leave some labelling of a neighbouring junction without a sponsor along their common line, so it too will be eliminated. This is how the effects propagate.

A major attraction of filtering is that it is suitable for parallel computation. Each junction, or even each junction label, could be allocated to a separate processor, which would be given links to the processors for neighbouring junctions. All the processors could then repeatedly check for sponsors in parallel.

A number of workers have attempted to extend the filtering approach. Mackworth (1975) and Freuder (1976) consider ways of checking more than just pairwise consistency, so as to cope with cases where there are many alternative labels each of which has the required sponsors, even though there is only one globally consistent labelling (Waltz handled such cases by resorting to a depth-first search). Rosenfeld et al, Barrow and Tennenbaum, and I, have tried to extend Waltz filtering so as to find optimal interpretations when there are preferences for particular labels, or the constraints are not binding (see Chapter 5).

1.5: The need for optimisation.

Consider the handwritten letters in figure 1.4.

Figure 1.4

The difference between the two m's is just like the difference between the h and the n. So why, on first inspection, isn't the h interpreted as a distorted n just as one of the characters is interpreted as a distorted m? There are two questions here. First, what makes the h interpretation preferable, since the distorted n interpretation also fits the data perfectly? Second, how does the existence of the h interpretation either suppress its rival or prevent it ever being explicitly formulated?

The obvious answer is that the h interpretation is preferred because it does not involve distortion, and that the distorted n interpretation is not noticed because such interpretations are only sought when attempts to find better ones have failed. It will be shown, however, that this kind of solution runs into difficulties if all the possible interpretations contain unattractive

features.

1.5.1: Consistency versus goodness in the blocks world.

Consider figures 1.5a and 1.5b. These have fairly obvious interpretations as a hole and as a solid respectively. There is some ambiguity about whether the solid is attached to the background along any of its boundaries, but apart from this, a program can easily give the pictures their appropriate line labellings. Notice, however, that the interpretations of the two pictures can be swapped if the bottom central junctions are seen as the result of a special viewpoint. The tee-junction in figure 1.5a could depict a trihedral vertex seen from a point lying in the same plane as the invisible surface. Similarly the lower arrow junction in figure 1.5b could depict the internal concave edge of a hole lying exactly behind a corner in the rim of the hole. Both these interpretations are ruled out by the assumptions of Huffman and Clowes, and so a program can discover the interpretations which people find obvious simply by using consistency. People, however, must use more complex criteria than simple consistency, since they also make interpretations based on non-general viewpoint when there are no better ones (See figure 1.5c).

There seems to be no way of redefining the notion of consistency so as to allow the obvious interpretation for

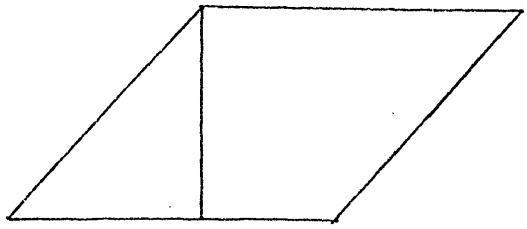


FIGURE 1.5a: A hole

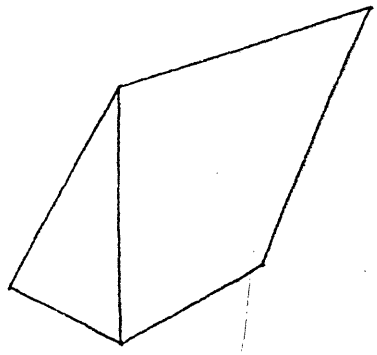


FIGURE 1.5b: A solid

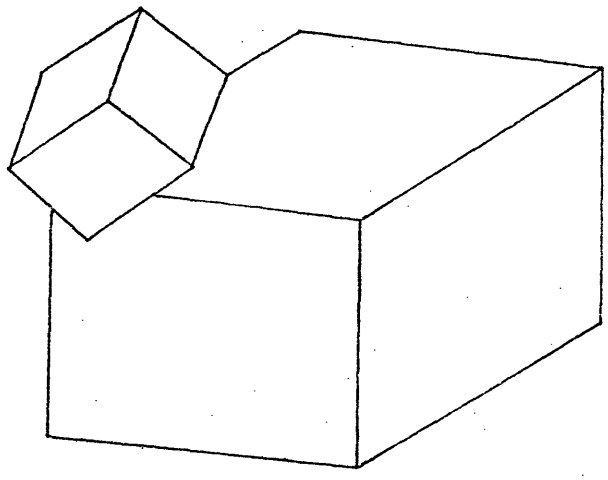


FIGURE 1.5c: A Y-junction between the two cuboids is interpreted as an accidental alignment of an edge at one depth with a closer vertex. (The picture was suggested by Steve Draper).

figure 1.5c whilst ruling out the unlikely interpretations of figures 1.5a and 1.5b. An alternative to consistency for characterising the interpretations which people come up with, is to introduce the idea of the goodness of an interpretation, and to define it in such a way that people's interpretations are optimal or nearly optimal. It is an interesting empirical question whether such a definition is possible. There is no a priori reason why it must be, though if good is equated with probable (see Section 1.7.1), then the desire for the best interpretation may be explained by the obvious value of finding the most probable interpretation of the visual input when perceiving the real world.

Chapter 2 discusses what makes an interpretation good in one domain. Another example of the meaning of "good", using the blocks-world, is given below, before discussing how good interpretations may be found.

For blocks-world pictures, there are many different aspects of an interpretation which affect how good it is. Some of these can be explained by the concept of general viewpioint (Roddy Cowie, personal communication). The perceiver is unlikely to be in such a position that certain important properties of the image would change with a small change of viewpoint. For example, it is unlikely that a straight line in the image is the projection of a curved edge, or that parallel lines in an orthographic image are caused by non-parallel edges seen from a spe-

cial viewpoint. The alternative interpretations of figures 1.5a and 1.5b provide further examples of non-general viewpoints.

A different kind of desirable feature in the interpretation of a blocks scene is that there should be three orthogonal directions with which many edges are aligned. This helps to explain why a line drawing of a cube is not seen as a non-rectangular parallelipiped. Potential symmetries may also determine which interpretations people perceive (see Perkins 1976).

1.6: Ways of finding good interpretations.

This section describes a variety of methods for finding good but not necessarily optimal interpretations. It is by no means a complete survey. What the methods have in common is that they lack an adequate mechanism for identifying trade-offs between the various ways in which an interpretation may be imperfect. Since they cannot identify complex trade-offs, decisions between rival sets of imperfections are not confronted. Consequently, the methods do not need any systematic way of evaluating combinations of imperfections of different kinds. Rather, they tend to make use of domain-specific heuristics for deciding commonly encountered types of conflict on a local basis. The term "procedurally embedded optimisation" will be used to refer to these methods,

because they are made to find good interpretations by embedding ideas about goodness in the procedures for deciding whether to develop a context, or to make an assumption. This contrasts with the use of explicit scores for systematic optimisation.

One of the simplest and commonest ways of making a program produce a good interpretation, is to investigate promising possibilities first and to accept the first solution. Roberts (1965), for example, uses this method in his program which interprets line drawings in terms of known three-dimensional models. Various configurations of lines and regions in the picture act as cues for particular models. The cues are ordered on the basis of how much of a model they depict, and then the program attempts to match models to the line drawing in that order. The first sufficiently good match is accepted. The problem with this approach is that the best cue may not give the best match. Also, after the first object has been found, the lines which remain may be very hard to explain in terms of other objects. Roberts ignores this trade-off between the quality of the first object and the quality of subsequent ones. This helps to explain why he can get led up the garden path when doing composite analysis (Mackworth 1977). Grape's (1973) program is also unable to make subsequent difficulties reverse a decision to interpret part of a picture as a particular view of a particular object.

1.6.1: Guided depth-first search.

A systematic way of using ordering to achieve good solutions is to combine it with a depth-first search which terminates as soon as any solution is found. Backtracking ensures that early choices are reconsidered if they lead to inconsistency, and hence guarantees that a consistent solution will be found if there are any. At each choice point, the possibilities are ordered on the basis of how they would contribute to the goodness of the global interpretation. Planner (Hewitt 1972) allows the user to specify the ordering so that he can guide the search towards good solutions.

Unfortunately, the rejection of a locally poor possibility may force the acceptance of many poor choices later, in order to achieve consistency. So the first consistent, complete interpretation to be found may be far from the best. For a guided depth-first search, the ordering of choices high in the search tree has far more effect than the ordering of lower ones in determining the order in which consistent solutions are generated. Using a guided depth first search to find good solutions first, is like using the values of integers to find those whose digits have a large sum.

1.6.2: Conniving.

Conniver embodies a more sophisticated way of combining the use of contexts with the investigation of promising possibilities first. The ability to jump to specified contexts means that a line of investigation can be abandoned as soon as it looks unpromising, but can be reopened if there turns out to be nothing better, or if evidence turns up suggesting that the abandoned context was better than it appeared. Also, the reasons for abandoning a context may suggest which other context to jump to. Adler (1977) has argued that these control facilities can be helpful in interpreting pictures. A deficiency of Conniver, as section 1.4.1 explains, is that the serial ordering of the suppositions which constitute a context can prevent facts discovered in one context from being made available wherever relevant. Another defect is apparent in tasks such as line-labelling where there are many strongly interrelated choices. It is not clear how the control facilities available to the Conniver user could help him to achieve anything like the efficiency of the Waltz filter. Brady and Wielinga (1976) mention further difficulties encountered in using a Conniver-like language for vision.

1.6.3: Assumptions and specialist error procedures.

In some domains, the need to store and develop many separate contexts can be avoided altogether. Instead of spawning a new context for each possibility at a choice point, a program can simply choose the possibility which seems best on the evidence available. If the program has a lot of domain-specific knowledge to help it choose , it should be able to make the right choice in most situations. In cases where there is no obvious right choice, it may be possible to delay the decision until more helpful evidence has emerged. Inevitably, such a program will sometimes make mistakes. Sooner or later it will arrive at a contradiction or notice that its combination of assumptions is too implausible. When this happens, it cannot jump or backtrack to another context, since it has not kept any. Instead, it must examine the difficulty it has discovered and uses domain-specific knowledge to decide which assumptions to abandon and which new ones to put in their place.

It is hard to see how such a process can be guaranteed not to oscillate, unless it keeps a record of previous combinations of assumptions (which begins to look like depth-first search). However, the emphasis placed on domain-specific knowledge means that the method cannot be fairly evaluated in the abstract. It may be that for the sorts of visual tasks at which people excel, there is so much available information suggesting the

correct interpretation, that systematic search is unnecessary. Several quite competent programs work in this way and two are described below.

1.6.4: Bar-finding in Popeye.

One part of the Popeye program (Sloman et al 1977) searches for bars in pictures like figure 1.1b. The program expects long lines of dots to depict bar walls (the longer sides), so if it finds two parallel lines appropriately positioned, it assumes they are opposite walls of a bar. If it subsequently discovers a good line of dots between the two previous lines, it may jettison the original bar, and replace it by two new ones (cracks are allowed). So the initial assumption can be undone on account of evidence discovered later.

In fact, bars in Popeye have a rather complex semantics which has similarities to the feature semantics discussed in section 1.3.2. A distinction is drawn between picture-bars which are correct if the picture contains good evidence for a scene bar, and scene-bars which are only correct if they occur in the correct global interpretation (i.e. the interpretation people see). So the only assumption involved in asserting the presence of a picture-bar is that the picture evidence is good. It is possible for the program to be mistaken about this, because it does not perform an exhaustive low-level

analysis before looking for high-level structures.

The use of concepts like picture-bar enables decisions about scene-bars to be left until evidence is provided by higher level considerations, such as how well picture bars combine with others to form letter-shaped laminae. The use of higher-level structures to make local decisions is an important way of avoiding making arbitrary assumptions.

1.6.5: Marr and Poggio (1976).

When each eye is presented with one of two random dot patterns, which are identical except for lateral displacements of some regions in one pattern, people see a number of surfaces at different depths (Julesz 1971). To do this we have to decide which dot in one pattern to pair with which dot in the other. Since all dots are the same, there are many potential mates for each one. However, each pairing will give a different angular disparity, and hence a different perceived depth for the dot. Using the assumption that each dot can only be paired with one other (based on the opacity of surfaces), and the assumption that neighbouring dots in the merged image should be at similar depths (based on the continuity of surfaces), it is possible to make the many potential pairings disambiguate one another.

Marr and Poggio show that the computation of a good

set of pairings can be done in an interesting way. They use a binary "neuron" for each potential piece of surface at each depth. Neurons corresponding to pieces of surface lying along a line of sight from an eye tend to inhibit one another (the opacity assumption), and neurons corresponding to adjacent pieces of surface tend to excite one another (the continuity assumption). A dot in a pattern tends to excite neurons corresponding to all the pieces of surface on that line of sight. The computation consists of an iterative process whereby each neuron is turned on or off by the combined effects of the other currently active neurons and the input. When the strengths and ranges of the effects have been tuned, the system works very well and settles down in only a few iterations.

Marr has expressed doubts (personal communication) about whether people solve the stereo correspondence problem in this way. However, it is a good illustration of the method of making assumptions and revising them if it seems necessary, since an active neuron corresponds to an assumption about surface depth. Notice how inappropriate it seems to find a solution by developing many separate consistent contexts. This illustrates that search methods appropriate in domains such as understanding natural language (e.g. micro-planner) may be inappropriate for low-level vision.

The difficulties that can be caused by the way in

which the consequences of an assumption can ramify do not seem to be encountered in the stereo correspondence task. This is partly explained by the fact that surfaces do not have to be continuous. Occasional discontinuities are allowed, and this means that no definite long-range consequences follow from an assumption about surface depth at one point. This, and the simplicity of the constraints, means that the mechanism used by Marr and Poggio is adequate, even though it cannot capture the kind of rigid complex logical constraints which the relaxation method handles (see Chapter 3).

1.6.6: The breakdown of Waltz filtering.

One search method which cannot easily be used for finding good interpretations, is the filtering technique which works so well for finding consistent labellings in a restricted domain (see section 1.4.3). The method depends on being able to show that labels are impossible because there are no compatible labels for neighbouring junctions. If, however, neighbouring junctions may have very unlikely labellings, based on non-general viewpoint, then it is hard to eliminate any labels. It can be disastrous to remove a label unless it is definitely impossible, since if a correct label is accidentally eliminated, this can cause the elimination of the correct labels from neighbouring junctions, and the effects can propagate until no labels are left anywhere. There is

little hope of noticing when a correct label has been eliminated and backtracking, since the correct label may not be the last one to be removed from a junction. Also, the divergent effects of some removals, which give Waltz filtering its power, make it very hard to trace and unpick the effects of an erroneous removal. This divergence of effects is also a major difficulty for the method of making assumptions and correcting errors when they are discovered. There seems to be no limit to the potential consequences of an assumption, and hence no limit to what an error-correcting procedure might need to do to unpick these consequences.

1.7: Explicit numerical scores

One way of determining how to make complex trade-offs between hypotheses is to give them explicit numerical scores, and to define the global best fit as the one which maximises the sum of the scores of its constituent hypotheses. This means that finding the best global interpretation becomes what is known in the operational research literature as a "linear programming problem" (often abbreviated to an "L.P. problem"). More specifically, it is a "zero-one" programming problem because in any solution the hypotheses must have truth values of zero or one. The following sections attempt to answer a number of issues concerning the validity and usefulness of explicit numerical scores:

1. What is the underlying justification for the individual scores used, and for the method of combining them?

2. What are the advantages of having a simple numerical definition of the optimum?

3. Is it sensible to introduce real numbers given that a major feature which differentiates the scene analysis approach from pattern recognition is its commitment to reasons and symbolic descriptions instead of numerical weights?

1.7.1: Probabilities and the costs of hypotheses

In Capital, Marx puts forward the idea that there must be some common underlying essence shared by all goods in order to explain how they can be given prices according to which they are exchanged. The same philosophical point seems to apply to hypotheses. There must be some property which they share in order to explain how they can be given scores according to which they are traded. The obvious candidate is probability. If the global best fit is defined as the least improbable set of consistent hypotheses, and if hypotheses are given negative scores (costs) corresponding to the logs of their probabilities, then minimizing the sum of the costs of the hypotheses will indeed produce the globally most probable interpretation, (assuming that the probabilities

are independent, so that the most probable interpretation is the one whose constituent hypotheses have the greatest product of probabilities).

It is not obvious how to apply probability theory to perception in order to assign costs to hypotheses, and it is particularly difficult to make the probabilities independent. However, Woods (1976) successfully employs explicit numerical scores based on probabilities in HWIM, a speech understanding system. The scores are necessary because conflicts arise between knowledge sources of quite different kinds. For example, a poor phonemic interpretation may be chosen because it allows a much better pragmatic interpretation or vice versa. The scores are discovered by collecting statistics in cases where the correct interpretation is known. The method used in HWIM to find the best global interpretation (see Section 5.5) is different from the method examined in this thesis, but it is encouraging that the theoretical arguments presented here in favour of explicit numerical scores are supported by the practical usefulness of such scores in a large program dealing with real data.

1.7.2: The advantages of a numerical definition of the optimum.

One major advantage of using explicit numerical values is that they provide a way of settling unforeseen conflicts between hypotheses of quite different types.

They also make it clear just how diverse, separate considerations can combine to overwhelm an hypothesis, a process which is hard to implement otherwise and tends to be avoided or glossed over within the framework of procedurally embedded optimization.

Another advantage of using explicit numerical scores is that they allow the problems of optimization to be abstracted from the welter of specific visual knowledge. There is, of course, a danger in attempting to impose a uniform optimisation system on visual processing. The appropriate use of domain-specific knowledge is often more helpful in deciding on the best interpretation than a lot of weighing of evidence based on an inadequate understanding. So an optimisation system is disadvantageous if its use of numbers rules out or discourages the use of any of the great variety of types of inference needed for scene analysis. This criticism, however, does not seem to be applicable to a system which can handle arbitrary logical relations between hypotheses.

1.8: Pattern Recognition and the Misuse of Numbers

The systematic use of real numbers and the accompanying mathematical arguments are regarded with suspicion by many workers in Artificial Intelligence. One of the main reasons for this suspicion is the inappropriate way real numbers were used in early attempts to produce shape recognition systems. Books and journals (e.g. Pat-

tern Recognition) were filled with papers discussing mathematical methods and theorems which assumed a formalisation of the process of perception that was inadequate. This section analyses the defects of the pattern recognition approach in order to show that the ways in which real numbers were inappropriately used there, do not necessarily rule them out for defining the global optimum.

1.8.1: The pattern recognition paradigm.

Given some fixed set of feature analysers, any spatial pattern can be described in terms of which features it has and which it lacks. If some standard, named patterns are described in this way then an unknown pattern can be classified as most similar to a particular standard pattern by comparing its feature set with the sets for the standard patterns. Different features may be given different real-number weights so that agreement with a standard pattern on some features is more important than on others. Major issues within the paradigm are how to select the best set of weights, and what features to use to achieve good separation of the standard patterns and to cope with size, position and orientation invariance.

1.8.2: Inadequacies of Pattern Recognition

The model outlined above suggests that the aim of perception is to classify a pattern, that the representations used are sets of features, and that the process consists of first extracting a feature set and then comparing it with stored sets. By contrast Artificial Intelligence research suggests that perception consists in producing a description of a scene using complex articulated representations (Minsky and Papert 1972), and that the processes involved are far more sophisticated than simply extracting and comparing sets of features.

The most obvious failing of the pattern recognition model is that it treats the input pattern as a whole. This presupposes that a sensible figure has already been segmented out (Hebb 1949, Neisser 1967) and it also precludes a recursive process in which description of the whole pattern may involve applying equally powerful descriptive apparatus to parts of the pattern (Minsky & Papert 1972). Except in special cases, such as the recognition of separate, upright, typewritten letters, the types of representation and processes needed for the presupposed initial segmentation are far more complex than the feature sets, and the process of comparing them which is meant to model recognition. For example, the programs of Guzman (1968), Clowes (1971) and Waltz (1972) use a relational network to describe the picture structure before starting on segmentation. This data structure

itself is much richer than a set of features.

Understandably, pattern recognition tends to avoid 2-D pictures of 3-D scenes. It has no way of coping with the way in which the appearance of a three-dimensional object is affected by occlusion, lighting and the picture-taking process. There is no simple way of initially normalizing the figure nor is there an adequate set of features which are invariant under the transformations.

It is true that people may have been attracted to the pattern recognition paradigm because it allowed known mathematical techniques to be applied to the selection of feature weights. It is also true that preoccupation with the weights and with ways of tuning them may have distracted people from noticing obvious inadequacies of the model. For example, a perceptron using local features cannot successfully discriminate between the connected and disconnected patterns in figure 1.6 (Minsky and Papert 1969). However, neither of these points implies that a successful formalisation of perception should avoid real numbers or systematic ways of manipulating them based on mathematical principles. Associating real number scores with hypotheses does not commit one to any particular kind of representation in the same way as the use of feature weights does. It will be shown (section 3.6) that any truth-functional logical relation can still be used, so that inferences based on occlusion, lighting, support,

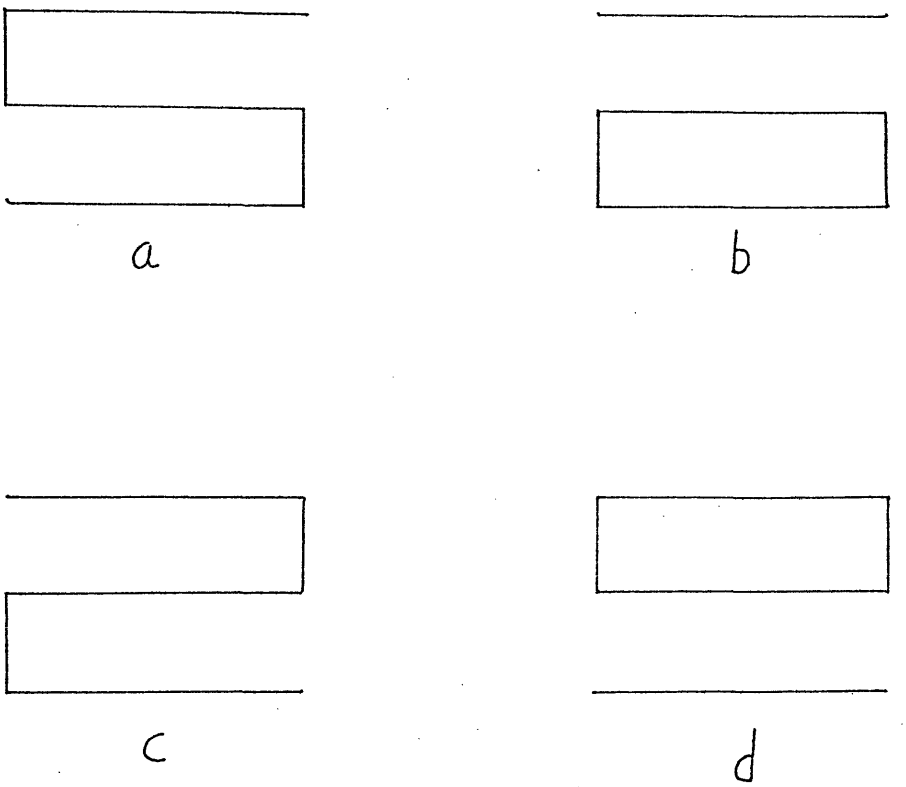


FIGURE 1.6: The connected figures (a and c) cannot be classified differently from the disconnected ones (b and d) by a perceptron with local feature detectors which are too small to encompass both ends of one of these figures simultaneously. The relationship between the sets of local features at the two ends is crucial, and it cannot be represented by a perceptron.

or the picture-taking process can, in principle, be integrated with the recognition of particular shapes. Similarly, giving hypotheses numerical values does not commit one either to a pass-oriented or to a heterarchical approach (Winston 1977) to the process of perception.

1.9: Branch-and-Bound search.

Explicit numerical scores for global interpretations, can be used to evaluate contexts (partial solutions). This allows many poor contexts to be abandoned before they have been completed or reached a contradiction. A systematic way of using evaluations to decide which context to develop is presented by Hart, Nilsson and Rapheal (1968). The method depends on being able to set an upper bound on the score which could be achieved by completing a context. For example, if all the local scores are negative (costs), then the combined score for an incomplete context is an upper bound on the score for any completion of the context. During the search, a list of alternative contexts is created by branching at choice points. At each stage, the list is examined to find the context with the highest upper bound (e.g. the lowest accumulated cost). This context is then replaced by several new ones which are made by branching at the next choice point. The search terminates when there is a complete solution with a score higher than any of the other upper bounds.

A branch-and-bound search can be very efficient if it can find upper bounds on contexts that are not much higher than the actually achievable scores, but this is hard to do in complex domains. Without tight upper bounds, many alternative contexts will be examined, and the same criticism applies as to depth-first search. There will be a lot of duplication of work as the same local combinations of possibilities are examined within the context of different, but irrelevant, higher level choices. A similar duplication occurs in the storage of the alternative contexts during the search.

1.10: The Relevance of Parallel Hardware.

A common criticism of artificial intelligence programs, as contributions to psychology, is that they are tailored to serial digital computers, whereas neurophysiological evidence shows that in the brain many activities occur in parallel. It has been claimed for example, (Dreyfus 1972, Weizenbaum 1976) that human abilities such as inuitive thought and Gestalt perception depend on parallel, holistic processes which are qualitatively different from the sequential steps generated by a normal computer program. These criticisms are simply not relevant to one of the main functions of artificial intelligence programs, which is to investigate the suitability of particular kinds of representation for particular tasks. Also, the difference in hardware cannot be

used to rule out computer models, since any desired parallel machine can be simulated on a general-purpose digital computer.

There is, however, a core of truth in the objections. Within artificial intelligence it is accepted that different programming languages encourage different programming styles by making some operations (the primitives of the language) particularly easy (Sussman & McDermott 1972). It seems likely that the relative ease of different basic computations will depend on the nature of the hardware. So, unless efficiency and convenience are disregarded, different hardware, like different languages, may encourage different programs.

It is sometimes claimed that the higher levels of organisation of a program are determined more by the nature of the task than by the hardware. The history of heterarchy however, shows that hardware considerations can be relevant even to general organisational principles. It was found that it was very difficult to derive a clean line drawing of some blocks from the mass of grey-level data produced by a camera. Shirai (1973) showed how higher-level knowledge could be used to guide line finding and his program was used to support the idea that truly intelligent programs need rich interactions between experts in different domains, rather than a sequential, pass-oriented organisation. The application of this idea to low-level vision was attacked by Marr

(1975) who argued that the enormously powerful, parallel hardware known to exist in the brain, could produce much richer symbolic descriptions about edges than conventional A.I. programs, without invoking knowledge of particular objects. The dispute has not been fully settled, but there seems no doubt that claims about the existing hardware are a major ingredient of Marr's case.

An early candidate for a useful computational primitive which might be more efficiently implemented on parallel hardware was associative memory. Willshaw and Longuet-Higgins (1969) went beyond suggestive analogies with holography and demonstrated an efficient method, the associative net, for associating pairs of bit-patterns so that one member of a pair could be produced in response to the other. This technique has not been used in A.I. programs, partly because of the need to translate to and from bit-patterns, but mainly because, given a serial digital computer, it is easier to use techniques such as hash-coding than to simulate a parallel machine.

Another candidate for an important computational process that might be more suited to parallel hardware, is the problem, of selecting an optimal interpretation from among a network of conflicting and co-operative hypotheses. The desire to show how this process could be decomposed into parallel interacting sub-processes was a crucial consideration in the design of the relaxation method presented in Chapter 3. This is a very different

approach to first writing a slow, serial program and then appealing to parallel hardware as a way of speeding it up. Some programs written for a serial computer (e.g. a breadth-first search) may, perhaps, be easily transferable to parallel hardware, but the serial nature of many programs makes it hard for them to use parallel hardware effectively.

1.11: Summary of Chapter 1.

The thread of the argument of this chapter may not have been obvious, so it will be stated here without the examples, elaborations and diversions:

The main problem in vision is to specify the types of representations and the inferences and heuristics that are available to build the representation of a particular scene, given a picture or image of it. Disregard for these issues can lead to futile efforts like perceptrons. Also, unnecessarily difficult search problems can be created by using poor representations (Amarel 1968). However, except in toy worlds, it is necessary to formulate tentative hypotheses, and important theoretical issues arise about how to manipulate these. Sometimes these issues can be side-stepped by using more knowledge, but not always. Any complex visual system, especially one dealing with messy data, needs systematic and principled ways of handling tentative hypotheses. So this be-

comes an issue in its own right.

Searching for consistent sets of hypotheses by developing separate contexts may involve unnecessary duplication in both time and storage space. For line labelling, a constraint propagation method, like that used by Waltz (1972) or Fikes (1970) is much more efficient.

In complex worlds it is not possible to specify a grammar of allowable interpretations which rules out all but one or a few global interpretations. The concept of a good or optimal interpretation is necessary.

There are several ways of finding good global interpretations. However they cannot handle the complex and unforeseeable trade-offs that may arise between difficulties of different kinds (e.g. missing line segments versus unknown words in the Popeye domain). It would be useful if we could find a principled way of making the trade-offs at run-time. Explicit numerical costs based on probabilities provide this. Some of the largest A.I. systems for handling real data work this way.

Given numerical evaluation criteria, a branch-and-bound search is the obvious candidate. However, the use of separate contexts can be inefficient. It would be better to represent constraints between hypotheses explicitly, if this allowed a parallel, constraint-propagation method, like Waltz filtering, to be used. However, the selection of hypotheses must be driven by the need for

optimality as well as consistency, and it is not obvious how to do this with Waltz filtering.

CHAPTER 2

THE TASK OF SEEING SOME OVERLAPPING RECTANGLES AS A PUPPET.

Figures 2.1 to 2.10 show, among other things, the input and output of a computer program designed to find the best puppet in a network of overlapping transparent rectangles. The puppet may have some parts missing and there may be some extra rectangles which are not puppet parts. The best puppet is taken to be the one with the greatest number of instantiated joints between parts, unless additional instructions are given.

2.1: The ease and purpose of the task.

By artificial intelligence standards the task is a simple one. The only difficulty lies in defining how two parts should be related so as to constitute an acceptable joint. Once this has been specified the search for the best fit can be done fairly simply by standard techniques such as a branch-and-bound search (Nilsson 1971) or a depth-first search. The existing program, however, uses a relaxation technique for selecting the best global combination from a network of rival, candidate part and joint hypotheses. This makes it considerably more com-

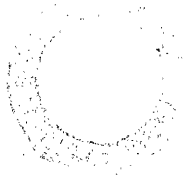
plex and probably slower than a conventional search for all the examples given. The point of the program is to illustrate and analyse the relaxation method in a simple domain. It is argued in chapter 4 that for more complex problems, especially with unreliable data and many layers of interpretation, a suitably modified form of relaxation is much faster than conventional search methods, especially if implemented on parallel processors.

2.2: Pictorial input.

Pictures are input on a graphics display terminal by drawing some overlapping rectangles with the cursor. Two sides of a rectangle are drawn and a program then completes it and gives it a single letter name. The names and corner coordinates of the rectangles are stored in a file. This file is the immediate input to the program.

2.2.1: The range of possible pictures.

Although it will happily accept parallelograms, the program is only intended for, and has only been tried on, scenes consisting of overlapping rectangles or near rectangles. Any configuration of these may be used. Isolated rectangles are simply ignored.



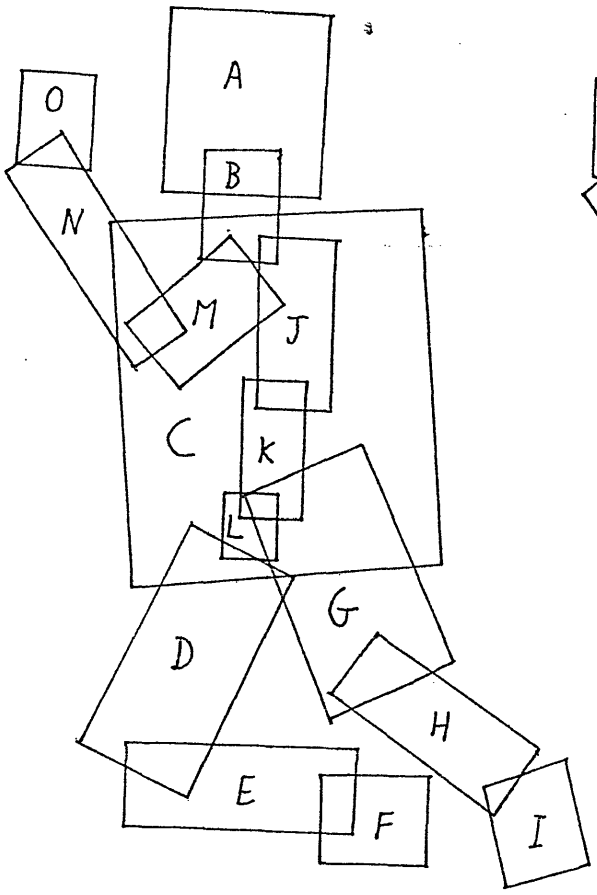


FIGURE 2.1a:

The input to the program.

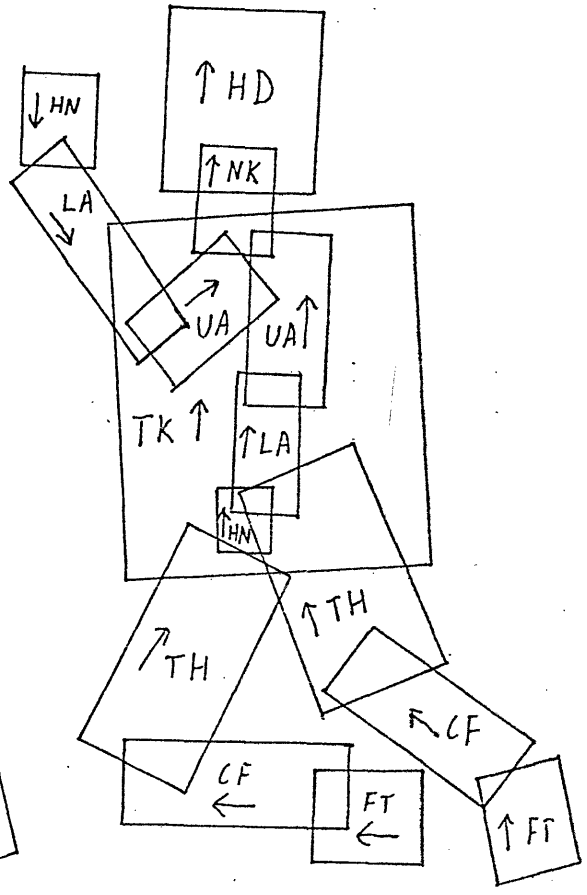


FIGURE 2.1b:

A pictorial interpretation of the program's output.

```
!.bestset;
A1   TOP   HEAD       NECK B1
B1   TOP   NECK       HEAD A1     TRUNK C2
C2   TOP   TRUNK       NECK B1     UPPERARM J1 M3     THIGH D3 G4
D3   TOP   THIGH       TRUNK C2     CALF E3
E3   TOP   CALF       THIGH D3     FOOT F1
F1   BOT   FOOT       CALF E3
G4   TOP   THIGH       TRUNK C2     CALF H5
H5   TOP   CALF       THIGH G4     FOOT I1
I1   TOP   FOOT       CALF H5
J1   TOP   UPPERARM       TRUNK C2     LOWERARM K6
K6   TOP   LOWERARM       UPPERARM J1     HAND L7
L7   TOP   HAND       LOWERAM K6
M3   TOP   UPPERARM       TRUNK C2     LOWERARM N4
N4   BOT   LOWERARM       UPPERARM M3     HAND O2
O2   BOT   HAND       LOWERARM N4
```

FIGURE 2.1c: The actual output of the program.

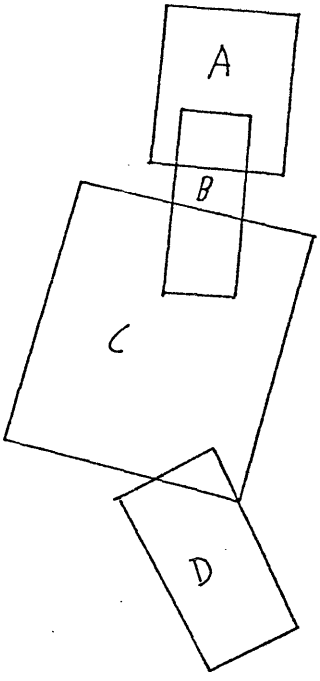


FIGURE 2.2a:
The input

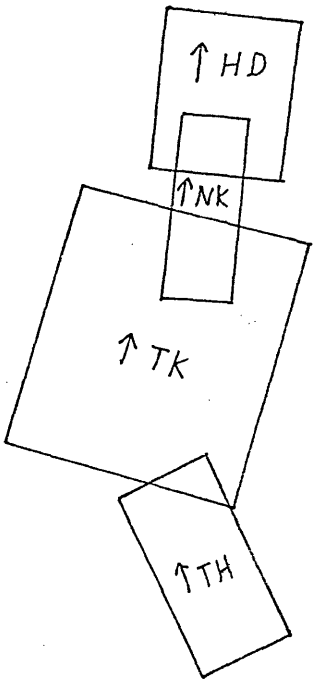


FIGURE 2.2b:
Interpretation of output

```
!.bestset;
A1   TOP   HEAD     NECK B1
B1   TOP   NECK     HEAD A1    TRUNK C3
C3   TOP   TRUNK     NECK B1    UPPERARM -    THIGH D3
D3   TOP   THIGH     TRUNK C3    CALF -
```

FIGURE 2.2c: The actual output of the program.

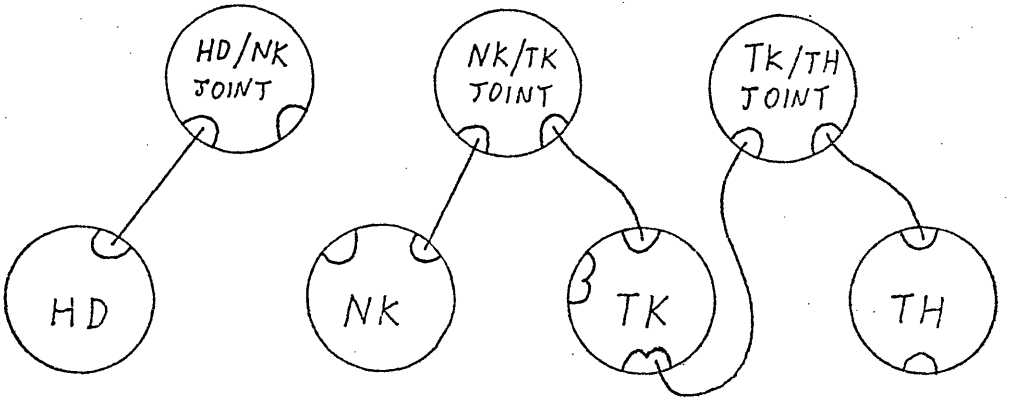
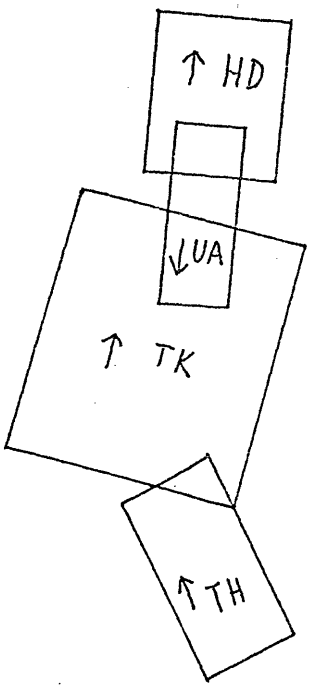


FIGURE 2.2d: The nodes in the relational network of part and joint hypotheses which form the best set. The indentations depict slots. The lines depict two-way links.

```
!.shownet;
A1   TOP   HEAD      NECK B1
B1   TOP   NECK      HEAD A1   TRUNK C3
B2   BOT   UPPERARM      TRUNK C3   LOWERARM -
C1   TOP   LOWERARM      UPPERARM -   HAND D2
C2   TOP   CALF      THIGH -   FOOT D1
C3   TOP   TRUNK      NECK B1   UPPERARM B2   THIGH D3
D1   TOP   FOOT      CALF C2
D2   TOP   HAND      LOWERARM C1
D3   TOP   THIGH      TRUNK C3   CALF -
```

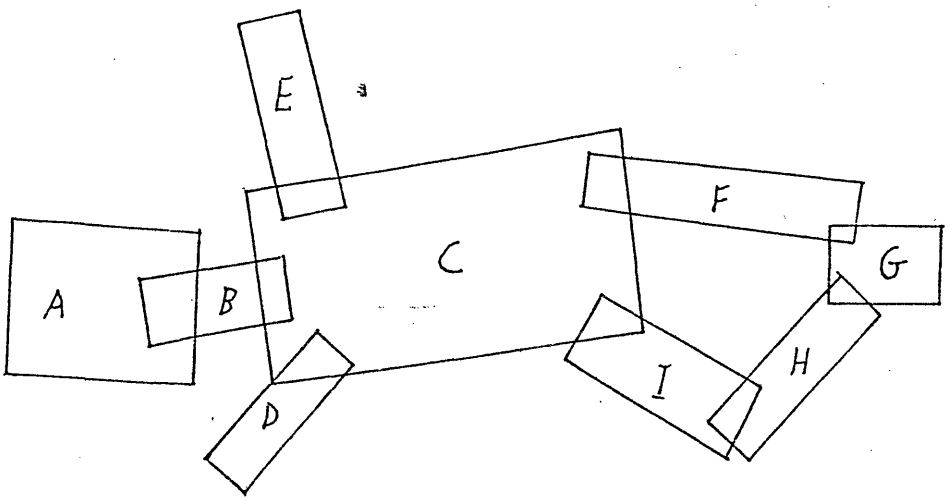
FIGURE 2.3a: The complete set of candidate hypotheses found by the program when given the picture in figure 2.2a.

```
!trytointerpret [b as upperarm importance=2];
```




```
!.bestset;
A1   TOP   HEAD      NECK -
B2   BOT   UPPERARM      TRUNK C3    LOWERARM -
C3   TOP   TRUNK      NECK -   UPPERARM B2    THIGH D3
D3   TOP   THIGH      TRUNK C3    CALF -
```

FIGURE 2.3b: An instruction given as additional input, with the resulting output, and its interpretation.



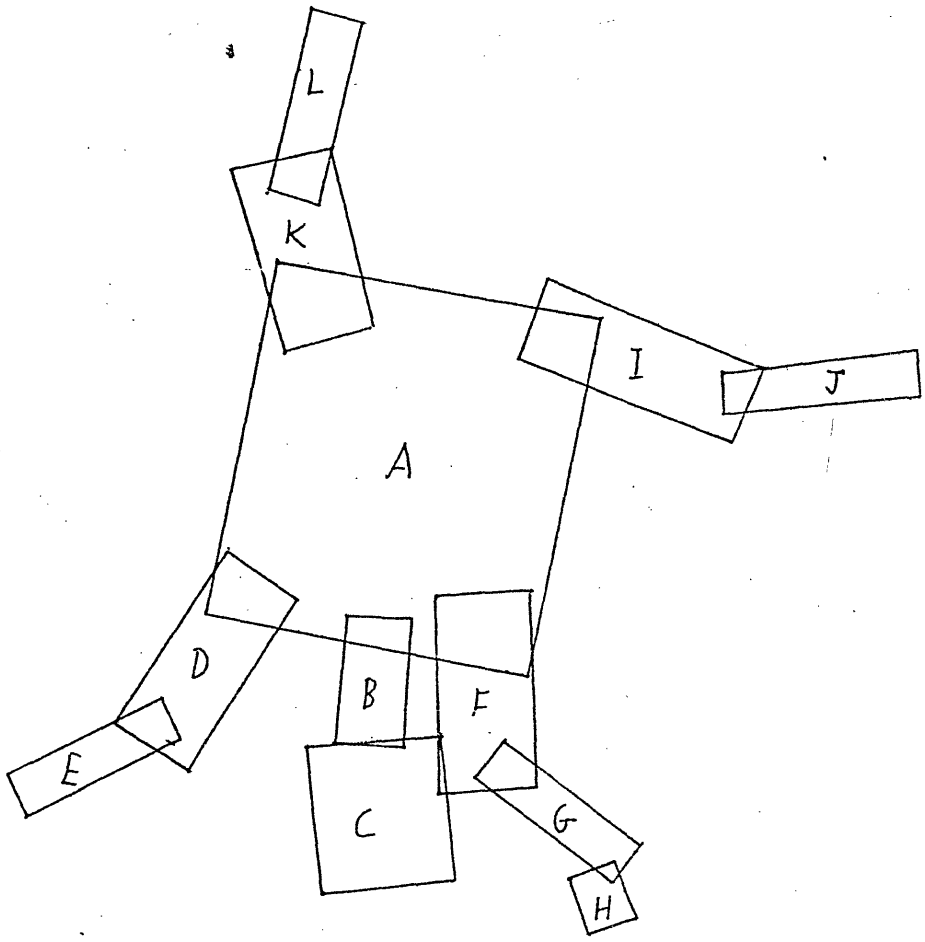

```
A1   TOP   HEAD     NECK B1
B1   BOT   NECK     HEAD A1    TRUNK C1
C1   BOT   TRUNK     NECK B1    UPPERARM D4 E4    THIGH F3 I3
D4   TOP   UPPERARM     TRUNK C1    LOWERARM -
E4   BOT   UPPERARM     TRUNK C1    LOWERARM -
F3   TOP   THIGH     TRUNK C1    CALF -
G3   BOT   FOOT     CALF H2
H2   BOT   CALF     THIGH I3    FOOT G3
I3   TOP   THIGH     TRUNK C1    CALF H2
```

FIGURE 2.4a: A picture and the program's output.

```
F1   BOT   NECK     HEAD G1   TRUNK C2
F2   TOP   UPPERARM     TRUNK C2   LOWERARM -
F3   TOP   THIGH     TRUNK C1   CALF -
F4   TOP   LOWERARM     UPPERARM C5   HAND G2
F5   TOP   CALF     THIGH C6   FOOT G3
```

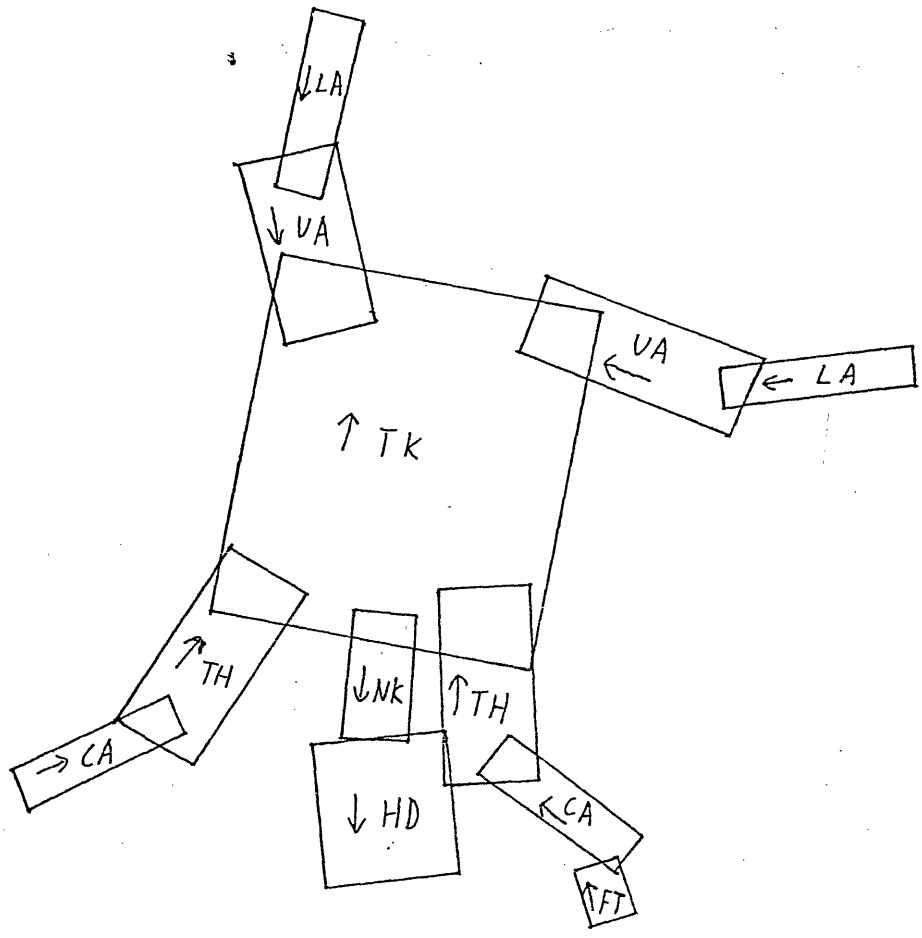
FIGURE 2.4b: The rival candidate hypotheses for F considered by the program.

