

Preface

The following are the proceedings of a Workshop on Machine Learning which was held at the University of Calgary on July 24th and 25th, 1987. Brian Barrow and John Elbert had just attended the IEEE First Annual International Conference on Neural Networks in June, 1987 in San Diego, California. Peter Andrews from Victoria University of Wellington in New Zealand had just attended the Fourth International Workshop on Machine Learning in June, 1987 held at the University of California at Irvine.

The occasion of these two conferences plus Peter Andrews' subsequent visit to the University of Calgary provided an impetus for the workshop. A schedule of the workshop at Calgary follows this preface, and contains a list of all participants. The first day of the workshop contained a synopsis of the two conferences discussed above, followed by a discussion of the current state of the art in Neural Nets and Machine Learning. The second day contained a presentation of Machine Learning projects taking place at both the University of Calgary and Victoria University of Wellington. The second day ended with a discussion of these different Machine Learning projects and possible ways of integrating those projects together.

The authors wish to note that the text of the workshop in their presentation of what transpired. Care has been taken to try to present what was actually said, and by whom in some cases. Some extra background information was added at several points, in the attempt to make the proceedings more understandable by those not totally familiar with Neural Nets or Machine Learning.

Contents

Preface	ii
Workshop Schedule	iii
1 What's going on in neural nets and machine learning	1
1.1 About the neural net conference	1
1.1.1 Best keynote address	1
1.1.2 Stochastic computing	1
1.1.3 Commercial exhibition	1
1.1.4 Conclusions	1
1.2 Overview of papers presented at the neural net conference	2
1.2.1 Theory	2
1.2.2 Hardware	2
1.2.3 Software and applications	2
1.2.4 General comments	3
1.3 Overview of the 4th International Workshop on Machine Learning	3
1.3.1 Introduction	3
1.3.2 Similarity based learning in detail	4
1.3.3 Explanation based learning in detail	5
1.3.4 Problems for future work	5
1.4 Discussion of the current state of the art in neural nets and machine learning	6
1.4.1 What is "learning"?	6
1.4.2 What are the goals of Machine Learning?	6
1.4.3 What functionality should a Machine Learning system have?	7
1.4.4 Can we hope to understand neural nets (in terms of concepts we already have)?	7
1.4.5 Can we understand other learning algorithms in terms of neural nets?	8
1.4.6 Methodological issues	8
2 Machine learning Projects; System integration	9
2.1 From MARVIN to ALVIN	9
2.1.1 What is MARVIN?	9
2.1.2 Problems with MARVIN	10
2.1.3 ALVIN to the rescue	10
2.1.4 Advantages of ALVIN	11
2.2 Some complexity results for the Version Space algorithm	11
2.2.1 What is Version Space?	11
2.2.2 Different kinds of search spaces	12
2.2.3 Results	12
2.2.4 Observations and questions	13
2.3 Reconstructing Noddy	13
2.3.1 Description of Noddy	13
2.3.2 How Noddy Works	13
2.3.3 An Implementation in Prolog	15

2.4	Architecture of a Knowledge Acquisition System	15
2.4.1	Grid elicitation	15
2.4.2	Grid analysis tools	15
2.4.3	Current work	16
2.5	Machine Learning in Wellington	16
2.5.1	Old Noddy	16
2.5.2	New Noddy: Big Ears and Mr. Plod	17
2.5.3	Other projects at Wellington	17
2.5.4	Discussion	17
2.6	Discussion	18
2.6.1	MARVIN	18
2.6.2	Version Space	18
2.6.3	Noddy	19
2.6.4	Knowledge representation	19
2.6.5	CONCLUSION: What are we trying to do, and would we know when we achieved it?	19
A	Neural Net Conference	20
A.1	Neural Nets Conference Synopsis	21
A.2	MacBrain Information Sheet	22
A.3	Membership Application to Neural Network Society	23
B	ID3 Counter-Example	24
B.1	Page 1	25
B.2	Page 2	26
C	Hierarchies	27
C.1	World	28
C.2	Comparison of Information Technology and Neural Nets	29
D	From Marvin to Alvin	30
D.1	Presentation Outline - Page 1	31
D.2	Presentation Outline - Page 2	32
D.3	Introduction to Marvin	33
D.4	Learning Swimming	34
D.5	Constructing Crucial Objects in Marvin	35
D.6	Marvin overgeneralizes	36
D.7	Tree structure and partial order structure	37
D.8	Domain with tree structure and partial order structure	38
D.9	Choosing crucial or significant objects	39
D.10	Animal Kingdom	40
D.11	Learning 2-leggedness	41
D.12	Learning 2-leggedness	42
D.13	Learning 2-leggedness	43

E	Version Space	44
E.1	Outline	45
E.2	Observation	46
E.3	Open Questions	47
E.4	Example of Non-near miss	48
E.5	Example of Near miss	49
E.6	Hierarchical Search space	50
E.7	Attributive Search space	51
E.8	Boolean Search space	52
E.9	Specificity Ordering	53
F	Reconstructing Noddy	54
F.1	Outline	55
F.2	Introduction	56
F.3	Example Traces	57
F.4	Turtle Procedure	58
F.5	Function Induction	59
F.6	Building the Expression	60
F.7	Constants	61
F.8	Composition of Operators	62
F.9	Representation of Knowledge	63
F.10	Representation of the Expression	64
G	Kitten	65
G.1	Kitten Diagram	66
H