Notice that the hypothesis selected by the relaxation process is one of the poorer ones in terms of its number of locally possible joints.



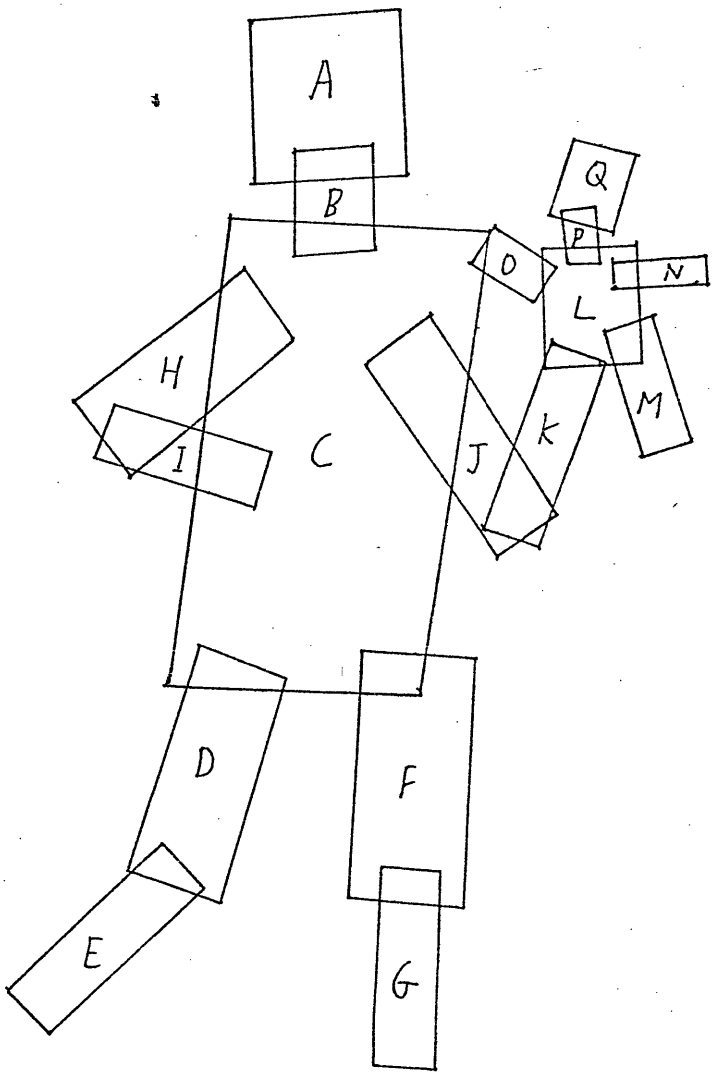

```
!.bestset;
A1   BOT   TRUNK      NECK B1    UPPERARM D2 F2    THIGH I3 K2
B1   BOT   NECK      HEAD C1    TRUNK A1
C1   BOT   HEAD      NECK B1
D2   TOP   UPPERARM      TRUNK A1    LOWERARM E4
E4   TOP   LOWERARM      UPPERARM D2    HAND -
F2   TOP   UPPERARM      TRUNK A1    LOWERARM G2
G2   TOP   LOWERARM      UPPERARM F2    HAND H2
H2   TOP   HAND      LOWERARM G2
I3   TOP   THIGH      TRUNK A1    CALF J4
J4   BOT   CALF      THIGH I3    FOOT -
K2   BOT   THIGH      TRUNK A1    CALF L4
L4   BOT   CALF      THIGH K2    FOOT -
```

FIGURE 2.5: A picture of an upside-down puppet, with the program's output. Unlike human perceivers, the program has no expectations about orientation, so it finds this picture no harder than one of an upright puppet.



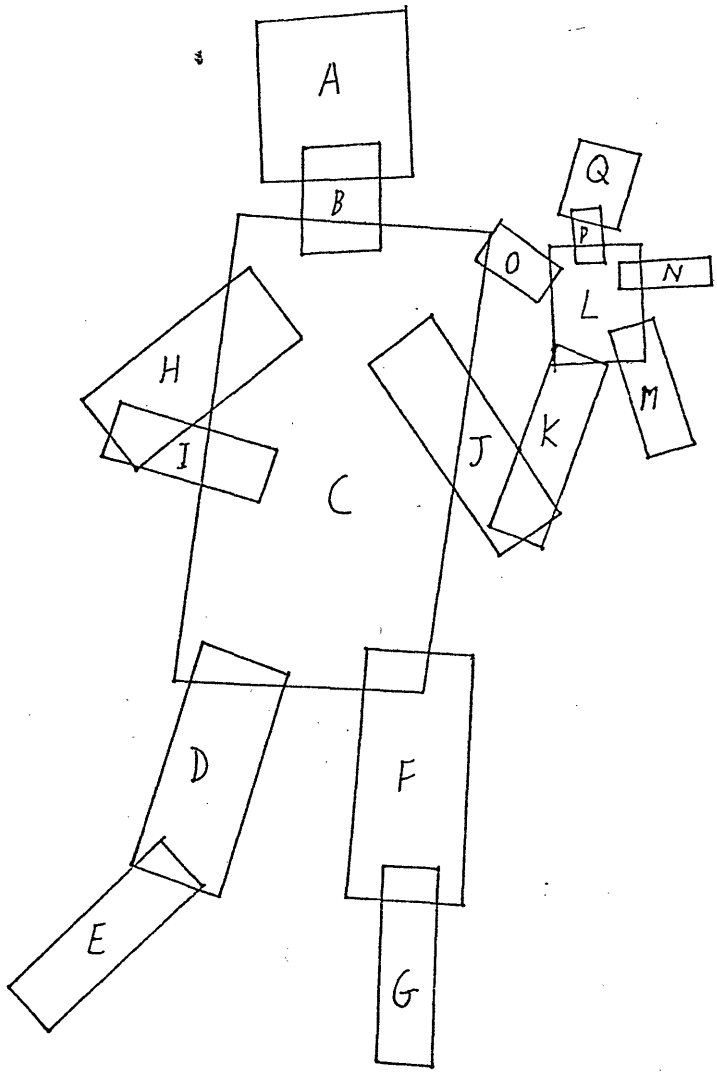

```
!trytointerpret [trunk as upright importance=1];
!trytointerpret [thish as upright importance=1];

!.bestset;
A2  TOP  TRUNK      NECK -   UPPERARM I2 K1    THIGH D3 F3
B1  BOT  NECK      HEAD C1    TRUNK -
C1  BOT  HEAD      NECK B1
D3  TOP  THIGH      TRUNK A2    CALF E3
E3  TOP  CALF      THIGH D3    FOOT -
F3  TOP  THIGH      TRUNK A2    CALF E3
G3  TOP  CALF      THIGH F3    FOOT H1
H1  TOP  FOOT      CALF G3
I2  TOP  UPPERARM      TRUNK A2    LOWERARM J3
J3  BOT  LOWERARM      UPPERARM I2    HAND -
K1  BOT  UPPERARM      TRUNK A2    LOWERARM L3
L3  BOT  LOWERARM      UPPERARM K1    HAND -
```

FIGURE 2.6: In addition to the picture, the program has been given instructions to attempt to find an interpretation in which the trunk and thighs are upright. It succeeds by finding an interpretation in which the trunk and neck are not connected.




```
!.bestset;
A1   TOP   HEAD      NECK B1
B1   TOP   NECK      HEAD A1    TRUNK C2
C2   TOP   TRUNK      NECK B1    UPPERARM H1 J1    THIGH D3 F3
D3   TOP   THIGH      TRUNK C2    CALF E3
E3   TOP   CALF    THIGH D3    FOOT -
F3   TOP   THIGH      TRUNK C2    CALF G3
G3   TOP   CALF     THIGH F3    FOOT -
H1   TOP   UPPERARM      TRUNK C2    LOWERARM I1
I1   TOP   LOWERARM      UPPERARM H1    HAND -
J1   TOP   LOWERARM      TRUNK C2    LOWERARM K4
K4   BOT   LOWERARM      UPPERARM J1    HAND L6
L6   BOT   HAND      LOWERARM K4
```

FIGURE 2.7: A picture in which people see two puppets, and the program's output, corresponding only to the best puppet.




```
!switchattention(0.5);

!.bestset;
L2   TOP   TRUNK      NECK P1     UPPERARM N5 O4     THIGH M3
M3   TOP   THIGH      TRUNK L2    CALF -
N5   TOP   UPPERARM      TRUNK L2     LOWERARM -
O4   BOT   UPPERARM      TRUNK L2     LOWERARM -
P1   TOP   NECK      HEAD Q1     TRUNK L2
Q1   TOP   HEAD      NECK P1
```

FIGURE 2.8: The output constitutes a "residual" interpretation consisting mainly of rectangles which were uninterpreted in the first interpretation (see figure 2.7). The "switch attention" instruction gives added importance to interpretations of the previously omitted rectangles.

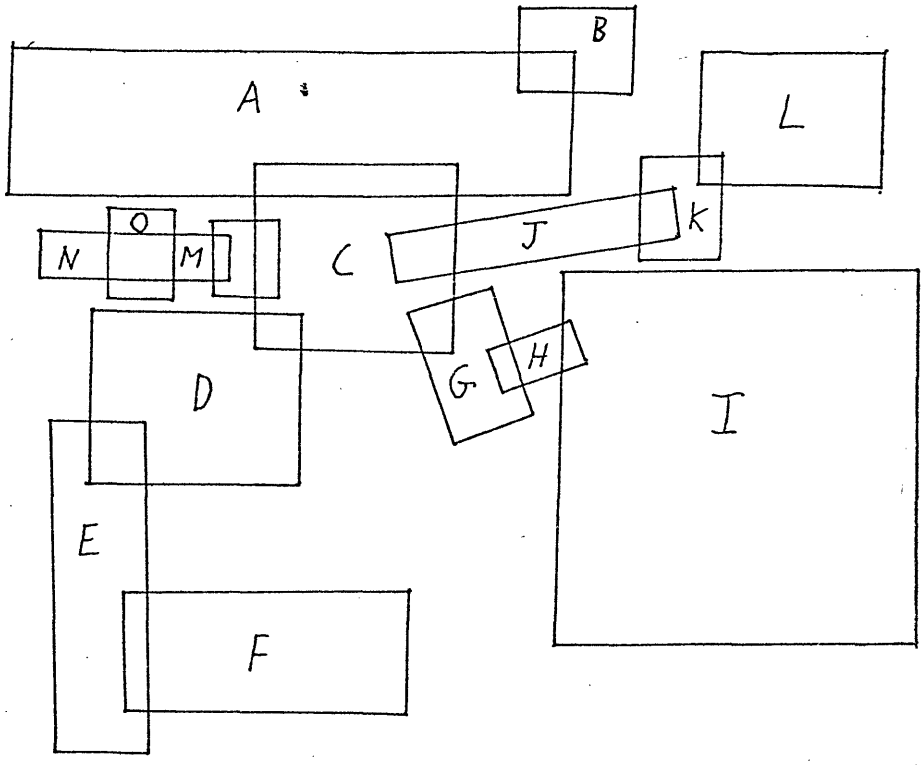
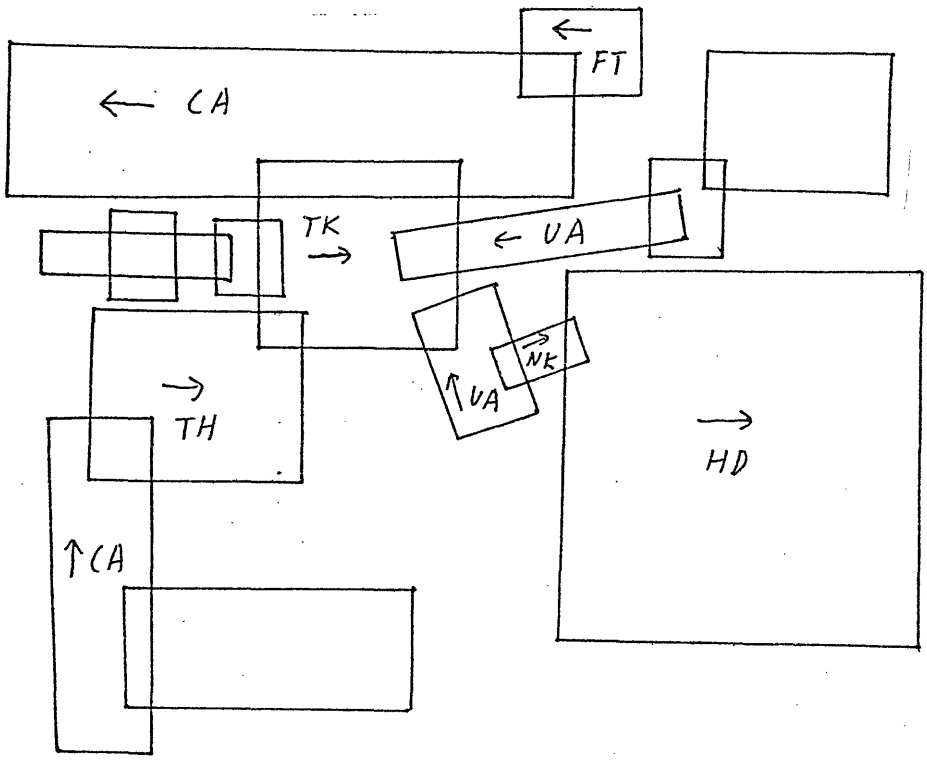


FIGURE 2.9a: A nonsense picture which has the same connectivity graph as a perfect puppet containing no "accidental" overlaps. The picture shows the importance of metric considerations.

```
?A1   BOT   LOWERARM     UPPERARM -   HAND ?B2
?A2   BOT   CALF      THIGH -   FOOT ?B1
?B1   TOP   FOOT      CALF ?A2
?B2   TOP   HAND      LOWERARM ?A1
C1   BOT   TRUNK      NECK -   UPPERARM G2 J2      THIGH D3
D3   BOT   THIGH      TRUNK C1    CALF E2
E2   TOP   CALF     THIGH D3    FOOT -
G2   TOP   UPPERARM      TRUNK C1    LOWERARM -
?H1   TOP   NECK      HEAD ?I1    TRUNK -
?I1   BOT   HEAD      NECK ?H1
J2   BOT   UPPERARM      TRUNK C1    LOWERARM -
?K1   TOP   NECK      HEAD ?L1    TRUNK -
?L1   TOP   HEAD      NECK ?K1
```

FIGURE 2.9b: The output of the program when it is given the picture above and allowed prolonged relaxation. The question marks indicate indecision. The way the program reacts to nonsense pictures is informative. It highlights the program's inadequacies as a model of human perception.

```
!trytointerpret [i as head importance=0.5];
!trytointerpret [a as calf importance=0.5];
```




```
!.bestset;
A2   BOT   CALF      THIGH -   FOOT B1
B1   TOP   FOOT      CALF A2
C1   BOT   TRUNK      NECK -   UPPERARM G2 J2    THIGH D3
D3   BOT   THIGH      TRUNK C1    CALF E2
E2   TOP   CALF      THIGH D3    FOOT -
G2   TOP   UPPERARM     TRUNK C1   LOWERARM -
H1   TOP   NECK      HEAD I1    TRUNK -
I1   BOT   HEAD      NECK H1
J2   BOT   UPPERARM     TRUNK C1    LOWERARM -
```

FIGURE 2.10: Two additional instructions are shown. When these are given with the picture in figure 2.9a, they break the deadlock between equally good, partial interpretations seen in figure 2.9b. The output of the program and its pictorial interpretation are shown.

2.3: Non-pictorial input.

Various kinds of instruction can be given about how to try to interpret a picture. The instructions always have an associated number which indicates how important it is to obey them (any real number is allowed). The types of instruction are:

1. Try to interpret a particular rectangle as a particular puppet part. The instruction may also indicate which way up the part should be, by saying whether its proximal end (see below) should be at the top or the bottom of the rectangle depicting it. A part is "upsidedown" if its proximal end is at the bottom of the rectangle depicting it.

   e.g. TRYTOINTERPRET [A AS HEAD IMPORTANCE = 1];

   or TRYTOINTERPRET [A AS UPSIDEDOWN HEAD IMPORTANCE = 0.5];

2. Try to interpret a particular rectangle as some part of the puppet.

   e.g. TRYTOINTERPRET [A AS SOMEPART IMPORTANCE =1];

If the importance is negative the instruction means: Try not to interpret the rectangle as any puppet part.

3. Try to find a global interpretation (i.e. a consistent set of local part and joint interpretations) in which a particular puppet part is a particular way up (only two orientations can be specified, though more could easily be allowed):

e.g. TRYTOINTERPRET [TRUNK AS UPRRIGHT IMPORTANCE = 3];

4. After the best global interpretation has been found the program may be instructed to try for a residual global interpretation which tends to contain those rectangles not included in the first interpretation and which also tends not to contain those rectangles previously included. The importance of containing or not containing rectangles of the two types is given as a parameter:

e.g. SWITCHATTENTION (0.3);

Any combination of instructions may be given. The effect is to alter the definition of what constitutes the best interpretation. The basic default requirement is to find as many compatible instantiated joints as possible with an importance of one for each joint. The additional instructions have the effect of assigning importances to particular interpretations of rectangles of puppet parts. If several instructions match the interpretation of a rectangle as a puppet part, then their importances are added to get the importance of including that interpretation. The best puppet instantiation is the one whose constituent parts and joints have the greatest sum of importances.

2.4: Output of the best global interpretation.

When the relaxation process has finished there will be a network of part and joint hypotheses which are regarded as correct. This network is output by showing its part hypotheses, each of which is specified by its name followed by its orientation, its type and the joints filling its slots. The names of the part hypotheses are made by appending successive integers to the names of the corresponding rectangles. Their orientations are two-valued and depend on whether the proximal end is at the top or the bottom of the depicting rectangle. (Every puppet part has a proximal end and a distal end. The proximal end is the one anatomically closest to the top of the head. The arrows in figure 2.1b indicate which is which). The joints in a slot are specified by following the slot name with the name of the related part hypotheses.

2.5: The puppet model

A perfect puppet consists of fifteen rectangular parts having the following properties and relationships:

1. Each part has a proximal end and a distal end. The proximal end is the one anatomically nearest to the top of the head. The length of a part measured along the proximal distal axis is greater than its width.

2. The trunk is wider than any of the upper limb

parts and each of these, in turn, is wider than its connected lower limb part. Also the head and trunk are wider than the neck.

3. The head is greater in area than the neck and the lower limb parts are greater in area than their associated hands or feet.

4. Anatomically connected parts overlap in the right way (see below) to depict a joint.

The precise details of the puppet model cannot be justified in terms of human perception, but something more than simple connectivity must be used to exclude cases like figure 2.9a.

2.5.1: Defining satisfactory joints.

Figure 2.12 shows some pairs of overlapping rectangles which have been assigned a distal-proximal direction. Some pairs can plausibly be interpreted as depicting knee-joints and others cannot. One method for defining these two classes is in terms of the way in which the sides and ends of the rectangles intersect, The examples given, however, show that these intersections are rather varied, and it is difficult to find a natural definition in terms of them. It seems as if the intersections of the edges are more the result of the way the rectangles overlap than a defining characteristic of their relationship. A simpler and more intuitively satisfactory way of

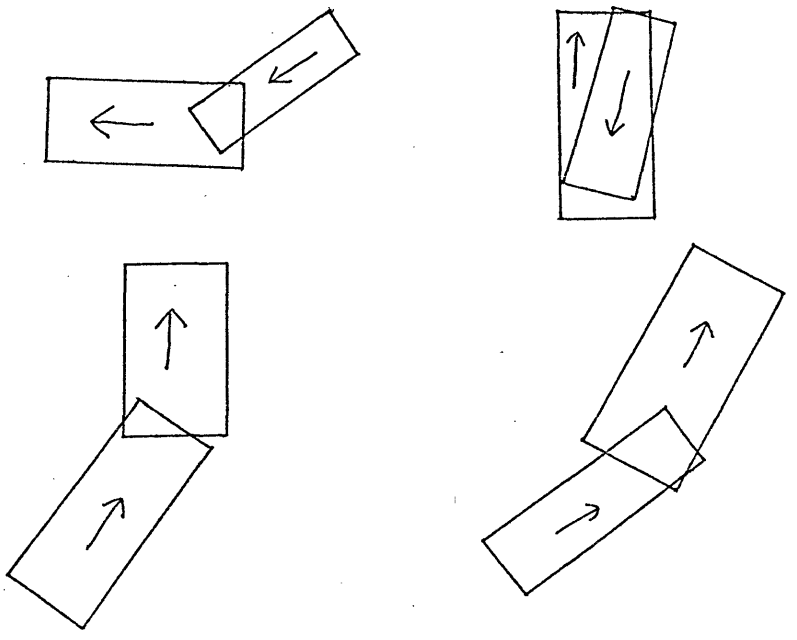


FIGURE 2.12a: Some examples of possible knee-joints. The arrows indicate the distal→ proximal direction. The thigh is the wider of the two parts. Notice the variety of ways in which the ends and sides intersect.

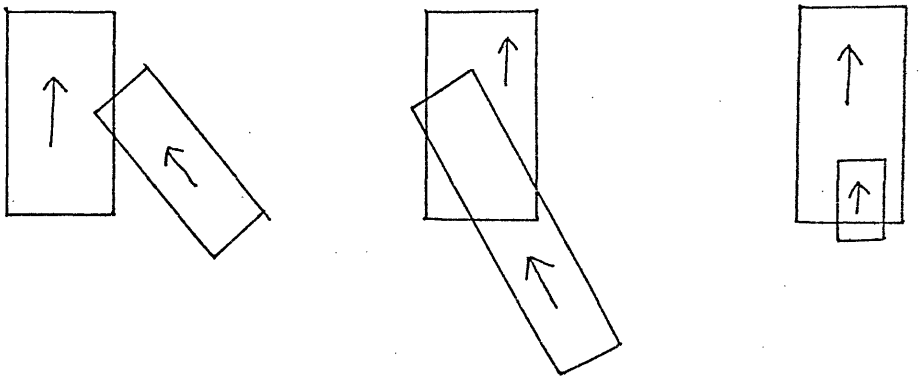


FIGURE 2.12b: Some unsatisfactory knee-joints.

articulating spatial relationships between rectangles is to specify a set of zones in each rectangle, and then to specify pairs of zones, one in each rectangle, which do or do not overlap. Using this method, the examples given in Figure 2.12 can easily be separated into satisfactory and unsatisfactory knee joints on the basis of the zone overlaps defined in Figure 2.13. The use of zones rather than edges to define spatial relationships is a simple example, in two dimensions, of the "space occupancy" idea referred to by Brady and Wielinga (1976). Paul (1977) defines satisfactory relationships between parts of a puppet in a similar way. The necessary and sufficient definitions of all the various joints in the puppet are shown in Figure 2.13b. These are not fully adequate because they are all or none. They do not allow for poor but not hopeless joints. One way in which people are more flexible (as perceivers) is that they will allow some relations or proportions to be stretched provided the rest are reasonable. The implications of this will be discussed in Section 4.7.

2.6: Definition of the required output.

2.6.1: What pictures depict.

When we perceive the real world there is a clear distinction between how things are and how they appear to be. We can make mistakes, and it is quite possible under

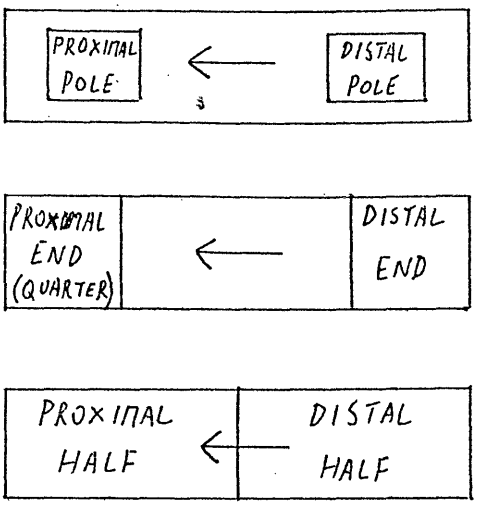


FIGURE 2.13a: Six zones of a puppet part which has been assigned a distal → proximal direction.

CALF or LOWER-ARM	THIGH or UPPER-ARM	OVERLAP?
P. E.	D. E.	MUST
P. E.	P. H.	MUST NOT
D. H.	D. E.	MUST NOT

FOOT or HAND	CALF or LOWER-ARM	OVERLAP
D. E.	WHOLE	MUST NOT
WHOLE	P. H.	MUST NOT

THIGH	TRUNK	OVERLAP?
P.E.	D.H	MUST
P. E.	P.H	MUST NOT
D.E.	D. POLE	MUST NOT

UPPER-ARM	TRUNK	OVERLAP?
P.E.	P.H.	MUST
P.E.	D.H.	MUST NOT
D.E.	P. POLE	MUST NOT

NECK	TRUNK	OVERLAP?
D.E.	P.E.	MUST
WHOLE	D.H.	MUST NOT
P.H.	WHOLE	MUST NOT

NECK	HEAD	OVERLAP?
D.E.	P.E.	MUST
WHOLE	D.H.	MUST NOT
P.H.	WHOLE	MUST NOT

P. = Proximal D. = Distal E. = End H. = Half

FIGURE 2.13b: Showing the definition of satisfactory joints used by the program. The two whole rectangles are assumed to overlap. There are also constraints on relative lengths, widths and areas (see section 2.5). Notice how the definition of a knee-joint applies to figure 2.12a.

suitable circumstances for an object to consistently appear to be something which it is not. The Ames Room is a compelling example. The same distinction holds for photographs, but for pictures there is no such simple distinction between what they appear to depict and what they actually depict. In some cases it may be possible to decide what a picture really depicts by appealing to the intentions of its creator, the conventions of the picture-making process, or how the picture appears to normal perceivers. For example, such appeals may enable us to decide whether a given picture is an imperfect depiction of a perfect wire-frame cube or a perfect depiction of an imperfect one. For many puppet pictures the difficult decision between incomplete or imperfect depiction and depictions of the incomplete or imperfect, could arise. It will be avoided by assuming that the depictions are perfect. So missing rectangles mean that the puppet is incomplete, not the picture.

2.6.2: Basic definition of the best puppet

When there is nothing better in the picture people happily find incomplete puppets. The program can do the same if it is given some way of evaluating incomplete puppets so that it can avoid poor global interpretations when there are better alternatives. Currently, the best puppet is defined as the one containing the greatest number of satisfactory joints whilst satisfying the fol-

lowing constraints:

1. No rectangle can be seen as more than one part.

2. A part may be involved in several joints but no part can have more joints than in a perfect puppet. A trunk, for example, can not have three thighs, nor can a calf enter into two knee joints.

3. No type of part can be instantiated more times than it occurs in the model: e.g. there must not be more than two thighs.

4. A joint cannot exist unless both parts are instantiated.

This definition produces results similar to the perceptions of a person who is experienced in the domain and knows what the task is. It is hard to assess how well it does because people seem to have the ability to learn to see the picture in the way the program does. The author's considerable perceptual experience of the domain, for example, may have evolved to fit the program as well as vice- versa. An interesting feature of the definition is that it allows disconnected instantiations as in Figure 2.6 . People can also see disconnected instantiations but they notice that they are disconnected which the program does not. Also if the best interpretation is severely disconnected, as in Figure 2.14, people often notice just how a rectangle would have to be added to unify the figure, and they may report what they see in

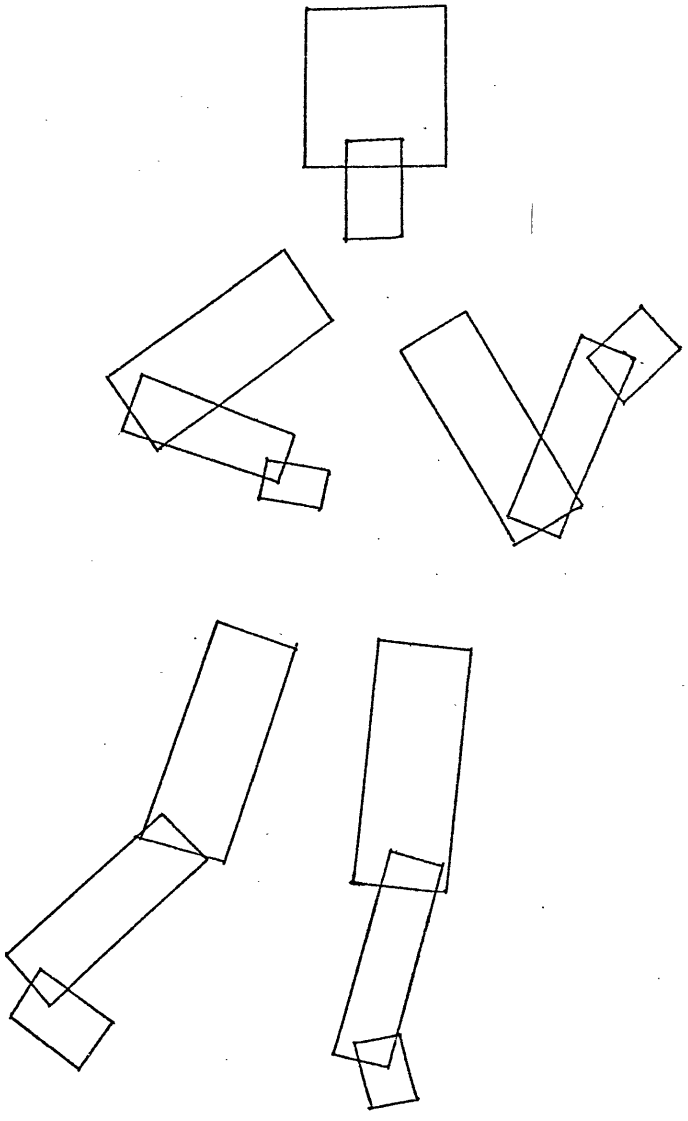


FIGURE 2.14: A puppet with a missing trunk. People notice that the limbs are correctly related despite the absence of the trunk. This is beyond the current program.

these terms. Such abilities are beyond the current program.

2.6.3: Modification of the definition of best.

The specific instructions which may be given as input, along with the picture, can alter the definition of the best puppet by attaching importances to the interpretation of rectangles as puppet parts, but the instructions cannot affect the four types of constraint that are listed above. So, for example, the program cannot be told to look for a one-legged or a three-legged puppet. The instructions are also unable to affect the relative proportions and the spatial relations which rectangles must have in order to depict a joint.

2.6.4: Equal rivals.

When there are several different optimal interpretations it is reasonable to demand that a program give them all. This could be achieved by adding control facilities to the current program, but that would raise issues beyond those which the program was designed to investigate. So when there are equal rivals the program is not required to give a decisive output until given additional specific instructions which favour one rival over the others as in Figure 2.10.

Chapter 3

THE PUPPET FINDING PROGRAM

The aim of this chapter is to describe the program at a level above that of its implementation in a particular language, but in sufficient detail to enable anyone familiar with the language to follow the code. First there is a description of how the puppet task is reduced to the problem of finding the best consistent set among some logically related hypotheses. Then the principles behind a relaxation method for solving the problem are given. Finally, there are detailed examples of the method applied to various puppet pictures.

3.0: The two main stages : An overview

The program works in two stages. First, many locally feasible part and joint hypotheses are created, and the constraints between them are explicitly represented. Each hypothesis is then assigned an arbitrary supposition value, which can be interpreted as the extent to which the program is currently supposing the hypothesis to be correct. The values are iteratively modified so as to satisfy numerical constraints, derived from the logical

relations between hypotheses, whilst maximizing the supposed number of instantiated joints. When this relaxation process finishes, the hypotheses corresponding to the best puppet will generally have supposition values of 1 and the rest will have values of 0.

3.1: The main data-structures and their creation

When given a picture, the program forms three different but interlinked networks whose nodes represent rectangles, hypotheses, and suppositions (see below). First it creates a data-structure for each rectangle and gives it a list of the overlapping rectangles and structures for the zones within the rectangle. Then it creates part hypotheses, which are interpretations of rectangles as puppet parts in particular orientations, and joint hypotheses, which are interpretations of the spatial relationships between rectangles as joints between puppet parts.

The reason for having an explicit structure for a joint, rather than simply giving each slot in a part hypothesis a pointer to the related part hypothesis, is so that the program can refer directly to the joint and can associate other information with it.

When the process of finding candidate local hypotheses terminates, there is, generally, a surfeit of hypotheses, and before the best consistent set of these

can be selected, it is necessary to instantiate the constraints between them. To do this, each hypothesis is given an associated supposition node which contains its importance (how important it is to include it in the final interpretation), its supposition value (which arbitrarily starts at zero), and lists of the constraints on its supposition value which are derived from the definition of the best puppet by the method described in section 3.8.

Figures 3.1, 3.2, and 3.3. show the three networks built by the program for a simple picture. Notice that constraints are not directly linked to hypotheses but rather to their associated supposition nodes. This allows a modular program in which the particular structures used for hypotheses need not be accessed during the relaxation process for finding the best consistent set. So the code for this process, can be independent of any particular domain.

3.1.1: Representing zones and computing their overlaps

There are six relevant, rectangular zones in each rectangle (see figure 2.13) as well as the whole rectangle itself. The only computation in which zones are used is for deciding whether or not two of them overlap, and so their representation is designed to make this judgement easy. Each zone is given pointers both to its corner points and to its four borders or half-spaces.

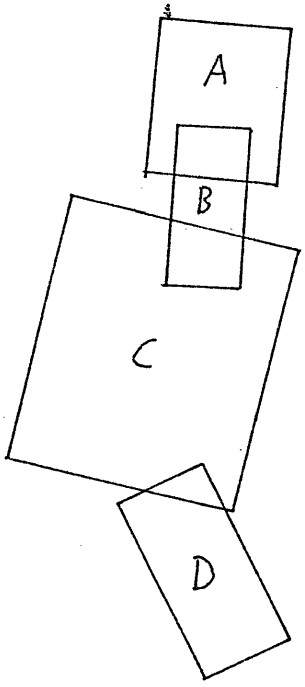


FIGURE 3.1a: A simple picture.

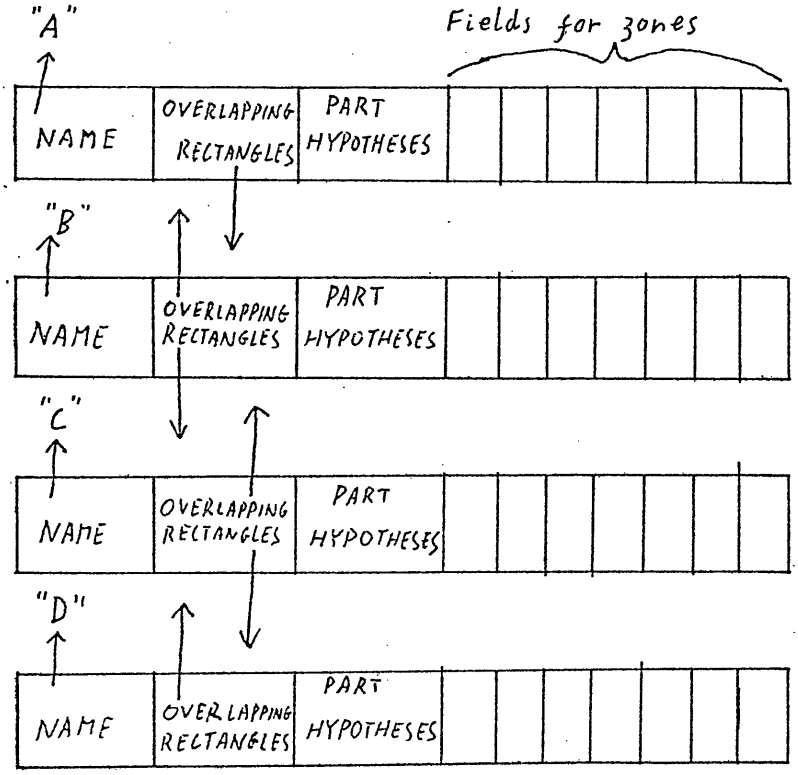


FIGURE 3.1b: The network of data-structures representing the rectangles in the picture above. Each structure also has pointers to all its corresponding part hypotheses.

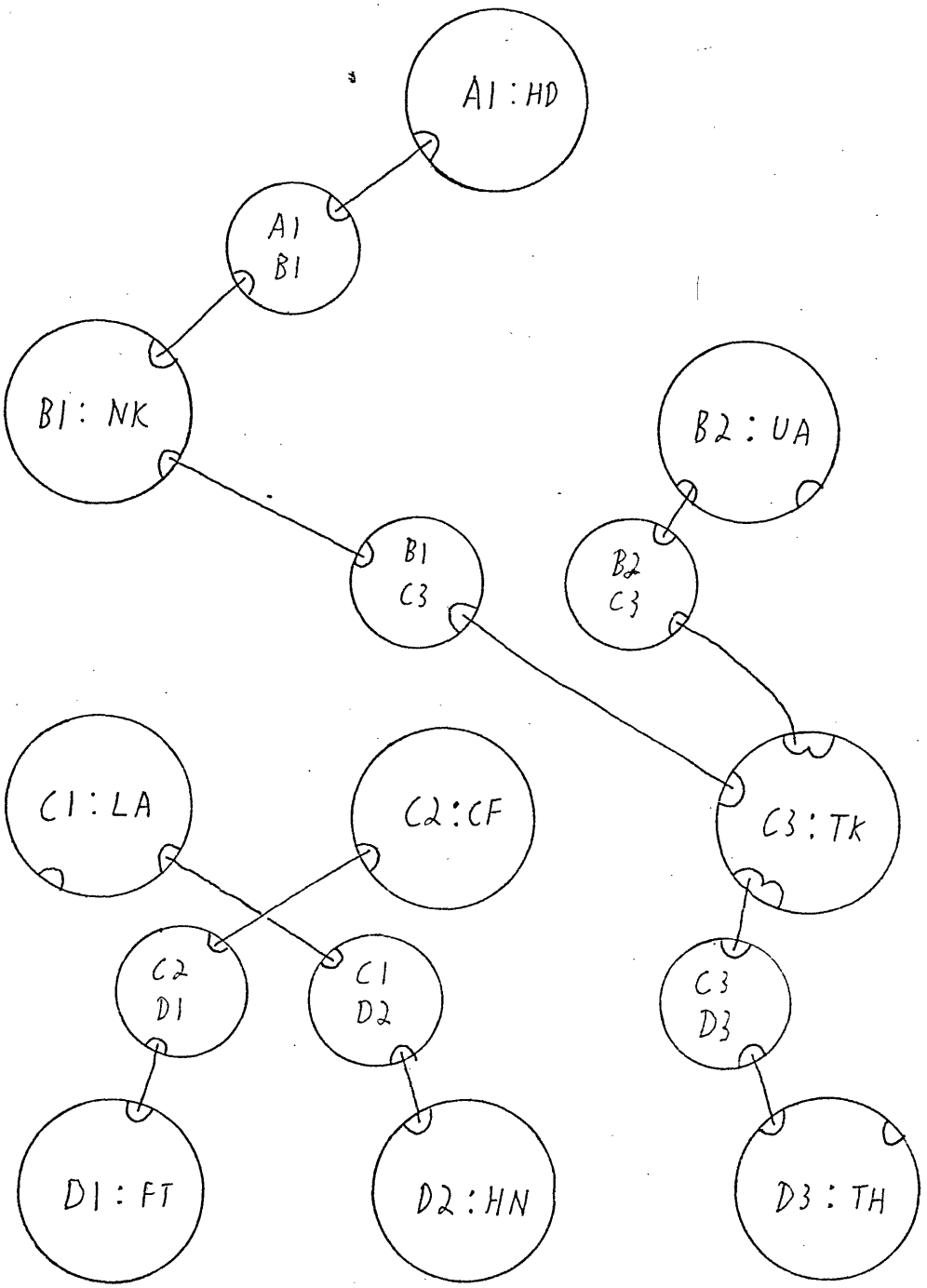


FIGURE 3.2: The network of candidate part and joint hypotheses for the picture in figure 3.1. (See figure 2.3a for an alternative representation). The indentations represent slots and the lines depict two way pointers. Every hypothesis also has a pointer to its supposition node, and part hypotheses have pointers to the data-structures for their rectangles.

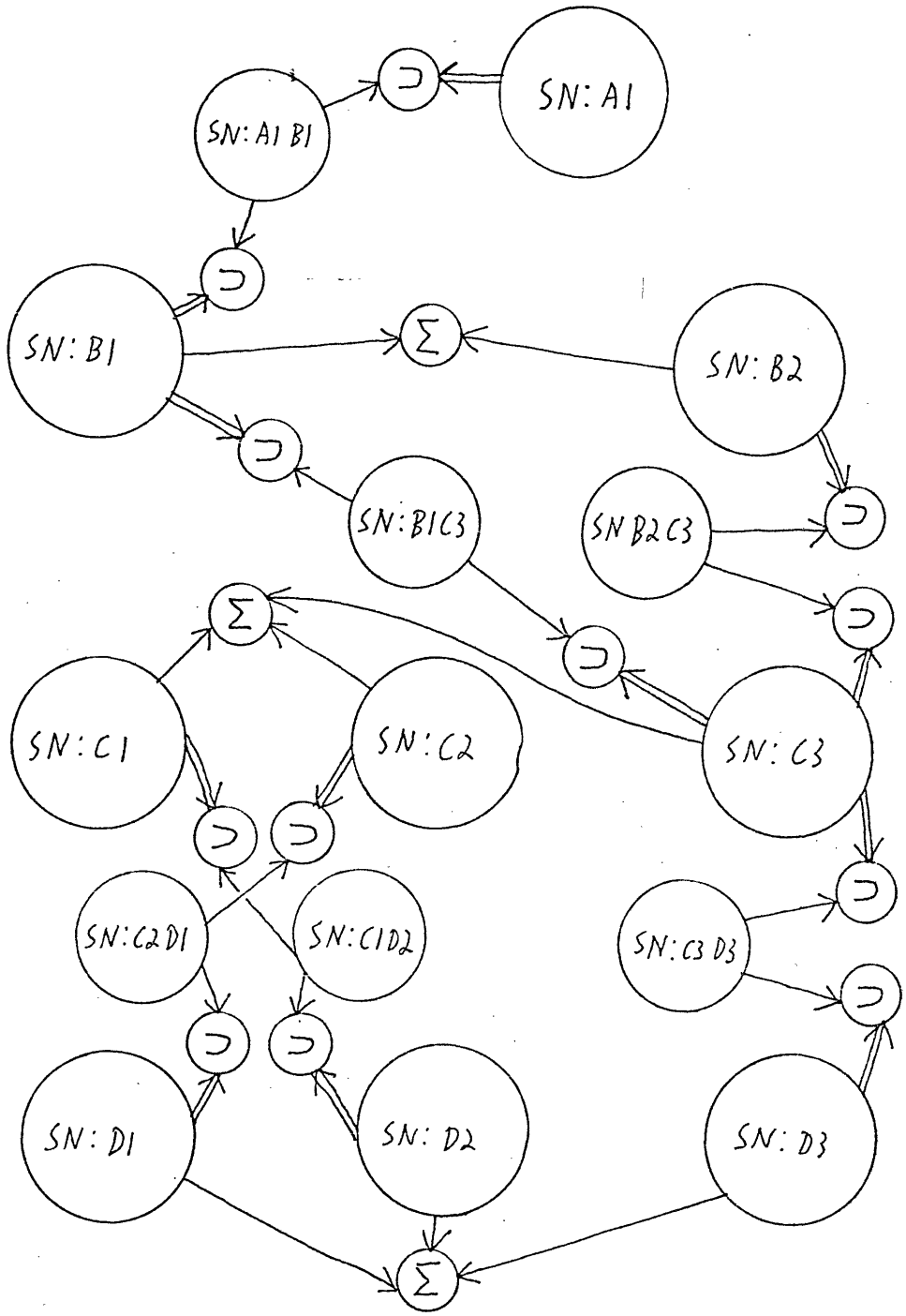


FIGURE 3.3: The network of supposition nodes associated with the hypotheses in figure 3.2. The prefix "SN:" is used to distinguish supposition nodes from hypotheses. Supposition nodes have pointers to constraints which, if violated, tend to raise (double arrow) or lower (single arrow) their supposition values.

The only constraints in this network are on the sum of the values for nodes corresponding to one rectangle(Σ), and on the relative values for joints and their parts(⊃).

Each border has an on-side and an off-side and the zone is the intersection of the four on-sides. Points actually on the border are taken to be on its on-side.

A border can always be expressed in the form: $a.x + b.y \geq c$ where the expression is true for points on the on-side. So if the border is represented by the coefficients a, b, c, it is easy to compute which side of it a given point lies on. Using this basic test, a procedure can quickly decide whether or not two zones overlap by using the fact that convex polygons are disjoint if and only if one of them has a border which has the other entirely on its off-side. This fact is not intuitively obvious, so in appendix 1 a construction is given which shows why there must be such a border if the polygons are disjoint.

The way in which zones are represented and overlaps are computed is not intended to have any psychological relevance.

3.2: Creating the network of candidate hypotheses

Creation of a network of conflicting and supporting hypotheses is the first stage in finding the best puppet. This section describes in detail how the network is made.

Since the relaxation process does not itself create new local interpretations, it is essential that all the correct hypotheses for the best puppet should exist be-

fore relaxation starts. One way of achieving this is to give each rectangle all possible part hypotheses and then to find all possible joints. This method is costly even for the puppet problem and would be worse for more complex cases. It does, however, guarantee that hypotheses will not be missed just because they are locally implausible, like the hand in figure 2.7. A more economical method, implemented in the program, is to start by creating hypotheses for those rectangles which have locally obvious interpretations. These initial hypotheses are called nuclei, because they act as a context which suggests interpretations for neighbouring, overlapping rectangles. (Woods (1976) uses "seeds" in a similar way in a speech understanding system.) These suggested interpretations can then, in turn, act as a context for interpreting their neighbours, and so on until a whole set of related hypotheses is formed around a nucleus. In fact, if the best puppet is connected and if it contains at least one nucleus then all its hypotheses will be found, however locally implausible some of them may be. In figure 2.4, for example, rectangle G is given one interpretation as a hand as a result of C being a trunk nucleus. Even if the best puppet has no nuclei it will still be found if any of its part hypotheses are created whilst developing other nuclei.

The program simulates the simultaneous spreading of interpretation from a number of independently discovered nuclei by using discrete time steps. On the first step

the nuclear hypotheses are made, and on each subsequent step attempts are made to fill the slots of the part hypothesis created during the previous step. For each slot, all overlapping rectangles are examined to find any which are related in the right way to depict the required puppet joint. Whenever a candidate joint is found, the program creates a joint hypothesis and also makes a new part hypothesis for the overlapping rectangle, unless one already exists. New part hypotheses act just like nuclei, and the process is continued until no new part hypotheses are created.

In more sophisticated uses of relaxation (see section 4.7), the process of growing candidate hypotheses is integrated with relaxation rather than being a separate first pass. An integrated approach is needed to avoid many of the enormous number of hypotheses that would be generated by a separate first pass in a complex domain.

3.2.1: Types of nuclei

The decision about what local configurations should constitute nuclei involves a compromise between having so many types of nuclei that a great number of irrelevant hypotheses are created and so few that the best puppet doesn't contain any. The program uses the following three types of nuclei which are normally adequate (but see figure 3.4).

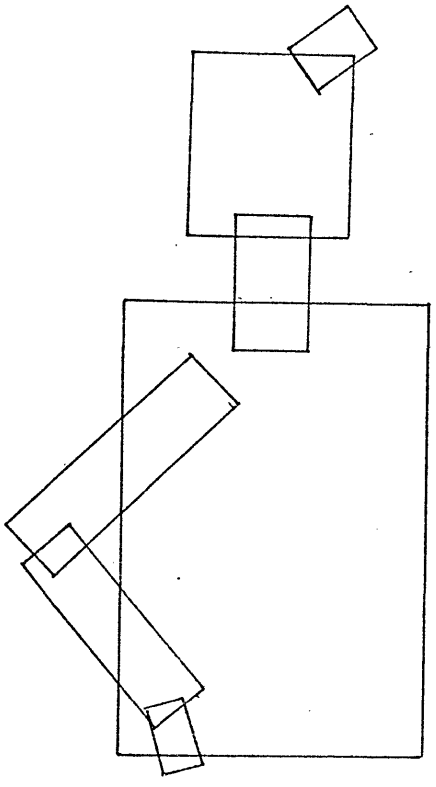


FIGURE 3.4: A picture in which there are no nuclei. The program cannot find the obvious interpration.

1. A rectangle which only overlaps one other and which is wider than it, is interpreted as a head if the other rectangle is so related that it could be a connected neck.

2. A rectangle which only overlaps one other and has less area than it, is given rival interpretations as a foot and a hand if the two rectangles satisfy the overlap requirements for a lower-arm/hand or a calf/foot joint.

3. A rectangle which overlaps three or more narrower rectangles is given an interpretation as a trunk if at least one of the overlapping rectangles is suitably related to depict an upperarm, thigh or neck. Usually, two rival trunk hypotheses with opposite orientations will be created.

There should, perhaps, also be a neck nucleus for a rectangle joining two wider ones. By stipulating that the central rectangle should have a smaller area than either of the other two, confusion with calves and lower-arms would be avoided.

A desirable feature of any set of nuclei, which helps to give it a reasonable performance over a wide range of pictures, is that some nuclei (e.g. 1 and 2 above) tend to remain, even when many rectangles are missing, whereas others (e.g. 3) are immune to extra irrelevant rectangles.

3.3: Numerical constraints between supposition values.

Consider the logical constraint $p \vee q$ and the numerical constraint $S_p + S_q \geq 1$ where S_p means the supposition value of the hypothesis p. The numerical constraint appears to be a good generalisation of the logical constraint because it rules out the same combination of integer values for p and q, (0,0). The advantages and weaknesses of this kind of generalization are discussed in the following sections.

3.3.1: The function of continuous supposition values.

The purpose of using continuous supposition values is to avoid explicit enumeration of combinations of the truth values of hypotheses during the process of searching for the best consistent set. The aim in choosing the numerical constraints between supposition values is to ensure that iterative adjustment on the basis of the numerical constraints leads to values of 1 for hypotheses in the best set and 0 for the rest.

3.3.2: States of supposition values: terminology

Sets of supposition values which satisfy all the numerical constraints will be called feasible states. States in which all the values are 1 or 0 will be called integer states, and states in which some values are non-integer will be called intermediate states.

3.3.3: Normalised linear combinations

This section is difficult and may be easier to understand if read in conjunction with section 3.5 which explains the same ideas using a spatial analogy.

Given some feasible integer states, a new state can be obtained by multiplying each state vector by a weight and adding the results. The resulting state vector is a linear combination of the original states. If the sum of the weights is 1, the result is a normalised linear combination. Figure 3.5 gives some examples. If the numerical constraints between supposition values are such that all the feasible states are normalised linear combinations of the feasible integer states, then the best integer state can be found by hill-climbing in the space of feasible states. An informal argument shows why this is so: If every feasible state is a normalised linear combination of feasible integer states, it can be expressed as a set of weights on these states. Also, the total preference, T_S, of a state, S, can be expressed in terms of the total preferences of the feasible integer states:

$$T_S = \sum_i w_i c_i$$

where w_i is the weight on an integer state and c_i is its total preference. Now, consider what happens to a feasible state if the weight on the best feasible integer state, B, is increased by δ and the weight on some other feasible integer state A, is decreased by δ. Provided

	P	q
V_1	1	1
V_2	0	1
V_3	0	0

FIGURE 3.5a: V1, V2, and V3 are the feasible combinations of truth values for p and q given the constraint $p \supset q$.

	P	q
$0.5\ V_1$	0.5	0.5
$+\ 0.7\ V_2$	0	0.7
$+\ 0\ V_3$	0	0
V_4	0.5	1.2

	P	q
$0.4\ V_1$	0.4	0.4
$0.6\ V_2$	0	0.6
$0.0\ V_3$	0	0
V_5	0.4	1.0

FIGURE 3.5b: V4 is a linear combination of V1, V2, V3. V5 is normalised linear combination because the weights on the vectors V1, V2, V3 add to 1.

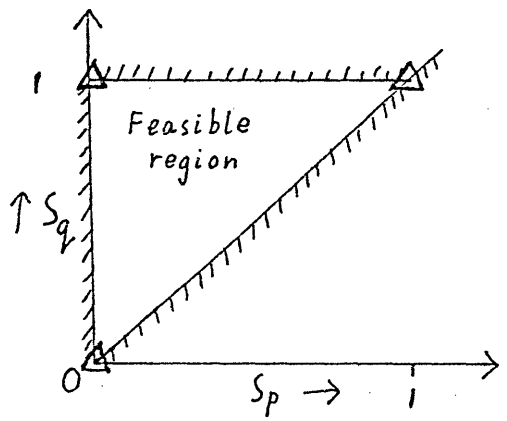


FIGURE 3.5c: The feasible region of supposition values for p and q given the constraint $Sq \geqslant Sp$ which is the numerical equivalent of $p \supset q$. Triangles denote the feasible integer states, and normalised linear combinations of these lie within the convex hull of the triangles.

no weights have become more than 1 or less than 0, the new state is also a normalized linear combination of feasible integer states and hence it is a feasible state. Its preference has increased by:

$$\delta \left(T_B - T_A \right)$$

where T_B, T_A are the total preferences of the states B, A. This is positive since B is better than A. So all feasible states except B can be improved by increasing the weight on B and decreasing some other weight. Notice that local maxima do not occur in this space, so the usual objection to hill-climbing, that it gets stuck at local maxima, does not apply. Figure 3.5 shows a simple example in which a logical constraint has been used to derive a numerical constraint on the supposition values. This constraint ensures that the only feasible states are normalized combinations of feasible integer states. The guiding principle used in deriving numerical constraints is to find the strongest inequality which is true of normalised linear combinations (i.e. the inequality which rules out the most states). By forcing the feasible states to satisfy these conditions one can usually force them to be normalised linear combinations. Cases where this approach fails, and ways of handling them, are discussed in Section 3.7.

3.4: Probabilities and supposition values

The constraints on the supposition values of hypotheses are like the constraints on the probabilities of events. The similarity of the calculus of supposition values to the calculus of probabilities suggests that supposition values may be interpretable as some kind of probability. It would be wrong to interpret them as the probability that the hypothesis is objectively correct, since a supposition value of 1 does not mean that the hypothesis is right, but only that it is part of the best consistent set. A more plausible candidate is, therefore, the probability that the hypothesis occurs in the best consistent set. This interpretation may be satisfactory when the values have all settled down to 1 or 0, but it is suspect as an interpretation of the changing values during the relaxation process, because they change without any change in the relevant knowledge or evidence. Even on the subjective interpretation of probabilities as degrees of belief, the belief should not change rapidly on the basis of no new evidence. It seems that suppositions and beliefs are different things, and this is confirmed by the fact that strong temporary suppositions need not imply strong temporary beliefs.

3.5: The hyperspace model.

Supposition values can be represented as distances along the axes of a multidimensional space. A set of

values is then a point in the space, and a numerical constraint corresponds to a hyperplane. To satisfy an equality or inequality constraint the point must lie on the hyperplane or on the appropriate side of it. The points representing the feasible states form a convex polyhedron because they lie in the intersection of some hyperplanes (equality constraints) and some half-spaces (inequality constraints). The total cost (or preference) of a state is defined as the scalar product of the cost vector with the supposition value vector. In spatial terms this means that the relative magnitudes of components of the cost vector define a direction in the hyperspace, and the optimal feasible state is the one furthest in that direction. In general, this will be a vertex of the polyhedron. The condition that the feasible states be the normalised linear combinations of the feasible integer states, is equivalent to the condition that the polyhedron defined by the constraint planes has only integer points as vertices, so that it is the convex hull of the feasible integer states.

3.6: Representing arbitrary logical constraints

The examples given so far have only shown the numerical constraints corresponding to simple logical expressions. If the method is to be applicable to sets of hypotheses related by arbitrary constraints in the propositional calculus, it is necessary to have an automatic

procedure for "cashing" any propositional form. The following four observations show how this is possible:

1. When a hypothesis is true its negation is false and vice versa. This suggest that the supposition values of a hypothesis and its negation should be related as follows:

$$s_{\bar{a}} = 1 - s_a$$

where $\bar{a}$ means the negation of a.

2. Any disjunction corresponds to the constraint that the sums of the supposition values must be at least 1:

$$a \vee \bar{b} \vee c \implies s_a + (1 - s_b) + s_c \geq 1$$

3. A conjunction of disjunctions can be cashed by simply cashing all the disjunctions separately:

$$(a \vee b) \wedge (c \vee d) \implies s_a + s_b \geq 1 \text{ and } s_c + s_d \geq 1$$

4. Any logical expression can be put into conjunctive normal form in which it becomes a conjunction of disjunctions:

$$(p \supset q) \equiv (q \vee r) \implies (p \vee \bar{q}) \wedge (q \vee r)$$

Although this approach allows one to derive a set of numerical constraints which rule out the same integer combinations of truth-values as any propositional form,

it may not lead to the strongest set of numerical constraints. For example, the constraints may not correspond to the convex hull of the feasible integer states.

3.7: Non-integer optima

Consider three hypotheses a,b,c which have equal, positive, unit preferences and are connected by the logical constraints: a/b, b/c, c/a where "/" means "not both". The corresponding numerical constraints are:

$$s_a + s_b \leq 1 \quad , \quad s_b + s_c \leq 1 \quad , \quad s_c + s_a \leq 1$$

The best feasible state, which has a total preference of $1\frac{1}{2}$ occurs when:

$$s_a = s_b = s_c = \frac{1}{2}$$

Clearly, this is a case where the obvious numerical constraints yield a larger polyhedron of feasible states than the convex hull of the feasible integer states. Figure 3.6a shows the polyhedron and its non-integer vertex $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. Such "bad" vertices are a serious threat to the use of continuous supposition values unless some way can be found to handle them. There are two possible times at which this can be done. Stronger numerical constraints than those obviously implied by the logical constraints can be sought when the constraints are made, and used to ensure that only the normalised linear combinations are feasible in the first place. Alternatively, the obvious

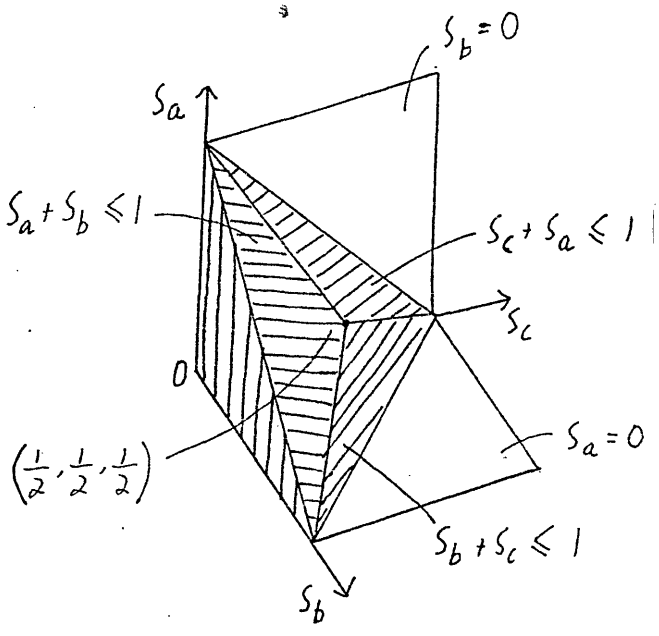


FIGURE 3.6a: The three constraint planes corresponding to a/b, b/c, c/a and the non-integer vertex where they intersect.

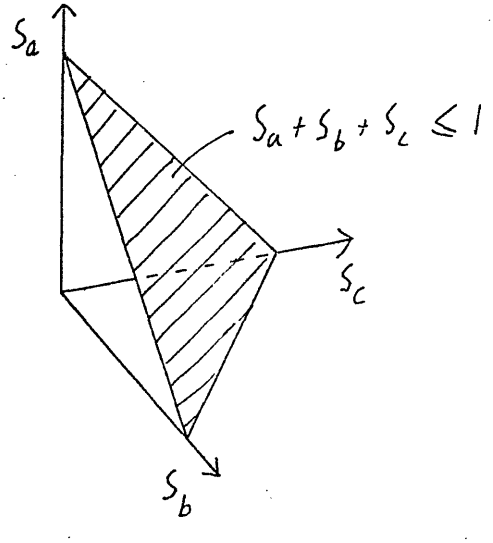


FIGURE 3.6b: A cutting plane corresponding to the constraint "at most one of a,b,c" which removes the non-integer vertex from the polyhedron of feasible states.

constraints can be used initially, and whenever the best vertex is non-integer, a stronger numerical constraint (called a cutting plane) can be constructed to eliminate it. This process of elimination can be continued until an integer vertex is best. The second method has the advantage that only those stronger constraints needed to rule out optimal bad vertices need to be found. All other discrepancies between the polyhedron of feasible states and the convex hull of the feasible integer states are irrelevant.

For the example above, the obvious stronger constraint is:

$$S_a + S_b + S_c \leq 1$$

Notice that this corresponds to the logical constraint that at most one of a,b,c be true. This can be derived logically from the three given logical constraints but it does not follow from the three corresponding numerical inequalities, because they lack the requirement that the values be 1 or 0. However, the integer requirement can be used in conjunction with the numerical constraints, to derive the stronger condition:

$$\begin{array}{r} S_a + S_b \leq 1 \\ S_b + S_c \leq 1 \\ \underline{S_c + S_a \leq 1} \\ 2S_a + 2S_b + 2S_c \leq 3 \end{array}$$

$$\therefore \quad S_a + S_b + S_c \leq 1\tfrac{1}{2} \qquad \cdots\cdots\cdots (1)$$

In any feasible integer state all values on the LHS of (1) must be integer. Therefore, no otherwise feasible integer states are ruled out by:

$$S_a + S_b + S_c \leq 1$$

There is a large literature on methods of deriving cutting planes to eliminate non-integer optimal vertices (see Garfinkel & Nemhauser 1972). In particular it was shown by Gomory (1958) that there are methods of constructing cutting planes which are guaranteed to eliminate all non-integer optimal vertices in a finite number of cuts.

An alternative to the use of cutting planes is to branch into two sub-problems whenever a bad optimal vertex is encountered, by fixing one of the intermediate supposition values at 0 in one case and at 1 in the other. The better of the optimal vertices of the sub-problems is then considered and if it also is non-integer, another intermediate supposition value is set at 1 or 0 to create two more sub-problems. Since the sub-problems must have worse optima than their parents, a branch-and-bound search is possible. Branching need only occur on the best of the remaining bad vertices and only until some integer vertex is better. This bound may prevent branching on many of the bad vertices. The combined use of branching and cutting planes is also possible (see Garfinkel and Nemhauser p.388).

The particular examples on which the final version of the puppet-finding program has been tried have never given rise to non-integer optima.. This may be due to the nature of the constraints in the puppet domain though this has not been proved. Since the problem has not arisen, no programs have been written for handling bad vertices, though it is recognised that such programs may be necessary for extending the use of continuous supposition values to other domains.

3.8: The numerical constraints in the puppet task

Section 2.6.2 lists four types of logical constraint that may occur between part and joint hypotheses. The corresponding numerical constraints between their supposition values are:

1. For part hypotheses corresponding to one rectangle:

$$\sum_{p} s_p \leqslant 1$$

This prevents a rectangle from having more than one interpretation as a puppet part.

2. For joint hypotheses of the same kind which compete for the same part p:

$$\sum_{j} s_j \leqslant 1$$

except for thigh/trunk or upper-arm/trunk joints which are competing for a shared trunk, for which:

$$\sum_j s_j \leq 2$$

since a trunk can have two thighs or upper-arms. These constraints prevent a part (e.g. a calf) from being used in several different joints of the same kind (e.g.knees).

3. For hypotheses about a type of part that occurs n times in a complete puppet:

$$\sum_p s_p \leq n$$

This prevents for example, two trunk hypotheses from both being accepted.

4. For each joint hypothesis j, relating part hypotheses p and q:

$$s_j \leq s_p \quad \text{and} \quad s_j \leq s_q$$

This prevents joint hypotheses being accepted unless both the related part hypotheses are accepted.

A stronger type of constraint based on a combination of (2) and (4) above is:

5. For joint hypotheses of the same kind competing for a part p: $\sum_j s_j \leq s_p$

$$\text{or} \quad \sum_j s_j \leq 2 s_p$$

for joints competing for thigh or arm slots in a trunk.

The numerical constraints used for the puppet program were designed to be as strong as possible in an attempt to remove non-integer optimal vertices. For all the examples tried they were successful in doing this. An earlier version of the program used constraints of types (2) and (4) separately, without combining them into type (5) constraints. As a result, the optimal vertices were occasionally non-integer.

3.9: The simplex algorithm

The use of continuous supposition values allows the problem of finding the best consistent set of hypotheses to be reduced to a linear programming task. There is a standard technique for solving such problems on a digital computer, based on the Simplex Algorithm. Pierre (1969) expounds the basic algorithm and variations of it which increase efficiency in particular cases. Only the basic strategy of the algorithm is explained here.

The problem is to find the vertex of a convex, multi-dimensional polyhedron which is best, i.e. furthest in the particular direction defined by the cost vector. The strategy is to find any vertex and then to compare its value (distance along the direction of decreasing cost) with the values of all the neighbouring vertices.

If none is better then the vertex is optimal, otherwise a better neighbour is chosen and the process repeated. Since each vertex is better than its predecessor, cycles cannot occur and since the number of vertices is finite, the process must terminate after a finite number of steps.

Neighbouring vertices are not too difficult to find. A vertex is defined by the intersection of a number of hyperplanes, corresponding to inequality constraints. In general a vertex in an n-dimensional space will be formed by the intersection of n hyperplanes, though in degenerate cases more planes may be involved. Neighbouring vertices are those which lie on n-1 of the original hyperplanes and on at least one new one. So by considering possible additions and deletions to the set of inequality constraints that are exactly satisfied, all neighbouring vertices can be generated.

Despite its guaranteed success, the simplex algorithm has serious deficiencies as a model of how the best consistent set of hypotheses might be found in a parallel computer. Although neighbouring vertices could be examined in parallel, the process of finding successively better vertices is inherently serial. For a polyhedron with many faces, the number of vertices traversed, and hence the number of serial steps, may be large. In fact, for the worst case, the number of vertices examined is an exponential function of the dimensionality of the space.

There is no polynomial upper bound. A further weakness is that the storage required may be large.

3.10: Assigning preferences to hypotheses

All part hypotheses have an initial preference of 0, and all joint hypotheses are given a standard initial preference of 1, in order to implement the basic aim of finding an interpretation with as many instantiated joints as possible. Additional input instructions such as:

TRYTOINTERPRET [B AS CALF IMPORTANCE = 0.5];

are implemented in a very simple way. For each such instruction, the whole list of part hypotheses is searched and any that fit the instruction have their preferences incremented by the specified amount.

3.11: The abstract optimization problem and the type of solution required.

The puppet-finding task has now been reduced to the following abstract problem: given some hypotheses, and logical constraints between them, and the importance of including each hypothesis in the final interpretation, how can the best consistent set of hypotheses be selected?

There are many ways of tackling this problem and

some of them have already been discussed in Chapter 1. This thesis is primarily concerned with examining one particular method in which each hypothesis is given an associated real number, and the numbers are iteratively modified to make the best consistent set of hypotheses stand out. There is a danger, when trying to develop a technique of this kind, of evolving a set of unprincipled number-juggling tricks which can be tuned to work moderately well in a restricted domain, but which are not clearly understood and can therefore only be extended to other domains by empirical parameter tuning. Further disadvantages of unprincipled tricks are that it is hard to characterise the set of domains for which the method works, or to express the nature of the computation being performed in any more illuminating way than by giving the particular implementation details. Marr & Poggio (1976) discuss the importance of separating the nature of the computation from particular implementations. Although an implementation constitutes an effective procedure and therefore has advantages over a purely verbal theory, simply describing an implementation may confuse arbitrary implementation decisions with important principles.

The following sections are intended to provide a sound theoretical basis for the way in which supposition values are adjusted by the relaxation operator, though the precise details of the operator are not fully determined by the theory.

3.12: Two types of relaxation

There are various relaxation operators which make iterative adjustments to the supposition values so as to converge on the best feasible state or on a state close to it. Methods in which the values are modified one at a time, and the updated state is used in deciding how to modify the next value, will be called serial relaxation. By contrast, parallel relaxation involves using the current supposition values to compute new values for all the hypotheses, and then changing all the values together. It is more suitable for a parallel digital computer, and is closer to the behaviour of an analogue system. Both types of operator were tried for the puppet-finding task. The parallel one was easier to analyse and needed less iterations than a serial one working on a round robin basis, though clever scheduling might well improve the serial operator significantly. Only the parallel operator was used for the final version of the program, and it is described below.

3.13: Two components of the relaxation operator

The relaxation operator consists of two components. One is defined to ensure that the supposition values are feasible or nearly feasible, and the other adjusts them to achieve optimality. There is a mechanical analogy, based on the hyperspace model. One component exerts strong forces on states which are outside the feasible

polyhedron and moves them towards it, whilst the other component is equivalent to a constant weak force in the direction defined by the preference vector. First, the component for achieving feasibility will be described and then ways of combining it with the optimality component will be discussed.

3.14: Achieving feasibility

The following discussion assumes that all constraints are in the form of inequalities. Equality constraints can always be removed by using them to eliminate a variable, or by simply representing them as two inequality constraints:

e.g. $a+b=n \Rightarrow a+b \geq n$ and $a+b \leq n$

One measure of how much a state of the supposition values violates a particular constraint is: the normal distance from the corresponding point to the corresponding hyperplane if the point is on the infeasible side of the plane, otherwise 0. Using this measure of violation, the infeasibility of a state can be defined as:

$$I = \sum_j \tfrac{1}{2} V_j^2$$

where j ranges over all the constraint planes, and V_j is the amount by which the state violates the j'th constraint.

Clearly, I is zero within the feasible region and

positive outside it. More significantly, the rate at which I changes as the violation of a constraint j changes, is given by:

$$\frac{\partial I}{\partial v_j} = V_j$$

I can be thought of as a potential energy function over the hyperspace, and $\partial I / \partial v_j$ is then the force exerted at a point by the j'th constraint plane. The equation above shows that the force is proportional to the normal distance of the point from the plane. This mechanical analogy allows physical intuitions to be brought to bear on the design of a relaxation operator for minimizing I.

One parallel relaxation operator for reducing the infeasibility of a state inolves choosing each new supposition value so as to minimize the infeasibility, assuming the old values for all the other suppositions. In mechanical terms this amounts to choosing the new supposition value so that the forces due to relevant violated constraints, assuming that the remaining suppositions have their old values, are in equilibrium. Unfortunately, this operator does not necessarily reduce the infeasibility. For states in which one supposition has the new value and the rest have old ones, the infeasibility is the same or less, but for the state with all the new values it may be considerably higher, as Figure 3.7 shows. The reason is that several different supposition values may be altered so as to reduce the violation of a particular constraint, and although the alterations

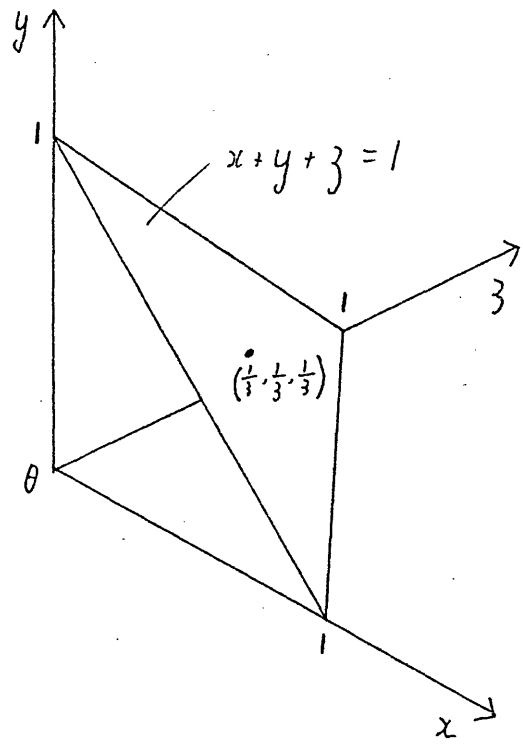


FIGURE 3.7: Suppose there are two constraints: $x + y + z \leq 1$ and $x + y + z \geq 1$ and the initial state is (0, 0, 0). Relaxation on any one dimension would produce one of the feasible states where the plane cuts an axis. Combining independent relaxation on three dimensions, however, yields the state (1, 1, 1). By symmetry, $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ is the foot of the perpendicular to the plane from both (0, 0, 0) and (1, 1, 1), so the final state has twice the violation of the initial one.

separately reduce infeasibility, together they may overshoot and cause other violations which outweigh the reduction in the original one.

An alternative method is to find the direction, at the current point in the hyperspace, in which the infeasibility decreases fastest and to move a small distance in this direction. This is equivalent to changing the individual supposition values in proportion to their partial derivatives. In terms of the mechanical analogy, the forces due to the violated constraints can be resolved into components parallel to the axes. The resultant of the force lies in the direction of steepest descent and so therefore does a vector whose components are proportional to the forces along the axes. The magnitude in the change of each supposition value is determined by a constant K_f :

$$S_i^{t+1} = S_i^t - K_f \left(\frac{\partial I}{\partial S_i}\right)^t$$

where S_i^t is the value of S_i at time t, and $\left(\frac{\partial I}{\partial S_i}\right)^t$ is the value of $\frac{\partial I}{\partial S_i}$ at time t.

In the simplest possible case, when only one constraint is being violated, and no other violations are caused by moving directly towards the constraint plane, the obvious value for K_f is 1. This has the effect of exactly satisfying the constraint in one iteration (see Figure 3.8). However, if several violated constraints are involved, or if new violations are caused by the change,

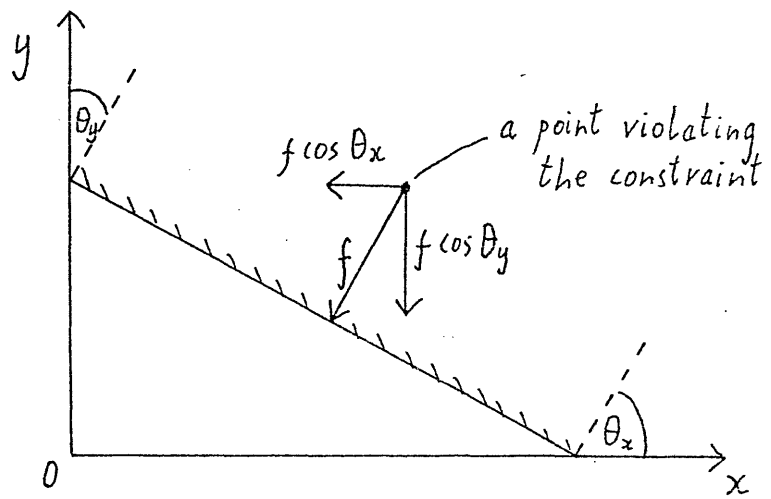


FIGURE 3.8: The force due to a single violated constraint plane, and its components in the x and y directions. Altering each supposition value by an amount equal to its component of the force would exactly satisfy the constraint.

a smaller value of K_f may be needed to minimize the reduction in the infeasibility. It is hard to compute the optimal value of K_f, partly because of the interactions between alterations of different supposition values, and partly because changes may activate previously satisfied constraints. However, the following theorem shows that for any particular set of constraints there is some finite value for K_f which ensures that the infeasibility is always decreased by a significant proportion.

Theorem

For any finite set of constraints which allows some feasible states there is a finite value for K_f such that moving a distance $K_f \frac{dI}{dl}$ from any infeasible point P in the direction of steepest descent at P, decreases I by at least $\frac{1}{2} K_f \left(\frac{dI}{dl}\right)^2$ where l is the distance along a line in the direction of steepest ascent at p.

Proof:

The proof depends on showing that there is a limit to the rate at which dI/dl can decrease, so that a sufficiently small step cannot move the state past the point at which dI/dl changes sign and the infeasibility starts increasing again. By definition: $I = \sum_j \frac{1}{2} V_j^2$

$$\therefore \frac{dI}{dl} = \sum_j V_j \frac{dV_j}{dl} = \sum_j V_j \cos\theta_j$$

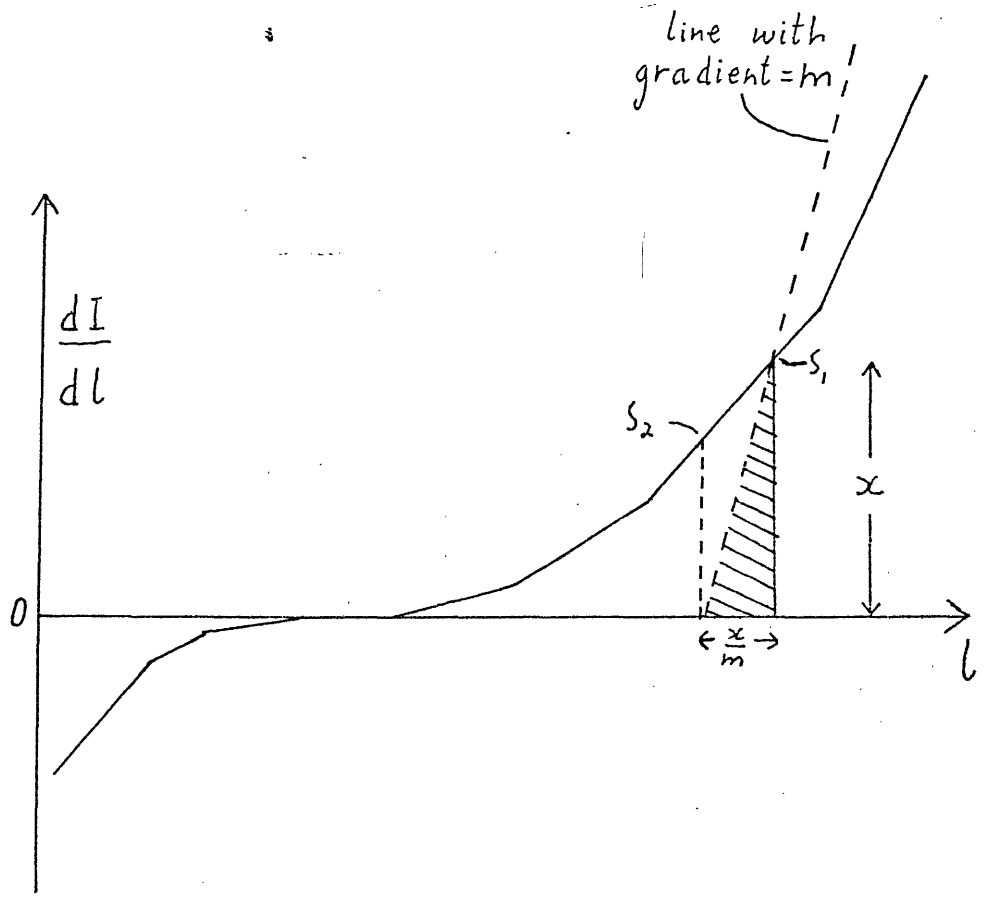


FIGURE 3.9: Showing how dI/dl changes with l . The slope changes by a discrete amount whenever a constraint plane is crossed. The effect of a move of $-\frac{1}{m}\, dI/dl$ from a state S_1 is shown. Even if dI/dl had its maximum gradient of m (indicated by the dotted line through S_1) the move could not reverse the sign of dI/dl . The reduction in the infeasibility is the area under the curve between S_1 and S_2. This is at least the area of the shaded triangle.

where θ_j is the angle between direction of steepest descent and the normal to the j'th constraint plane. Hence:

$$\frac{d^2I}{dl^2} = \sum_j \cos^2\theta_j$$

For each constraint plane, $\cos^2\theta_j$ is at most one, so a weak upper bound on d^2I/dl^2 is therefore m, the number of constraint planes. This corresponds to the case where the constraints are all violated and all the corresponding planes are normal to the direction of steepest ascent. Generally, the maximim value of d^2I/dl^2 will be much smaller than m.

Now suppose $K_f = \frac{1}{m}$ so that the size of the move in the direction of steepest descent is $\frac{1}{m}\frac{dI}{dl}$. Figure 3.9 shows that $\frac{dI}{dl}$ cannot reverse its sign as a result of such a move. Also, the decrease in I is at least the area of the shaded triangle, which is $\frac{1}{2m}\left(\frac{dI}{dl}\right)^2$.

3.15: The speed of convergence on a feasible state.

Figure 3.10 shows that in some cases the feasible region may never be reached. However, if the infeasibility is reduced by at least some constant proportion on each iteration, it will decay exponentially and can be reduced to any finite level in a finite number of iterations. The theorem above shows that there is a value for K_f which ensures that the infeasibility decreases by at least $\frac{1}{2m}\left(\frac{dI}{dl}\right)^2$ on each iteration. So provided $\left(\frac{dI}{dl}\right)^2 \geq cI$

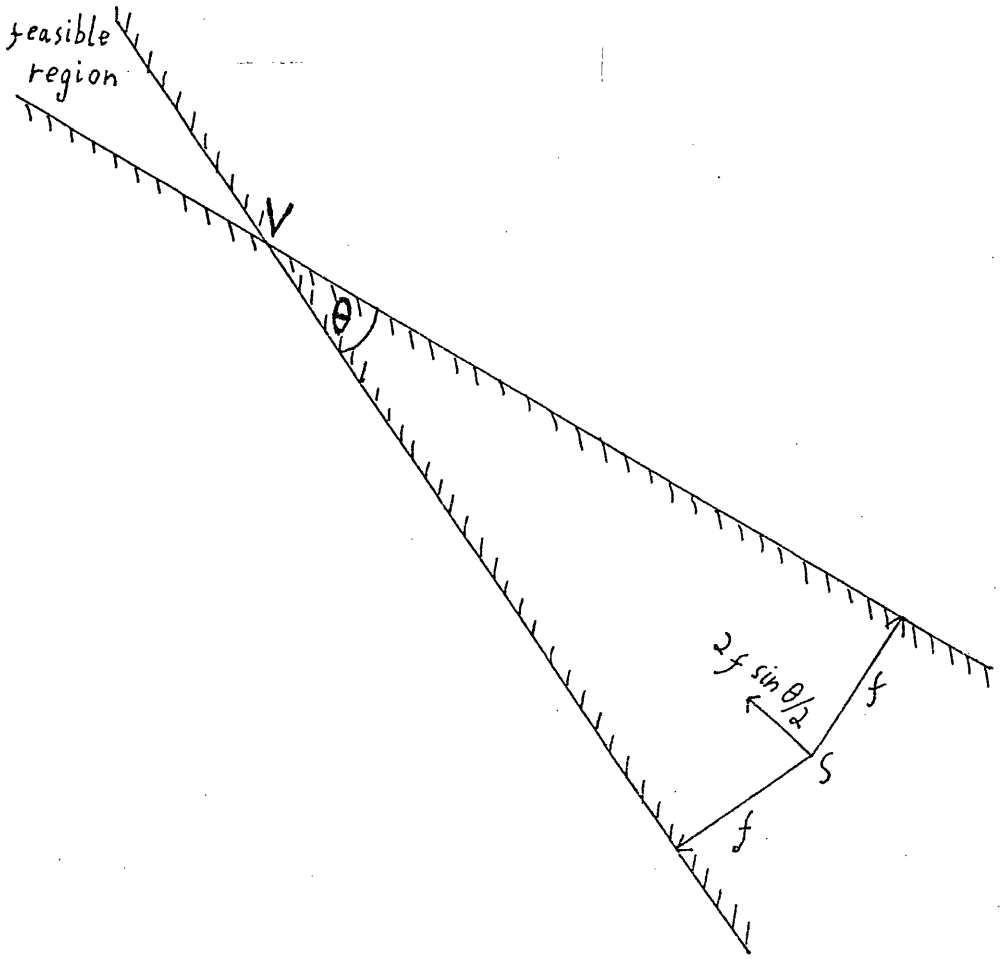


FIGURE 3.10: If an infeasible state, S, violates two constraint planes equally, the resultant force will be towards, V, the intersection of the planes. Unless K_f is large enough to make the state feasible in one move, the same situation, but on a smaller scale, will occur after each move, and the infeasibility will never reach zero. The expression for the resultant force shows that by making θ, the angle between the planes, sufficiently small, the resultant, for any given violation, can be reduced indefinitely.

where c is some finite constant, the infeasibilty will decrease by at least $c/2m$ on each iteration and so there will be a lower bound on the rate of exponential decay of the infeasibilty. Figure 3.10 shows that the constant c may be made indefinitely small by choosing opposed constraint planes which are sufficiently close to parallel. For any given set of constraints, however, there will be a most closely opposed pair of planes and these will presumably set a lower bound on c, though I have been unable to discover an expression for this bound in terms of the constraints. (Parallel opposed planes are irrevelevant since if there are any feasible states there can be no infeasible ones which violate both planes). Assuming there is a lower bound on c it can be combined with the conservative value of $1/m$ for K_f to give a very conservative lower bound to the speed of convergence for any given set of constraints. I cannot see how to establish a realistic estimate of the speed other than by empirical observation. Similarly, a suitable value for K_f rather than a conservative lower bound, can be found by observing the behaviour of the system for any particular problem. Small values cause slow convergence but large ones cause oscillations which may be divergent. In the puppet program a suitable value was found empirically and the same value was used in all the examples, though it would have been possible to optimize K_f at run time by monitoring the changes in infeasibility and altering K_f appropriately.

3.16: Achieving optimality

Using the mechanical analogy, suppose that in addition to the forces caused by constraint violations, there is a constant weak force in the direction of the preference vector. A simple example of the behaviour which results is shown in figure 3.11a. Notice that the system converges on a point which is near the best vertex and just outside the feasible region. Adding a force in the direction of the preference vector is equivalent to adding to each supposition a force proportional to the preference of the corresponding hypothesis, where the constant of proportionality K_p is 1 if a unit preference has the same affect as a unit violation of a constraint plane normal to the axis defined by the supposition.

Increasing the value of K_p increases the speed of convergence but it also makes the equilibrium point further from the best vertex. Figure 3.12 shows the effect of different values of K_p on a particular puppet problem. A good practical strategy used for the examples in Chapter 2 is to start with a large value for K_p which gives rapid convergence on roughly the right region, and then to lower K_p to obtain slower convergence on a point closer to the best vertex. For the puppet task, the values to be used for K_p were determined in advance (see section 3.19), rather than being dynamically controlled at run time.

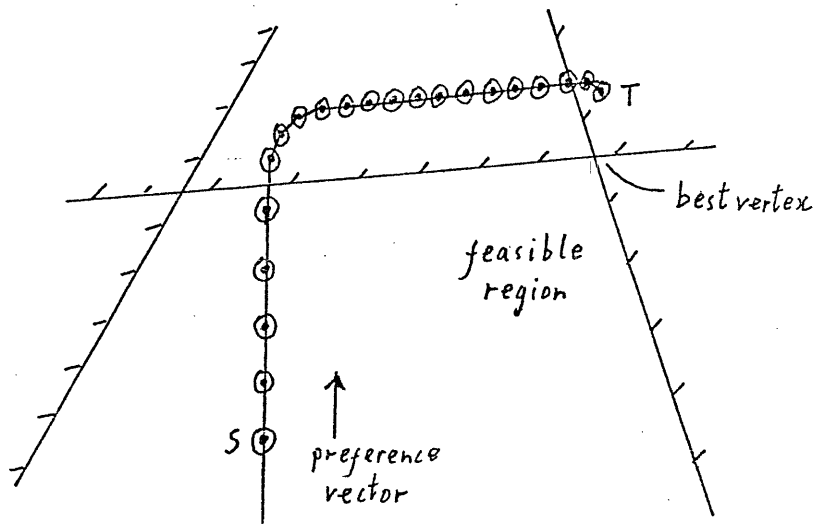


FIGURE 3.11a: Showing how the state moves from S to the equilibrium state, T, under the combined influence of the preferences and the violated constraints.

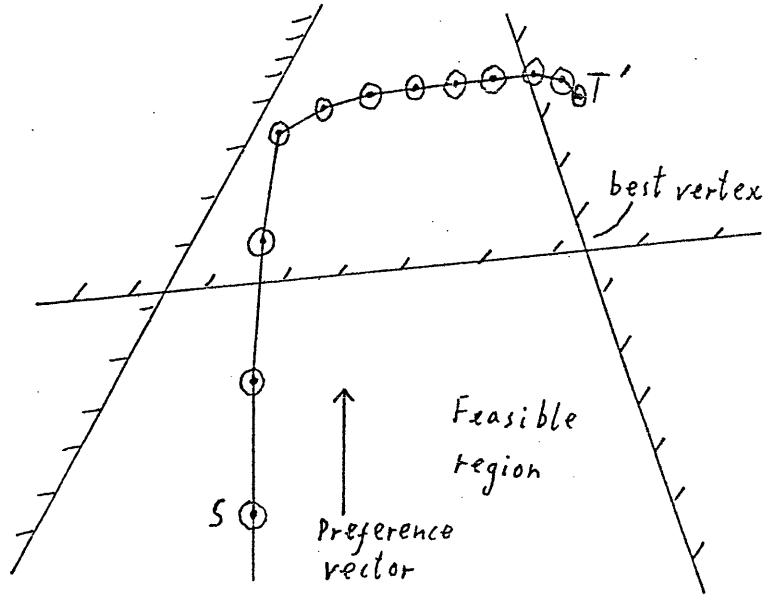


FIGURE 3.11b: Showing how the speed of convergence is increased by magnifying the forces due to the preference vector. Notice that the equilibrium state is further from the best vertex.

```
!showconversence(0,4);
                                                C3 C3 B1 D1 D2 A1
A1 B1 B2 C1 C2 C3 D1 D2 D3 D3 B2 C3 C2 C1 B1
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
43 52 47 34 34 51 35 35 46 60 60 61 46 46 56
87 79 64 33 33 77 35 35 66 86 86 90 49 49 94
99 87 64 26 26 89 27 27 82 99 91 99 41 41 99
99 86 66 23 23 90 24 24 85 99 93 99 38 38 99
99 86 66 22 22 91 23 23 86 99 93 99 37 37 99
99 86 67 22 22 91 23 23 86 99 93 99 37 37 99
99 86 67 22 22 91 23 23 86 99 93 99 37 37 99
99 86 67 22 22 91 23 23 86 99 94 99 37 37 99
99 86 67 22 22 91 23 23 86 99 94 99 37 37 99
99 86 67 22 22 91 23 23 86 99 94 99 37 37 99

99 86 67 22 22 91 23 23 86 99 94 99 37 37 99
```

FIGURE 3.12a: Each row of numbers shows the supposition values (X 100) for the part and joint hypotheses for the picture in figure 3.1a. The values are printed on every tenth iteration, except for the final row which is the equilibrium state found by continuing for 250 iterations. The headings indicate the identity of the hypotheses. Joints have a double heading giving the names of the two related parts. For formatting reasons, only the integer parts of the numbers are shown and 100 is printed as 99.

The values of the coefficients were: $K_p = 0.4$ $K_f = 0.3$. The remaining coefficients (see below) were both zero.

```
!showconversence(0.2);
                                                C3 C3 B1 D1 D2 A1
A1 B1 B2 C1 C2 C3 D1 D2 D3 D3 B2 C3 C2 C1 B1
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
21 26 23 18 18 27 18 18 23 30 30 30 23 23 28
48 52 50 33 33 50 34 34 46 54 56 56 40 40 54
68 64 56 30 30 62 31 31 56 66 67 69 38 38 72
80 72 56 26 26 69 27 27 64 74 69 77 34 34 82
87 77 53 22 22 75 23 23 72 81 68 83 30 30 88
93 82 49 19 19 82 20 20 79 88 64 88 27 27 94
98 88 44 16 16 88 16 16 85 94 60 94 24 24 99
99 91 40 13 13 93 13 13 91 99 55 99 21 21 99
99 92 37 11 11 94 12 12 92 99 52 99 19 19 99
99 92 36 11 11 95 11 11 93 99 50 99 18 18 99


99 92 35 11 11 95 11 11 93 99 49 99 18 18 99
```

FIGURE 3.12b: As in figure 3.12a, but with the value of K_p halved. Notice that the number of iterations required to approximately reach the final state is doubled, but that the equilibrium state is half as far from the optimal integer vertex.

```
!showconversence(0.1);
                                                C3 C3 B1 D1 D2 A1
A1 B1 B2 C1 C2 C3 D1 D2 D3 D3 B2 C3 C2 C1 B1
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
10 13 11  9  9 13  9  9 11 15 15 15 11 11 14
24 26 25 19 19 26 19 19 25 28 28 28 21 21 27
37 39 38 29 29 40 29 29 38 41 41 41 31 31 40
50 51 50 31 31 49 31 31 45 50 53 52 35 35 53
59 57 53 28 28 55 29 29 51 56 58 59 32 32 61
65 60 52 26 26 59 26 26 56 61 59 63 30 30 66
68 63 51 24 24 62 24 24 60 65 59 66 28 28 69
71 66 50 22 22 65 22 22 63 68 57 68 26 26 71
73 68 47 20 20 68 21 21 67 71 55 71 24 24 74
76 71 45 19 19 71 19 19 70 74 52 74 23 23 77


99 96 17  5  5 97  5  5 96 99 24 99  9  9 99
```

FIGURE 3.12c: Halving K_p to 0.1 again halves the distance of the equilibrium state from the optimal vertex, but doubles the time to reach equilibrium.

3.17: A method of increasing the convergence speed

When K_p is small and the best vertex is only slightly better than some other one, the supposition values tend to change very slowly. Figure 3.11a shows an abstract example in two dimensions and Figure 3.12c shows a real puppet example. The reason for the slowness is that the preference vector is almost normal to the direction in which the state needs to move if its to improve without increasing its infeasibility. Under such circumstances the state moves in small steps in a roughly constant direction. If the steps are made to depend not only on the currently active forces, but also on the previous step, it is possible to make them increase steadily in size when the supposition values are moving in a constant direction. So the formula used to determine the next move, $\underline{M}_{t+1}$, is:

$$\underline{M}_{t+1} = K_f \left(K_p \underline{p} + \underline{V} \right) + K_d \cdot \underline{M}_t$$

where $\underline{p}$ is the preference vector, $\underline{V}$ is the resultant of all the violations, and $\underline{M}_t$ is the previous move.

The effect of the term containing $\underline{M}_t$ is to give the system a simple kind of memory so that each move depends on the history of previous moves. The forces exerted on the state at time t contribute to each subsequent move at time t+n , but by an exponentially decaying factor of K_d^n (assuming $K_d < 1$). When K_f is small and $K_d = 1$ the system behaves as if the state has inertia, so that once

it has been made to move it can only be stopped by opposite forces. This leads to oscillations which may be divergent for $k_d > 1$ but which are damped for $k_d < 1$. A value of 0.8 was found to significantly reduce the number of iterations required in examples of the puppet problem without causing other problems. Indeed, the introduction of k_d may actually reduce oscillations caused by a high value of k_p as figure 3.13 shows.

3.18: The method of selecting the final set of hypotheses

When the system is nearing its equilibrium point, the supposition values will generally be near 1 or 0 if the optimal vertex is integer and k_p is small enough for the equilibrium point to be near it. One might use a simple threshold of 0.8, say, and choose the hypotheses with a higher supposition value as the best set. However, there is no guarantee that the set will be consistent, since one of the high values may only be allowed if several of the low values are not zero. For example the constraint $A \leq B+C+D+E$ is satisfied by A=0.8 and B=C=D=E=0.2. An alternative to thresholding is to introduce small extra forces which pull high values towards 1 and low values towards 0. If the equilibrium point is near an integer vertex, then small extra forces will cause relaxation to actually achieve an integer state and, provided the extra forces are too weak to cause a significant constraint violation, the final state will be

									C3 D3	C3 B2	B1 C3	D1 C2	D2 C1	A1 B1
A1	B1	B2	C1	C2	C3	D1	D2	D3						
99	96	17	6	6	97	6	6	96	99	24	99	8	8	99
99	96	17	5	5	97	5	5	96	99	24	99	9	9	99
99	96	17	5	5	97	6	6	96	99	24	99	9	9	99
99	96	17	5	5	97	5	5	96	99	24	99	9	9	99
99	96	17	5	5	97	5	5	96	99	24	99	9	9	99
99	96	17	5	5	97	6	6	96	99	24	99	9	9	99
99	96	17	5	5	97	5	5	96	99	24	99	9	9	99
99	96	17	5	5	97	6	6	96	99	24	99	9	9	99
99	96	17	5	5	97	5	5	96	99	24	99	9	9	99
99	96	17	5	5	97	5	5	96	99	24	99	9	9	99
99	96	17	5	5	97	5	5	96	99	24	99	9	9	99

FIGURE 3.13a: A stable state in which the large value of 0.7 for K_f does not cause problems because K_d also is large (0.8) and therefore smoothes out rapid oscillations. K_p = 0.1 as in figure 3.13c.

									C3 D3	C3 B2	B1 C3	D1 C2	D2 C1	A1 B1
A1	B1	B2	C1	C2	C3	D1	D2	D3						
99	96	17	5	5	97	5	5	96	99	24	99	9	9	99
99	96	17	5	5	97	5	5	96	99	24	99	9	9	99
99	96	17	5	5	97	5	5	96	99	24	99	9	9	99
99	96	17	6	6	97	6	6	96	99	24	99	8	8	99
99	96	17	4	4	97	4	4	96	99	24	99	10	10	99
99	97	18	7	7	98	8	8	97	99	24	99	6	6	99
99	95	17	2	2	94	2	2	94	99	25	99	13	13	99
99	98	18	12	12	99	12	12	99	98	24	97	0	0	99
99	91	15	2	2	89	2	2	89	99	25	99	6	6	99
99	99	22	6	6	99	6	6	99	91	22	88	5	5	98
99	88	11	1	1	94	1	1	94	98	29	95	12	12	99
99	99	28	11	11	97	11	11	98	99	18	95	0	0	96
99	85	14	3	3	89	2	2	90	99	25	99	7	7	99
99	99	25	6	6	99	6	6	99	92	21	82	6	6	92
99	87	12	1	1	94	1	1	94	99	28	89	13	13	99
99	99	28	13	13	96	13	13	99	98	20	94	0	0	94

FIGURE 3.13b: When K_d is reduced to 0 oscillations start because of the large K_f.

consistent. The last minute flips from low to high (or vice versa) which cause problems for thresholding are precipitated by forcing the other values to 1 or 0. The magnitude of the extra force on a supposition value, s_i is determined by the coefficient k_h in the expression:

$$Extra\ force = \left| k_h (s_i - 0.5) \right|$$

3.19: The final form of the relaxation operator

When all the above modifications are incorporated, the expression used to compute a new supposition value s_i^{t+1} is:

$$s_i^{t+1} = s_i^t + K_f \left(K_p \cdot p_i + \left(\sum_j v_j \cos\theta_{ji} \right) + K_h \left(s_i^t - 0.5 \right) \right) + K_d \left(s_i^t - s_i^{t-1} \right)$$

where s_i^t is the i'th supposition value at time t, p_i is the preference for the i'th hypothesis, v_j is the violation of the j'th constraint, and $\cos\theta_{ji}$ is the angle between the normal to the j'th constraint plane and the axis defined by the i'th supposition value (the angle is 90 degrees for the constraints not involving the supposition value).

If the new value for s_i is outside the range 0 to 1 it is rounded up or down accordingly.

For the examples in Chapter2, 50 iterations were used with the values of the coefficients set as shown:

K_p	K_d	K_f	K_h	Iterations
0.4	0.5	0.3	0	10
0.2	0.5	0.3	0	10
0.1	0.8	0.3	0	10
0.1	0.8	0.3	0.1	20

Appendix 4 shows, for the examples in Chapter 2 how the supposition values of the hypotheses changed during relaxation.

Chapter 4

THEORETICAL ANALYSES OF RELAXATION, AND SOME POSSIBLE EXTENSIONS.

This chapter starts by analysing what is happening during relaxation. Comparisons are made with search methods in which partial solutions are formulated explicitly, and there is a discussion of how the time taken by relaxation depends on the number of hypotheses. However, the "technical" problems of achieving rapid convergence on a state sufficiently close to the optimum, and of removing non-integer optima have not been fully investigated. The fact that the puppet program works is a start, but more theoretical analysis is required. This may prove fruitful because the linear programming formulation not only makes the relaxation operator easy to understand but also facilitates analysis of the effects of modifying the basic operator.

Later sections discuss ways in which relaxation needs to be extended to be applicable to more complex problems. A recurring theme is the need to integrate the process of creating hypotheses with the process of selecting between them. A major weakness of simple L.P. relaxation is its separation of these two processes into distinct phases so that the selection performed by relax-

ation is unable to guide hypothesis creation. The development of an integrated system for an extended version of the puppet task is discussed towards the end of the chapter.

4.1: The avoidance of Explicit Enumeration

The number of feasible combinations of hypotheses is, generally, an exponential function of the number of hypotheses, so that, for large problems, exhaustive explicit enumeration is out of the question. The use of continuous supposition values allows intermediate states, which can be thought of as normalised linear combinations of many different integer states, and when an intermediate state is modified the system is typically moving towards a very large number of integer states and away from many others. Thus, particular combinations of hypotheses are dealt with implicitly, which gives a potentially exponential saving in space or time.

4.2: Decomposition into Interacting Sub-Systems

Perhaps the most attractive feature of L.P. relaxation is the way in which it is naturally suited to parallel hardware. Each supposition value and each constraint can be given its own processor thus achieving a linear but large saving in speed over a serial system. Of course, there are still problems about how to set up the configuration of processors and the interconnections

needed for a specific task, but the way in which the processes should interact once they have been set up is clearly specified by L.P. relaxation. The space required is only a linear function of the number of hypotheses and constraints because explicit enumeration of combinations of hypotheses is avoided. By contrast implementing a breadth-first search on parallel hardware, is simply a way of trading a combinatorial explosion in time for one in space.

It is interesting to try to analyse the whole system in terms of sets of hypotheses which have dense internal connections but which are relatively sparsely connected with one another. In an extreme case, for example, there might be two independent sets, and given parallel hardware, the time to reach equilibrium would then be the longer of the times for each set separately. Notice that for a serial depth-first or breadth-first search the combined time is the product of the separate times. Of course a serial search could be modified so that it first checked whether there were two independent sets, and if so performed two separate searches. If, however, the sets are largely but not completely independent, there is no simple way of using the near-independence in a conventional search. An interesting simple case is when two subsystems are linked by constraints that allow a combined optimum which is simply the combination of the optima for the separate subsystems. If the linking constraints rule out combinations of independently feasible,

near-optimal states of the subsystems, then the whole system may converge faster then either subsystem alone. Figure 4.1 shows an example of this effect.

There is a way of viewing the interactions between subsystems which helps to clarify the relationship between L.P. relaxation and a technique known as dynamic programming (see Pierre 1969 for an exposition). In L.P. relaxation, each subsystem can be seen as optimising its own internal state, subject to the boundary conditions imposed by those other supposition values which are linked to the subsystem by constraints. A subsystem exerts pressure on its current boundary conditions tending to change them so as to allow a higher optimum for the subsystem. In dynamic programming, a table or function is created for a subsytem, which gives its optimum internal state for each possible combination of boundary conditions. This is the only information about the subsystem which is of relevance to the determination of the global optimum. Dynamic programming works by expanding the subsystem (incorporating new hypotheses), and simultaneously modifying the associated table or function. When the subsystem has engulfed all the hypotheses, there will only be the null boundary condition, and its associated optimal state will be the solution. Dynamic programming is particularly effective if subsystems have simple boundary conditions, for then the tables or functions are simple. In a puppet task, a subsystem containing about half the hypotheses will, typically, be linked by con-

```
!relax(50,5)$
```

A	B	C
0	0	0
50	22	0
89	34	1
99	25	7
99	23	6
99	22	5
99	20	4
99	19	3
99	18	1
99	17	0
99	15	0

```
!relax(50,5)$
```

D	E	F
0	0	0
24	22	0
49	45	0
61	54	0
63	53	0
64	51	0
65	50	0
66	49	0
68	48	0
69	46	0
70	45	0

FIGURE 4.1a: Showing the speed of convergence for two independent sets of hypotheses $\{A,B,C\}$ and $\{D,E,F\}$. Fifty iterations are shown with printing every fifth iteration. The constraints are $A \wedge B \wedge C$ and $D \wedge E \wedge F$. In numerical form these are $S_A + S_B - 1 \leqslant S_C$ and $S_D + S_E - 1 \leqslant S_F$. The preferences for the set $\{A,B,C\}$ are (2, 0.9, -1) so the best feasible state is (1, 0, 0) with (1, 1, 1) a close second. For $\{D,E,F\}$ the preferences are (1, 0.9, -2) giving an optimum of (1, 0, 0) with (0, 1, 0) a close second.

```
!relax(50,5)$
```

A	B	C	D	E	F
0	0	0	0	0	0
41	22	0	33	22	0
75	41	0	70	43	0
86	37	2	80	38	0
94	30	3	88	31	0
99	23	3	95	24	0
99	19	1	98	19	0
99	16	0	99	16	0
99	15	0	99	15	0
99	15	0	99	15	0
99	15	0	99	15	0

FIGURE 4.1b: Showing the faster convergence when there are linking constraints: $S_A = S_D$, $S_B = S_E$, $S_C = S_F$. The best feasible state is then much better than its nearest rival.

straints to many others. The boundary conditions are all the feasible combinations of truth values of these other hypotheses, which may be a large number. Relaxation avoids this explosion by avoiding explicit enumeration of the possible boundary conditions of a subsystem.

4.3: The Time Taken to Reach Equilibrium

An important factor in determining whether L.P. relaxation is a good search method is the number of iterations required to reach the equilibrium state. The puppet examples have few enough hypotheses for serial search techniques to be relatively quick, but as the number of hypotheses increases, the time required for these methods increases exponentially. By contrast, it will be shown that the time required for relaxation, using parallel hardware, is independent of the number of hypotheses, given certain reasonable assumptions.

The puppet examples (appendix 4) show that much of the time required to reach equilibrium is spent in creeping towards the optimum state and away from a very different integer state with a slightly lower score. The reason progress is so slow is that the state is moving parallel to an edge which is almost normal to the preference vector, so that the componenent of the preference vector in the direction of motion is very small (see figure 3.11a).

As a first step to analysing how the time depends on

the number of hypotheses, it will be shown that the time is related to the rate of travel along the ridge which is most nearly normal to the preference vector. Let us call the direction of the preference vector "vertical". The optimal feasible state then corresponds to the highest vertex (the peak). The relaxation process can be divided into two stages. First, the state is made roughly feasible, and then it moves to a point near the peak, either by going through the interior of the polyhedron or by staying just outside it and moving roughly parallel to its surface. The first stage, achieving near-feasibility, may not be necessary, and even if it is, it is generally relatively quick compared with the second stage. So only the time for the second stage will be considered. The problem, therefore, is to find the time taken to travel in the local direction of steepest ascent from an arbitrary point within or nearly within the polyhedron, to a point near the peak, given that the rate of travel depends on the cosine of the angle with the vertical. The problem is made more tractable if the starting point is approximated by the nearest point, S, which is actually on or within the polyhedron, and the equilibrium point is approximated by B, the peak. If there are n hypotheses, the distance between S and B cannot exceed $\sqrt{n}$ since the feasible polyhedron lies within a unit hypercube whose longest diagonal has length $\sqrt{n}$. So, if the shallowest ridge (the most nearly horizontal one) connects S to B the time taken is at most $\sqrt{n}/r$, where r is the rate of travel along the shallowest ridge. If S and B are not

connected by this ridge, then the point representing the current state will travel at an angle closer to the vertical and will therefore travel faster, but it may also have to travel much further, since it may follow a zig-zag path. It can be shown, however, that the time taken cannot exceed $\frac{3}{2}\frac{\sqrt{n}}{r}$.

Theorem

Let a particular direction in an n-dimensional space be called "vertical". Let B be the "highest" point on a convex polyhedron enclosed within a unit hypercube, and let S be any point on or within the polyhedron. Following the path of locally steepest ascent, the time taken to travel from S to B is not more than $\frac{3}{2}\frac{\sqrt{n}}{K\cos\phi}$ where ϕ is the angle between the shallowest ridge of the polyhedron and the vertical, and the rate of travel in a direction which makes an angle of θ with the vertical is $K\cos\theta$.

Proof

Rather than considering the distance travelled and the rate of travel, it is easier to consider the difference in the heights, h_s , h_B, of S and B and the rate at which this difference is reduced. For a direction making an angle θ with the vertical, the rate of travel is $K\cos\theta$, so the rate at which the height increases is given by: $\frac{dh}{dt} = K\cos^2\theta$.

The difference in height between S and B can be divided into two parts by using a height h_c such that:

$$h_B - h_c = \sqrt{n} \cos\phi$$

The total time, t_{SB}, to rise from h_S to h_B is the time t_{SC} taken to rise to h_c plus the time t_{CB} to rise from there to h_B.

The reason for using h_c to divide up the height interval is to enable different types of argument to be used about the maximum values of the component times t_{SC} and t_{CB}. A maximum time for t_{CB}, the last part of the journey, can be determined from the slope of the shallowest ridge in the polyhedron of feasible states (see below). By contrast, a stronger upper limit can be set on t_{SC}, the first part of the journey, by relating the minimum rate of gain of height to the distance below the peak. This limit is only stronger if the height difference is at least $\sqrt{n} \cos\phi$, hence the definition of h_c.

The minimum value for dh/dt occurs when travelling along the shallowest ridge, and is given by:

$$\min\left(\frac{dh}{dt}\right) = K \cos^2\phi$$

Hence:

$$t_{CB} \leq \frac{h_B - h_c}{K \cos^2\phi}$$

$$\therefore \quad t_{CB} \leq \frac{\sqrt{n} \cos\phi}{K \cos^2\phi}$$

$$\therefore \quad t_{CB} \leq \frac{\sqrt{n}}{K \cos\phi}$$

An upper bound on the time taken to rise from h_s to h_c can be found by[3] using the fact that, for a convex polyhedron the direction of steepest ascent at a point must always be at least as steep as the direct line from the point to the peak. Since the point cannot be further than $\sqrt{n}$ from the peak the direct line has a cosine with the vertical of at least $(h_B - h)/\sqrt{n}$. So:

$$\frac{dh}{dt} = K\cos^2\theta \qquad \text{and} \qquad \cos\theta \geq (h_B - h)/\sqrt{n}$$

$$\therefore \frac{dh}{dt} \geq K\left(\frac{h_B - h}{\sqrt{n}}\right)^2$$

$$\therefore \frac{dt}{dh} \leq \frac{n}{K(h_B - h)^2}$$

$$\therefore t_{sc} \leq \int_{h_s}^{h_c} \frac{n}{K}(h_B - h)^{-2}\, dh$$

$$\therefore t_{sc} \leq \left[\frac{n}{2K(h_B - h)}\right]_{h_s}^{h_c}$$

$$\therefore t_{sc} \leq \frac{n}{2K(h_B - h_c)} - \frac{n}{2K(h_B - h_s)}$$

Since the term $n/2k(h_B - h_s)$ is positive, it can be omitted, and by definition:

$$h_B - h_c = \sqrt{n}\cos\phi$$

$$\therefore t_{sc} \leq \frac{\sqrt{n}}{2K\cos\phi}$$

So combining t_{sc} and t_{cB}, the total time t_{sB} is bounded by: $t_{sB} \leq \frac{3}{2}\frac{\sqrt{n}}{K\cos\phi}$

A simple example will now be used to illustrate the application of the above expression, and then the expression will be used to illuminate more complex cases. Suppose there are two identical sets of hypotheses with no inteconnecting constraints. Given parallel hardware, the time taken to reach equilibrium is the same for the two sets as for either set alone. Comparing the expression for the two sets with that for a single set, $\cos \phi$ stays the same because the gains in height and the distances travelled both increase by a factor of $\sqrt{2}$. The term $\sqrt{n}$ increases by a factor $\sqrt{2}$, but this is offset by a similar increase in K due to the greater magnitude of the combined preference vector. The larger preference vector does not drag the equilibrium supposition values further from the values at the best vertex, because it is opposed by twice as many constraints, each of which is less effective by a factor of $\sqrt{2}$ because the corresponding plane makes a smaller angle with the preference vector.

Now, consider what happens to the time taken to reach equilibrium when the number of hypotheses is increased by a factor of f, but the magnitudes of the individual preferences and the number of constraints per hypothesis remain the same. Even if the hypotheses cannot be split into disconnected sets, the same reasoning as above can be applied, so $\cos \phi$ will remain roughly the same, whereas $\sqrt{n}$ and K will both increase by a factor of $\sqrt{f}$. The time therefore, will be unaffected.

4.4: Introducing non-linearity.

If there are two equally good interpretations, the ridge joining the corresponding points in hyperspace will be horizontal (assuming the direction of the preference vector is taken as vertical). So the system will not reach either vertex. This is clearly unsatisfactory. Human perception of pictures like the Necker cube suggests that it would be better to somehow select one interpretation arbitrarily. This can be done using the coefficient K_h (section 3.18).

The effect of a non-zero value for K_h is to change the forces acting at each point in the hyperspace. As well as the forces due to the preference vector and any violated constraint planes, an extra force is added, whose magnitude and direction differs at different places. Near a corner of the unit hypercube (i.e. an integer vertex), the extra force is at its greatest and points towards the corner. At the centre of the unit hypercube the force is zero. In fact, the force is radically symmetrical, and its magnitude at a point is proportional to the distance of the point from the centre of the unit hypercube.

One way of thinking about the effect of K_h is in terms of a non- uniform force field like that shown in figure 4.2b. Alternatively, provided K_h is small, a topological transformation can be applied which makes the force field uniform at the expense of bending and non-

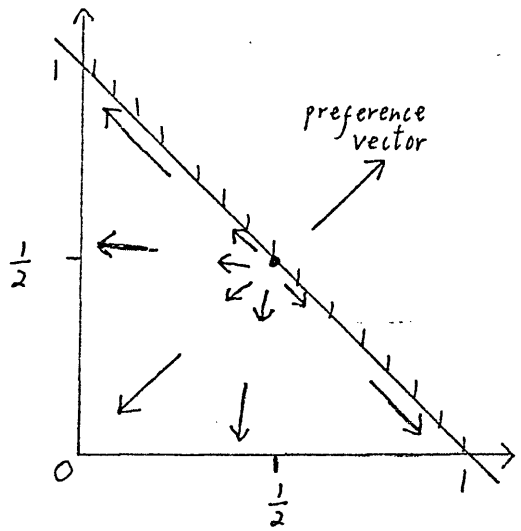


FIGURE 4.2a: Showing two equal rivals, and the additional forces caused by K_h.

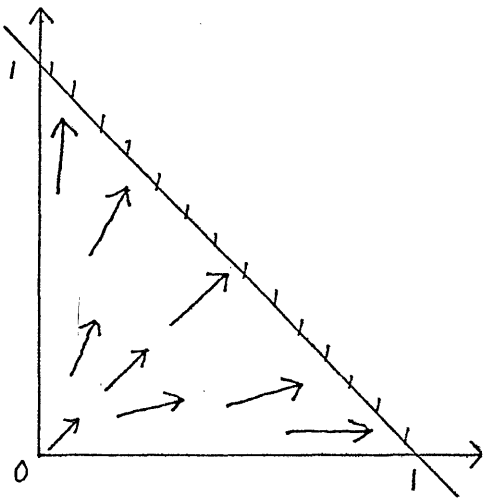


FIGURE 4.2b: Showing the force field obtained by combining the preference vector and the extra forces.

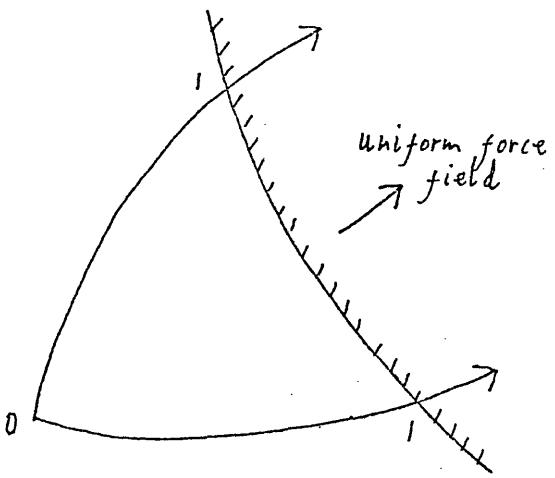


FIGURE 4.2c: Showing the effect of a topological transformation designed to make the force field uniform.

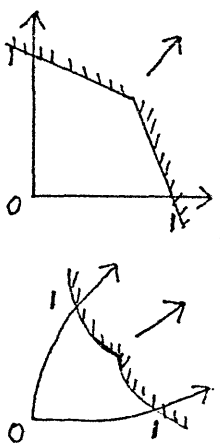


FIGURE 4.2d: Showing a non-integer optimum vertex and the effect of K_h.

uniformly compressing the constraint planes and axes, as in figure 4.2c. This representation has the disadvantage that the forces due to violated constraints need not act normally to the constraint planes. This means that intuitions about the speed at which the state moves can be misleading, though it can never make the state move downhill along a constraint plane (taking the force vector to be vertical).

Using the representation in which k_h distorts the constraint planes, but leaves a uniform force field, it is clear that the two equal rivals have become local optima. It is also clear that a sufficiently high value for k_h can turn a globally optimal non-integer vertex into a very local optimum, as in figure 4.2d.

Although k_h has been used to speed up the puppet program, its effects have not been rigorously analysed. This needs doing because of its apparent helpfulness with the important problems of equal rivals, speed, and non-integer optima. The representation in which k_h causes non-linear, curved constraint planes may be helpful for further analysis, though its value has not yet been demonstrated.

4.5: The Need for Intermediate Level Hypotheses.

An important and valid criticism of the puppet program is that it lacks explicit representations of signi-

ficant groups of parts such as complete arms or legs, or even whole puppets. This lack is a characteristic feature of "holistic" systems (e.g. cellular automata) in which global patterns emerge on the basis of local interactions. Its advantage is that it avoids the potentially explosive number of combinations of local hypotheses. Its disadvantage is that it is generally impossible to express all the required characteristics of the global optimum in terms of preferences and constraints on low level constituents. The puppet task was chosen precisely because much of our knowledge of the human form is reducible to knowledge of the relationships between its rigid parts, but even here, there may be irreducible aspects. Suppose, for example, that good puppet instantiations should have both arms the same length, but that the comparative sizes of the corresponding parts of the two arms are immaterial. A preference for equal arm lengths can be incorporated into the puppet program by creating explicit hypotheses for pairs of arms. Alternatively, pairs of hypotheses for single arms of different lengths could have their suppositions linked by weak incompatability constraints (see section 4.6). Either way, an explicit hypothesis of at least the complexity of an arm is required for the expression of knowlege about arm lengths.

The kind of relaxation used in the puppet program, is quite capable of handling hierarchially structured hypotheses, provided the logical constraints are specified.

In this respect it differs from the intrinsically "flat" relaxation techniques described in sections 5.1 and 5.4. These methods are restricted to tasks in which the problem is to decide which labels (interpretations) to assign to various entities (picture structures). However, although L.P. relaxation can handle many levels of hypotheses simultaneously, it could prove extremely expensive to create all hypotheses at all levels before doing any selection, and it would contradict a major aim of relaxation, which is to avoid explicit enumeration. What is needed is a way of using the initial results of relaxation to guide the creation of plausible higher level hypotheses, so that explicit nodes are not created for combinations of local hypotheses unless they fit in well globally. As mentioned above, the use of relaxation to guide hypothesis creation is discussed later, though not in the context of hierarchically structured hypotheses.

4.6: Weak rules

So far, the only constraints used have been ones which must be satisfied in any allowable global interpretation. This requirement seems too strict to capture the flexibility of human perception. People are capable of violating normal constraints if by doing so they can achieve a much better global interpretation. If a puppet has three well-connected, perfect legs for example, that's how people will see it. Similarly, in interpreting

some lines as capital letters, people will drop the usual perceptual assumption that one line can depict only one letter stroke, if they can thereby arrive at a more sensible interpretation. Ideally, an optimization system should allow a trade-off between preferences for hypotheses and violations of weak rules in arriving at the optimum interpretation.

One way of attempting to implement such a trade-off is to make the constraints corresponding to breakable rules have a much weaker effect on the relaxation operator. If the forces due to violated weak constraints are of roughly the same magnitude as the forces due to the preferences, then the equilibrium position may well involve some weak constraints being significantly violated as a result of the pull in the direction of the preference vector. The disadvantage of this approach is that the forces due to a constraint violation are proportional to the magnitude of the violation, whereas the preference forces are constant. As a result, the system will tend to settle down at an intermediate state where some weak constraints are being violated a bit, but not too much. Such a state is senseless if the weak rules are of the type that either hold or are broken. Suppose, for example, there is a weak rule that a puppet has only two legs. Given a picture in which there is a candidate for a third leg, the best interpretation should either included it or leave it out. It should not contain the third leg to a certain extent, at the cost of violating the weak con-

straint a little.

There is a simple way of incorporating breakable rules which does not run into the above difficulties. Whenever a weak rule gives rise to a constraint, an extra hypothesis is created to represent the possibility that the rule is broken. The hypothesis is given an associated cost depending on the strength of the rule, and instead of the obvious constraint, a more complex one involving the extra hypothesis is created. Suppose, for example, that a weak rule implies the constraint $p \vee q$. An extra hypothesis e (equivalent to $\tilde{p} \wedge \tilde{q}$) is made together with the strong constraint $p \vee q \vee e$. So it is possible to break the rule and have neither p nor q, but only by paying the cost associated with e. An implemented example in which weak inference rules are handled in this way is described in chapter 7.

4.7: Using relaxation to guide hypothesis creation

The puppet-finding program described in chapter 3 is unrealistically simple as a model of how people percieve the puppet pictures. One deficiency of the task is that the number of potential part and joint hypotheses is small enough to allow all the hypotheses to be created before relaxation commences. If the definition of a satisfactory part or joint is extended to allow poor instances (see figure 4.3), then the number of potential hypotheses becomes much larger, so it becomes important

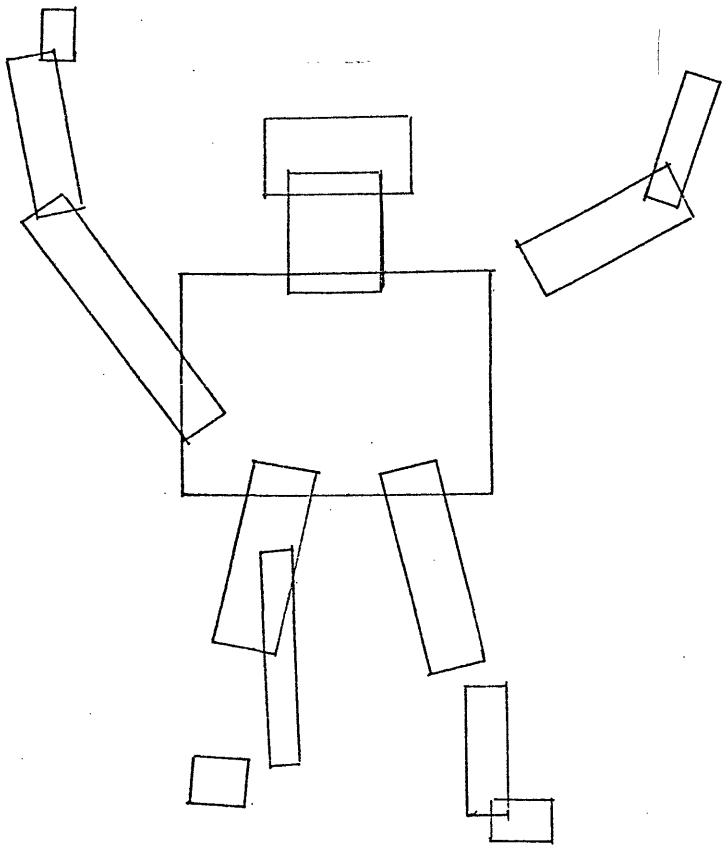


FIGURE 4.3: People see this as a puppet even though the knee and shoulder joints are poor, and the head and trunk have the wrong proportions. The program needs extending to handle such locally poor joints and parts.

to avoid ever formulating many of the possible hypotheses.

This section describes how relaxation and hypotheses creation can be integrated so that the globally best interpretation is achieved without formulating many of the possible hypotheses. No program has been written for this extended version of the puppet task, so there may be unforeseen snags in the method proposed.

4.7.1: The extended puppet-finding task

For human perception, there seem to be many different degrees of acceptability of parts and joints, but for simplicity only three categories will be considered: perfectly acceptable, poor, and unacceptable. Precise definitions of what constitutes a poor part or joint have not been formulated. They should, however, present no problem as they can be of the same form as the definition of good parts and joints, but with less restrictive requirements on the proportions and overlaps.

For reasons which will become apparent later, it is desirable to use only negative scores for hypotheses. Clearly, an interpretation is worse if it has poor parts or joints rather than good ones, but worse still if some parts or joints are missing altogether. A simple, though somewhat ad hoc, method of scoring global interpretations is as follows:

1. For each poor joint or poor part score -1.

2. For each missing slot filler in a part hypothesis score -1.

3. Since all the scores are negative, it is necessary to prevent a global interpretation in which there are no hypotheses at all. This can be done by forcing the program to have a single, obligatory puppet-instance with slots for each part, and imposing penalties on unfilled slots. These penalties need to be large enough to force the slots in the puppet-instance to be filled by rather poor, largely disconnected, parts where necessary, but not so large as to encourage filling by entirely unsupported part hypotheses.

4.7.2: Generators

There is a simple trick which allows relaxation to be started before all possible joints and parts have been found. As well as the normal part and joint hypotheses, slot fillers of a new type called generators are introduced. These have the property that if relaxation makes their supposition values high, they are "run" and replaced by the part or joint hypotheses which are discovered. Generators can be thought of as representing sets of potential hypotheses which have not yet been explicitly created.

If all the good joints and parts are found before doing any relaxation, then all of the hypotheses in the set represented by a generator will be poor ones and will have an associated cost. So the generator can itself be given a cost equal to that of the best hypotheses that might be in its set. If the relaxation process gives a high supposition value to a generator, this means that it is worth searching for the hypotheses which it implicitly represents. If, however, relaxation rejects the generator, then there is no point in running it since any hypotheses so produced would also be rejected.

Figure 4.4 shows a simple case in which relaxation applied to the initial set of good hypotheses could guide the search for poorly connected parts without jeopardising the guarantee of finding the best puppet instantiation. Those poor joints which were never explicitly formulated could not be relevant, since they could not be better than their generator which was rejected by relaxation. The guarantee of optimality stems from the fact that expanding a generator can never improve the state reached by relaxation. It may, of course, make the state worse, since running the generator may produce no hypotheses at all, so that some other, more costly, slot filler would have to be used instead of the generator.

The simple type of generator described above could be elaborated to cope with many different degrees of acceptability of slot fillers. Initially, a generator with

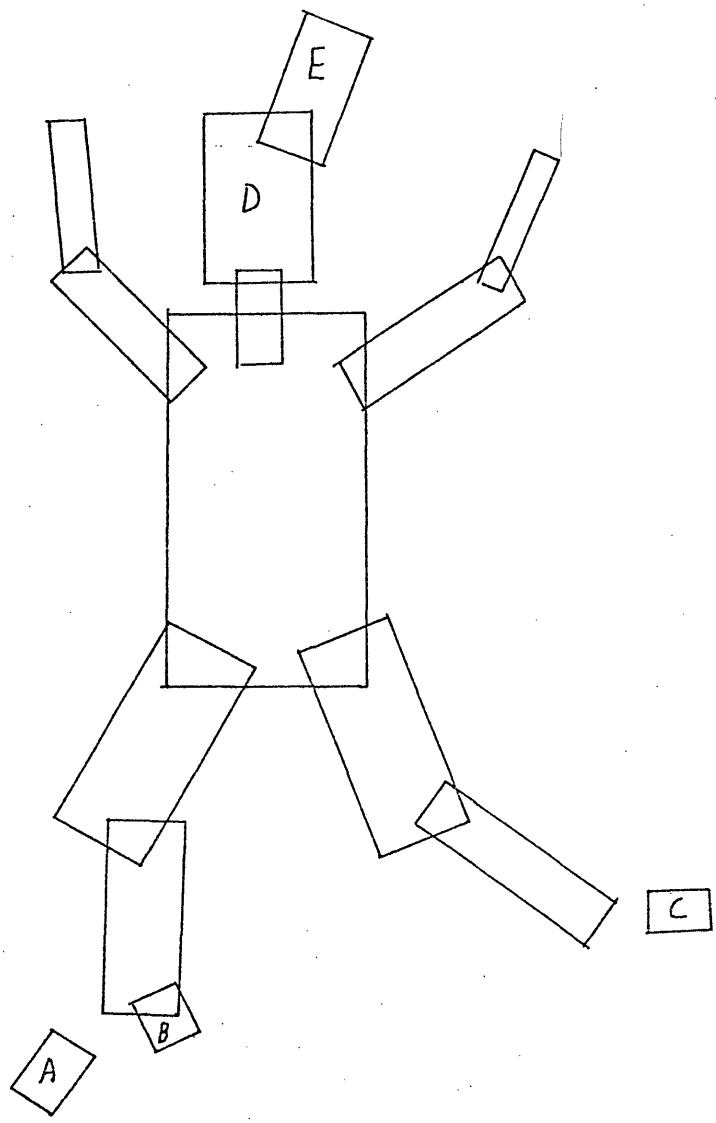


FIGURE 4.4: A puppet with some poor joints. If generators are used to control the search for poor parts, no search will be made for A because the generator will be suppressed by the interpretation of B. There would, however, be a search for C. Similarly, the initial candidate interpretation of E as foot and D as calf would be suppressed by competition, and so the generator for a related thigh would not be run.

a low cost equal to the best of the potential hypotheses would be used. If relaxation gave this generator a high supposition value, a search would be performed for the fairly good hypotheses and the original generator would be replaced by these hypotheses plus a new generator with a higher associated cost equal to that of the best hypotheses which might still be found by further search. Provided the search can be organised to find the hypotheses in order of increasing cost, it should always be possible to avoid searching for hypotheses which are so poor that relaxation would reject them anyway.

The decision about what cost to associate with a generator may be complicated by the fact that a hypothesis produced when the generator is run can fill several slots. For example, a joint hypothesis produced by a joint generator will fill slots in two different part hypotheses. Although each part hypothesis separately may be too weakly supported to bear the cost of a poor joint, together they may be able to bear it. Now, if slots in both parts are filled with separate joint-generators and these generators have the cost of a poor joint, relaxation may reject the generators even though it would accept a shared, poor joint. One solution is to associate with a generator the cost of the best potential hypothesis divided by the number of slots the hypothesis would fill. If each slot contains a generator with this cost, then the search for the potential poor slot-fillers will only be avoided if none of the generators are well

enough supported to bear their share of the cost. This guarantees that hypotheses which might form part of the optimal solution will not be missed, but also means that generators may be run even when relaxation will reject the best hypothesis they might produce.

4.8: Optimising real-valued parameters

So far, relaxation has only been used to find the optimal combination of truth values for sets of inter-related hypotheses. Many problems also involve determining the optimal combination of values for sets of real-valued variables. For example, in finding edges in grey-level data, parameters such as orientation of each piece of edge need to be optimised (Zucker 1976). This section will show how L.P. relaxation can be used for determining real values, though no program has been written. It is important not to confuse supposition values with values of parameters such as orientation. It would be absurd to apply L.P. relaxation directly to the the later. Quantitive decisions need to be reduced to qualitative ones before applying relaxation.

First, an abstract version of the problem will be defined. Suppose there is a finite set of variables, and a finite set of functions each of which takes as input a set of values for a subset of the variables and returns a cost. The task is to find the set of values which minimizes the sum of the costs returned by all the functions.

Provided the costs do not vary too rapidly as the values of the variables change, a simple but expensive way of using relaxation to find an approximate optimum is to consider a number of evenly-spaced values for each variable. A variable-value hypothesis must be created for each possible assignment of a value to variable. Also, a cost-hypothesis must be created for each possible combination of argument values of each cost function. The cost-hypotheses should have associated costs equal to the results of their cost functions and should be bound by constraints which demand that a cost-hypothesis be accepted if all its relevant variable-value hypotheses are accepted. There must also be constraints which require that each variable has exactly one value. For example, if among the variables there are two, A, B, for which values of 1, 2 are considered, then there will be variable-value hypotheses corresponding to A=1, A=2, B=1, B=2. If there is a cost function which accepts values for A and B and returns the difference as the cost, then there would have to be two cost hypotheses with a cost of 0 and two with a cost of 1. The conjunction of the variable-value hypotheses A=1 and B=1 would imply one of the cost-hypotheses which had a cost of 0, and there would be a constraint representing this implication.

Clearly, if the cost functions have many arguments or if many values are considered for each variable, an enormous number of hypotheses and constraints may be needed, so the simple method of formulating all the

variable-value hypotheses before relaxation, is infeasible. However, by using a technique similar to the generators described above, relaxation can be integrated with the formulation of variable-value hypotheses and a great many irrelevant hypotheses can be avoided. The basic idea is to consider intervals in which the value of a variable may lie. Initially the range of possible values for each variable can be covered by a few large intervals, so that instead of many variable-value hypotheses there are a few variable-interval hypotheses.

In order to use relaxation to establish the most promising set of variable-interval hypotheses, it is necessary to modify the cost functions so that instead of taking specific values and returning a cost, they take intervals for the values and return a lower bound on the cost that could be achieved using values within the intervals. For example, if a particular cost function took two numerical arguments and returned their difference as the cost, then its modified version would take two intervals and return either zero (if the intervals overlapped) or the difference between the top of the lower interval and the bottom of the higher one.

Using the modified cost functions to create cost-hypotheses, a promising set of variable-interval hypotheses can be selected by relaxation, and the intervals involved can then be further sub-divided, so that the selected variable-interval hypotheses are replaced by

finer ones. Repetition of this process of selection and sub-division allows the optimal values to be determined accurately without requiring detailed consideration of values within unpromising intervals. If n is the ratio of the range of possible values divided by the accuracy to which the optimal values are required, then, provided there is no back-tracking (see below), the number of interval-hypotheses needed is proportional to log n instead of n for the simpler method described earlier.

Interval-hypotheses which are initially rejected by relaxation must, nevertheless, be retained in the network of possible hypotheses, since when the initially promising intervals are sub-divided it may be impossible to find a combination of the smaller intervals which gives as low a cost as the lower bound estimated for the larger intervals. A simple example shows how this can happen. Suppose there are three variables, A, B, C with real-values in the range 0 to 9, and suppose that there are six cost functions which return costs of:

$$|A-0|,\ |B-4.5|,\ |C-9|,\quad |A-B|,\ |B-C|,\ |C-A|$$

These functions "try" to make A=0, B=4.5, and C=9, but also try to make A=B=C. The best solution is A=B=C=4.5 which has a cost of 9. Suppose the initial intervals used are 0 to 3, 3 to 6, and 6 to 9. Relaxation would select the combination of hypotheses A_{0-3} , B_{3-6} , C_{6-9} where A_{0-3} means that the value of A is in the interval 0 to 3. This combination has zero cost, since for each cost

function there are values yielding a cost of zero within the chosen intervals. However, different values within the intervals are required to satisfy different cost functions. So when the selected interval hypotheses are replaced by more specific ones involving smaller intervals, relaxation may select one of the previously rejected, coarser intervals. If, for example, the selected intervals are sub-divided into intervals of size 1, then relaxation would reject all the more specific hypotheses for A and C and backtrack to the hypotheses A_{3-6} and C_{3-6} which together with B_{4-5} give a total cost of 6.

CHAPTER 5

COMPARISONS BETWEEN L.P. RELAXATION AND ALTERNATIVE SYSTEMS.

In this chapter a number of alternatives to L.P. relaxation are described and criticized. A section is also included on the use of L.P. relaxation for Huffman/Clowes line labelling, since this is the domain chosen by one rival system.

5.1: Rosenfeld, Hummel and Zucker (1975).

In their paper "Scene labelling by relaxation operations", Rosenfeld et al discuss ways of extending Waltz filtering so as to incorporate degrees of compatibility between labels, rather than the simple all or none compatibilities used by Waltz. They describe three models. The first and least interesting is based on fuzzy set theory and associates fuzzy weights with labels. It is like one of the methods used by Barrow and Tennenbaum (see section 5.4.3) and will not be discussed further. The remaining two models use probabilistic weights for labels. These weights are similar in many respects to supposition values, but it will be argued that there are crucial differences which make these methods less satisfactory than L.P. relaxation.

5.1.1. The linear probabilistic model.

A weight between 0 and 1 is associated with each possible label (e.g. +, −, ⊢ or ⊣) for each object (e.g. a line). The weights on the labels for an object sum to 1, so they can be interpreted as the probabilities that the labels are correct (if the distinctions discussed in section 3.4 are ignored). The weights are said to be consistent when each one has a required value which can be calculated (see below) from the weights and compatibilities of the labels on neighbouring objects. If the weights are inconsistent, each is replaced by the value determined by the label weights on neighbouring objects. It can be shown that if this relaxation operator is repeatedly applied in parallel to all the weights a consistent state will eventually be reached. The expression used to determine the required weight $p_i(\lambda)$ on the label λ for the i'th object is:

$$p_i(\lambda) = \sum_j c_{ij} \left[\sum_{\lambda'} p_{ij}(\lambda|\lambda')\, p_i(\lambda') \right]$$

where the c_{ij} are coefficients such that $\sum_j c_{ij} = 1$ for all i. The inner sum in the expression is the expected probability of λ, $E_j(\lambda)$, given the weights and conditional probabilities of the labels at j. The outer sum is a weighted average of the $E(\lambda)$ over all i's neighbours. The magnitude of the constants c_{ij}, c_{iK} indicates the relative importances of the estimates $E_j(\lambda)$,

$E_k(\lambda)$ ... provided by the neighbouring objects in determining the weights of the labels at i.

Rosenfeld et al give no justification for their definition of a consistent set of weights. It is hard to see how it can be reconciled with probability theory because of the following example: suppose that for an object, j, the label λ' has a weight of 1 and all the other labels have a weight of 0. Suppose, also, that $P_{ij}(\lambda \mid \lambda') = 0$ i.e. given that j has label λ', i cannot have label λ. The inner sum of expression 1 (above) correctly yields $E_j(\lambda) = 0$, but because of the weighted averaging of the $E(\lambda)$ this does not force the outer sum to be zero. So a non-zero value for $p_i(\lambda)$ may be allowed by the expression even though it is inconsistent with the conditional probabilities.

The linear model has the interesting property that it converges on a set of weights which is entirely determined by the values of the c_{ij} and the conditional probabilities, and is independent of the initial set of label weights. Rosenfeld et al assume, as do Barrow and Tennenbaum, that the initial weights for particular labels should be used to implement the preferences, which may arise from their a priori probabilities or their goodness of fit to the local data. This assumption leads them to reject the linear model in favour of a non-linear one in which the final state depends on the initial one. They do not discuss the alternative, used by L.P. relaxation

and by Marr and Poggio (1976) of implementing preferences by an extra term in the relaxation operator.

5.1.2: The non-linear model.

The example with $p_{ij}(\lambda/\lambda') = 0$ which was used to criticise the expression (1) above, is actually an extreme case of an undesirable property which Rosenfeld et al discuss. If a label λ' at j has a high weight then it should have a strong tendency to reduce the weights of labels on neighbouring objects with which it has a low compatibility. Expression 1 does not work like this, so Rosenfeld et al suggest replacing the conditional probabilities by correlations, which can have a negative value and can therefore cause the maximum reduction in $p_i(\lambda)$ when the weights on the incompatible labels for j are high. The new expression gives the required change in $p_i(\lambda)$ rather than its required value, and there is no guarantee that the weights will stay positive or that the new weights for labels of a single object will add to 1. These two desirable properties can be restored by modifying the relaxation operator so that it effectively renormalises the new label weights.

The same criticism applies as in the linear model. The way in which the $E(\lambda)$ are averaged in the relaxation operator means that a weight of 1 for a label λ' on j can coexist with a non-zero weight for a label λ on i

even though their correlation is -1.

The convergence properties of the non-linear operator have not been established. It has been tried on the simple problem of choosing the best Huffman/Clowes labelling for a triangle, where a good labelling is defined as one which assigns highly correlated labels to the two lines at an ell junction. The lines were the objects, and the correlations between line-labels at a junction the compatibility functions. The weights converged fairly rapidly, often on integer values, and the initial weights were capable of determining which of the possible unambiguous labellings was chosen.

The main weakness of this model is that it is not clear what computation is being performed. The underlying idea is to enhance label weights by local interactions, but there is no definition of what counts as a good enhancement. A consequence of this lack of a precise problem is that the relaxation operator cannot be derived so as to satisy well specified criteria. Instead, an operator is chosen which has qualitative characteristics which are thought to be desirable. By contrast, L.P. relaxation is designed to perform a well specified task which provides clear-cut criteria for evaluating the relaxation operator.

Zucker (1976) reviews the applications of the non-linear model to "image enhancement" in a number of domains. It is hard to assess the usefulness of some of

the applications since they are intended as a pre-processing stage. In the absence of any clear definition of what this stage is intended to achieve, it can only be evaluated by seeing how much it helps later stages and these are generally non-existent.

One application which is similar in some respects to the puppet task is the enhancement of combinations of parts which match a model (Davis and Rosenfeld 1976). The model used is an upright square of fixed size whose parts are simply its four corners. Nodes are created for candidate corners which are found in a noisy grey-scale picture. Dummy nodes are also created to represent corners which were not found in the grey-level data, but which can be predicted from the corners which were found. Each node has five possible labels corresponding to the four corner types and "no match". The initial label weights at a node reflect the goodness of fit of the corresponding corner types to the local grey level data. The compatibilities between label weights depend on the relative positions of the nodes. For two nodes which are horizontally or vertically separated by exactly the side-length of the square, there will be some pairs of labels, one on each node, which agree and some which disagree. These have compatibilities of +1 and -1 respectively. For pairs of nodes whose relative positions are approximately but not precisely correct, the label compatibilities have correspondingly smaller magnitudes, and for all other pairs of nodes the label weights

do not affect each other. This approach to model matching suffers from all the criticisms already made of the non-linear relaxation method. There is no clear specification of the task, so it is hard to justify the initial label weights or the compatibility functions, or the relaxation operator, or to say precisely what the relaxation process achieves.

One of the aims of the non-linear model is to make use of probabilistic constraints between labellings as well as local biases for particular labellings. It is instructive to see how these types of knowledge can be captured by L.P. relaxation in the example used by Rosenfeld et al. The local biases can obviously be implemented as preferences, but the probabilistic constraints are obviously different from the logical constraints used in L.P. relaxation. Nevertheless, L.P. relaxation can handle probabilistic constraints if they are reduced to logical ones by introducing extra hypotheses with associated costs or preferences (see section 4.6). For the line labelling example used by Rosenfeld et al the extra hypotheses take the form of junction labels. A formulation of the task suitable for L.P. relaxation is given in section 5.2. Compared with the non-linear model, the time taken to reach equilibrium is longer and the number of nodes required is larger. However, it is clear what the computation achieves, and the relaxation process can be analysed.

5.2: Line Labelling using LP relaxation

Huffman/Clowes labelling is explained in section 1.4.1 . There are two reasons for wanting to get the best labelling rather than just a list of all the feasible ones:

1. The number of feasible labellings can become enormous if the set of junction labels is extended to allow for accidental alignment of edges with vertices of different depths, or to accommodate laminae as well as solid objects (Draper - personal communication).

2. People are quite capable of interpreting junctions as accidental alignments, yet they never see more than a few of the interpretations which are possible if such accidentals are allowed.

There are several quite different reasons for associating costs or preferences with particular labellings:

1. If an expanded set of labels is used, costs can be attached to labels which require either accidental alignment or non-solid objects. This can be viewed as a way of providing a set of unusual labels which are to be used sparingly, and only when the usual set is inadequate.

2. If the input is a noisy grey-scale image, rather than a line drawing, there may be weak evidence which suggests particular labels. For example, under some

conditions of illumination, convex edges have slight highlights along them, and concave ones have slight shadows (Rosenfeld et al 1975). Also, shadow edges have distinctive grey level characteristics (see appendix 2). To incorporate this extra information, the idea of extracting a line drawing from the grey-level data needs to be extended to include extracting preferences for particular line labels. The process of finding a consistent labelling for the picture can then operate on richer data than the line drawing alone.

3. When people view a scene they do not perform a detailed analysis of all parts of it simultaneously. It appears that they perceive it in a sequence of glances whose results are synthesised into a representation of the whole scene (Hochberg 1968). Each glance will be accompanied by expectations based on the representation of the scene derived from previous glances. So when a person attends to one part of a scene and attempts to discern its 3-D structure he may already expect it to contain certain types of edge or vertex. It would be possible to mobilise expectations of this kind to aid the interpretation of lines as particular kinds of edges. If a hole is expected, for example, there could be a higher preference for the labelling of those ell junctions which correspond to an interpretation in which the reflex angle lies in the nearer occluding surface.

So it is interesting to see how a program might discover the optimal consistent labelling of a line drawing, where the optimum is defined in terms of preferences or costs for particular line and junction labels.

In a consistent interpretation each line and each junction have exactly one label, so the supposition values in all normalised linear combinations of consistent interpretations satisfy the following constraints:

For each line, l, and for each junction, j,

$$\sum_{\lambda} s_{j:\lambda} = 1 \qquad \text{and} \qquad \sum_{\lambda} s_{l:\lambda} = 1$$

where λ ranges over the possible labels for a line or junction and $l:\lambda$ means that the line l has label λ.

Also, in a consistent interpretation, if a line, l, has label λ then a junction at the end of it, j, must have a label, λ', which is compatible with λ. Hence for line labels:

$$s_{l:\lambda} = \sum_{\lambda'} s_{j:\lambda'}$$

where λ' ranges over the labels of j which give the label λ to line l.

Using these constraints, a network of line label and junction label hypotheses was created for a line drawing of a triangle. Figure 5.1 shows the possible line and

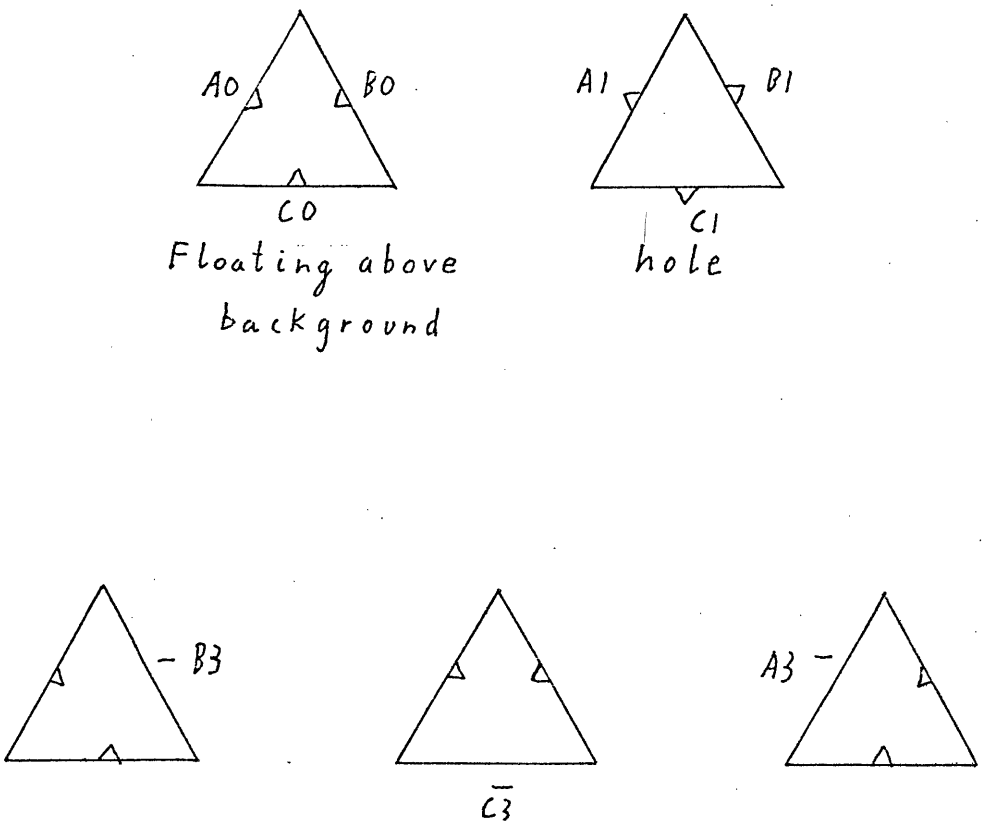


FIGURE 5.1: The possible labellings of a triangle, given the Huffman/Clowes labels for an ell-junction (see figure 1.2). The names A1, A2 etc., are used to refer to particular line labels in figure 5.2 .

```
!relax50();
```

A0	A1	A2	A3	B0	B1	B2	B3	C0	C1	C2	C3
0	0	0	0	0	0	0	0	0	0	0	0
65	22	14	1	65	22	14	1	65	22	14	1
90	7	4	0	90	7	4	0	90	7	4	0
99	0	0	0	99	0	0	0	99	0	0	0
99	0	0	0	99	0	0	0	99	0	0	0
99	0	0	0	99	0	0	0	99	0	0	0

FIGURE 5.2a: Showing how the supposition values change during relaxation for the line labels on the three sides of a triangle. The lines are A, B, C and the suffixes 0, 1, 2, 3 indicate the labels. The meanings of A1, A2 etc., are shown in figure 5.1.

Junction label hypotheses were also involved but are not shown. The preferences were 0.5 for each of the three junction labels corresponding to occluding convex corners, and 0 for all other hypotheses.

```
![h5 1 J5 1 k5 1].set-refs;
!
!relax50();
```

A0	A1	A2	A3	B0	B1	B2	B3	C0	C1	C2	C3
0	0	0	0	0	0	0	0	0	0	0	0
39	89	0	0	39	89	0	0	39	89	0	0
14	95	0	0	14	95	0	0	14	95	0	0
3	99	0	0	3	99	0	0	3	99	0	0
0	99	0	0	0	99	0	0	0	99	0	0
0	99	0	0	0	99	0	0	0	99	0	0

FIGURE 5.2b: If the junction labels corresponding to concave occluding corners are given preferences of 1, the triangular hole interpretation becomes the best.

```
!relax50();
```

A0	A1	A2	A3	B0	B1	B2	B3	C0	C1	C2	C3
0	0	0	0	0	0	0	0	0	0	0	0
48	56	5	0	34	0	99	0	48	56	5	0
41	65	0	0	26	0	99	0	41	65	0	0
23	81	0	0	15	0	99	0	23	81	0	0
0	99	0	0	0	0	99	0	0	99	0	0
0	99	0	0	0	0	99	0	0	99	0	0

FIGURE 5.2c: A preference of 3 for the "convex edge" label, B2, overrides preferences of 0.5 for the "acute occluding corner" junction-labels (since $3 > 0.5 \times 3$), causing the equilibrium state to be the best containing B2.

junction labels and figure 5.2 gives examples of relaxation with various label preferences.

5.3: Yakimovsky and Feldman (1973)

One way of segmenting an image of a natural scene is to start with a large number of small, roughly homogenous regions and to merge them into larger regions which correspond to meaningful parts of the scene. Yakimovsky and Feldman describe a way of arriving at good partitions of images into regions and good interpretations of the regions, which utilizes knowledge about the scenes. The two kinds of knowledge employed are the probabilities of the regions of different kinds depicting particular scene constituents and the probabilities of boundaries of different kinds existing between regions with particular interpretations. For example, blue regions are unlikely to be trees and regions interpreted as road and sky are unlikely to share a vertical boundary. If the probabilities are assumed to be independent and there are no other a priori probabilities, then a global interpretation G, is optimal if it maximizes the product:

$$\prod_i P\left(\begin{array}{l}\text{region } i \text{ has} \\ \text{interpretation int}(i,G)\end{array} \middle| \begin{array}{l}\text{region } i \text{ has the} \\ \text{measured values}\end{array}\right)$$

$$\times \prod_{\substack{\text{for neighbouring} \\ \text{regions } i,j}} P\left(\begin{array}{l}\text{boundary B}(i,j) \text{ is between} \\ \text{int}(i,G) \text{ and int}(j,G)\end{array} \middle| \begin{array}{l}\text{B}(i,j) \text{ has the} \\ \text{measured values}\end{array}\right)$$

where int(i,G) is the interpretation given to region i in

the global interpretation G, and B(i,j) is the boundary between region i and region j.

Using conventional techniques it would be extremely expensive to evaluate the product for all the combinations of region interpretations for all partitions of the image into regions. To avoid this, the part of the program discussed by Yakimovsky and Feldman only considers a sequence of partitions generated by removing possible boundaries one at a time in a particular order, and for each partition it only computes upper and lower bounds on the product. Given these bounds, graph searching techniques can be used to find good interpretations of particular partitions. The upper bounds are found by relaxing the consistency constraints, so that the individual terms in the product are simply the probabilities of the locally best interpretation for each region . The lower bounds are found by choosing interpretations for the regions one at a time, the extent by which the most probable interpretation of a region outstrips the others being used to decide which region to interpret next. This is an example of a method discussed in Section 1.6.1 for finding good but not necessarily optimal interpretations.

There are serious objections to the way in which Yakimovsky and Feldman have formulated the segmentation problem. They have ommitted general knowledge about 3-D structure whilst including specific knowledge about the probabilities of particular scene constituents being dep-

icted by neighbouring regions. At a low enough level both types of knowledge may be absent, and at a high enough one both may be present, but it seems unlikely that really good segmenters (people) invoke knowledge of particular objects before invoking general 3-D knowledge (Marr 1975). The most impressive segmentation programs use inferences based on 3-D structure and not on specific types of object (Guzman 1968, Clowes 1971, Waltz 1972). (Guzman's program does not appear to use 3-D knowledge. However, the reason his program works so well is that it uses 2-D cues which allow powerful inferences because of the 3-D structures they imply).

The abstract problem presented by Yakimovsky and Feldman suggests a relaxation approach, and it is informative to see how relaxation can be applied, what difficulties it runs into, and how they may be overcome.

5.3.1: A relaxation formulation

The task of maximising the product given above is equivalent to minimizing the total cost of a set of hypotheses about region and boundary interpretations, where the individual costs are the logs of the probabilities. It seems to be necessary to have hypotheses about regions, boundaries, region interpretations and boundary interpretations. The constraints are:

1. Larger regions are produced by merging small ini-

tial regions. In any global interpretation, an initial region must be either unmerged or part of exactly one larger region. So for all regions which share an initial region:

$$\sum s_r = 1$$

2. Every region should be given exactly one interpretation:

$$\sum_i s_{ri} = 1$$

where s_{ri} is the supposition value of the hypothesis that region r has interpretation i.

3. If two neighbouring regions q, r exist in the best interpretation then so does the boundary between them. So for all neighbouring pairs q, r:

$$q \wedge r \supset B(q,r)$$
$$\therefore \bar{q} \vee \bar{r} \vee B(q,r)$$
$$\therefore (1-s_q) + (1-s_r) + s_{B(q,r)} \geq 1$$
$$\therefore s_{B(q,r)} \geq s_q + s_r - 1$$

4. Similarly, if two neighbouring regions q, r have interpretations i, j then the boundary between them has interpretation $B(q_i, r_j)$:

$$s_{B(q_i,r_j)} \geq s_{q_i} + s_{r_j} - 1$$

where $s_{B(q_i,r_j)}$ is the supposition value of the hy-

pothesis $B(q_i, r_j)$ and S_{q_i} is the supposition value of the hypothesis that region q has interpretation i.

There are three main objections to straightforwardly creating all the relevant hypotheses and constraints and then finding the best state.

1. It is not clear in advance how many region hypotheses to make. Yakimovsky and Feldman continued removing boundaries until the upper and lower bounds on the best possible interpretation of the current partition fell sharply. This relies on the assumption that once the product falls significantly, further merging will not raise it again. If the assumption is valid, relaxation could be used on some initial partitions, and further merging to produce new partitions might only be necessary if the best solution found by relaxation involved one of the later partitions, that is, one with many merges.

2. If there are i interpretations for each region, and r regions, the number of region-interpretation hypotheses is i.r, which may be of the order of a thousand for the data given. For boundary interpretations, however, the number is about $\frac{1}{2}$ b.i^2.r where b is the boundaries per region. This is a formidable number if i is large. Fortunately, it is possible to avoid ever formulating many of the boundary interpretation hypotheses. Hypotheses about the interpretation of a boundary need only be added when the relaxa-

tion process raises to a significant level the supposition values of particular interpretations of the regions on either side of the boundary. This is because boundary interpretations have associated costs and so will not be included in the best global intepretation unless they have to be. The only thing that can force the inclusion of a boundary interpretation is a constraint of type (4) above which does not become operative until the sum of the supposition values for the alternative interpretations of a region exceeds 1. This is another example of the important technique of avoiding irrelevant hypotheses by integrating hypothesis creation with relaxation.

3. Since all the preferences are negative, there is a tendency for constraints like (4) above to lead to non-integer optima, so that relaxation does not produce a clear-cut answer and it is necessary to use cutting planes or branching (see Section 3.7). The reason for expecting non-integer optima is that if many region interpretations are given supposition values of a half or less, constraints of type (4) do not constrain the supposition values of the boundary interpretations, and so the associated costs are not incurred. Constraints of type (2) above can still be satisfied by several different interpretations of a region, each of which has a small supposition value.

If a relaxation program of the kind described could

be made to work then apart from the advantage that it could use parallel hardware, it would be capable of finding a solution in which there were late merges in one part of the image without being too committed to earlier merges in another part. Yakimovsky and Feldman have a strict ordering for boundary removal and this sequential strategy prevents them from ever considering most of the complete partitions involving subsets of the candidate regions they generate. This point may be clarified by a simple example. Suppose there are four initial regions R1, R2, R3, R4. If merges are considered in the order R1 + R2 $\longrightarrow$ R12, R3+R4 $\longrightarrow$ R34, R12 + R34 $\longrightarrow$ R1234 then the partition R1, R2, R34 will never be considered, even though it only involves existing regions.

5.4: Barrow and Tennenbaum (1976)

5.4.1: The task

Barrow and Tennenbaum describe a system, MSYS, which is designed to find the optimal consistent set of interpretations for regions in a hand- partitioned image of a room scene. Regions correspond to entities like the back of a chair, a picture, a door or a patch of floor. Region interpretations are given a priori likelihoods on the basis of their height in the scene and their surface orientations, which are discovered using a laser range finder. There are various constraints between the in-

terpretations of different regions. A picture, for example, cannot be adjacent to a door, and two patches of floor must be of similar brightness.

5.4.2: The general strategy

Barrow and Tennenbaum describe several versions of their system. Only the version for which there is a guarantee of finding the best solution is described below.

The optimal set of region interpretations can be found by using a branch- and-bound search (Duda 1970). Branches are created by opting for or against a particular region interpretation, and an upper bound is set on the best terminal state reachable along a given branch by combining the likelihoods of the locally best surviving interpretations for each region. MSYS uses a branch-and-bound search, but for each intermediate state it attempts to get a much tighter upper bound. Instead of simply combining the best surviving a priori likelihoods, it enters a relaxation phase in which the constraints are used to lower the likelihoods. It then uses the best lowered a posteriori likelihood for each region, the local optimum, to compute the global upper bound. The hope is that given sufficiently rich constraints the upper bound will be so tight that hardly any branching is required.

5.4.3: Likelihoods and their modification

The actual method MSYS uses for modifying the likelihoods during the relaxation phase is hard to grasp because it is not clear what the likelihoods are, and so it is not clear how they should be manipulated. The real logic behind the way the likelihoods change seems to be the requirement that they always fall so that the highest value in intermediate states can be used to set an upper bound on the values obtainable in terminal states. Given this requirement on how the numbers should behave it is not clear that any sensible interpretation of them exists. The a priori likelihoods of the different interpretations of a region sum to 1 which suggests that they are probabilities. However, after a phase of relaxation the sum is no longer 1. The numbers cannot be renormalised, because this might raise some of them. Also, although the numbers start off looking like probabilities, the way the local optima are combined to get a global upper bound is by addition, not multiplication. This may suggest that likelihoods are logs of probabilities, but the way they interact via constraints argues against it. The basic form of a constraint is that an interpretation R_i of one region, R, must be supported by particular interpretations S_j, T_k ...of other regions S,T.... If these interpretations have low or zero likelihoods then so must R . The actual numerical constraints may be based either on set theory or on fuzzy set theory (see fig. 5.3).

$$R_i \supset S_j \Rightarrow l(R_i) \leqslant l(S_j)$$

$$R_i \supset S_j \wedge T_k \Rightarrow l(R_i) \leqslant l(S_j) \times l(T_k)$$

$$R_i \supset S_j \vee T_k \Rightarrow l(R_i) \leqslant 1-(1-l(S_j))\times(1-l(T_k))$$

Figure 5.3 Showing how logical constraints give rise to numerical ones using set theory. "$\supset$" means "must be supported by" and $l(R_i)$ is the likelihood of R_i .

$$R_i \supset S_j \Rightarrow l(R_i) \leqslant l(S_j)$$

$$R_i \supset S_j \wedge T_k \Rightarrow l(R_i) \leqslant \inf(\, l(S_j),\ l(T_k)\,)$$

$$R_i \supset S_j \vee T_k \Rightarrow l(R_i) \leqslant \sup(\, l(S_j),\ l(T_k)\,)$$

Figure 5.3b Showing the numerical constraints derived using fuzzy set theory.

5.4.4: An abstract example

Suppose there are two regions, R,S, each with three interpretations. Figure 5.4b shows some a priori likelihoods, and the a posteriori likelihoods reached after relaxation using the constraints shown in figure 5.3a. When equilibrium is reached a branch is made on the likelihood of R_3, say, by setting it or all its rivals to zero. This gives the states shown in figure 5.3c. After relaxation, a terminal state is reached which has value 0.24 + 0.2. Since this is better than the combined local optima in the other, intermediate state, it is the best solution according to the criterion used by MSYS. Notice, however that the solution R S is consistent and that both sum and the product of its a priori likelihoods are better than for MSYS's choice. The reason why MSYS does not find the solution R S is that it only uses the constraint $R_1 \supset S_1 \vee S_2 \vee S_3$ to lower the likelihood of R_1 whereas if likelihoods are anything like probabilities, the constraint should also have the effect of raising S_1, S_2 or S_3 when R_1 is high and they are all low, as it does in L.P. relaxation.

$R1 \supset S1 \vee S2 \vee S3$, $S1 \supset R1$, $S3 \supset R3$

$l(R1) \leqslant \sup(\, l(S1),\, l(S2),\, l(S3)\,)$

$l(S1) \leqslant l(R1)$, $l(S2) \leqslant l(R2)$, $l(S3) \leqslant l(R3)$

FIGURE 5.4a : Some constraints between interpretations of R and S (first line) and the corresponding numerical constraints between likelihoods (second and third lines).

$l(R1) = 0.76 \rightarrow 0.7 \rightarrow 0.2$	$l(S1) = 0.1$
$l(R2) = 0.01$	$l(S2) = 0.7 \rightarrow 0.01$
$l(R3) = 0.24$	$l(S3) = 0.2$

Figure 5.4b : Some a priori likelihoods, and the results of a relaxation phase (indicated by arrows) using the constraints above.

Choose R3		Reject R3	
$l(R1)=0$	$l(S1)=0.1\rightarrow 0$	$l(R1)=0.2\rightarrow 0.1$	$l(S1)=0.1$
$l(R2)=0$	$l(S2)=0.01\rightarrow 0$	$l(R2)=0.01$	$l(S2)=0.01$
$l(R3)=0.24$	$l(S3)=0.2$	$l(R3)=0$	$l(S3)=0.2\rightarrow 0$

FIGURE 5.4c : Two states obtained by branching on R3 from the state obtained after the relaxation phase in figure 5.4b above.

5.4.5: Comparison of MSYS with LP relaxation

The main criticism of MSYS is the lack of a precise interpretation for likelihoods. From the point of view of LP relaxation, the reason for the confusion is the lack of a distinction between preferences and supposition values. Likelihoods seem to be an attempt to combine these two different types of number into one. A priori probabilities (preferences) are represented as initial values for likelihoods, so when the likelihoods change, the a priori probabilities are lost and the criterion of the optimal consistent state cannot be in terms of their product. The criterion of maximizing the sum of the likelihoods seems like an unprincipled choice for facilitating the branch-and-bound search. By contrast, when LP relaxation is combined with a branch-and-bound search as a way of handling non-integer vertices (see Section 3.7) the measure which is being optimized, and is used as a bound, is a principled one.

Despite these criticisms of detail, the general view of the way computations may be performed in vision is shared by the authors of MSYS and LP relaxation. In particular, the importance of constraint propogation for avoiding search, as illustrated by Waltz's program and REF-ARF (Fikes 1970), was first explained to me by Barrow.

5.5 Growing islands of consistent hypotheses.

In the revised puppet task (Section 4.7), distorted parts and poor relations are allowed but have an associated cost. The problem is to find the consistent set of interpretations of the rectangles and overlaps with the minimal total cost. One alternative to relaxation is a branch-and-bound search (see Section 1.9) in which the cost of a partial solution is the sum of the costs of its constituent hypotheses. The first complete solution whose cost is lower than any of the uncompleted partial solutions is the optimum. Unfortunately, a partial solution which is nearly complete will tend to have a much higher cost than one which contains only a few hypotheses, especially for a puppet picture in which the best interpretation contains many poor joints or parts. Consequently, the optimal solution will not be reached until all the other partial solutions have been developed to contain a considerable number of costly hypotheses. This means that the bound will not prune the search tree very effectively.

The reason for the ineffectiveness of the branch-and-bound search is that large partial solutions are unfairly penalised compared with small ones. A better measure of the promise of a partial solution can be obtained by comparing the total cost it has incurred with its size. More precisely, the "shortfall density" of a partial solution can be defined as the mean value, over

all its hypotheses, of the difference between the cost of the hypothesis chosen and the cost of the locally best hypothesis for explaining the same data (i.e. the rectangle or overlap). The smaller the shortfall density the better the partial solution. This measure cannot be used in a branch-and-bound search because the best overall solution might start life as a very unpromising partial solution, and so there is no guarantee that a complete solution which has a lower shortfall density than any currently existing partial solution is the optimum. However, an island growing technique used in the HWIM speech understanding system and described by Woods (1977) can make effective use of shortfall density to prune the search space. The way in which a modified version of the technique would be applied to the revised puppet task is described below. At present this application is entirely hypothetical.

The first stage is to create a number of seed hypotheses which will act as the initial islands. These are like the nuclei used in the puppet program (See section 3.2), though they differ in that it is not always sufficient simply to find just one seed in the best interpretation. To be sure of finding the optimum it is necessary that all its good constituent hypotheses be seeds (see below). One way of ensuring this is to make all the good local hypotheses act as initial islands.

The second stage consists in growing islands either

by merging two islands, or by creating and adding new interpretations of the rectangles or overlaps neighbouring an island. The island with the lowest shortfall density is always selected as the next one to be grown until there is an island which covers all the rectangles and overlaps and still has a lower shortfall density than any other. This is taken to be the best global interpretation. The reason that it can be accepted is the best in this case though it would be unacceptable in a branch-and-bound search, is that if there were a better complete solution, it would have to contain a partial solution with a lower shortfall density and if there were such a good partial solution it would already have been grown from one of the seeds. To put it another way, a tree search imposes an ordering on the rectangles and ovelaps which may force the best global interpretation to grow from a partial solution with a high shortfall density, whereas island growing from a sufficient number of seeds allows the best parts of a global interpretation to be grown first.

Compared with LP relaxation, island growing has both strengths and weaknesses. It avoids all the messy problems associated with the use of continuous supposition values. Also, by combining the constraints imposed by the hypotheses in an island, it should be possible to restrict the search for the new hypotheses which may act as extensions to the island. A potential weakness of island growing is that whenever a new island is created, a

check must be made to ensure that it is not a copy of an island which already exists. Given a large quantity of data and hence many islands, the checking process can be very time consuming. A further difficulty is that there is no simple, economical way of handling minor variations of an island. The obvious strategy is to allow islands to contain "OR" nodes, but there may be interactions between the choices at different "OR" nodes. Suppose, for example, that at one place in an island there is a choice of A or B, and at another place there is a choice of C or D. It may be that A is incompatible with D, and B with C. If "OR" nodes are to be used, these dependencies need to be explicitly represented, perhaps by something like the connectivity matrices of Hearsay II (Erman and Lesser 1976). Also, "OR" nodes greatly complicate the process of using the content of an island to restrict the search for possible extensions. So perhaps the best strategy is the simple but expensive one of creating two completely separate islands for each minor variation.

In the absence of a detailed example of the use of island growing and shortfall density for picture interpretation, it is hard to assess the importance of the above criticisms or to discover the effectiveness of shortfall density in limiting the number of islands. The fairly successful use of island growing in HWIM (Woods 1976) seems to be the best available guide to its value.

5.6: Matching by Clique finding

Ambler et al (1975) describe an efficient matching technique which is well-suited to the puppet task. In their example, the problemsmof finding the best match between a data-graph and a model-graph, is transformed into the problem of finding maximal completely connected subgraphs (cliques) of a third graph, in which each node corresponds to an interpretation of a data-node as a model-node. Two interpretation nodes are linked by an undirected arc if and only if the interpretations are compatible. In the puppet example, there is no explicit model-graph, but the part and joint hypotheses are equivalent to interpretation nodes and the clique-finding technique can be applied if all compatible pairs of hypotheses are linked by arcs.

An efficient clique finding algorithm is described by Bron and Kerbosch (1973). It works by extending totally connected subgraphs, but unlike island growing, it manages to avoid ever creating the same clique twice, and hence avoids checking for duplicates.

Although it may be the best solution to the simple puppet task described in Chapter 2, it is not clear how clique finding can incorporate additional input instructions favouring certain solutions over others, or how it can be extended to the revised puppet task (see Section 4.7) in which the hypotheses do not all have preferences of one or zero. Ambler et al suggest using thresholds to

eliminate poor hypotheses and also poor arcs between pairs of hypotheses which are only poorly compatible. All remaining hypotheses and the compatibility arcs between them are then treated as equally good, thus reducing the problem to the previous form. However, something is lost in the reduction. Maximising the number of consistent good hypotheses is not the same problem as finding the best consistent set of hypotheses. So although clique-finding is efficient for some matching problems, there is no obvious way of extending it to the more general problems to which LP relaxation can be applied.

5.7: Hierarchical synthesis

Barrow et al (1972) describe a very efficient graph matching technique stemming from work by Selfridge and Neisser (1960). Rather than having a single model-graph, there is a hierarchy of them corresponding to the hierarchy of parts in the model. Each part has a corresponding graph or relational net whose nodes correspond to smaller parts. In the program which implements hierarchical synthesis, each part of a model has a corresponding program module which contains the relational network of smaller parts, pointers to the modules for smaller parts, and back-pointers to all the modules whose relational networks contain the part. During matching, activated modules search for all reasonable instantiations of their

relational nets. To do this they need instantiations of their lower level modules so they activate them. When a module finds any successful instantiation it returns it to its higher level modules which are in turn, activated. Top-down matching is caused by initially activating the top-level module, and bottom-up matching by activating all the lowest level modules. The reason that hierarchical synthesis is efficient is that modules remember their instantiations, so that time is not wasted in repeated efforts to match the same subgraph. As this suggests, the method is particularly effective if many different higher level modules share a lower level one.

Some kind of hierarchical structuring seems inevitable in visual perception, but there are a number of ways in which the simple version of hierarchical synthesis described above is not an entirely adequate model:

1. When a module is activated by a lower level one, it requests it, in effect, to search for all reasonable instantiations. This is not a rich enough interaction between modules, since under many circumstances the search could be restricted by mobilising constraints imposed by the instantiations of sibling modules. For example, suppose a leg module has pointers to lower level foot, calf and thigh modules. If a thigh and foot have already been found, then when the leg module activates the calf module, it should give additional information about the expected size,

position and orientation of the calf.

2. There is evidence (Navon 1977) that in human perception, an awareness of coarse, global structure precedes the analysis of details. In hierarchical synthesis this is impossible since the only way of discovering that a high level module is instantiated is via its lower level modules. What is needed is a more direct link between higher modules and the grey-level data.

3. For many objects, there is no natural unambiguous hierarchical decomposition into parts, so each module may need to have alternative relational networks using different decompositions (see Turner 1974). Another reason for wanting modules corresponding to many different, overlapping fragments of an object is that when an object is partially occluded, the remaining fragment is probably easier to recognise if it can be seen as one of a few known fragments than if it can only be analysed as fragments of fragments.

4. In general, modules will not find perfect instantiations, so some mechanism is needed for making the best of imperfect ones. Turner (1974) uses linear threshold functions to decide whether an imperfect instantiation is acceptable. However, this means that a high level module may accept an instantiation consisting of many barely acceptable parts, but reject one with several perfect parts and one just unacceptable

one. As in clique-finding, local thresholding cannot guarantee the global optimum.

5. Perhaps the greatest potential advantage of LP relaxation over graph-matching techniques like hierarchical synthesis or clique-finding, lies in the way that occlusion, lighting, and support might be handled. It is hard to see how knowledge of these effects can be mobilised in graph-matching. In fact, occlusion is typically treated as if it were inexplicable noise (Turner 1974). By contrast, LP relaxation provides a mechanism which should be able to incorporate specific inferences based on explicit hypotheses about occlusion, lighting, or support, so that relaxation could integrate decisions about these effects with decisions about three-dimensional shape. Naturally, a great deal of work would be required to write a program which demonstrated that this promise could actually be fulfilled.

CHAPTER 6

PERCEPTUAL SCHEMAS AND THEIR RELATIONSHIP TO PERCEPTUAL AWARENESS.

The main aim of this thesis is to investigate relaxation as a method of finding optimal interpretations of scenes, and so many important perceptual issues have been deliberately avoided in discussing the relatively simple applications of relaxation which have been described so far. However, the next application to be descibed is a system which allows relaxation to be used in the construction of more complex perceptual representations, and in order to implement the system, it was necessary to face up to some difficult general issues. Decisions had to be taken about the types of representation used in perception, and about the relationship between stored knowledge and the current awareness of a particular scene. So this chapter discusses these issues, and then Chapter 7 shows, in detail, how a particular approach to them can be incorporated in a working system.

6.1: Current awareness and stored knowledge

It will be assumed that the representation of a particular scene is some kind of relational nework, (see

Guzman 1968, Winston 1970). An important issue is how these representations are related to those of stored general knowledge about the forms of objects. In psychological terms this amounts to the relationship between the contents of current awareness and the contents of long term memory. There is a view, common in Psychology and Artificial Intelligence, that the two types of representation are similar in form, so that the contents of long term memory are like copies of the contents of current awareness. This view will be criticised and contrasted with an alternative model, a simple version of which has been implemented.

The following two assumptions constitute a model of how objects are remembered and recognised which seems to be used implicitly by many psychologists.

1. Long term memory consists of a store of something like copies of percepts, and recalling consists in retrieving things from this store, or in activating them.

2. Recognition involves comparing percepts with stored memory images.

Some of the plausibility of this model of recognition and memory may come from its similarity to well known systems which work in just this way. For example, finger-prints are recorded by taking copies of them and suspect prints are recognised by comparing them with the

stored copies. Also, the contents of current awareness seem, introspectively, to be similar when we perceive an object and when we recall it.

A quite different model of memory, which was supported by Bartlett's (1932) experiments, is that recalling is a constructive process of creating a coherent, articulated representation rather than simply reactivating or retrieving a copy. A good analogy is "remembering" a sweater by keeping the knitting instructions so that the sweater can be recreated, as opposed to remembering it by keeping another similar sweater. On this model, the contents of current awareness resulting from recall may be different from the contents of long-term memory, so that the expression "memory image" must be reserved for one or the other. If we use "memory image" to mean a representation in current awareness created in the absence of the relevant perceptual input, then the contents of long-term memory may be nothing like a memory image.

Perceiving is also a constructive process which uses some of the same long- term memory information as remembering, but this does not mean that any remembering goes on when we perceive. We may deliberately choose to compare a perceived object with a memory image, but this is introspectively quite different from the perception and recognition of a familiar object.

The evidence against the stored copies model comes

mainly from the generative nature of perception and memory. Bartlett, for example, showed that if people are asked to recall a story after progressively longer intervals, they produce stories which contain less and less of the detail of the original and are more and more in accordance with general expectations. This seems to fit the idea that what are stored are rules for constructing the story and that if any of the rules are lost, general principles are used in their place. The idea of stored rules also seems to be necessary to explain how we can perceive objects which have never before been encountered, such as a flight of stairs with nineteen steps. Stored copies of previously perceived flights of stairs would presumably contain a particular number of steps, but what we need is an awareness of the grammar of stairs, the way in which risers and treads alternate. The similarity between structures built during perception and the structures which Linguists assign to sentences, has been expounded by Narasiman (1966) and Clowes (1969) among others. The linguistic analogy is particularly helpful here, for supposing that our knowledge of spatial structures resides in stored copies of percepts, is like supposing that our knowledge of grammatical structure resides in a set of stored sentences.

6.2 Frames

In a widely read paper, Minsky (1975) expounded a theory of the way in which knowledge is structured and used in perception and understanding. His theory will be discussed at some length, mainly in order to attack his view that current awareness and long term memory have the same form, but also because many of his ideas about the structuring of knowledge into frames are incorporated in the system to be described.

6.2.1: An example of a schema

The idea that we understand the world by assimilating it to our own schemas (or frames) is far from new, having been expounded by Kant (1781). Piaget (1954) and Bartlett (1932) among others. The difficulty of the following task is a striking illustration of the existence of schemas and their powerful influence on our awareness of reality. Imagine a solid, regular tetrahedron, and then try to imagine a plane which cuts it so as to give a square cross-section. Most people cannot imagine such a plane. Their schema for a tetrahedron gives it a triangular base and three sloping triangular faces. There are three horizontal base-edges and a tripod of other edges. Not only are there no right angles, but edges and faces naturally fall into groups of three.

There is, however, a quite different schema for a

tetrahedron, which is more appropriate when the tetrahedron is in a different orientation, since then the edges and faces which are grouped together have similar inclinations to the vertical. Imagine a horizontal edge resting on the support plane, with another horizontal edge at right angles to it and some distance above it, so that the centres of the edges are vertically aligned. Now join each end of one edge to each end of the other as in figure 6.1. This is a quite different way of thinking of a tetrahedron. The faces naturally form two pairs each of which is hinged across a horizontal edge. The edges fall into a group of two horizontal ones and four sloping ones. In volumetric terms, the tetrahedron can be seen as a stack of rectangular laminae which are very elongated at the bottom, become progressively squatter nearer the middle, and are elongated the other way at the top. Half way up is a square.

6.2.2: Minsky's theory

Minsky puts forward a theory of how frames are used and inter-related:

> "Here is the essence of the theory: When one encounters a new situation (or makes a substantial change in one's view of the present problem) one selects from memory a substantial structure called a frame. This is a remembered framework to be adapted to fit reality by changing details as necessary.

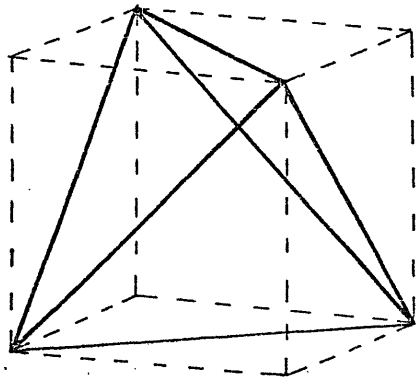


FIGURE 6.1: A tetrahedron inscribed in a cube (after Hilbert and Cohn-Vossen 1952). The top/bottom direction suggested by the cube can be used for understanding the tetrahedron, but it gives rise to a different schema from the normal one (see section 6.2.1). Conversely, the normal schema for a tetrahedron involves an intrinsic top/bottom direction which can be imposed on the cube to reveal a different schema in which the hexagonal cross-section is apparent. (This takes practice).

"A frame is a data-structure for representing a sterotyped situation, like being in a certain kind of living room, or going to a child's birthday party. Attached to each frame are several kinds of information. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are not confirmed.

"We can think of a frame as a network of nodes and relations. The "top levels" of a frame are fixed, and represent things that are always true about the supposed situation. The lower levels have many terminals - "slots" that must be filled by specific instances or data. Each terminal can specify conditions its assignments must meet. (The assignments themselves are usually smaller "sub-frames".) Simple conditions are specified by markers that might require a terminal assignment to a person, an object of sufficient value, or a pointer to a sub-frame of a certain type. More complex conditions can specify relations among the things assigned to several terminals.

"Much of the phenomenological power of the theory hinges on the inclusion of expectations and other kinds of presumptions. A frame's terminals are normally already filled with "default" assignments. Thus, a frame may contain a great many de-

tails whose supposition is not specifically warranted by the situation. These have many uses in representing general information, most-likely cases, techniques for bypassing "logic", and ways to make useful generalizations."

One of the main aims of the theory is to show how our apparently rich and complex immediate awareness of the scene can be compatible with serial processing. Minsky believes that, although parallelism may be useful at lower levels, it offers little help to hypothesis formation and confirmation methods that seem necessary at higher levels. Instead of the parallel formation and parallel interaction of many hypotheses, expounded in this thesis, he proposes the serial manipulation of complex pre-existing structures so that the richness of awareness comes from selecting the correct existing structure rather than from constructing one.

6.2.3: Some Difficulties for Frames

Minsky implies that frames are data-structures which get joined together by making terminal assignments during perception. This creates a problem for rooms with two windows. Presumably there is only one window frame, so what happens when both window slots in the room are filled? If the details of the windows differ, there will be rival fillers for the slots in the window frame. It

seems that we must be able to copy the window frame and use separate copies for the two slots in the room frame. So the economical idea that all the main high-level data-structures used in perception are ones that already exist has to be abandoned.

A more serious difficulty is that some frames, such as those for a polygon or a zebra crossing, need to have a variable number of slots. This suggests that frames contain generators for instances rather than simply being copied to produce instances, just as in computing languages structures like arrays are not made by copying a standard array but by a procedure which can take parameters. Even when the number of slots is fixed, as in a POP-2 record, there is no need to generate instances by copying a standard example. There is an important issue here about the value of a particular example of the structure - a structural template - as a model of structures of that kind. At first sight such a direct representation seems to have many advantages (see Sloman 1971). However, it also has many disadvantages. For example, our knowledge that a square has square corners is more economically represented as a single rule that can be applied to any corner of the square rather than as four separate pieces of knowledge attached to the four corner slots, and the same goes for our knowledge that each white stripe in a zebra crossing is bounded by two black ones.

Another difficulty for structural templates, stems from the hierarchy of types of object. For example, an ostrich is a type of bird, so it seems to be redundant to have a frame for an ostrich which contains two slots for wings, since this structural information is already contained in the frame for a bird. Although it may be convenient, as an implementation detail, to store knowledge about ostriches within a bird-frame, this structure need neither be used nor copied to create the bird instances used for representing a particular ostrich, since we may create a representation of a bird before deciding whether it is an emu or ostrich and hence before the ostrich frame has been selected at all. The view that instances are created by copying frames leads to awkward questions about whether to copy the bird frame or the ostrich frame or both in order to represent a particular ostrich. Such questions do not arise if stored knowledge consists of schemas which define roles and rules (see below) since then the instance representing an ostrich in current awareness can derive roles and rules from both schemas simultaneously.

A further unsatisfactory feature of frames is their use of default fillers. One reason for having defaults seems to be that since frames are structural templates the slots are available, so they might as well be filled with something. A default is a simple direct way of representing a particular expectation, but it is clearly inadequate for representing a range of possible frame-

types, or restrictions which any particular instance of a frame must satisfy in order to fill the slot. Given that some more sophisticated kind of representation is needed for this more complex information, it is questionable how much is added by using specific defaults. Minsky's claims that defaults are useful for by-passing logic and making generalisations have yet to be substantiated. Reasoning with defaults is a tricky business because of their peculiar status. They may be suggestive but in particular cases no firm conclusions can be drawn because the defaults may be wrong.

The main motivation for defaults is to explain the apparent richness of immediate awareness without appealing to parallel processing at high levels. There is no need, however, to suppose that decisions have already been taken about specific details when we first perceive a scene. Much of the detail may only be apparently present, owing to the peculiar properties of introspection. When we examine real objects such as a television picture we can assume the picture does not change simultaneously with our attention, so if we examine one part of it in detail we can assume that all those details were then even when we were not looking at that part. We have no such guarantee for introspection, so it may well be that people use a kind of "demand processing" whereby slots are filled only when their values are needed. If this process is smooth, rapid and unconscious it might well appear to naive introspection that the fillers were

there all the time. This line of argument has its own problems because decisions about how to fill one slot normally involve decisions about filling other slots, so that slots cannot be filled one at a time when needed. However, demand processing seems like a good alternative to defaults, if one wants to explain how the apparent richness of awareness could be compatible with relatively slow serial processing.

CHAPTER 7

A SYSTEM WHICH USES RELAXATION TO COORDINATE NETWORK GROWING RULES.

If one accepts the view that perception consists in using stored rules to grow a network of instances from the low-level data, then two of the major issues which arise are:

1. How is it possible to notice the occurrence of subsets of instances which satisfy the left hand sides of rules, without extensive searching?

2. When the low-level data or the rules are dubious, how can relaxation be used to find the best consistent interpretation?

A system called SETTLE has been implemented which incorporates answers to both these questions. SETTLE is described in detail in the rest of this chapter, and is illustrated using the domain of family relationships.

7.1: Overview of the SETTLE system.

SETTLE provides a set of facilities which are designed to make it easy to write programs of a particu-

lar kind. The aim of the system is to allow the user to concentrate on defining the schemas needed for a particular domain, and the inference rules which apply to combinations of instances of the schemas. The business of noticing when rules apply, setting up the relevant constraints between hypotheses, and achieving a consistent network of instances is handled by the system.

The term Schema will be reserved for stored knowledge about a particular type of entity and the term Instance will be used to refer to a representation in current awareness of an entity of that type. Schemas are thought of as far more like grammars than like instances. A Schema specifies a number of roles or slots which have associated restrictions on individual fillers, or on the relationships which should hold between the fillers of different slots. Schemas do not, at present, contain procedural information about how to search for fillers of slots. It is hard to use knowledge gained at run-time about properties of the thing that should be in a slot, to constrain the search for candidate fillers. This problem has been temporarily ignored.

Instances and the connections between them are created by the action parts of rules. An action is performed when the pattern specified on the left hand side of a rule matches a subset of the existing instance-network. For example, a rule might say that if a person A has a spouse B, and A also has a child C, then the

"child" slot of B should be filled by C (only conventional families are allowed!). Once this rule has been entered in the person schema, the system ensures that it is invoked whenever it is appropriate. In this respect rules resemble Planner antecedent theorems (Hewitt 1972).

Each instance and each filling of a slot is a hypothesis. It has an associated supposition value and is bound by constraints. For example, when the rule described above is invoked, the action part not only creates the hypotheses that B has C as a child, but also sets up a constraint so that relaxation will ensure that this hypothesis is accepted if the hypotheses which matched the left hand side of the rule (called its key) are accepted. The use of relaxation means that instances and connections can be added to the network even though they are not definitely correct. If costs dependent on the probabilities of tentative hypotheses are associated with their rejection, then relaxation will find the most probable combination of instances and connections.

7.2: Schemas.

The person-schema which will be used to illustrate the SETTLE system is created by the command:

```
MAKESCHEMA ("PERSON", [SPOUSE PARENT 2 CHILD 0 SEX SURNAME]);
```

The words following "Person" are the names of the slots. Slots are assumed to be limited to one filler unless they are followed by a number indicating a higher limit (0 is used to mean no limit). When an instance of the schema is required, a one-dimensional array (a strip) is created. The function "makeschema" assigns strip-accessing functions to the slot names so that they can be used to access the slots of instances.

7.3: Slots.

A slot is not simply a location for holding a pointer to some other instance. It is a complex data-structure with the following components:

1. A pointer back to the part of the schema which contains information about the slot, such as the rules involving its fillers.

2. A list of demons which are waiting for new slot fillers (see below).

3. A list of hypotheses about potential slot fillers.

7.4: Bonds

Connections between instances involve each instance filling a slot in the other. Slot fillings are hypotheses which are bound by constraints and have their supposition values manipulated by relaxation, so they need to be represented by data-structures rather than simply being pointers. The system uses structures called "bonds" to implement slot fillers. As figure 7.1 shows, a bond has pointers to the two instances which it joins, and also an associated record, called a supposition-node, which contains the supposition value of the bond and the constraints involving it. The relevant slots in the two connected instances have pointers to the bond in their lists of candidate fillers.

In the domain for which the system was designed, when A fills a particular slot in B, it generally follows that B must fill a known slot in A, and so it is unnecessary to have separate hypotheses about the two reciprocal fillings. This is the reason for using a single two-way bond rather than two one-way ones. When a slot is filled with something other than an instance, or when the inverse slot is unknown, a single slot filling can be represented by simply omitting one of the pointers to the bond.

The way in which slot fillings are implemented is expensive and cumbersome, but the complexity seems to be a necessary consequence of the need to refer to fillings

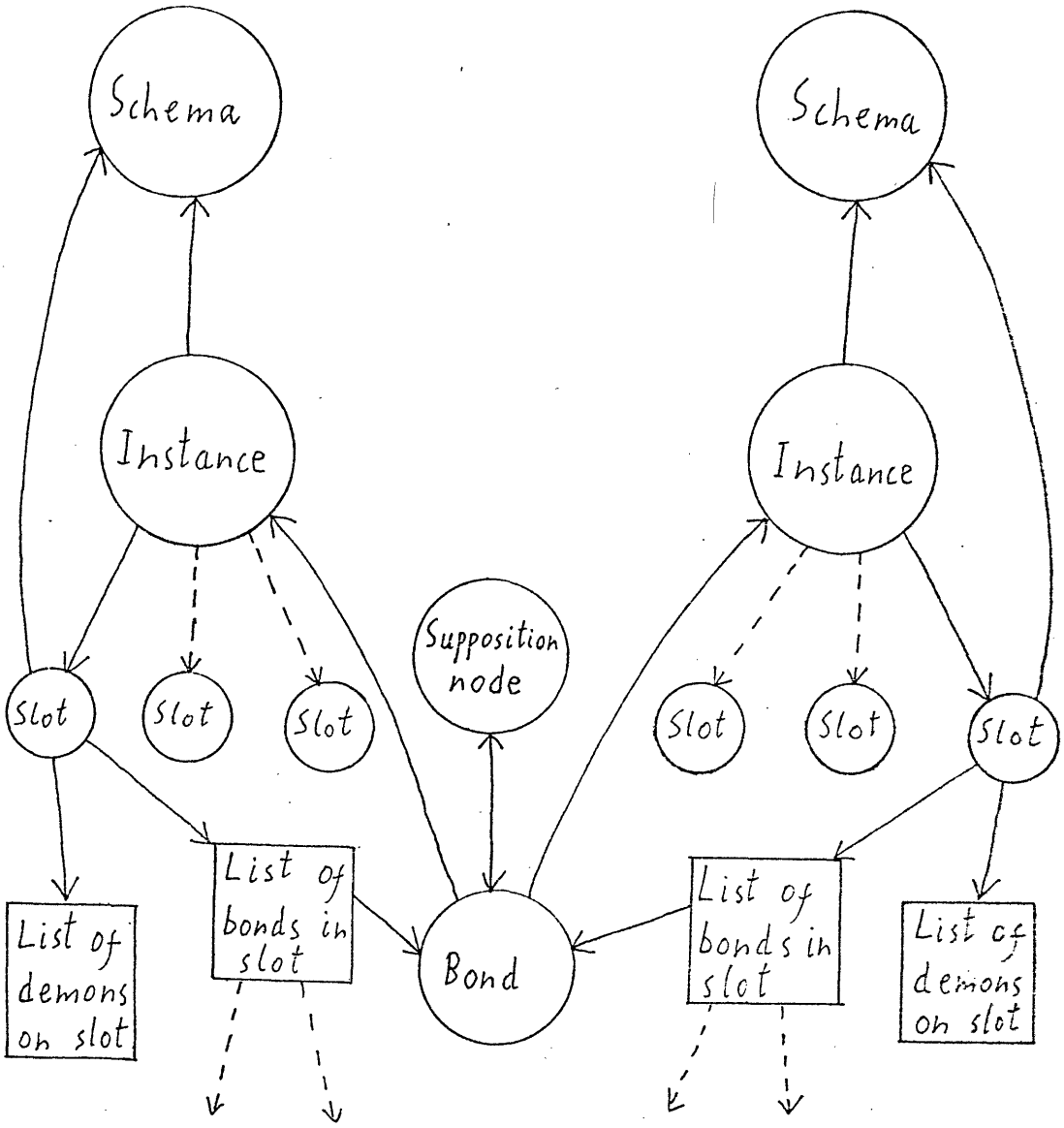


FIGURE 7.1: Showing some of the data-structures used in the SETTLE system. The use of demons, and explicit constraints means that the connections between instances are considerably more complex than simple pointers from a field in one instance to the other instance.

as hypotheses.

7.5: Specifying Rules.

Once a schema has been created, rules can be added to it. These determine how instances of the schema can combine with other instances. Rules typically specify that a particular subset of instances is illegal, or that it implies some other instance or bond between instances. Rules are entered in a list format that is convenient for typing, but they are compiled into records containing a key and an action, before they are stored in schemas.

The left hand side of a rule contains a list of bond specifications and a list of other conditions which must be satisfied by the matching instances. For example, the rule that a person's child is also their spouse's has the following form:

```
[3 PERSON [ A CHILD C] [ A SPOUSE B]  ] ==> [inferbond
([B CHILD C])];
```

The square brackets are list brackets. The first two items of the LHS are the number of the rule and the schema to which it should be added. The remaining items are bond specifications. These declare variables which are bound to instances during a match. There are conventions that if a variable is repeated it must be matched to the

same instances, and that different variables must be matched to different instances. The specification [A CHILD C] should be read "A has child C" .

There are several elaborations to the basic way of specifying a bond. [A BROTHER B SISTER] is equivalent to the two specifications [A BROTHER B], [B SISTER A]. Such a specification may be useful when slots do not have unique inverses. [A SEX = MALE] is used to indicate that the filler of A's sex-slot should be the word "male" rather than an instance. [A SEX /= MALE] means that an instance will only match A if it does not have male in the sex-slot. Any part of a specification can be preceded by the "&" sign which causes the value of the following word to be used. For example, if the value of the variable SLOTNAME is "CHILD" and the value of the variable X is C, then [A & SLOTNAME & X] is equivalent to [A CHILD C]. This facility is useful when rules are being generated by a function rather than being typed in directly.

Finally, items starting with a "." on the LHS of a rule specification are conditions which must be satisfied. For example, to ensure that children are younger than their parents, the following rule could be added:

```
[ 4 PERSON [ A CHILD C] [ A AGE X] [C AGE Y] [.LESS  X
Y]] ==> [CONTRADICTION ()];
```

Condition specifications consist of a dot followed by a function or function name followed by arguments (as in lisp). The match fails unless the function returns true. Variations in the way in which conditions can be specified are explained in comments in the code in Appendix 6.

7.6: Rule invocation.

This section starts by describing a method of rule invocation which assumes that all the instances and bonds are present before any matching starts, and then shows how the method can be extended to the harder problem of noticing when a rule becomes applicable through the addition of a new instance or bond.

The LHS of a rule is compiled into a key, which is a data structure that is designed to facilitate rapid matching. A key is a rooted, directed graph of keynodes, each of which gets bound to a different instance during matching. The basic strategy is to bind the keynodes one at a time and to generate candidate bindings for new keynodes by looking in the slots of instances which are already bound. For example, if the bonds specified are:

[A CHILD C] and [A SPOUSE B]

then once A has been bound, the fillers of its child and spouse slots are the candidates for C and B respectively.

Only perfect matches to the key are required, which means that the keynodes can be bound in a predetermined order, and a match can fail as soon as it reaches a keynode for which there are no suitable instances. The supposition values of instances and bonds are ignored during matching, so several alternative bindings may be possible for a keynode and a depth-first search is used to find all the ways in which a key is instantiated in the instance network.

The candidate bindings for the first keynode are all the instances of the schema with which the rule is associated. Bond specifications are used to give each keynode, except the first, a pointer to an earlier keynode and an associated slot name. It uses these to generate candidate instances from the instance bound to the earlier keynode. The candidates are not always feasible, because they may already have been bound to an earlier keynode, or they may violate one of the conditions specified later in the LHS. Each such condition is associated with a particular keynode and, in order to prune the search, it is tested as soon as that keynode is bound. Conditions which take as arguments the instances bound to several different keynodes are associated with the last one to be bound.

If more bonds are specified than there are non-root keynodes, then the key will be a lattice or graph rather than a tree. In this case the system selects a subset of

the bond specifications which form a rooted tree, and uses these for generating candidate bindings, as above. The remaining, extra bond specifications are handled like the conditions. They are associated with the last of their keynodes to be bound and are tested before further bindings.

If the bond specifications do not contain a rooted, directed tree, then there may be no economical way of generating candidate bindings for some keynodes, so keys of this form are not allowed and any such rule specification is rejected by the system.

So far the description of rule invocation has ignored the fact that the instance network grows, so that a match which initially fails may later succeed. When a new bond is added to the network the system needs to have some way of deciding which keys may now match. It would be possible to index each key under all the types of bond involved. However, if a potentially relevant key was found in this way, then a fresh match would have to start at the new bond and so the simplicity and speed gained by being able to match the keynodes in a predetermined order would have to be sacrificed. Also, if matching started afresh with each new bond, there would be a great deal of duplication of the work done during earlier, failed matches.

An alternative strategy, which again depends on the fact that only perfect matches are required, is to set up

a demon whenever a match fails on account of a missing bond. The demon "sits" on the slot in which the bond will go, and so no searching is required to activate it. The demon keeps a list of the instances to which keynodes were bound in the earlier match before it failed. So when a new bond is put in the slot, the keynodes can be rebound and the match continued using the new bond, without any duplication of previous work. The demon is, in effect, a suspended partial match.

Since any slot may gain another filler after the first attempt at matching a key, it is not sufficient only to leave demons on slots containing no suitable filler. Every slot which is used to generate candidate instances for a keynode needs to be given a demon. This leads to a lot of demons and so implementation tricks (explained in comments in the code), are used both to keep down the number of demons and to make them compact.

7.7: Jobs

It would be possible, when a key matched, to perform the corresponding action immediately. However, actions often create new bonds which cause other keys to match or the same key to match in a different way, so actions would be called within other actions. If this embedding occurred in any depth, it would cause inconveniently deep calling sequences. Like several other programs (Sloman

1977, Paul 1977) the system avoids this problem by using a job queue. Whenever a new bond is added to the instance network, all the resulting matches are found. For each match, a job-record is created which contains the bindings of the variables in the key, the bonds matched by the key, and the action part of the rule. The job is added to the queue. When the job is run it restores the bindings of the variables used in the key, so that the code for the action can use the variables to refer to the same instances. The matching bonds are stored because actions typically infer some other bond from them and so, for the purposes of relaxation, they need to set up a constraint between the matching bonds and the inferred one.

There is another and more important reason for using jobs. Any system which is based on forward chaining (antecedent theorems) and also keeps alternative possibilities, is liable to explode. Some method of limiting the forward chaining is needed, and the SETTLE system uses relaxation coupled with the assumption that an action is only relevant if all the bonds which matched the key have high supposition values. For example, if a rule involves inferring a new bond from the old ones matching the key, then the action will set up a constraint which requires the new bond to be true if all the old ones are. This constraint has no effect if any of the old bonds are rejected, so there is no point even making the constraint unless all the old bonds have high supposition values.

It would be possible, but not easy, to take supposition values into account during matching. When a match failed because there was no suitable bond with a high enough supposition value, a demon would be set up waiting for such a bond. Unfortunately, by the time a suitable bond arrived, the supposition value of some bond used earlier in the suspended partial match might have fallen. So whenever high supposition values fell, it would be necessary to garbage-collect all the demons which were waiting to complete the partial matches which were no longer valid. A further difficulty would be that oscillations in the supposition value of a bond would cause the same match to be rediscovered several times.

The system ignores supposition values when finding matches but takes them into account in deciding whether or not to run a job. It examines the first job on the queue to ensure that all the bonds which matched its key have high supposition values. If they do, the job is run, but if any are low, the job is removed from the queue and hung on the bond responsible. Whenever the supposition value of a bond rises to a high enough value, a check is made for hanging jobs, which are then put back on the job queue. The effect of this procedure is that jobs are only actually run when all the bonds matching their key have high supposition values, so that many ineffective constraints and unsupported bonds and instances are never added to the instance network. Provided all the scores are negative, hanging jobs cannot lead to

the best global interpretation being overlooked. Running a job can only make matters worse for the set of bonds and instances currently favoured by relaxation. Any set of hypotheses which is rejected by relaxation would still be rejected after running hanging jobs which added further constraints or costs to that set.

7.8: An example of the SETTLE system in action.

Although SETTLE is intended as a way of applying relaxation to vision tasks, the domain of family relationships has been chosen to illustrate, in detail, how the system works. The reason for the choice is that people are very familiar with family relationships, so there should be no confusions about the domain to add to the difficulties of understanding the system. The example is not intended as a model of how people handle information about family relationships.

7.8.1: Specifying rules about family relationships.

Only one schema is used in this example. Figure 7.2 shows how it is defined, and how the system is told about rules to be applied to instances of the schema. When this code is compiled, the structures made from the the rule specifications are associated with the relevant parts of the person schema. For example, rule 1 is kept

```
MAKESCHEMA("PERSON",[SPOUSE PARENT 2 CHILD 0 SEX SURNAME]);

COMMENT SOME SLOTS HAVE KNOWN INVERSES;
SPOUSE<->SPOUSE;PARENT<->CHILD;

COMMENT  A PERSONS PARENTS ARE MARRIED;
[1 PERSON [X PARENT P1][X PARENT P2] ]
==> [INFER([P1 SPOUSE P2])];

COMMENT  A PERSONS CHILDREN ARE ALSO HIS SPOUSES CHILDREN;
[2 PERSON [P CHILD C][P SPOUSE Q] ]
==> [INFER([Q CHILD C])];

COMMENT  A PERSONS SPOUSE IS OF THE OPPOSITE SEX;
[3 PERSON [P SPOUSE Q][P SEX S] ]
==>[IF S="MALE" THEN INFER([Q SEX =FEMALE])
    ELSEIF S="FEMALE" THEN INFER([Q SEX =MALE])
    ELSE INSTPR(P);PR(' HAS FUNNY SEX !);PR(S);
    CLOSE];

COMMENT  SPOUSES HAVE THE SAME NAME;
[4 PERSON [P SPOUSE Q][P SURNAME N] ]
==> [INFER([Q SURNAME N])];

COMMENT  MALE CHILDREN HAVE THEIR PARENTS NAME;
[5 PERSON [P CHILD C][C SEX =MALE] ]
==> [SAMEFILLER(P,SURNAME,C,SURNAME)];

COMMENT  UNMARRIED FEMALE CHILDREN HAVE THEIR PARENTS NAME;
[6 PERSON [C SEX =FEMALE][C SPOUSE =NONE][C PARENT P] ]
==> [SAMEFILLER(C,SURNAME,P,SURNAME)];

COMMENT  FEMALE CHILDREN WITH THEIR PARENTS NAME ARE
PROBABLY UNMARRIED;
[7 PERSON [C SEX =FEMALE][C PARENT P][C SURNAME N]
          [P SURNAME N] ]
==> [SOFTINFER([C SPOUSE =NONE],0.7)];

COMMENT  MARRIED CHILDREN WHO HAVE THEIR
PARENTS NAME ARE PROBABLY MALE;
[8 PERSON [C SPOUSE /=NONE][C PARENT P][C SURNAME N]
          [P SURNAME N] ]
==> [SOFTINFER([C SEX =MALE],0.7)];
```

FIGURE 7.2: The person schema and some rules about family relationships.

in the part of the schema which stores information relevant to the PARENT slots of the instances. When an instance has its parent slot filled, the key of rule 1 will start matching by binding the keynode for X to the instance and the keynode for P to the filler.

There are several features of figure 7.2 which have not, so far, been explained. Rule 3 demonstrates the convenience of being able to use arbitrary POP-2 code to specify the action part of the rule. It allows error messages and tracing to be included, as well as allowing arbitrarily complex actions.

Rules 5 and 6 show the use of the SAMEFILLER function. It is often possible to infer that two slots must have the same filler, without knowing what it is. This knowledge could be captured in two rules each of which required a filler for one of the slots as part of its condition, and then inferred that the filler also filled the other slot as its action. However, it is more economical to have a single rule with a simpler condition which sets up special demons on both slots, so that any fillers of one are inferred to fill the other, subject to the continued truth of the conditions which caused the demons to be set up.

Rules 6, 7, and 8 show how the filler "NONE" can be used to represent the fact that there is no filler for a slot of a type which can have at most one filler. For

such a slot, the system automatically keeps a constraint, which it modifies when new candidate fillers are found, to prevent more than one filler being accepted as true. So by supporting the filler "NONE", real fillers can be kept out. Some kind of mechanism like this is needed, since the known absence of any filler cannot be represented simply by the absence of fillers from the slot. However, it may be that using "NONE" fillers is just an unprincipled hack. The method cannot be used when slots which can potentially have any number of fillers, are discovered to have none (as opposed to not being discovered to have any). I suspect that this apparently minor difficulty is the tip of an iceberg. Sometimes, the implication of a rule involves quantifiers rather than being about particular fillers. These are hard to handle in the current SETTLE system. "SAMEFILLER" demons and "NONE" fillers cope with the two cases that have arisen so far, but a more general mechanism for handling quantifiers would be better.

Rules 7 and 8 show how non-binding inferences can be handled. The function SOFTINFER causes a constraint to be set up, so that if the conditions of the rule are accepted, but the implication is rejected, then a penalty of 0.7 is paid. (See section 4.6). This particular number is given meaning by its magnitude relative to other costs which determine the trade-offs made in deciding which hypotheses to accept and which to reject.

7.8.2: Interpreting claims about specific people.

Figure 7.3a shows one way of inputting data about a particular set of people and their relationships. The claims give preferences to particular bonds. Their strength, 1, means that a claim can override one soft inference, but not two, since 0.7 + 0.7 > 1. The instances and candidate bonds are shown in figure 7.3b. This also indicates the way in which bonds generated by inference rules depend on other bonds. The result of 29 rounds of relaxation is shown in figure 7.4b. It is the best consistent set of beliefs given the claims and inference rules. Figure 7.4a shows the job statistics as relaxation proceeds. In this case relaxation is automatically terminated after 15 clear rounds in which no jobs are made or roused. Notice how three jobs made on the second round of relaxation do not get run until the eighth round, when the bonds matching the rule keys have all attained high supposition values. The way the supposition values change during relaxation is shown in figure 7.5.

7.8.3: The effect of more, incompatible claims.

Figure 7.6 shows some more claims and the network of candidate bonds and instances which is caused by these extra claims and by the inference rules which they

trigger off. Some previously accepted bonds now have to be rejected in order to reach the best consistent set of beliefs in the light of the new data. Figure 7.7b shows this optimal set, which is discovered by the program. Notice that one of the original claims (about the sex of person2) has been rejected. Figure 7.7a again shows the job statistics as relaxation proceeds until there are 15 clear rounds. The way the values change during this phase of relaxation is shown in figure 7.8.

```
COMMENT THIS IS HOW PEOPLE ARE MADE;
2 <* MAKEINST(PERSON) *>;

CLAIM([PERSON1 CHILD PERSON2],1);

CLAIM([PERSON1 SURNAME =JONES],1);

CLAIM([PERSON2 SURNAME =JONES],1);

CLAIM([PERSON2 SEX =FEMALE],1);
```

FIGURE 7.3a: Some claims about people.

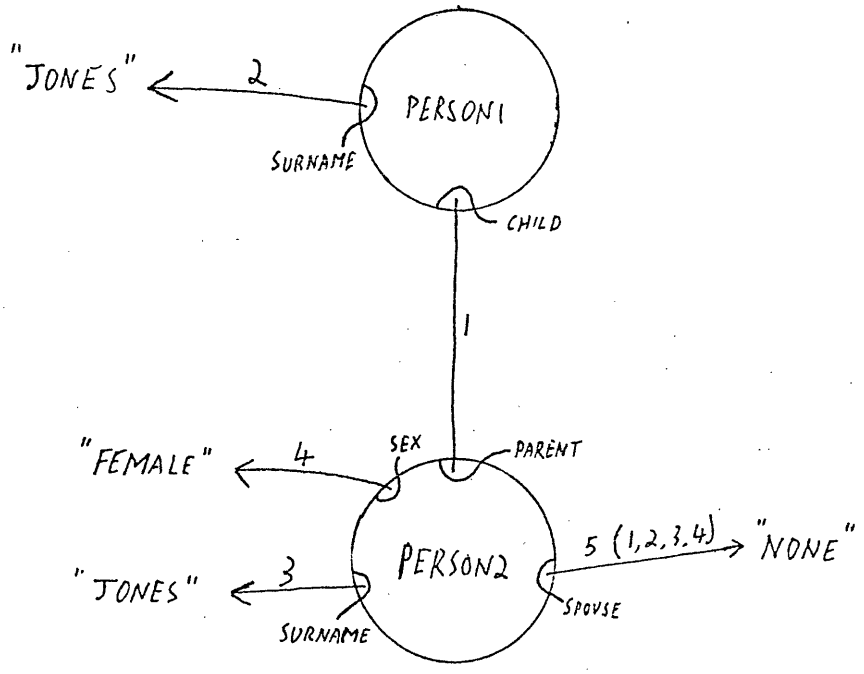


FIGURE 7.3b: The candidate bonds created by the claims and the inferences they trigger off. Bonds are given numbers, and implications between bonds are indicated by following a bond number with the numbers of a conjunction of bonds that imply it.

```
!15.settle$

ROUSED RUN STORED MADE
   0     0    1     0
   0     0    0     0
   1     1    0     3
   0     0    3     0
   0     0    0     0
   0     0    0     0
   3     3    0     2
   0     2    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
   0     0    0     0
!
```

FIGURE 7.4a: Job statistics. Each row corresponds to a round of relaxation. Jobs are roused when a bond which they depend on reaches a high enough supposition value (0.7 in this case). Jobs are stored if any bond they depend on has a lower value than this.

```
!.showtrue$
 PERSON2
 SPOUSE [NONE ] PARENT [ PERSON1 ] CHILD [] SEX [FEMALE ]
 SURNAME [JONES ]

 PERSON1
 SPOUSE [] PARENT [] CHILD [ PERSON2 ] SEX []
 SURNAME [JONES ]
```

FIGURE 7.4b: The optimal interpretation of the claims, reached by relaxation (see figure 7.5).

```
!20.runmore}
 50  50  60  60  60  60
 57  60  68  71  71  71
 69  76  76  82  84  84  50   0
 83  90  88  94  98  98  50   0
 96  99  97  99  99  99  56   0
 99  99  99  99  99  99  67   0
 99  99  99  99  99  99  80   0
 99  99  99  99  99  99  91   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
 99  99  99  99  99  99  99   0
        P1  P1  P2  P2  P2
 P1  P2  P2  JO  JO  FE  NO  CN
!
```

FIGURE 7.5: Showing how the supposition values change during relaxation after the claims. The "headings" are at the bottom because not all the hypotheses are known in advance.

The column headings in this figure are rather cryptic. Headings with just one row refer to a person instance (e.g. P1), or to the extra hypothesis set up by a soft inference rule (rule 7 or 8 in figure 7.2). Constraints force such an extra hypothesis to be accepted if the rule is broken, and a cost is then paid. Unfortunately, the relevant inference cannot be identified from the heading. Headings on two rows refer to bonds, either between two instances, or between an instance and a word which is abbreviated to its first two letters.

```
2 <* MAKEINST(PERSON) *>;

CLAIM([PERSON2 SPOUSE PERSON3],1);
CLAIM([PERSON2 CHILD PERSON4],1);
CLAIM([PERSON3 CHILD PERSON4],1);
CLAIM([PERSON3 SEX =FEMALE],1);
```

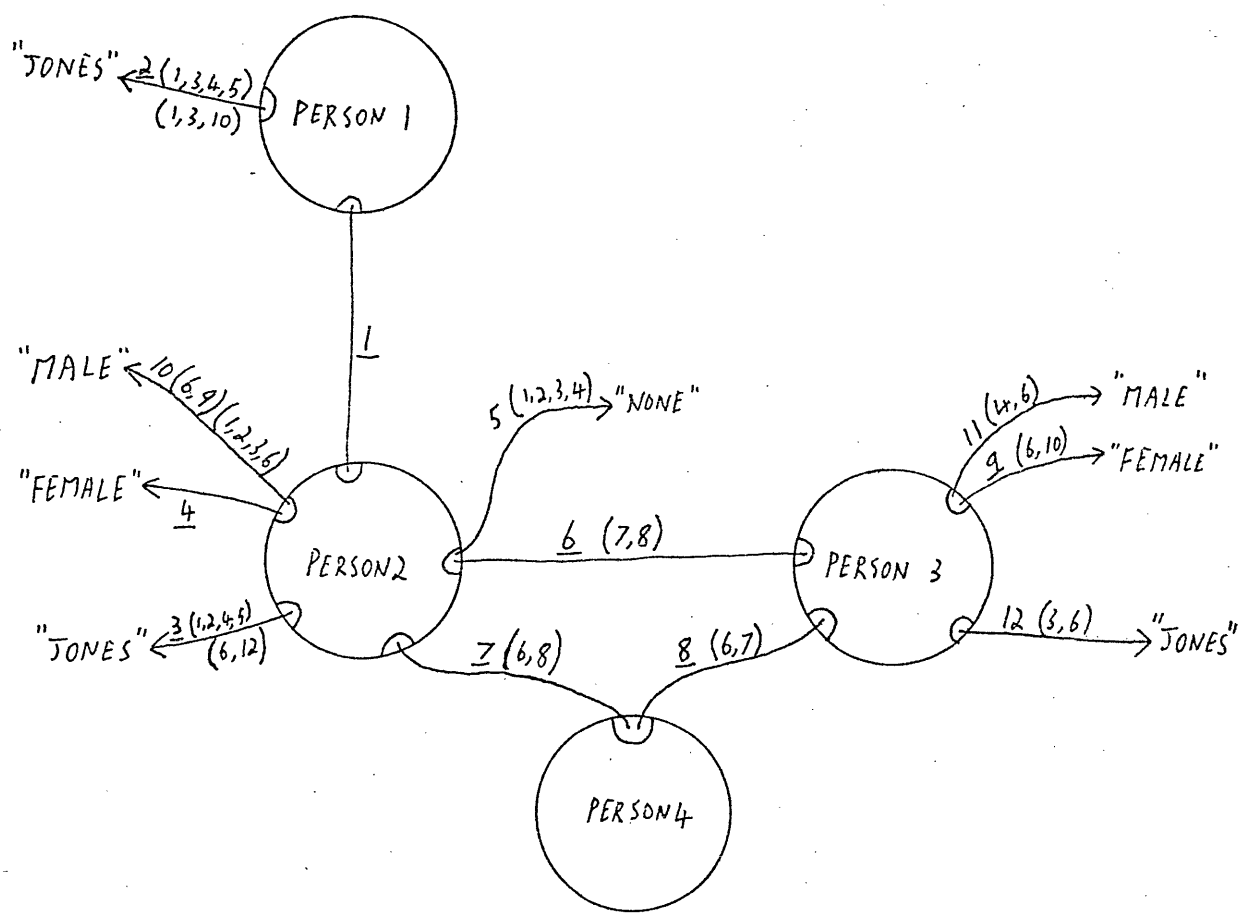


FIGURE 7.6: Some more claims, and the resulting network of candidate bonds. The slot names have been ommitted, but should be obvious from the fillers. The bonds are numbered in order of creation. The numbers in brackets after each bond number are the sets of other bonds which imply it. Some of these implications may only be weak ones, derived from rules 7 and 8.

Bonds which were entered as claims are underlined. The claims may be rejected (e.g. bond 4). Competition between bonds for slots is not shown.

```
!15.settle;

ROUSED RUN STORED MADE
   0    0     9    0
   1    1     0    0
   2    2     0    0
   0    0     0    0
   6    6     0    5
   0    0     5    0
   0    0     0    0
   0    0     0    0
   0    0     0    0
   1    1     0    0
   0    0     0    0
   0    0     0    0
   2    2     0    2
   0    2     0    0
   0    0     0    0
   0    0     0    0
   0    0     0    0
   0    0     0    0
   0    0     0    0
```

FIGURE 7.7a: Job statistics for the relaxation following the claims in figure 7.5. The last ten rows of zeros are not shown.

```
!.showtrue;
PERSON4
SPOUSE [] PARENT [ PERSON3  PERSON2 ] CHILD [] SEX []
SURNAME []

PERSON3
SPOUSE [ PERSON2 ] PARENT [] CHILD [ PERSON4 ]
SEX [FEMALE ] SURNAME [JONES ]

PERSON2
SPOUSE [ PERSON3 ] PARENT [ PERSON1 ] CHILD [ PERSON4 ]
SEX [MALE ] SURNAME [JONES ]

PERSON1
SPOUSE [] PARENT [] CHILD [ PERSON2 ] SEX []
SURNAME [JONES ]
```

FIGURE 7.7b: The best interpretation of all the claims, found after the 29 rounds of relaxation shown above. Some beliefs in figure 7.4b have been rejected.

```
!20.runmore;
 99  99  99  99  99  99  83   0  50  50  42  60  60  60
 99  99  99  99  99  99  70   0  57  57  39  71  68  71
 99  99  99  99  99  99  66   0  69  69  43  82  74  82
 99  99  99  99  99  99  68   0  83  83  60  86  77  94
 99  99  99  99  99  99  66   0  94  91  70  96  87  99  50  50  50   0
 99  99  99  99  94  80  59   0  99  98  58  99  95  88  42  38  55   0
 99  99  99  99  97  70  52   0  99  99  74  91  89  82  33  22  59   0
 99  99  99  99  95  64  43   0  99  99  62  94  93  82  41  18  64   0
 99  99  99  99  99  64  39   0  99  99  72  92  92  91  47  18  67   0
 99  99  99  99  99  59  37   0  99  99  74  95  96  98  56  19  70   0
 99  99  99  99  99  53  36   0  99  99  77  98  99  99  63  16  73   0
 99  99  99  99  99  46  34   0  99  99  84  99  99  99  68  12  77   0
 99  99  99  99  99  40  28   0  99  99  85  99  99  99  73  10  81   0
 99  99  99  99  99  35  22   0  99  99  89  99  99  99  77   9  85   0
 99  99  99  99  99  30  17   0  99  99  93  99  99  99  81   8  89   0
 99  99  99  99  99  26  12   0  99  99  95  99  99  99  85   8  93   0
 99  99  99  99  99  22   9   0  99  99  98  99  99  99  89   8  97   0
 99  99  99  99  99  19   6   0  99  99  99  99  99  99  91   7  99   0
 99  99  99  99  99  17   4   0  99  99  99  99  99  99  94   7  99   0
 99  99  99  99  99  15   3   0  99  99  99  99  99  99  95   6  99   0
      P1  P1  P2  P2  P2                  P2  P2  P3  P3  P2  P3  P3
 P1  P2  P2  JO  JO  FE  NO  CN  P3  P4  P3  P4  P4  FE  MA  MA  JO  CN
!
```

FIGURE 7.8: Showing how the supposition values change during the first 20 rounds of relaxation after the extra claims in figure 7.6.

CHAPTER 8

SUMMARY

This chapter summarises the presuppositions behind the relaxation approach. It then mentions the main inadequacies in the treatment given to relaxation, including the failure to relate it to human vision. Finally there is a brief summary of what has actually been achieved.

8.1: Presuppositions of the relaxation approach

L.P. relaxation is only relevant to vision if the following claims are correct:

1. During the process of building the internal representation of a scene, tentative hypotheses must be formulated and selections must be made from among rival hypotheses.

2. A visual system cannot arrive at the same kinds of interpretation as people do, if inconsistency is its only way of ruling out interpretations. It must have a way of arriving at good interpretations and avoiding poor ones.

3. A sensible way of resolving complex and unforeseeable conflicts between sets of hypotheses of different kinds, is to use numerical scores for the constituent hypotheses of a global interpretation and to maximize the sum of these scores.

The first two claims are defended in Chapter 1, and though they may be false, they are not unduly speculative. The third claim is the one which many artificial intelligence researchers regard with suspicion. Some workers (e.g. Paul 1977) regard the avoidance of real-numbers for evaluating hypotheses as a positive virtue, and have demonstrated that, for some vision problems, explicit numerical scores are unnecessary. If it is accepted that numerical scores are an undesirable last resort, then their use can only be defended by showing that no other method will work. This would be very difficult, and has not been attempted. Instead, the prejudice against numerical scores has been attacked. It has been argued that the properties of real numbers are particularly appropriate for resolving conflicts (section 1.7.2); that the past misuse of numbers is irrelevant (section 1.8); and that the choice of numerical values need not be arbitrary (section 1.7.1).

However, it has not been established that the resolution of complex conflicts between hypotheses of different kinds is a necessary part of normal vision, or that the interpretations people notice can be defined in

terms of the probabilities of their constituent hypotheses. So numerical scores, and hence relaxation, may be simply irrelevant to vision.

8.2: The choice of numerical scores

In section 1.7.1 it was argued that probabiliies could provide a systematic basis for the choice of numerical scores. Woods (1976), has shown that this idea can be applied in speech perception, but the programs in this thesis use scores which were chosen so as to give sensible interpretations, rather than being based on probabilities. More work is required to show how scores can be based on probabilities without running into problems caused by combining non-independent probabilities.

8.3 Details of the relaxation operator

A lot of effort has gone into analysing and improving the basic relaxation operator, but many problems remain unsolved:

1. How can relaxation be made to select one of a pair of equally good, rival global interpretations?

2. What should be done about non-integer optima if they cannot be removed by a better numerical formulation of the logical constraints?

3. How can the time to reach the equilibrium state be decreased?

4. How can the system decide when it is sufficiently close to the equilibrium state?

The coefficient K whose qualitative effects are discussed in section 4.4 can help with all these problems. Its quantitative effects need to be investigated both empirically and analytically.

8.4: The SETTLE system

The most advanced and promising use of relaxation is in the SETTLE system described in Chapter 7, but this system still needs a lot of development. An attempt has been made to use it for interpreting Popeye pictures (like figure 1.1b). This application is not described here since several major problems have been encountered and have not yet been resolved. Until the SETTLE system has been successfully applied to a vision task which requires its skill at handling messy data and dubious inferences, it will be hard to asses its value.

8.5: Relaxation and human vision

There are two rather different sets of considerations which are relevant when developing a theory about

the mechanism of human vision. On the one hand, a mechanism must be clearly defined and shown to be adequate for its postulated role. This is the main purpose of most Artificial Intelligence programs and the only aim of this thesis. On the other hand, evidence must be found to show that people use the mechanism. No attempt has been made to find evidence for relaxation in human vision. An obvious first step would be to show that for a task such as the interpretation of line drawings of polyhedra, the interpretations which people perceive can be distinguished from other consistent interpretations by giving them scores on the basis of their constituent hypotheses. It would also be interesting to try to analyse in detail our perception of pictures like the Necker cube or the Penrose triangle. However, these projects would inevitably involve many other difficult issues, some of which are outlined below.

8.5.1: The temporal structure of vision

People move their eyes, so their visual input consists of a sequence of retinal images. For each new fixation, low-level representations of what the retinal image contains have to be re-computed. (These low-level representations will be called the primal sketch, by affinity with Marr's primal sketch). However, the world appears stable as we move our eyes or move around, so we presumably have some representations which do not change

with our retinal images (Hochberg 1968). These will be called the cognitive map. Given this distinction between types of representation, there are a number of possible roles for relaxation which have not been distinguished in the simple tasks to which it has been applied:

1. The creation of the primal sketch. This needs to be fast and there may not be time for L.P. relaxation unless it can be speeded up. Also, it may not be necessary to decide between alternatives at this level (see section 1.3.2).

2. The discovery in the primal sketch of objects to be represented in the cognitive map. This stage of perception is the one which the puppet-finding program is intended to model.

3. The construction of a consistent cognitive map. The evidence provided by one retinal image may contradict representations based on an earlier image. Relaxation could be used to resolve such conflicts.

A great deal of work needs to be done to clarify the various ways in which relaxation might be used in a visual system as complex at the human one.

8.6: What has been shown.

A relaxation method for selecting the best consistent set of hypotheses has been clearly defined. The method does not appear to suffer from a combinatorial explosion in time or space as the number of hypotheses increases. It can make effective use of parallel hardware, and is one of the first cleally defined ways of organising parallel interactions between conflicting and cooperating hypotheses so as to make a good "Gestalt" emerge.

It has been shown how to handle any logical constraint that can be expressed in the propositional calculus. The successful application of the method to the two simple tasks of puppet-finding and line-labelling has been demonstrated.

Several ways of changing the relaxation operator have been discussed and their effects have been investigated empirically. They have also been analysed theoretically using a hyperspace representation. The difficulties caused by non-integer vertices and equal rivals have been revealed.

It has been shown, using an extended version of the puppet task, that as well as selecting from among existing hypotheses, relaxation can be used to control which hypotheses are created. The application of the technique to the choice of numerical values for parameters has also

been discussed.

Finally, the SETTLE system has shown how relaxation can be used to control a data-driven system which grows a relational network by noticing when complex conditions become true and using forward chaining. This is a novel way of organising a search within a kind of production system.

APPENDIX 1

COMPUTING WHETHER CONVEX POLYGONS OVERLAP

This is not a formal proof. It is a construction to show how an unobvious fact follows from obvious ones.

Corresponding to each infinite straight line there are two borders. A border has an on-side (including the points in the line) and an off-side. The sides of a convex polygon are segments of infinite lines which can be assigned corresponding borders in such a way that the polygon contains all and only the points on the on-side of all the borders.

We want to show that if two convex polygons are disjoint (have no common points), then at least one border of one of them has the other polygon entirely on its off-side.

Let us say that a line separates two polygons if their interior points lie on opposite sides of it. For any pair of disjoint, convex polygons there are some separating lines (unproved but obvious). In particular, there is one separating line which cannot be rotated clockwise about any of its points without intersecting the interior of one polygon (see figure APP1). Similarly there is a most-anti-clockwise separating line. Call these two lines b and c, and their Point of intersection

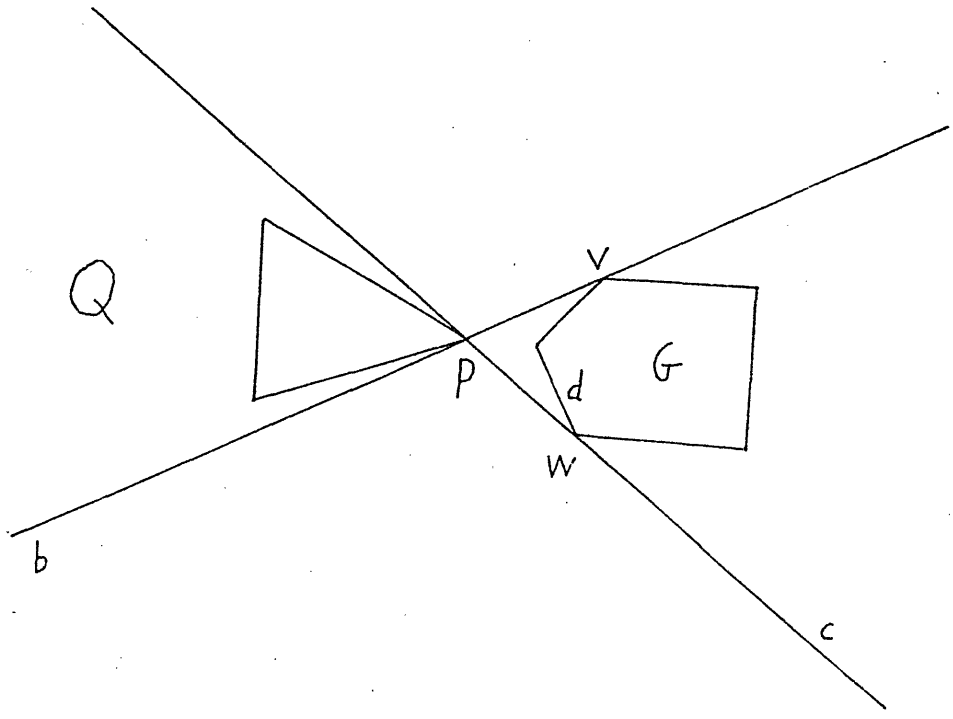


FIGURE App1: Showing the construction involving the most-clockwise separating line, b, and the most-anticlockwise, C.

P. Since b and c are separating lines, P cannot lie in the interior of either polygon and since the polygons are disjoint they cannot both have vertices at P. So at least one of them, call it G, must have P outside it. For P to be outside G it must be on the off-side of at least one of G's borders, call it d. Since d is a border of G, all the vertices of G are on its on-side. In particular, the vertices of G which lie on b and c must be on the on-side So, considering figure APP1, d must have P on its off-side and V and W on its on-side. Hence d must intersect b between P and V (or at V) and it must intersect c between P and W (or at W). Because d can only intersect the lines b and c once, it is obvious (though unproved) that the quadrant Q must lie entirely on the off-side of d, and hence so must the polygon within Q.

Note: The idea that one polygon must contain a separating border was suggested to me by Frank O'Gorman.

APPENDIX 2

USING PENUMBRAS TO AID LINE LABELLING

Waltz (1972) shows how it is possible to extend Huffman/Clowes labelling to line drawings in which some lines depict shadow edges. Waltz uses perfect line drawings and so he ignores the question of whether the grey-level data can provide information about the type of an edge as well as about it existence. Evidence which suggests the type of an edge, but which is not conclusive, is interesting because it is easily incorporated into a relaxation approach as a preference for a particular labelling.

Under some conditions of illumination there should be direct grey-level evidence suggesting that some edges are shadow edges. Figure APP2 shows the shadow cast by an object when there is a single source of illumination which is not infinitely small. The shadow edges have penumbras which diverge as the distance from the casting edge increases. For small sources this should be detectable as a fuzziness which increases linearly in the direction away from the casting edge, provided this edge is straight and the shadow lies on a flat surface. An example of the usefulness of such information is seen at junctions J and K in Figure APP2. The degree of fuzziness caused by the penumbra supports the interpretation

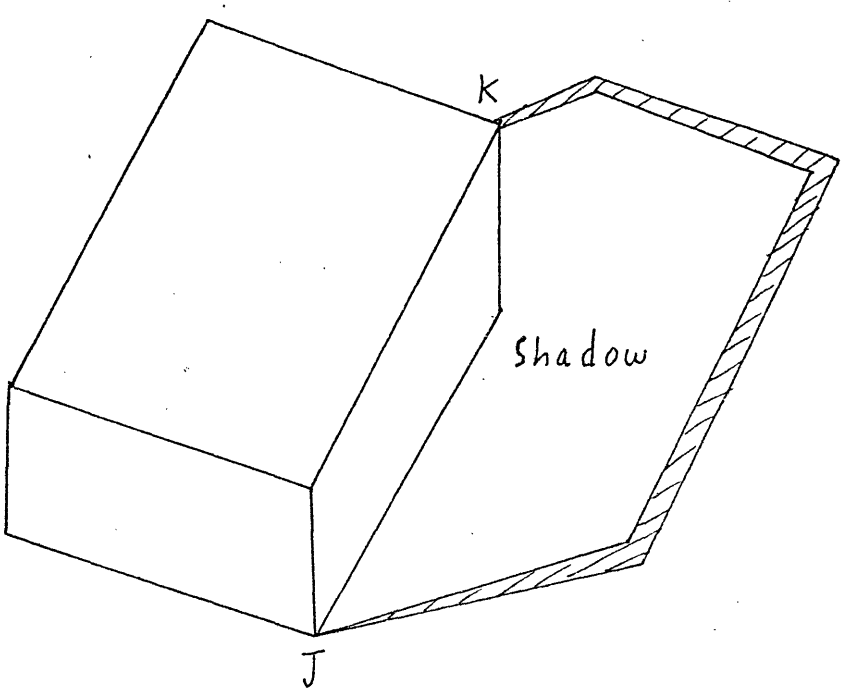


FIGURE App2: A cuboid casting a shadow. The width of the lines depicting shadow edges indicates the width of the penumbras caused by a light source of finite magnitude. Notice that the fuzziness of the shadow edge at K suggests an accidental alignment of vertex and shadow edge.

of junction J as involving an attached shadow, but suggests an accidental alignment of vertex and shadow at K.

It is not clear whether human perception makes use of the way in which the penumbras diverge along shadow edges.

APPENDIX 3

CODE FOR THE PUPPET-FINDING PROGRAM

A number of basic functions and macros are used but are not listed below. The meanings of most of them are evident from their their names and the context, but the following need some explanation:

FILTLIST: This filters a list through a predicate, returning a list of all the elements satisfying the predicate.

RIG: This takes a list constant and returns a list in which all elements preceded by "&" have been evaluated.

RECORD: This is a macro for declaring records. The default field size is COMPND, but full-word fields can be selected by using a 0 after the field name declaration. Constructor and destructor functions are made by concatenating the class name with "cons" or "dest".

RHLOOP: This is a looping macro. On each iteration, an item in the list preceding RHLOOP is assigned to the variable RH. The macro ENDRH terminates the loop.

The printing functions are not listed.

```
                    *** SOME RECORD CLASSES AND GLOBAL VARIABLES ***

VARS RECTS PERCEPTS RELATIONS ;
NIL->PERCEPTS;NIL->PERCEPTS;NIL->RELATIONS;NIL->RECTS;
COMMENT'triples and quadruples already exist.
this allows their components to be given
more meaningful names!;

0->POPCOMMENT;
TRIP1->RELSLOT1;TRIP2->RELSLOT2;TRIP3->RELCRED;
QUAD1->SLOTPER;QUAD2->SLOTFUN;QUAD3->SLOTTYPE;
QUAD4->SLOTRELS;

COMMENT'these are the zones in a rectangle
which has been given a top/bottom direction!;
[1 1 0.8 0]->DEFTOPEND;[0.2 1 0 0]->DEFBOTEND;
[1 1 0.5 0]->DEFTOPHALF;[0.5 1 0 0]->DEFBOTHALF;
[0.9 0.8 0.7 0.2]->DEFTOPPOLE;[0.3 0.8 0.1 0.2]->DEFBOTPOLE;

1->POPCOMMENT;

COMMENT'interpretations of rectangles as puppet
parts used to be called "percepts". interpretations
of overlaps as joints were called "relations".
the morphemes "per" and "rel" are used with
this sense.!;


RECORD PERCEPT PERNAME PERRECT PERPROX PERTYPE PERSLOTS
        PERCRED;
RECORD RECT RECTNAME RECTCON RECTPERS WHOLE TOPEND BOTEND
TOPHALF BOTHALF TOPPOLE BOTPOLE;


   *** CODE FOR MAKING CONSTRAINTS ***

ENSURELIST CONSTRAINTS;

RECORD CONSTR CONVIOL 0 HYPLENGTH 0 OLDCONVIOL 0;

COMMENT'conviol stores the amount by which
the constraint is violated. whenever a supposition
value changes, the violations of all constraints
involving it are changed appropriately. each
supposition node will cause some of its constraints
to be more violated when its value goes up, and
will also cause others to be less violated. it
keeps these two sets of constraints in separate
lists called ceilings and floors!;
```

```
COMMENT'the number stored in conviol is the
difference between the two sides of the
algebraic inequality.(positive numbers mean
violation).this number is not the same as the
distance in hyperspace of the point from the
plane. however, for any given plane the
violation and the distance have a fixed ratio.
this is kept in hyplength.
!;

COMMENT'consider ,for example, the constraint
x-2y > 0 . when this has a violation of 1,
then the force in the x direction should be
1/sqrt(5) and in the y direction it should
be -2/sqrt(5). the hyplength is sqrt(5),
which the root of the sum of the squares
of the coefficients in the inequality.
!;
FUNCTION  CREDSUM L=>SUM;
0->SUM;
L RHLOOP;RH.CREDVAL+SUM->SUM;ENDRH;
END;

FUNCTION REMOVEALL X L=>N REM;
COMMENT'removes all occurences of x from l and returns
their number and the remaining list!;
NIL->REM;1->N;
L RHLOOP;
IF RH=X THEN N+1->N ELSE RH::REM->REM CLOSE;
ENDRH;
END;

FUNCTION SUMSQUARES L;
COMMENT'returns the sum of the squares of the occurence
        numbers!;
VARS N;IF L.NULL THEN 0 EXIT;
REMOVEALL(L.HD,L.TL)->L->N;
N*N+L.SUMSQUARES;
END;


FUNCTION RETURNCONSTR FLIST CLIST N=>C;
COMMENT'n+the credvals in clist musnt exceed the credvals
in flist i.e. the sum of the credvals in flist-the sum in
 clist must be at least n.!;
CONSCONSTR(CLIST.CREDSUM-FLIST.CREDSUM+N,
        SQRT(FLIST.SUMSQUARES+CLIST.SUMSQUARES),UNDEF,FLIST,CLIST)
            ->C;
C:::CONSTRAINTS;
FLIST RHLOOP;C::RH.FLOORS->RH.FLOORS;ENDRH;
CLIST RHLOOP;C::RH.CEILINGS->RH.CEILINGS;ENDRH;
END;


FUNCVAR MAKECONSTR RETURNCONSTR FNCOMP ERASE;
```

```
FUNCTION MAKECRED OBJ P=>C;
CONSCREDNODE(OBJ,NIL,NIL,NIL,0,0,P)->C;
END;

FUNCTION ATMOSTONE L;
MAKECONSTR(NIL,L,-1);
END;

FUNCTION ATMOSTTWO L;
MAKECONSTR(NIL,L,-2);
END;


FUNCTION MORECRED A B;
MAKECONSTR(A,B,0);
END;

FUNCTION ATLEASTONE L;
MAKECONSTR(L,NIL,1);
END;

FUNCTION INFERFROM L B;
MAKECONSTR(B::NIL,L,1-L.LENGTH);
END;


   *** CODE FOR CHANGING SUPPOSITION VALUES ***

COMMENT'supposition nodes used to be called
crednodes. the morpheme "cred" is used like
this!;

ENSURELIST CREDNODES;

RECORD CREDNODE CREDOBJ FLOORS CEILINGS CREDVAL 0 CREDINC 0
CREDPREF 0;

COMMENT'credobj is the hypothesis, credpref
is its preference, and credval is its
supposition value. credinc is the next
increment in credval, which is computed
and then stored until the other
credincs have also been computed using
the current set of supposition values.
this is necessary for parallel
relaxation. floors
and ceilings are lists of the constraints
which, when violated, tend to hold the
supposition value up(floors) or down
(ceilings).!;
```

```
FUNCTION CHANGEVALS CREDNODE INC;
CREDNODE.CEILINGS RHLOOP;
   RH.CONVIOL+INC->RH.CONVIOL;
ENDRH;
CREDNODE.FLOORS RHLOOP;
   RH.CONVIOL-INC->RH.CONVIOL;
ENDRH;
CREDNODE.CREDVAL+INC->CREDNODE.CREDVAL;
END;

FUNCTION ENDFIXINC INC X;
VARS N;INC+X->N;
IF N>1 THEN INC+1-N
ELSEIF N<0 THEN INC-N
ELSE INC
CLOSE;
END;



FUNCTION CHANGETO C VAL;
CHANGEVALS(C,VAL-C.CREDVAL);
END;


FUNCTION RESETVALS;
APPLIST(CREDNODES,CHANGETO(%0%));
END;

'*** SET INITINCS ***
!.PRSTRING;

VARS COEFFLIST DCOEFF PCOEFF FCOEFF HCOEFF;
[PCOEFF DCOEFF FCOEFF HCOEFF]->COEFFLIST;
VARS COARSE MEDIUM FINE TERMINAL;
[0.4 0.5 0.3   0]->COARSE;
[0.2 0.5 0.3 0 ]->MEDIUM;
[0.1 0.8 0.3 0]->FINE;
[0.1 0.8 0.3 0.1]->TERMINAL;

FUNCTION SETCOEFFS L;
POP L->PCOEFF;
POP L->DCOEFF;
POP L->FCOEFF;
POP L->HCOEFF;
END;


COARSE.SETCOEFFS;
```

```
FUNCTION INITINCS;
APPLIST(CREDNODES,
   LAMBDA C;0->C.CREDINC;END);
END;

FUNCTION CEILFORCE CON;
VARS V;CON.CONVIOL->V;
IF V>0 THEN (-V)/CON.HYPLENGTH ELSE 0 CLOSE;
END;

FUNCTION FLOORFORCE CON;
VARS V;CON.CONVIOL->V;
IF V>0 THEN V/CON.HYPLENGTH ELSE 0 CLOSE;
END;

FUNCTION UPFORCE C;
APPSUM(C.CEILINGS,CEILFORCE)+APPSUM(C.FLOORS,FLOORFORCE);
END;

FUNCTION STORESTEP C;
COMMENT'this stores the size of the next step in credinc!;
VARS INC;
C.CREDINC*DCOEFF
+(C.UPFORCE+C.CREDPREF*PCOEFF+(C.CREDVAL-1/2)*HCOEFF)*FCOEFF
->INC;
ENDFIXINC(INC,C.CREDVAL)->C.CREDINC;
END;

FUNCTION TAKESTEP C;
CHANGEVALS(C,ENDFIXINC(C.CREDINC,C.CREDVAL));
END;

FUNCTION MOVE;
APPLIST(CREDNODES,STORESTEP);
APPLIST(CREDNODES,TAKESTEP);
END;


FUNCTION GETSHOWLIST;
IF CREDNODES.LENGTH>20 THEN FIRST(20,CREDNODES)
ELSE CREDNODES CLOSE;
END;

FUNCTION RELAXANDSHOW STEPS PRFREQ PRINTLIST;
VARS N;PRFREQ->N;
PRINTLIST.SHOWNAMES;
PRINTLIST.SHOWCREDS;
STEPS<* .MOVE;
        IF N>1 THEN N-1->N ELSE PRINTLIST.SHOWCREDS;
                                PRFREQ->N;CLOSE;*>;
END;
```

```
FUNCTION RELAX;
.GETSHOWLIST.RELAXANDSHOW;
END;

FUNCTION RELAXINSTAGES STAGES;
VARS L;.GETSHOWLIST->L;
L.SHOWNAMES;L.SHOWCREDS;
APPLIST(STAGES,
        LAMBDA X;
        IF X.ISWORD THEN X.VALOF.SETCOEFFS
        ELSE X<* .MOVE *>;L.SHOWCREDS CLOSE;
        END);
1.NL;
END;

FUNCTION RELAX50;
.RESETVALS;.INITINCS;
[COARSE 10 MEDIUM 10 FINE 10 TERMINAL 10 10].RELAXINSTAGES;
END;



     *** ZONE  GEOMETRY  ***

FUNCTION GETBORD P Q;
VARS A B C D;
P.DESTPAIR->B->A;Q.DESTPAIR->D->C;
CONSTRIPLE(D-B,A-C,A*D-B*C);
END;

FUNCTION ONSIDE P B;
B.TRIP1*P.FRONT+B.TRIP2*P.BACK>=B.TRIP3;
END;

FUNCTION OFFSIDE P B;
ONSIDE(P,B).NOT;
END;

FUNCTION NOTSEP X;
COMMENT'tests whether all points in one
rectangle are on the off-side of the
boundary x!;
ONSIDE(A,X) OR ONSIDE(B,X) OR ONSIDE(C,X) OR ONSIDE(D,X);
END;


FUNCTION NOSEPARATOR PTS BDS;
VARS A B C D;PTS.DESTQUAD->D->C->B->A;
BDS.QUAD1.NOTSEP AND BDS.QUAD2.NOTSEP AND
BDS.QUAD3.NOTSEP AND BDS.QUAD4.NOTSEP
END;
```

```
FUNCTION OVERLAP Z1 Z2;
COMMENT'if two convex polygons dont overlap there must be
a line which separates them, and one of their borders must
be such a line!;
NOSEPARATOR(Z1.FRONT,Z2.BACK)
AND NOSEPARATOR(Z2.FRONT,Z1.BACK)
END;



FUNCTION BORDSECT B1 B2;
COMMENT'finds the point of intersection of two borders!;
VARS A B C D E F DIV;
B1.DESTTRIPLE->C->B->A;B2.DESTTRIPLE->F->E->D;
B*D-A*E->DIV;
IF DIV=0 THEN "PARALLEL";EXIT;
CONSPAIR( (B*F-C*E)/DIV, (C*D-A*F)/DIV);
END;

FUNCTION AVBORD E F P;
COMMENT'checks that borders e and f are parallel and makes
a new one         which is a weighted average using p of e
and q of f!;
        OPERATION 7 === X Y;
        APPROXEQ(X,Y,1);
        END;
VARS Q R;1-P->Q;
IF E.TRIP2===0 OR F.TRIP2===0
THEN IF E.TRIP2===F.TRIP2
     THEN CONSTRIPLE(P*E.TRIP1+Q*F.TRIP1,0,
                     P*E.TRIP3+Q*F.TRIP3)
     ELSE "AVBORD".POPERR;
     CLOSE;
ELSEIF E.TRIP1/E.TRIP2===F.TRIP1/F.TRIP2
THEN E.TRIP2/F.TRIP2->R;
     CONSTRIPLE(E.TRIP1,E.TRIP2,P*E.TRIP3+Q*R*F.TRIP3)
ELSE "AVBORD".POPERR;
CLOSE;
END;

FUNCTION REVBORD B;
CONSTRIPLE(-(B.TRIP1),-(B.TRIP2),-(B.TRIP3));
END;



FUNCTION ZONEPTS Z;
COMMENT'gets the corners of a zone from the borders!;
VARS A B C D;Z.DESTQUAD->D->C->B->A;
CONSQUAD(BORDSECT(D,A),BORDSECT(A,B),BORDSECT(B,C),
         BORDSECT(C,D));
END;
```

```
FUNCTION MKZONEBDS P L;
COMMENT'makes zone borders from rectangle borders and a
list of relative positions of ymax ,xmax, ymin, xmin!;
VARS A B C D F;P.BACK.DESTQUAD->D->C->B->A;REVBORD->F;
CONSQUAD(AVBORD(A,C.F,POP L),AVBORD(B,D.F,POP L),
         AVBORD(A.F,C,POP L),AVBORD(B.F,D,POP L));
END;

FUNCTION GETZONE DEFZONE P;
COMMENT'returns a pair consisting of the corners and borders
        for a specified zone relative to p!;
VARS B;
MKZONEBDS(P,DEFZONE)->B;
CONSPAIR(B.ZONEPTS,B);
END;

FUNCTION LASTCORN L;
VARS A B C;POP L->A;POP L->B;POP L->C;
CONSPAIR(A.FRONT+C.FRONT-B.FRONT,A.BACK+C.BACK-B.BACK);
END;

FUNCTION CONVPAIR L;
CONSPAIR(L.HD,L.TL.HD);
END;

FUNCTION MAKEWHOLE L;
COMMENT'makes the points and borders of the whole from a
        list of its corner points!;
VARS BORDS CORNS;
APPLIST(L,IDENTFN).CONSQUAD->CORNS;
CONSQUAD(GETBORD(CORNS.QUAD1,CORNS.QUAD2),
         GETBORD(CORNS.QUAD2,CORNS.QUAD3),
         GETBORD(CORNS.QUAD3,CORNS.QUAD4),
         GETBORD(CORNS.QUAD4,CORNS.QUAD1))->BORDS;
CONSPAIR(CORNS,BORDS);
END;

FUNCTION CONWHSUB R S;
IF OVERLAP(R.WHOLE,S.WHOLE)
THEN R::S.RECTCON->S.RECTCON;S::R.RECTCON->R.RECTCON;
CLOSE;
END;

FUNCTION CONWHOLE L;
LOOPIF L.ISLINK
THEN APPLIST(L.TL,CONWHSUB(%L.HD%));L.TL->L;
CLOSE;
END;
```

```
FUNCTION MAKERECT L =>RECT;
VARS POINTS NAME P F;L.HD->NAME;
MAPLIST(L.TL,CONVPAIR)->POINTS;
POINTS.MAKEWHOLE->P;
GETZONE(%P%)->F;
CONSRECT(NAME,NIL,NIL,P,DEFTOPEND.F,DEFBOTEND.F,
          DEFTOPHALF.F,DEFBOTHALF.F,DEFTOPPOLE.F,
           DEFBOTPOLE.F)->RECT;
RECT->NAME.VALOF;
END;


FUNCTION MYDIST P Q;
SQRT( (P.FRONT-Q.FRONT)^2 + (P.BACK-Q.BACK)^2);
END;

FUNCTION WIDTH R;
VARS PTS;R.WHOLE.FRONT->PTS;
MYDIST(PTS.QUAD1,PTS.QUAD2);
END;

FUNCTION HEIGHT R;
VARS PTS;R.WHOLE.FRONT->PTS;
MYDIST(PTS.QUAD2,PTS.QUAD3);
END;


FUNCTION AREA R;
R.HEIGHT*R.WIDTH;
END;

FUNCTION PUPIN FILENAME;
COMMENT'the data files give lists of lists
of coordinates when compiled!;
MAPLIST(FILENAME.COMPILE,MAKERECT)->RECTS;
RECTS.CONWHOLE;
END;
```

```
          *** CODE FOR DECIDING WHETHER AN  ***
   *** OVERLAP COULD DEPICT A JOINT  ***


MACRO MACP;
VARS Z1 Z2;.ITEMREAD->Z1;.ITEMREAD->Z2;
MACRESULTS([LAMBDA P;IF P.PERPROX="TOP" THEN P.PERRECT.&Z1
            ELSE P.PERRECT.&Z2 CLOSE;END;].RIG);
END;

FUNCTION WIDER P Q;
P.PERRECT.WIDTH>Q.PERRECT.WIDTH;
END;

FUNCTION ALL P;
P.PERRECT.WHOLE;
END;

VARS PROXEND DISTEND PROXPOLE DISTPOLE PROXHALF DISTHALF;
MACP TOPEND BOTEND->PROXEND;
MACP BOTEND TOPEND->DISTEND;
MACP TOPPOLE BOTPOLE->PROXPOLE;
MACP BOTPOLE TOPPOLE->DISTPOLE;
MACP TOPHALF BOTHALF->PROXHALF;
MACP BOTHALF TOPHALF->DISTHALF;

FUNCTION KNEEJOIN P Q;
P.PERRECT.WIDTH=<Q.PERRECT.WIDTH
AND OVERLAP(P.PROXEND,Q.DISTEND)
AND OVERLAP(P.PROXEND,Q.PROXHALF).NOT
AND OVERLAP(P.DISTHALF,Q.DISTEND).NOT
END;



FUNCTION TERMJOIN P Q;
COMMENT'for hands or feet (terminal parts)!;
P.PERRECT.HEIGHT<Q.PERRECT.HEIGHT
AND P.PERRECT.AREA<Q.PERRECT.AREA
AND OVERLAP(P.DISTEND,Q.ALL).NOT
AND OVERLAP(P.ALL,Q.PROXHALF).NOT
END;

FUNCTION ARMJOIN P Q;
WIDER(Q,P)
AND OVERLAP(P.PROXEND,Q.PROXHALF)
AND OVERLAP(P.PROXEND,Q.DISTHALF).NOT
AND OVERLAP(P.DISTEND,Q.PROXPOLE).NOT
END;

FUNCTION LEGJOIN P Q;
WIDER(Q,P)
AND OVERLAP(P.PROXEND,Q.DISTHALF)
AND OVERLAP(P.PROXEND,Q.PROXHALF).NOT
AND OVERLAP(P.DISTEND,Q.DISTPOLE).NOT
END;
```

```
FUNCTION HEADJOIN P Q;
WIDER(P,Q) AND P.PERRECT.HEIGHT<2*P.PERRECT.WIDTH
AND OVERLAP(P.DISTEND,Q.PROXEND)
AND OVERLAP(P.ALL,Q.DISTHALF).NOT
AND OVERLAP(P.PROXHALF,Q.ALL).NOT
END;

FUNCTION NECKJOIN P Q;
COMMENT'for joint between neck and trunk!;
P.PERRECT.WIDTH<Q.PERRECT.WIDTH
AND OVERLAP(P.DISTEND,Q.PROXEND)
AND OVERLAP(P.ALL,Q.DISTHALF).NOT
AND OVERLAP(P.PROXHALF,Q.ALL).NOT
END;

FUNCTION JOINHEAD P Q;HEADJOIN(Q,P);END;
FUNCTION JOINNECK P Q;NECKJOIN(Q,P);END;
FUNCTION JOINKNEE P Q;KNEEJOIN(Q,P);END;
FUNCTION JOINTERM P Q;TERMJOIN(Q,P);END;

FUNCTION JOINARM P Q;ARMJOIN(Q,P);END;
FUNCTION JOINLEG P Q;LEGJOIN(Q,P);END;

VARS HEAD NECK HAND LOWERARM UPPERARM TRUNK CALF THIGH FOOT;

[[HEADJOIN NECK]]->HEAD;
[[JOINHEAD HEAD][NECKJOIN TRUNK]]->NECK;
[[TERMJOIN LOWERARM]]->HAND;
[[KNEEJOIN UPPERARM][JOINTERM HAND]]->LOWERARM;
[[ARMJOIN TRUNK][JOINKNEE LOWERARM]]->UPPERARM;
[[TERMJOIN CALF]]->FOOT;
[[KNEEJOIN THIGH][JOINTERM FOOT]]->CALF;
[[LEGJOIN TRUNK][JOINKNEE CALF]]->THIGH;
[[JOINNECK NECK] [JOINARM UPPERARM][JOINLEG THIGH]]
->TRUNK;
```

```
          *** CODE  FOR GROWING THE NETWORK ***
    *** OF PART AND JOINT HYPOTHESES   ***

FUNCTION GIVERECT P;
P::P.PERRECT.RECTPERS->P.PERRECT.RECTPERS;
END;

FUNCTION MAKESLOT L P;
CONSQUAD(P,L.HD,L.TL.HD,NIL);
END;

FUNCTION MAKEPER RECT PROX TYPE=>PER;
CONSPERCEPT(UNDEF,RECT,PROX,TYPE,UNDEF,UNDEF)->PER;
MAKECRED(PER,PERPREF)->PER.PERCRED;
MAPLIST(TYPE.VALOF,MAKESLOT(%PER%))->PER.PERSLOTS;
PER::PERCEPTS->PERCEPTS;
PER::RECT.RECTPERS->RECT.RECTPERS;
END;

FUNCTION OTHERPER R P;
VARS X;R.RELSLOT1.SLOTPER->X;
IF X=P THEN R.RELSLOT2.SLOTPER ELSE X CLOSE;
END;

FUNCTION ALREADY RELS Q;
SOMETRUE(RELS,LAMBDA R;
   R.RELSLOT1.SLOTPER=Q OR R.RELSLOT2.SLOTPER=Q ; END));
END;

FUNCTION FINDSLOT P Q;
VARS T FUN;Q.PERTYPE->T;
P.PERSLOTS RHLOOP;
   IF RH.SLOTTYPE=T
   THEN RH ;EXIT;;
ENDRH;
.POPERR;
END;

FUNCTION ADDREL P Q PSLOT;
VARS QSLOT REL;
P.PERRECT.RECTNAME.PR;1.SP;Q.PERRECT.RECTNAME.PR;3.SP;
FINDSLOT(Q,P)->QSLOT;
CONSTRIPLE(PSLOT,QSLOT,UNDEF)->REL;
MAKECRED(REL,RELPREF)->REL.RELCRED;
REL::RELATIONS->RELATIONS;
REL::PSLOT.SLOTRELS->PSLOT.SLOTRELS;
REL::QSLOT.SLOTRELS->QSLOT.SLOTRELS;
END;
```

```
FUNCTION GETEXISTINGPER RECT TYPE ORIENT;
RECT.RECTPERS RHLOOP;
        IF RH.PERTYPE=TYPE AND RH.PERPROX=ORIENT THEN RH;EXIT;
ENDRH;
FALSE;
END;

FUNCTION TRYTHEPER P SLOT RECT ORIENT;
VARS DONE Q REQTYPE FUN;
SLOT.SLOTTYPE->REQTYPE;SLOT.SLOTFUN.VALOF->FUN;
GETEXISTINGPER(RECT,REQTYPE,ORIENT)->Q;
IF Q
THEN IF Q=P THEN EXIT;
        IF ALREADY(SLOT.SLOTRELS,Q).NOT AND FUN(P,Q)
        THEN Q.ANYREL->DONE; ADDREL(P,Q,SLOT);
                UNLESS DONE THEN Q CLOSE;
         CLOSE;
ELSE MAKEPER(RECT,ORIENT,REQTYPE)->Q;
        IF FUN(P,Q) THEN ADDREL(P,Q,SLOT);Q;CLOSE;
        COMMENT'this is where future members of livelist are
                dumped!;
CLOSE;
END;


FUNCTION TRYFILLSLOT SLOT PERCEPT;
PERCEPT.PERRECT.RECTCON RHLOOP;
        TRYTHEPER(PERCEPT,SLOT,RH,"TOP");
        TRYTHEPER(PERCEPT,SLOT,RH,"BOT");
ENDRH;
END;

FUNCTION TRYGROW P;
APPLIST(P.PERSLOTS,LAMBDA S;TRYFILLSLOT(S,P);END);
END;

FUNCTION GROWPERS LIVELIST;
COMMENT'this takes the most recently created percepts and
        tries to fill their slots,possibly making more percepts!;
IF LIVELIST.NULL THEN EXIT;
[%APPLIST(LIVELIST,TRYGROW)%].GROWPERS;
END;

FUNCTION MAKEBOTH R T;
MAKEPER(R,"TOP",T);MAKEPER(R,"BOT",T);
END;
```

```
FUNCTION GETPOSSNUCLEI;
COMMENT'there are three types of nucleus:
        a trunk requires 3 feasible connected rectangles.
        a head requires the right proportions and exactly
       one  other connected rectangle ,which must
       be narrower. a hand or foot requires
       exactly one connected rectangle ,with greater area.!;
MAPLIST(RECTS,
LAMBDA R;
VARS L;R.RECTCON->L;
IF L.LENGTH=1
THEN IF L.HD.WIDTH<R.WIDTH
     THEN MAKEBOTH(R,"HEAD")
     CLOSE;
     IF L.HD.AREA>R.AREA
     THEN MAKEBOTH(R,"HAND");MAKEBOTH(R,"FOOT")
     CLOSE;
ELSEIF L.LENGTH>2
AND FILTLIST(L,LAMBDA X;X.WIDTH<R.WIDTH;END).LENGTH>2
THEN MAKEBOTH(R,"TRUNK")
CLOSE;
END);
END;

FUNCTION ANYREL P;
SOMETRUE(P.PERSLOTS,SLOTRELS FNCOMP ISLINK);
END;


FUNCTION GIVEPERNAME P L;
CONCATWORD(P.PERRECT.RECTNAME,NUMWORD(ITEMNUM(P,L)))
->P.PERNAME;P->P.PERNAME.VALOF;
END;

FUNCTION NEATPERS;
VARS L;
NIL->PERCEPTS;
RECTS.REV RHLOOP;
   RH.RECTPERS->L;
   APPLIST(L,GIVEPERNAME(%L%));
   L<>PERCEPTS->PERCEPTS;
ENDRH;
END;

FUNCTION MAKEPERNET;
VARS LIVELIST;NIL->PERCEPTS;NIL->RELATIONS;
.GETPOSSNUCLEI->LIVELIST;
LIVELIST->PERCEPTS;NIL->RELATIONS;
GROWPERS(PERCEPTS);
FILTLIST(PERCEPTS,ANYREL)->PERCEPTS;
APPLIST(RECTS,LAMBDA X;NIL->X.RECTPERS;END);
APPLIST(PERCEPTS,GIVERECT);
.NEATPERS;
END;
```

```
   *** CODE FOR HANDLING EXTRA INPUT ***
   *** INSTRUCTIONS LIKE "TRYTOINTERPRET" ***


FUNCTION CLEARPERPREFS;
APPLIST(PERCEPTS,LAMBDA P;0->P.PERCRED.CREDPREF;END);
END;

FUNCTION CLEARRELPREFS;
APPLIST(RELATIONS,LAMBDA R;0->R.RELCRED.CREDPREF;END);
END;

FUNCTION ISOFTYPE X T;
X.PERTYPE=T;
END;

FUNCTION ISOFRECT X NAME;
X.PERRECT.RECTNAME=NAME;
END;

FUNCTION HASPROXAT X W;
X.PERPROX=W;
END;

FUNCVAR ISUPRIGHT HASPROXAT(%"TOP"%);


FUNCTION HELPPERS N PRED;
APPLIST(PERCEPTS,
        LAMBDA P;
        IF P.PRED THEN N+P.PERCRED.CREDPREF->P.PERCRED.CREDPREF
        CLOSE;
        END);
END;

FUNCTION HELPRELS N PRED;
APPLIST(RELATIONS,
        LAMBDA R;
        IF R.PRED THEN N+R.RELCRED.CREDPREF->R.RELCRED.CREDPREF
        CLOSE;
        END);
END;

FUNCTION HELPPER P N;
N+P.PERCRED.CREDPREF->P.PERCRED.CREDPREF;
END;

FUNCTION THERELBETWEEN P Q;
VARS X Y;
RELATIONS RHLOOP;
   RH.RELSLOT1.SLOTPER->X;RH.RELSLOT2.SLOTPER->Y;
   IF (X=P AND Y=Q) OR (X=Q AND Y=P) THEN RH;EXIT;
ENDRH;
FALSE;
END;
```

```
FUNCTION HELPREL R N;
N+R.RELCRED.CREDPREF->R.RELCRED.CREDPREF;
END;

FUNCTION SETORIENTPREF PART ORIENT N;
HELPPERS(N,LAMBDA P;ISOFTYPE(P,PART) AND HASPROXAT(P,ORIENT)
            ;END);
END;

FUNCTION SETPARTPREF RECT L N;
VARS W PART;L.HD->W;
IF W="TOP" OR W="BOT"
THEN L.TL.HD->PART;
        HELPPERS(N,LAMBDA P;
        ISOFRECT(P,RECT) AND HASPROXAT(P,W)
        AND ISOFTYPE(P,PART);
                        END);
ELSEIF W="SOMEPART"
THEN HELPPERS(N,LAMBDA P;ISOFRECT(P,RECT);END)
ELSE HELPPERS(N,LAMBDA P;ISOFRECT(P,RECT) AND ISOFTYPE(P,W);
                END)
CLOSE;
END;

OPERATION 3 TRYTOINTERPRET L;
VARS W N;
MAPLIST(L,LAMBDA W;
                IF W="UPRIGHT" THEN "TOP"
                ELSEIF W="UPSIDEDOWN" THEN "BOT"
                ELSE W CLOSE;
        END)->L;
COMMENT'the program likes "top" and "bot" but people dont
understand them!; l.rev.hd->n;l.hd->w;
IF MEMBER(W,[HEAD NECK TRUNK UPPERARM LOWERARM HAND THIGH
              CALF FOOT])
THEN SETORIENTPREF(W,L.TL.TL.HD,N);
ELSE SETPARTPREF(W,L.TL.TL,N)
CLOSE;
END;

FUNCTION INHIBIT N;
APPLIST(CREDNODES,
        LAMBDA C;IF C.CREDVAL>0.5 THEN C.CREDPREF-N->C.CREDPREF
                CLOSE;
        END);
END;

FUNCTION SWITCHATTENTION N;
VARS L CHANGE;
RECTS RHLOOP;
        MAPLIST(RH.RECTPERS,PERCRED)->L;
        IF SOMETRUE(L,LAMBDA C;C.CREDVAL>0.5;END)
        THEN -N->CHANGE ELSE N->CHANGE CLOSE;
        APPLIST(L,LAMBDA C;C.CREDPREF+CHANGE->C.CREDPREF;END);
ENDRH;
END;
```

```
   *** CODE FOR CREATING THE CONSTRAINTS ***

FUNCTION SLOTSIZE S;
IF S.SLOTPER.PERTYPE="TRUNK"
AND S/=S.SLOTPER.PERSLOTS.HD
THEN 2 ELSE 1 CLOSE;
END;

VARS RELPREF PERPREF;1->RELPREF;0->PERPREF;

FUNCTION THREECOLPR X EXTRALINE;
VARS F;PERNAME FNCOMP PR->F;
IF X.DATAWORD="PERCEPT" THEN IF EXTRALINE THEN 3.SP
                             ELSE 1.SP;X.F CLOSE;
ELSE IF EXTRALINE THEN 1.SP;X.RELSLOT1.SLOTPER.F
     ELSE 1.SP;X.RELSLOT2.SLOTPER.F
     CLOSE;
CLOSE;
END;


FUNCTION SETRECTCONSTR RECT;
MAPLIST(RECT.RECTPERS,PERCRED).ATMOSTONE;
END;

FUNCTION SETTYPECONSTR TYPE PERLIST FUN;
VARS L;FILTLIST(PERLIST,LAMBDA P;P.PERTYPE=TYPE;END)->L;
MAPLIST(L,PERCRED).FUN;
END;

FUNCTION SETSLOTCONSTR S;
VARS FLIST P;
S.SLOTPER.PERCRED->P;
IF S.SLOTSIZE=2 THEN [%P,P%] ELSE [%P%] CLOSE->FLIST;
MORECRED(FLIST,MAPLIST(S.SLOTRELS,RELCRED));
END;

FUNCTION SETCONSTRAINTS;
APPLIST(RELATIONS,
        LAMBDA R;VARS L;R.RELCRED::NIL->L;
        MORECRED(R.RELSLOT1.SLOTPER.PERCRED::NIL,L);
        MORECRED(R.RELSLOT2.SLOTPER.PERCRED::NIL,L);
        END);
APPLIST(RECTS,SETRECTCONSTR);
APPLIST([HEAD NECK TRUNK],SETTYPECONSTR(%PERCEPTS,ATMOSTONE%));
APPLIST([HAND FOOT LOWERARM UPPERARM CALF THIGH],
        SETTYPECONSTR(%PERCEPTS,ATMOSTTWO%));
APPLIST(PERCEPTS,
        LAMBDA  P;APPLIST(P.PERSLOTS,SETSLOTCONSTR);END);
END;
```

```
 *** THE TOP LEVEL FUNCTION FOR CREATING ***
 *** THE NETWORK OF CANDIDATE HYPOTHESES ***

FUNCTION FIRST N L;
IF  L.NULL OR N=0 THEN NIL
ELSE  L.HD::FIRST(N-1,L.TL) CLOSE;
END;

FUNCTION GETPUPNET FNAME;
NIL->PERCEPTS;NIL->RECTS;NIL->CONSTRAINTS;NIL->CREDNODES;
FNAME.PUPIN;.MAKEPERNET;1.NL;
.SETCONSTRAINTS;
MAPLIST(PERCEPTS,PERCRED)->PLIST;
PLIST<>MAPLIST(RELATIONS,RELCRED)->CREDNODES;
FIRST(20,PLIST)->PLIST;
'PLIST CREATED.
!.PRSTRING;
END;
```

APPENDIX 4

This shows the way the supposition values change during relaxation for the examples in chapter 2. Only the first nineteen part-hypotheses are shown in many cases. The function RELAX50 causes fifty rounds of relaxation with printing initially and after every ten rounds. Supposition values X 100 are shown, and for formatting reasons, 100 is printed as 99. The coefficients in the relaxation operator are set at:

K_p	K_d	K_f	K_h	Iterations
0·4	0.5	0·3	0	10
0·2	0·5	0·3	0	10
0·1	0·8	0·3	0	10
0·1	0·8	0·3	0·1	20

The hypotheses which get selected can be identified by referring to the figures in chapter 2.

```
!relax50();

 A1  B1  B2  B3  C1  C2  D1  D2  D3  D4  D5  E1  E2  E3  E4  E5  F1  F2  F3
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 59  65  25  26  63  68  23  53  54   5  10  20  60  61   0   0  57  57  18
 85  83  12  10  55  66  14  53  55   0   0   4  55  56   0   0  54  53   3
 98  94   6   2  50  63   4  49  57   0   2   4  49  56   0   0  54  49   2
 99  99   0   0  44  76   0  42  69   0   0   0  42  65   0   0  62  41   0
 99  99   0   0  17  99   0  16  96   0   0   0  15  92   0   0  89  14   0
```

For the example in figure 2.1.

```
!
!.relax50;
                                          C3  C3  B1  D1  D2  A1
 A1  B1  B2  C1  C2  C3  D1  D2  D3  D3  B2  C3  C2  C1  B1
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 73  65  99  36  36  78  38  38  68  90  99  84  53  53  80
 86  60  98  17  17  87  18  18  83  93  99  75  25  25  75
 86  39  93   9   9  93   8   8  95  99  97  47  12  12  47
 99  19  99   0   0  99   0   0  99  99  99  24   1   1  24
 99  12  99   0   0  99   0   0  99  99  99  16   1   1  16
```

For the example in figure 2.2. The double column headings indicate joint hypotheses.

```
!.relax50;

 A1  B1  B2  B3  B4  B5  C1  C2  C3  C4  C5  C6  D1  D2  D3  D4  D5  E1  E2
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 60  47  32  34  10  10  61  61  18  18   4   6  17  17   7  39  41  17  17
 76  70  28  22   0   0  68  55  10  10   0   0  10  10   0  48  42  10  10
 84  75  19   6   2   2  68  46   4   4   0   0   3   3   0  57  42   3   3
 99  98   0   0   0   0  87  31   0   0   0   0   0   0   0  80  28   0   0
 99  99   0   0   0   0  99   0   0   0   0   0   0   0   0  99   0   0   0
```

For the example in figure 2.4.

```
!.relax50;

 A1  A2  A3  A4  A5  A6  B1  B2  B3  B4  B5  C1  D1  D2  D3  D4  D5  E1  E2
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 72  60  12   0  14   0  49  36  33   1   1  82   0  45  41  27  28  35  35
 70  51   6   0   6   0  61  32  20   0   0  96   0  44  35  19  18  22  23
 71  47   0   0   0   0  76  24   6   0   0  99   0  48  35  14  10  15  17
 86  34   0   0   0   0  99   0   0   0   0  99   0  76  31   0   0   0   0
 99   0   0   0   0   0  99   0   0   0   0  99   0  99   0   0   0   0   0
```

For the example in figure 2.5.

```
!
!.relax50;

 A1  A2  A3  A4  A5  A6  B1  B2  B3  B4  B5  C1  D1  D2  D3  D4  D5  E1  E2
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 60  94   2   0  21   0  43  24  70   0   0  81   0  39  62  20  27  33  29
 50  88   0   0   8   0  52   8  60   0   0  94   0  36  59   9  18  21  13
 40  86   0   0   3   0  69   0  45   0   0  96   0  36  63   3  12  16   3
 25  99   0   0   0   0  99   0   8   0   0  99   0  18  96   0   0   0   0
  6  99   0   0   0   0  99   0   5   0   0  99   0   0  99   0   0   0   0
```

For the example in figure 2.6.

```
!.relax50;

 A1  A2  B1  B2  B3  B4  B5  C1  C2  C3  C4  C5  D1  D2  D3  D4  D5  E1  E2
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 49  14  49  23  13   8   7  55  62  19  13  13   0  38  40  29  29  38  38
 60   7  59  11   6   0   0  45  59   9   7   8   0  34  40  21  21  27  27
 66   0  65   4   0   1   1  40  61   1   4   7   1  34  46  14  14  19  18
 92   0  95   0   0   0   0  39  84   0   0   0   0  39  75   0   0   0   0
 99   0  99   0   0   0   0   3  99   0   0   0   0   0  99   0   0   0   0
```

For the example in figure 2.7.

```
!.relax50;

 A1  A2  B1  B2  C1  C2  D1  D2  D3  E1  E2  G1  G2  G3  H1  I1  J1  J2  J3
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 63  63  63  63  72  63  21  71  74  67  68  15  67  60  42  55  15  67  60
 56  56  56  56  62  56   0  61  67  56  62   3  60  54  57  57   3  60  54
 54  54  54  54  58  52   3  53  59  51  57   2  57  51  53  54   2  57  51
 53  53  53  53  62  48   0  51  65  48  62   0  62  48  56  54   0  62  48
 54  54  54  54  75  36   0  40  78  36  75   0  74  36  55  54   0  74  36

!relaxinstages([terminal 10 10 10]);

 A1  A2  B1  B2  C1  C2  D1  D2  D3  E1  E2  G1  G2  G3  H1  I1  J1  J2  J3
 54  54  54  54  75  36   0  40  78  36  75   0  74  36  55  54   0  74  36
 54  54  54  54  99   3   0   4  99   2  99   0  99   3  55  54   0  99   3
 54  54  54  54  99   1   0   1  99   0  99   0  99   0  55  54   0  99   0
 54  54  54  54  99   1   0   1  99   0  99   0  99   0  55  54   0  99   0
```

For the example in figure 2.9.

Thirty extra rounds of relaxation are shown, with the coefficients at their terminal settings

```
!relaxinstages([terminal 10 10]);

 A1  A2  B1  B2  C1  C2  D1  D2  D3  E1  E2  G1  G2  G3  H1  I1  J1  J2  J3
 54  54  54  54  99   1   0   1  99   0  99   0  99   0  55  54   0  99   0
 44  66  64  44  99   1   0   0  99   0  99   0  99   0  66  67   0  99   0
 11  99  97  12  99   1   0   0  99   0  99   0  99   0  99  99   0  99   0
```

For the example in figure 2.10.

The deadlock is broken by additional input instructions.

APPENDIX 5

CODE FOR THE EXAMPLE IN SECTION 5.2

```
FUNCTION SAMESUM L;
COMMENT'makes constraints which force the head of the list
to have the same sum as the rest!;
VARS X L;MAPLIST(L,VALOF)->L;
L.HD::NIL->X;L.TL->L;
MORECRED(X,L);MORECRED(L,X);
END;

FUNCTION UNITSUM L;
COMMENT'sets up two constraints to ensure that the
supposition values of the nodes in l add to one.!;
MAPLIST(L,VALOF)->L;
ATMOSTONE(L);ATLEASTONE(L);
END;

FUNCTION SETPREFS L;
COMMENT'l is a list of nodenames and numbers!;
VARS X N;
UNTIL L.NULL
DO POP L->X;POP L->N;
        N->X.VALOF.CREDPREF;
ENDDO;
END;

FUNCTION MAKENODE W;
VARS C;
CONSCREDNODE(W,NIL,NIL,0,0,0)->C;
C->W.VALOF;C::CREDNODES->CREDNODES;
END;

APPLIST([A0 A1 A2 A3 B0 B1 B2 B3 C0 C1 C2 C3 H4 H5 H6 H7
  H8 H9 J4 J5 J6 J7 J8 J9 K4 K5 K6 K7 K8 K9],MAKENODE);
APPLIST([ [A0 A1 A2 A3] [B0 B1 B2 B3] [C0 C1 C2 C3]
        [H4 H5 H6 H7 H8 H9] [J4 J5 J6 J7 J8 J9]
        [K4 K5 K6 K7 K8 K9] ],UNITSUM);

APPLIST([ [A0 H4 H6][A1 H5 H9][A2 H8][A3 H7]
          [A0 K4 K7][A1 K5 K8][A2 K9][A3 K6]
          [B0 J4 J6][B1 J5 J9][B2 J8][B3 J7]
          [B0 H4 H7][B1 H5 H8][B2 H9][B3 H6]
          [C0 K4 K6][C1 K5 K9][C2 K8][C3 K7]
          [C0 J4 J7][C1 J5 J8][C2 J9][C3 J6] ],
        SAMESUM);

VARS PLIST;
MAPLIST([A0 A1 A2 A3 B0 B1 B2 B3 C0 C1 C2 C3],VALOF)
        ->PLIST;
```

APPENDIX 6

CODE FOR THE "SETTLE" SYSTEM.

*** CODE FOR MAKING SCHEMAS AND INSTANCES ***

```
COMMENT'this file is for making schemas and instances for
a settle system. rules have to be added after the schema
is made.!;

ENSURELIST SCHEMAS;

 COMMENT'some slots have known inverses. knowing these
 facilitates  bond specifications!;


OPERATION 4 <-> X Y;
CONSPAIR(X,Y)::INVERSES->INVERSES;
END;

ENSURELIST INVERSES;

FUNCTION INVERSE F;
INVERSES RHLOOP;
        IF RH.FRONT=F THEN RH.BACK,RETURN
        ELSEIF RH.BACK=F THEN RH.FRONT,RETURN
        CLOSE;
ENDRH;
UNDEF;
END;



 COMMENT'instances are strips,butslot names are used to
 access components of them, so accessing functions are
 assigned to slot names. to avoid creating unnecessary
 functions, or creating copies of them, there is a
 dynamic list of them!;

FUNCTION NEXTFUN N SELF;
 COMMENT' a closure of this produces a selector function
 for the n th  component of a strip!;
VARS FUN;
POPVAL([LAMBDA S;SUBSCR(&N,S);END;].RIG)->FUN;
CONCATWORD("SUB",N.NUMWORD)::FUN.FNPROPS->FUN.FNPROPS;
POPVAL([LAMBDA C S;C->SUBSCR(&N,S);END].RIG)->FUN.UPDATER;
FUN;
N+1->FROZVAL(1,SELF);
END;

VARS GENERATOR SUBSCRFUN;
NEXTFUN(%1,UNDEF%)->GENERATOR;
GENERATOR->FROZVAL(2,GENERATOR);
ITEM(%GENERATOR.FNTOLIST%)->SUBSCRF;
 COMMENT'subscrfun takes an integer n and returns a
 selector for the n'th component of a strip.!;
```

```
FUNCTION NAMESLOT N W;
^A%% W IS MADE THE NAME OF THE N'TH COMPONENT OF AN INSTANCE.
POPVAL([%"VARS",W,";"%]);
N.SUBSCRFUN->W.VALOF;
END;


^A%% ALL INSTANCES START WITH THREE SPECIAL COMPONENTS
^A%% CALLED INSTNAME, INSTOF AND INSTCRED.
^A%% INSTNAME CONTAINS THE NAME OF THE INSTANCE.
^A%% INSTOF CONTAINS THE SCHEMA.
^A%% INSTCRED CONTAINS THE ASSOCIATED "CREDNODE".

NAMESLOT(1,"INSTNAME");NAMESLOT(2,"INSTOF");
NAMESLOT(3,"INSTCRED");

RECORD SCHEMA SCHNAME SCHKNOWLEDGE SCHINSTS SCHNUMOF 0;

 COMMENT' the schknowledge of each schema is a strip whose
 components are  slotknowledge records. these contain
 knowledge about the sizes  of the slots, the types of
 fillers allowed and the constraints.!;

RECORD SLOTKNOWLEDGE SKNAME SKSIZE 0 SKTYPECHECKS SKRULES;


 COMMENT'an instance of a schema is a strip whose
 components (apart from the first three defined above)
  re records of type slot.
 each slot has a list of demons, a list of bonds, and a
 pointer to the part of the schema which contains rules
 whose keys may start matching when the slot is filled.!;

RECORD SLOT SLOTKNOWLEDGE SLOTTRIGS SLOTBONDS;
RECORD BOND BONDINST1 BONDINST2 BONDCRED;



VARS SLOTNAME;SLOTKNOWLEDGE FNCOMP SKNAME->SLOTNAME;

FUNCTION SCHSLOTNAMES SCHEMA;
 COMMENT'produces the names of slots from a schema!;
MAPLIST(SCHEMA.SCHKNOWLEDGE.DATALIST.BACK.BACK.BACK,SKNAME)
END;

FUNCTION MAKENEXTNAME SCHEMA=>W;
 COMMENT'all instances of a schema have names consisting
 of the schema name followed by an integer!;
VARS N;SCHEMA.SCHNUMOF+1->N;N->SCHEMA.SCHNUMOF;
CONCATWORD(SCHEMA.SCHNAME,N.NUMWORD)->W;
[%"VARS",W,";"%].POPVAL;
END;
```

```
FUNCTION GETNEWINST SCHEMA=>NEW;
^A%% THIS MAKES A NEW INSTANCE OF A SCHEMA.
VARS C;
</ SCHEMA.MAKENEXTNAME, SCHEMA, UNDEF,
        APPLIST(SCHEMA.SCHKNOWLEDGE.DATALIST.TL.TL.TL,
                CONSSLOT(%NIL,NIL%)) /> -> NEW;
CONSCREDNODE(NEW,NIL,NIL,NIL,0.5,0,0)->C;
        C->NEW.INSTCRED;C::CREDNODES->CREDNODES;
NEW->NEW.INSTNAME.VALOF;
NEW::SCHEMA.SCHINSTS->SCHEMA.SCHINSTS;
END;

VARS MAKEINST;GETNEWINST FNCOMP ERASE->MAKEINST;

FUNCTION MAKESCHEMA SCHNAME L;
VARS SCHEMA SLOTSIZE NAME N  KSTRIP;
4->N;
</ UNDEF, UNDEF, UNDEF,
        LOOPIF L.ISLINK
        THEN POP L->NAME;NAMESLOT(N,NAME);
                IF L.ISLINK AND L.HD.ISINTEGER
                THEN POP L->SLOTSIZE;
                ELSE 1->SLOTSIZE
                CLOSE;
        CONSSLOTKNOWLEDGE(NAME,SLOTSIZE,NIL,NIL);
        N+1->N;
        CLOSE /> ->KSTRIP;
CONSSCHEMA(SCHNAME,KSTRIP,NIL,0)->SCHEMA;
SCHEMA::SCHEMAS->SCHEMAS;
SCHEMA->SCHNAME.VALOF;
END;



        *** CODE FOR CREATING KEYS FROM THE   ***
        *** BOND AND CONDITION SPECIFICATIONS ***


RECORD EXTRABOND EBSOURCE EBFUN EBGOAL;
RECORD KEYNODE KNBINDING KNCONDS KNGEN KNEXTRAS;
RECORD RULE RULENUM 0 RULEKEY RULEACTION;

FUNCTION UNPACKBONDS L;
 COMMENT'this destructively alters l substituting two one
 way specifications for one two way one!;
VARS R B;
UNTIL L.NULL
DO L.HD->B;
        IF B.LENGTH=4
        THEN ITEM(4,B)->R;NIL->B.TL.TL.TL;
                [%B.TL.TL.HD,R,B.HD%]::L.TL->L.TL;
                L.TL.TL->L;
        ELSE L.TL->L;
        CLOSE;
ENDDO;
END;
```

```
FUNCTION COMESFIRST A B L;
L RHLOOP;
   IF RH=B THEN FALSE;RETURN ELSEIF RH=A THEN TRUE;EXIT;
ENDRH;
.POPERR;
END;

FUNCTION GETORDEREDNODES BONDSPECS=>REACHABLE SPECS;
 COMMENT'this takes a list of bond specifications and
 ensures that the firstnode in each bond can be reached
 from a previously mentioned node. i. e. it
 will reorder [ [a fun b] [c fun d] [b fun c] ].
 it also returns a list of node names!;
VARS SUSPECT B;
        FUNCTION TRYADD B;
        VARS X;B.TL.TL.HD->X;
        UNLESS MEMBER(X,REACHABLE)
        THEN NCJOIN(REACHABLE,X::NIL)->REACHABLE CLOSE;
        END;
POP BONDSPECS->B;[%B.HD,B.TL.TL.HD%]->REACHABLE;
NIL->SUSPECT;
[%B,
  LOOPIF BONDSPECS.ISLINK
  THEN POP BONDSPECS->B;
        IF MEMBER(B.HD,REACHABLE)
        THEN B;B.TRYADD;
                SUSPECT RHLOOP;
                        IF MEMBER(RH.HD,REACHABLE)
                        THEN RH;RH.TRYADD;
                        REMOVE(RH,SUSPECT)->SUSPECT
                        CLOSE;
                ENDRH;
        ELSE B::SUSPECT->SUSPECT
        CLOSE;
  CLOSE%]->SPECS;
UNLESS SUSPECT.NULL
THEN 'INVALID KEY SPECIFICATION. CULPRITS: !.PRSTRING;
     SUSPECT.PR;
        .POPREADY;
CLOSE;
END;

FUNCTION GETNAMEDNODE W;
 COMMENT'assumes global keylist!;
KEYLIST RHLOOP;
IF RH.KNBINDING=W THEN RH;EXIT;
ENDRH;
FALSE;
END;
```

```
 COMMENT'some complex bond specifications are split into
 bonds and tests for conditions.i.e. [a spouse =none] is
 handled by translating it into something like:
        [a spouse b] and [.equal b none]
 so dummy names (like "b") are needed.!;

VARS DIFFERENT NEXTDUMMYNAME;
EQUAL FNCOMP NOT->DIFFERENT;
GENSYM("DUMMYNODE")->NEXTDUMMYNAME;

FUNCTION EXTRACTCOND BONDSPEC;
 COMMENT'this looks for a special symbol( = or /=) before
 the second node and destructively changes the bondspec and
 stacks the required condition!;
VARS W LASTBIT FUN;
BONDSPEC.TL.TL->LASTBIT;LASTBIT.HD->W;
IF W="=" THEN EQUAL->FUN
ELSEIF W="/=" THEN DIFFERENT->FUN
ELSE RETURN
CLOSE;
.NEXTDUMMYNAME->W;[%W%]->BONDSPEC.TL.TL;
[%FUN,W,LASTBIT.TL.HD%];
END;

FUNCTION GETCONDARG W;
 COMMENT'the arguements specified in a cond may or may
 not be keynodes!;

VARS X;W.GETNAMEDNODE->X;
IF X THEN X ELSE W CLOSE;
END;


FUNCTION LASTNODE L M;
 COMMENT'returns the member of l which occurs last in m!;
COMMENT'if no member of l occurs in m then this
returns m.hd!;
M.REV->M;
UNTIL M.TL.NULL OR MEMBER(M.HD,L) DO M.TL->M ENDDO;
M.HD;
END;

FUNCTION ADDKEYCOND COND;
 COMMENT'cond is turned into a list of keynodes or other
arguments preceded by a function and stored under the last
named node. keylist is assumed to be global.!;
VARS F L K;
COND.RIG->COND;
IF COND.HD.ISWORD THEN COND.HD.VALOF->F ELSE COND.HD->F
CLOSE;
MAPLIST(COND.TL,GETCONDARG)->L;
LASTNODE(L,KEYLIST)->K;
(F::L)::K.KNCONDS->K.KNCONDS;
END;
```

```
FUNCTION ADDKEYBOND BOND NODENAMES;
 COMMENT'this takes a bond specification, and uses it to
 modify the key appropriately.
 a bond specification may contain either a function
 or a word for the slotfun!;
VARS KNA KNB A F B;BOND.DL->B->F->A;
UNLESS F.ISFUNC THEN F.VALOF->F CLOSE;
A.GETNAMEDNODE->KNA;B.GETNAMEDNODE->KNB;
IF COMESFIRST(A,B,NODENAMES)
THEN IF KNB.KNGEN=UNDEF
     THEN CONSPAIR(F,KNA)->KNB.KNGEN
     ELSE CONSEXTRABOND(KNA,F,KNB)::KNB.KNEXTRAS
          ->KNB.KNEXTRAS
     CLOSE;
ELSE CONSEXTRABOND(KNA,F,KNB)::KNA.KNEXTRAS->KNA.KNEXTRAS
CLOSE;
END;

FUNCTION MAKEKEY NODENAMES CONDS BONDS=>KEYLIST;
 COMMENT'during the creation of the keynodes we keep their
 names in knbinding!;
MAPLIST(NODENAMES,CONSKEYNODE(%NIL,UNDEF,NIL%))->KEYLIST;
APPLIST(CONDS,ADDKEYCOND);
APPLIST(BONDS,ADDKEYBOND(%NODENAMES%));
END;

FUNCTION MAKERULE N BONDS CONDS ACTION;
VARS KEYLIST NODENAMES;
MAPLIST(BONDS,RIG)->BONDS;
MAPLIST(BONDS,EXTRACTCOND)<>CONDS->CONDS;
BONDS.UNPACKBONDS;BONDS.GETORDEREDNODES->BONDS->NODENAMES;
MAKEKEY(NODENAMES,CONDS,BONDS)->KEYLIST;
CONSRULE(N,KEYLIST,
        POPVAL([%"LAMBDA","FROZRULE","FROZBONDS",
                NODENAMES.DL,";",ACTION.DL,"END",";"%]));
END;

FUNCTION TRYFRESHRULE INST F L;
 COMMENT'when a new rule is added to a schema, this tries
 to match its key to all the existing instances in the
 appropriate slot
 of all instances of the schema!;
APPLIST(INST.F.SLOTBONDS,
        LAMBDA B;
        STARTKEY(L,INST,OTHERINST(B,INST));
        END);
END;
```

```
FUNCTION ADDRULE SCHEMA RULE ;
 COMMENT'for adding rules to schemas so that when an
 instance of one of the schemas is created, each slot in it
  will be able to look at the corresponding component of
 schknowledge to find its initial rules!;
VARS SK L F;RULE.RULEKEY.TL.HD.KNGEN.FRONT->F;
SCHEMA.SCHKNOWLEDGE.F->SK;
[%RULE%]->L;
L::SK.SKRULES->SK.SKRULES;
APPLIST(SCHEMA.SCHINSTS,TRYFRESHRULE(%F,L%));
END;

OPERATION 4 ==> LHS RHS;
VARS SCHEMA RULE N BONDS CONDS;
NIL->BONDS;NIL->CONDS;POP LHS->N;
POP LHS->SCHEMA;
APPLIST(LHS,LAMBDA L;
        IF L.HD="." THEN L.TL::CONDS->CONDS
        ELSE L::BONDS->BONDS;
        CLOSE;END);
MAKERULE(N,BONDS,CONDS,RHS)->RULE;
IF SCHEMA.ISWORD THEN ADDRULE(SCHEMA.VALOF,RULE)
ELSE APPLIST(SCHEMA,VALOF FNCOMP ADDRULE(%RULE%))
CLOSE;
END;



        *** SOME MISCELLANEOUS FUNCTIONS ***



FUNCTION OTHERINST BOND INST;
 COMMENT'halfbonds have bondinst2=undef!;
VARS X;BOND.BONDINST1->X;
IF X=INST THEN BOND.BONDINST2 ELSE X CLOSE;
END;


VARS WHERESLOTSSTART;4->WHERESLOTSSTART;

FUNCTION ISINSTANCE X;
X.ISSTRIP AND X.INSTOF.DATAWORD="SCHEMA"
END;

FUNCTION BEFORE A B;
 COMMENT'checks whether a was made before b.
        the instances in a schema are in reverse order!;
A.INSTOF.SCHINSTS RHLOOP;
   IF RH=B AND RH/=A THEN TRUE;RETURN
   ELSEIF RH=A THEN FALSE;EXIT;
ENDRH;
.POPERR;
END;
```

```
        *** CODE FOR CREATING AND ***
        *** MANIPULATING  BONDS   ***


FUNCTION FILLERS INST SLOTORFUN;
IF SLOTORFUN.ISFUNC THEN INST.SLOTORFUN->SLOTORFUN CLOSE;
MAPLIST(SLOTORFUN.SLOTBONDS,
         LAMBDA B;OTHERINST(B,INST);END);
END;


FUNCTION GETBOND SOURCE FUN GOAL;
SOURCE.FUN.SLOTBONDS RHLOOP;
        IF OTHERINST(RH,SOURCE)=GOAL THEN RH;EXIT;
ENDRH;
FALSE;
END;



FUNCTION GETTHEBOND SOURCE FUN GOAL;
GETBOND(SOURCE.KNBINDING,FUN,GOAL.KNBINDING);
END;

FUNCTION GETEXTRABONDS K;
APPLIST(K.KNEXTRAS,DESTEXTRABOND FNCOMP GETTHEBOND);
END;

FUNCTION GETGENBOND K;
VARS B;K.KNGEN->B;
GETTHEBOND(B.BACK,B.FRONT,K);
END;

FUNCTION GETBONDSUSED KEYLIST;
 COMMENT'this assumes that the nodes in keylist are
 correctly bound and returns all the bonds used in matching
  the key!;
[%APPLIST(KEYLIST.TL,GETGENBOND),
  APPLIST(KEYLIST,GETEXTRABONDS)%];
END;


FUNCTION COMMONMEM LL;
VARS COMMON;
FILTLIST(LL.HD,LAMBDA X;
                ALLTRUE(LL.TL,LAMBDA L;MEMBER(X,L);END);
                END)->COMMON;
IF COMMON.LENGTH/=1 THEN .POPERR CLOSE;
COMMON.HD;
END;
```

```
FUNCTION COMMONCEIL BONDS;
MAPLIST(BONDS,BONDCRED FNCOMP CEILINGS).COMMONMEM;
END;

FUNCTION ONEFILLERCONSTR BOND OLDBONDS;
 COMMENT'this type of constraint is only added if there
 is more than one filler. if a constraint already exists
 it is modified to include the new bond!;
VARS COM C;BOND.BONDCRED->C;
IF OLDBONDS.NULL THEN
ELSEIF OLDBONDS.TL.NULL
THEN ATMOSTONE([%C,OLDBONDS.HD.BONDCRED%])
ELSE OLDBONDS.COMMONCEIL->COM;C::COM.CONCEILINGS
     ->COM.CONCEILINGS;
        COM::C.CEILINGS->C.CEILINGS
CLOSE;
END;

FUNCTION ADDBOND INST FUN BOND;
 COMMENT'puts the bond in the slot and adds the constraint
 that the instance must be at least as true
 as the bond.  it also adds the
 constraint between the fillers of the slot,where
  applicable!;
VARS SLOT SLOTSIZE OLDBONDS;INST.FUN->SLOT;
SLOT.SLOTBONDS->OLDBONDS;
BOND::OLDBONDS->SLOT.SLOTBONDS;
MORECRED(INST.INSTCRED::NIL,BOND.BONDCRED::NIL);
INST.INSTOF.SCHKNOWLEDGE.FUN.SKSIZE->SLOTSIZE;
IF SLOTSIZE=1 THEN ONEFILLERCONSTR(BOND,OLDBONDS) CLOSE;
 COMMENT'assumes new credval=0!;
END;

FUNCTION RETURNBOND INST1 FUN1 INST2 FUN2=>B;
 COMMENT'this either returns an existing bond, or if
 there is none,it makes a new one.
 if inst2 isnt an instance fun2 must be undef!;
VARS CREDNODE;
GETBOND(INST1,FUN1,INST2)->B;
IF B AND (FUN2=UNDEF OR MEMBER(B,INST2.FUN2.SLOTBONDS))
 THEN EXIT;
CONSBOND(INST1,INST2,UNDEF)->B;
CONSCREDNODE(B,NIL,NIL,NIL,0.5,0,0)->CREDNODE;
        CREDNODE::CREDNODES->CREDNODES;CREDNODE->B.BONDCRED;
ADDBOND(INST1,FUN1,B);
IF FUN2/=UNDEF
THEN ADDBOND(INST2,FUN2,B);
CLOSE;
RUNTRIGS(INST1,FUN1,INST2);
IF FUN2/=UNDEF
THEN RUNTRIGS(INST2,FUN2,INST1) CLOSE;
END;
```

```
FUNCTION MAKEBOND;
.RETURNBOND.ERASE;
END;

VARS LINK;MAKEBOND(%UNDEF%)->LINK;
```

*** CODE FOR MAKING AND RUNNING JOBS ***

```
VARS JOBSRUN TRIGTHRESH;0->JOBSRUN;0.7->TRIGTHRESH;

FUNCTION JOBRULE J;
FROZVAL(1,J);
END;

FUNCTION JOBBONDS J;
FROZVAL(2,J);
END;

VARS JOBLIST;NIL->JOBLIST;

FUNCTION CHECKBONDVALS JOB;
 COMMENT'this either returns true or puts the job in a list
 on the crednode of an implausible bond!;
VARS CRED;
JOB.JOBBONDS RHLOOP;
        RH.BONDCRED->CRED;
        IF CRED.CREDVAL=<TRIGTHRESH
        THEN JOB::CRED.CREDJOBS->CRED.CREDJOBS;FALSE;
        EXIT;
ENDRH;
TRUE;
END;

FUNCTION ADDJOB J;
NCJOIN(JOBLIST,J::NIL)->JOBLIST;
END;

FUNCTION RUNJOB J;
J.APPLY;1+JOBSRUN->JOBSRUN;
END;

FUNCTION TRYDORMANTJOB JOB;
IF JOB.CHECKBONDVALS THEN JOB.ADDJOB CLOSE;
END;

FUNCTION TRYACTIVEJOB JOB;
 COMMENT'assumes that the job has been removed from joblist!;
IF  JOB.CHECKBONDVALS THEN JOB.APPLY; 1+JOBSRUN->JOBSRUN;
CLOSE;
END;
```

```
FUNCTION TRYJOB J;
J.TRYACTIVEJOB;REMOVE(J,JOBLIST)->JOBLIST;
END;

FUNCTION TRYALLJOBS;
VARS L;JOBLIST->L;NIL->JOBLIST;
APPLIST(L,TRYACTIVEJOB);
END;

FUNCTION TRYJOBN N;
ITEM(N,JOBLIST).TRYJOB;
END;


FUNCTION ADDSAMEFILLERJOB T INST FUN FILLER;
 COMMENT'this adds the job to infer the appropriate bond
 when a samefiller demon is activated.jobs are assumed to
 be closures of functions with frozrule and frozbonds as
 their first two formal parameters,so the function
 sfenviron is provided!;
        FUNCTION SFENVIRON FROZRULE FROZBONDS BONDSPEC;
        INFER(BONDSPEC);
        END;
SFENVIRON(%T.SFRULE,GETBOND(INST,FUN,FILLER)::T.SFBONDS,
                [%T.SFOTHERINST.CONSREF,
          T.SFOTHERFUN.CONSREF,FILLER.CONSREF%] %).ADDJOB;
 COMMENT'the function that interprets bondspecs expects
  words or references!;
END;

FUNCTION ADDRULEJOB RULE;
 COMMENT'assumes that the key will be bound!;
VARS KEY;RULE.RULEKEY->KEY;
RULE.RULEACTION(%RULE,KEY.GETBONDSUSED,
                  APPLIST(KEY,KNBINDING)%).ADDJOB;
END;
```

*** CODE FOR MAKING AND TRIGGERING DEMONS ***

```
RECORD EBTRIG EBTGOAL EBTREM EBTBINDINGS;

 COMMENT'ebtrig records are used as demons which
 wait for extra bonds, i.e. ones not used to
 generate candidate bindings for the next keynode!
 'these records sit on a slot in one instance and
 wait for a bond to another particular instance
 (ebtgoal). the remaining extra bond needed from the
 instance are held in ebtrem, and the bindings of
 previous keynodes in ebtbindings.!;


VARS ISKEYNODE;
SAMEDATA(%CONSKEYNODE(NIL,NIL,NIL,NIL)%)->ISKEYNODE;



FUNCTION CHECKCOND COND;
 COMMENT'assumes cond is a list of keynodes
 words or integers preceded by a function!;
VARS FUN;COND.HD->FUN;
COND.TL RHLOOP;
   IF RH.ISKEYNODE THEN RH.KNBINDING ELSE RH CLOSE;
ENDRH;.FUN;
END;

FUNCTION CHECKCONDS KEYNODE;
ALLTRUE(KEYNODE.KNCONDS,CHECKCOND);
END;

FUNCTION EBPRESENT EB;
 COMMENT'checks that an extra bond is present assuming that
 the keynodes have the right bindings!;
 COMMENT'information about the required extra bonds
 is kept in a keynode in an extrabond record. the
 keynode is in ebsource, and ebgoal contains another
 keynode. the extra bond must be between the instances
 bound to these two key nodes, and should be in the
 ebfun slot of the ebsource instance!;
GETBOND(EB.EBSOURCE.KNBINDING,EB.EBFUN,EB.EBGOAL.KNBINDING);
END;
```

```
FUNCTION CHECKEXTRABONDS EXTRAS BINDINGS;
 COMMENT'tests whether all the extras are present.if not it
 leaves a demon on the appropriate slot!;
VARS EB SLOT;
IF EXTRAS.NULL THEN TRUE
ELSEIF EXTRAS.HD.EBPRESENT
THEN CHECKEXTRABONDS(EXTRAS.TL,BINDINGS)
ELSE EXTRAS.HD->EB;
        (EB.EBFUN)(EB.EBSOURCE.KNBINDING)->SLOT;
        CONSEBTRIG(EB.EBGOAL.KNBINDING,EXTRAS.TL,BINDINGS)
        ::SLOT.SLOTTRIGS->SLOT.SLOTTRIGS;
        FALSE;
CLOSE;
END;


FUNCTION CANBIND INST KEYNODE BINDINGS;
MEMBER(INST,BINDINGS).NOT AND
(INST->KEYNODE.KNBINDING;KEYNODE.CHECKCONDS) AND
CHECKEXTRABONDS(KEYNODE.KNEXTRAS,BINDINGS);
END;



FUNCTION TRYTOBIND REMKEY RULE BINDINGS;
 COMMENT'this attempts to bind the remaining keynodes.
 it generates candidate instances for a keynode by looking
 at the instances filling the slot specified by the knbond
 in kngen. it also leaves a demon on this slot in case more
 fillers turn up later!;
VARS INST KEYNODE GENBOND SOURCEINST GENSLOT;
IF REMKEY.NULL THEN RULE.ADDRULEJOB;EXIT;
 COMMENT'when a match succeeds a job is made!;
REMKEY.HD->KEYNODE;KEYNODE.KNGEN->GENBOND;
GENBOND.BACK.KNBINDING->SOURCEINST;
(GENBOND.FRONT)(SOURCEINST)->GENSLOT;
BINDINGS::GENSLOT.SLOTTRIGS->GENSLOT.SLOTTRIGS;
 COMMENT'bindings is a list whose last element is a
 rule.implementing demons this way is economical
 because descendants of a demon can be have one new
 binding and a pointer back to the smaller demon
 i.e. the tail of a demon is its parent!;
GENSLOT.SLOTBONDS RHLOOP;
   OTHERINST(RH,SOURCEINST)->INST;
   IF CANBIND(INST,KEYNODE,BINDINGS)
   THEN TRYTOBIND(REMKEY.TL,RULE,INST::BINDINGS)
   CLOSE;
ENDRH;
END;
```

```
FUNCTION REBIND BINDINGS=>RULE REMKEY;
 COMMENT'used for rebinding keynodes when a demon fires.
 remkey will be the nodes not yet bound!;
VARS X;
"ENDOFBINDINGS",BINDINGS.DL->RULE;
RULE.RULEKEY->REMKEY;
UNTIL (->X;X="ENDOFBINDINGS")
DO X->REMKEY.HD.KNBINDING;REMKEY.TL->REMKEY;
ENDDO;
END;

FUNCTION STARTKEY RULELIST STARTINST NEWINST;
 COMMENT'rulelist is a list of the rules whose keys
 can start matching when a filler (newinst) is put
 in the appropriate slot of an instance (startinst).
 if binding the instances to the keynodes violates
 a condition in the key, the match fails before
 calling trytobind. so no demons are set up unless
 at least two instances and a bond between them
 fits the key. this avoids many demons.!;
VARS RULE  KEYLIST;
RULELIST.HD->RULE;RULE.RULEKEY->KEYLIST;
IF CANBIND(STARTINST,KEYLIST.HD,RULELIST)
AND CANBIND(NEWINST,KEYLIST.TL.HD,STARTINST::RULELIST)
THEN TRYTOBIND(KEYLIST.TL.TL,RULE,
            NEWINST::(STARTINST::RULELIST))
CLOSE;
END;

FUNCTION GBCONTINUE NEWINST BINDINGS;
 COMMENT'called when a new instance fills a slot
 which has a demon on it!;
VARS NEWBINDINGS REMKEY RULE;BINDINGS.REBIND->REMKEY->RULE;
NEWINST::BINDINGS->NEWBINDINGS;
IF CANBIND(NEWINST,REMKEY.HD,BINDINGS)
THEN TRYTOBIND(REMKEY.TL,RULE,NEWBINDINGS)
CLOSE;
END;

FUNCTION EBCONTINUE REMEB BINDINGS;
 COMMENT'called when the required instance fills a
 slot which has a demon waiting for an extra bond.!;
VARS REMKEY RULE;BINDINGS.REBIND->REMKEY->RULE;
IF CHECKEXTRABONDS(REMEB,BINDINGS)
THEN TRYTOBIND(REMKEY,RULE,BINDINGS)
CLOSE;
END;
```

```
FUNCTION RUNTRIGS STARTINST FUN NEWINST;
 COMMENT'demons are of two kinds. one is looking for a
 candidate for the next keynode and is represented by a
 list of the bindings so far sitting on the slot from which
 the next knbinding will have to be generated! 'the other
 is looking for an extra bond involving the last bound
 keynode and is represented by an ebtrig record containing
 the goal instance, the remaining extrabonds in the last
 bound keynode, and the bindings. the record sits on the
 appropriate slot of the bonds source instance.! 'in both
 cases the bindings list has the rule as last item!
 'finally, the rules in the schema need to be examined in
 case any key matches start with the new bond!;
 VARS SLOT;STARTINST.FUN->SLOT;
APPLIST(SLOT.SLOTKNOWLEDGE.SKRULES,
          STARTKEY(%STARTINST,NEWINST%));
APPLIST(SLOT.SLOTTRIGS,
        LAMBDA T;
        IF T.ISLIST THEN GBCONTINUE(NEWINST,T)
        ELSEIF T.DATAWORD="SAMEFILLER"
        THEN ADDSAMEFILLERJOB(T,STARTINST,FUN,NEWINST)
        ELSEIF T.EBTGOAL=NEWINST
        THEN REMOVE(T,SLOT.SLOTTRIGS)->SLOT.SLOTTRIGS;
             EBCONTINUE(T.EBTREM,T.EBTBINDINGS)
        CLOSE;
        END);
END;
```

```
        *** CODE FOR MAKING CONSTRAINTS        ***
        *** (MOSTLY LISTED IN PUPPET PROGRAM) ***


RECORD CONSTR CONVIOL 0 HYPLENGTH 0 OLDCONVIOL 0
      CONFLOORS CONCEILINGS;
 COMMENT'constraints have been given extra fields
 compared with the puppet program. the fields
 conceilings and confloors are used to hold lists
 of the nodes whose supposition values may be
 held down or held up by the constraint!;


FUNCTION INFERCONSTR L B;
RETURNCONSTR(B::NIL,L,1-L.LENGTH);
END;

FUNCTION DENYCONSTR L B;
RETURNCONSTR(NIL,B::L,(-L.LENGTH));
END;

FUNCTION NOTALLCONSTR L;
RETURNCONSTR(NIL,L,1-L.LENGTH);
END;
```

```
        *** CODE FOR THE FUNCTIONS USED IN ***
        *** THE ACTION PARTS OF RULES      ***


RECORD SAMEFILLER SFRULE SFBONDS SFOTHERINST SFOTHERFUN;

FUNCTION ADDANDTRYSFDEMON INST FUN TOTHERINST OTHERFUN;
 COMMENT'this adds a samefiller demon to a slot and also
         runs the demon on all existing fillers!;
VARS T S;INST.FUN->S;
CONSSAMEFILLER(FROZRULE,FROZBONDS,TOTHERINST,OTHERFUN)->T;
T::S.SLOTTRIGS->S.SLOTTRIGS;
APPFILLERS(INST,S,
        LAMBDA FILLER;ADDSAMEFILLERJOB(T,INST,FUN,FILLER);
        END);
END;

FUNCTION SAMEFILLER INSTA SFA INSTB SFB;
 COMMENT'this assumes it is called in the action part
         of a rule!;
UNLESS INSTA.ISINSTANCE AND INSTB.ISINSTANCE THEN EXIT;
ADDANDTRYSFDEMON(INSTA,SFA,INSTB,SFB);
ADDANDTRYSFDEMON(INSTB,SFB,INSTA,SFA);
END;

FUNCTION EVALSPEC X;
IF X.ISWORD THEN X.VALOF
ELSEIF X.DATAWORD="REF" THEN X.CONT
ELSE .POPERR CLOSE;
END;

FUNCTION CASHSPEC L;
 COMMENT'takes a bond specification and returns false,
       or true and the bond!;
VARS X INST1 INST2 SF1 SF2;
EVALSPEC(POP L)->INST1;
UNLESS INST1.ISINSTANCE THEN  0;EXIT;
EVALSPEC(POP L)->SF1;POP L ->X;
IF X="=" THEN POP L->INST2 ELSE X.EVALSPEC->INST2 CLOSE;
IF INST2.ISINSTANCE
THEN IF L.ISLINK THEN L.HD.EVALSPEC->SF2;
        ELSE SF1.INVERSE->SF2 CLOSE
ELSE UNDEF->SF2
CLOSE;
RETURNBOND(INST1,SF1,INST2,SF2);TRUE
END;


FUNCTION CLAIM LIST PREF;
VARS CRED;LIST.CASHSPEC.ERASE.BONDCRED->CRED;
PREF+CRED.CREDPREF->CRED.CREDPREF;
END;
```

```
FUNCTION MAKESOFTCONSTR FLIST CLIST N PENALTY;
VARS PENNODE C;
IF PENALTY
THEN CONSCREDNODE(UNDEF,NIL,NIL,NIL,0,0,-PENALTY)
        ->PENNODE;
        PENNODE:::CREDNODES;PENNODE::FLIST->FLIST;
CLOSE;
RETURNCONSTR(FLIST,CLIST,N)->C;
IF PENALTY THEN C->PENNODE.CREDOBJ CLOSE;
END;

FUNCTION SOFTINFERBOND B PENALTY;
VARS CLIST C;
MAPLIST(FROZBONDS,BONDCRED)->CLIST;
MAKESOFTCONSTR(B.BONDCRED::NIL,CLIST,1-CLIST.LENGTH,
                PENALTY);
END;

FUNCTION SOFTDENYBOND B PENALTY;
VARS CLIST C;
B.BONDCRED::MAPLIST(FROZBONDS,BONDCRED)->CLIST;
MAKESOFTCONSTR(NIL,CLIST,0-CLIST.LENGTH,PENALTY);
END;

FUNCTION SOFTCONTRADICTION PENALTY;
VARS CLIST C;
MAPLIST(FROZBONDS,BONDCRED)->CLIST;
MAKESOFTCONSTR(NIL,CLIST,1-CLIST.LENGTH,PENALTY);
END;

FUNCVAR CONTRADICTION SOFTCONTRADICTION(%0%);


 COMMENT'there are several formats for inferring or denying
  a bond.the bond,or its instances and functions, or a list
 of them, can all be used!;

FUNCTION SOFTINFER4 P;
SOFTINFERBOND(.RETURNBOND,P);
END;

FUNCTION SOFTDENY4 P;
SOFTDENYBOND(.RETURNBOND,P);
END;

FUNCTION SOFTINFER L PEN;
IF L.CASHSPEC THEN PEN.SOFTINFERBOND CLOSE;
END;

FUNCTION SOFTDENY L PEN;
IF L.CASHSPEC THEN PEN.SOFTDENYBOND CLOSE;
END;

VARS INFER DENY INFERBOND DENYBOND INFER4 DENY4;
SOFTINFER(%0%)->INFER;SOFTDENY(%0%)->DENY;
SOFTINFER4(%0%)->INFER4;SOFTDENY4(%0%)->DENY4;
SOFTINFERBOND(%0%)->INFERBOND;SOFTDENYBOND(%0%)->DENYBOND;
```

```
        *** CODE FOR RUNNING RELAXATION        ***
        *** (MOSTLY LISTED IN PUPPET PROGRAM) ***


FUNCTION UPFORCE C=>SUM;
 COMMENT'this computes the total force on c due to
          constraints!;
 COMMENT'more efficient than the separate functions
 used in the puppet program!;
VARS V;0->SUM;
C.CEILINGS RHLOOP;
        RH.CONVIOL->V;
        IF V>0 THEN SUM-V/RH.HYPLENGTH->SUM CLOSE;
ENDRH;
C.FLOORS RHLOOP;
        RH.CONVIOL->V;
        IF V>0 THEN SUM+V/RH.HYPLENGTH->SUM CLOSE;
ENDRH;
END;


FUNCTION RUNMORE STEPS;
STEPS<*.MOVE;.TRYALLJOBS;CREDNODES.REV.SHOWCREDS;*>;
CREDNODES.REV.SHOWNAMES;
END;

0.2->PCOEFF;
0.5->DCOEFF;
0.5->FCOEFF;
0.05->HCOEFF;


FUNCTION SETTLE CLEARROUNDS;
 COMMENT'after each round of relaxation this shows the
 number of dormant jobs
 aroused,and the number of jobs run and stored
 by a tryalljobs,and the number
 of new jobs created by those run!;
VARS N TOTAL ROUSED MADE;0->N;JOBLIST.LENGTH->MADE;
'
ROUSED RUN STORED MADE
!.PRSTRING;
UNTIL N=CLEARROUNDS
DO .MOVE;
        IF JOBLIST.ISLINK THEN 0->N ELSE N+1->N CLOSE;
        JOBLIST.LENGTH->TOTAL;TOTAL-MADE->ROUSED;
        0->JOBSRUN;
        .TRYALLJOBS;
        JOBLIST.LENGTH->MADE;
        INTPR(ROUSED,3);INTPR(JOBSRUN,4);
        INTPR(TOTAL-JOBSRUN,4);
       INTPR(MADE,4);
        1.NL;
ENDDO;
END;
```

BIBLIOGRAPHY

Adler M.R. (1976)
Recognition of peanuts cartoons.
In Proc. A.I.S.B. Summer Conf. July 1970. pp 1-13.

Amarel S. (1968)
On representations of problems of reasoning about actions.
In Machine Intelligence 3.
Ed. Michie, pp 131-172.
Edinburgh University Press, Edinburgh.

Ambler A.P., Barrow H.G., Brown C.M., Burstall R.M. & Popplestone R.J. (1975)
A versatile system for computer controlled assembly.
Artificial Intelligence 6, pp 129-156.

Barrow H.G., Ambler A.P. & Burstall R.M. (1972)
Some techniques for recognising structures in pictures.
In Frontiers of Pattern Recognition.
Ed Watanabe pp 1-29.
Academic Press, New York.

Barrow H.G & Tenenbaum J.M. (1976)
MSYS: A system for reasoning about scenes.
A.I. Center. Stanford Research Institute.

Bartlett F.C. (1932)
Remembering: A study in experimental and social psychology
Cambridge University Press, Cambridge.

Brady J.M. & Weilinga B.J. (1976)
Seeing a pattern as a character.
In Proc. A.I.S.B. Summer Conference.
University of Edinburgh, Edinburgh.

Bron C. & Kerbosch J. (1973)
Algorithm 457. Finding all cliques of an undirected graph (H).
Comm.Assoc.Comp.Mach. 16, No 9.

Burstall R.M., Collins J.S. & Popplestone R.J. (1971)
Programming in POP-2.
Edinburgh University Press, Edinburgh.

Clowes M.B. (1969)
Pictorial relationships - a syntactic approach.
In Machine Intelligence 4.
Ed. Meltzer and Michie, pp 361-383.
Edinburgh University Press, Edinburgh.

Clowes M.B. (1971)
On seeing things.
Artificial Intelligence, 2, pp 79-112.

Davis L.S. & Rosenfeld A. (1976)
Applications of relaxation labelling,
2: Spring-loaded template matching.
Technical Report 440.
Computer Science Center, University of Maryland.

Dreyfus H.L. (1972)
What computers can't do.
Harper & Row, New York.

Duda (1970)
Some current techniques for scene analysis.
Technical Note 46
A.I. Center, Stanford Research Institute.

Erman L.D. & Lesser, V.R. (1975)
A multi-level organisation for problem solving using
many diverse cooperating sources of knowledge.
In Proc. 4th Inter.Joint.Conf. on Artificial Intelligence.
pp 483-490.

Fikes R.E. (1970)
REF-ARF: A system for solving problems stated
as procedures.
Artificial Intelligence 1 pp 27-120

Freuder E. (1976)
Synthesizing constraint expressions.
M.I.T. A.I. Memo 378

Garfinkel R.S. & Nemhauser G.L. (1972)
Integer Programming
Wiley: New York

Gomory R.E. (1958)
An algorithm for integer solutions to linear programs.
Bull.Amer.Math.Soc. 64, pp 275-278.

Grape G.R. (1973)
Model-based (intermediate level) computer vision.
Stanford A.I.Memo AIM-201
Computer Science Dept. Stanford University.

Guzman A. (1968)
Decomposition of a visual scene into three-dimensional bodies.
A.F.I.P.S. Proc. Fall Joint Comp.Conf. 33, pp 291-304.

Guzman A. (1971)
Analysis of curved line drawings using context and global information.
In Machine Intelligence 6.
Ed. Meltzer & Michie, pp 325-376.
Edinburgh University Press, Edinburgh.

Hart D., Nilsson N. & Raphael B. (1968)
A formal basis for the heuristic determination of minimum cost paths.
IEEE Transactions.Sys.Sci. and Cybernetics.
Vol SSC-4 No 2 pp 100-107.

Hebb D.O. (1949)
The Organisation of Behaviour.
Wiley, New York.

Hewitt C. (1972)
Description and theoretical analysis (using schemata) of PLANNER.
Ph D thesis, M.I.T. AI Lab. AI-TR-258.

Hilbert D. & Cohn-Vossen S. (1952)
Geometry and the imagination.
Chelsea, New York.

Hochberg J. (1968)
In the mind's eye.
In Contemporary theory and research in visual perception. Ed. Haber

Huffman D.A. (1971)
Impossible objects as nonsense sentences.
In Machine Intelligence 6.
Ed. Meltzer and Michie, pp 295-323.
Edinburgh University Press, Edinburgh.

Julesz B. (1971)
Foundations of cyclopean perception.
University of Chicago Press, Chicago.

Kant I. (1781)
Critique of pure reason.
(Many editions).

Mackworth A.K. (1975)
Consistency in networks of relations.
Technical report 75-3.
Department of Computer Science, University of British Columbia, Vancouver.

Mackworth A.K. (1977)
How to see a simple world.
In Machine Intelligence 8.
Ed Elcock & Michie,.
Ellis Horwood Ltd., Chichester.

Marr D. (1975)
Analysing Natural Images.
M.I.T. A.I. Memo 334

Marr D. (1976)
Early processing of visual information.
Phil.Trans.Roy.Soc. B. 275 pp 483-524

Marr D. (1977)
Representing Visual Information.
M.I.T. A.I.Memo 415

Marr D. & Poggio T. (1976)
Cooperative computation of stereo disparity.
M.I.T. A.I. Memo 364.

Marx K. (1883)
Capital. Vol I.
Translated from the 1883 edition, Ed. Engels.
Lawrence & Wishart, London 1970.

Minsky M.L. (1975)
A framework for representing knowledge.
In The psychology of computer vision.
Ed. Winston P.H. pp 211-277.
McGraw-Hill, New York.

Minsky M.L. & Papert S. (1969)
Perceptrons: An introduction to computational geometry.
M.I.T. Press, Cambridge, Mass.

Minsky M.L. & Papert S. (1972)
Progress Report.
M.I.T. A.I. Memo 252.
Cambridge, Mass.

Narasimhan R. (1966)
Syntax-directed interpretation of classes of pictures.
In Comm. Assoc. Comp. Mach. 9.

Navon D. (1977)
Forest before trees: The precedence of global features in visual perception.
Cognitive Psychology 9, pp 403-411.

Neisser U. (1967)
Cognitive Psychology.
Appleton Century Crofts, New York.

Nilsson N.J. (1971)
Problem-solving in artificial intelligence.
McGraw-Hill, New York.

Paul J.L. (1977)
An image interpretation system.
D.Phil. thesis. Sussex University.

Perkins D.N. (1976)
How good a bet is good form?
Perception 5, pp 393-406.

Piaget J. (1954)
The construction of reality in the child.
Basic Books, New York.

Pierre D.A. (1969)
Optimisation theory with applications.
John Wiley & Sons Inc, New York.

Roberts L.G. (1965)
Machine perception of 3-D solids.
In Optical and electro-optical information processing.
Ed. Tippett et al pp 159-197.

Rosenfeld A., Hummel R.A. & Zucker S.W. (1975)
Scene labelling by relaxation operations.
Technical Report TR-379.
Computer Science Center, University of Maryland.

Rosenfeld A., Hummel R.A. & Zucker S.W. (1976)
Scene labelling by relaxation operations.
I.E.E.E. Trans. SMC-6 420.

Selfridge O.G. & Neisser U. (1960)
Pattern recognition by machine.
Scientific American 203 (Aug) pp 60-68.

Shirai Y. (1973)
A context sensitive line finder for recognition of polyhedra.
Artificial Intelligence 4 pp 95-120.

Sloman A. (1971)
Interactions between philosophy and artificial intelligence: The role of intuition and non-logical reasoning in intelligence.
Artificial Intelligence 2, pp 209-225.

Sloman A., Owen D., Hinton G. & O'Gorman F. (1977)
Popeye's progress through a picture.
Unpublished manuscript.
Cognitive Studies Programme, Sussex University.

Sloman A. & Hardy S. (1976)
Giving a computer Gestalt experiences.
In Proc A.I.S.B;. Summer Conference, pp 242-255.

Stallman R.M. & Sussman G.J. (1976)
Forward reasoning and dependency - directed backtracking in a system for computer-aided circuit analysis.
M.I.T. A.I. Memo 380.

Sussman G.J. & McDermott D. (1972)
Why conniving is better than planning.
M.I.T. A.I. Memo 255 A.

Turner K.J. (1974)
Computer perception of curved objects using a television camera.
Ph D Thesis, University of Edinburgh.

Waltz D.L. (1972)
Generating semantic descriptions from drawings of scenes with shadows.
MAC AI-TR-271.
M.I.T. Cambridge Mass.

Weisenbaum J. (1976)
"Computer thought and human reason".
W.H. Freeman, San Francisco.

Willshaw D.J. & Longuet-Higgins H.C. (1969)
Associative memory models.
In Machine Intelligence 5.
Ed Meltzer & Michie, pp 351-359.
Edinburgh University Press Edinburgh.

Winston P.H. (1970)
Learning structural descriptions from examples.
MAC AI-TR-76, M.I.T. Cambridge, Mass.

Winston P.H. (1972)
The M.I.T. Robot.
In Machine Intelligence 7.
Ed Meltzer & Michie, pp 431-462.
Edinburgh University Press, Edinburgh.

Winston P.H. (1977)
Artificial Intelligence.
Addison Wesley: New York

Woods W.A. et al (1976)
Speech understanding systems - final technical progress report. BBN Report No 3438 Vols 1-5
Bolt Beranek & Newman Inc. : Cambridge, Mass.

Woods W.A. (1977)
Shortfall and density scoring strategies for speech understanding control.
In Proc 5th Inter.Joint,Conf. on Artificial Intelligence. pp 18-26.
Available from Carnegie Mellon University.

Yakimovsky Y. & Feldman J. (1973)
A semantics-based decision theory region analyser.
Proc. 3rd IJCAI pp580-588.

Zucker S.W. (1976)
Relaxation labelling and the reduction of local ambiguities.
Technical report 451.
Computer Science Dept. University of Maryland.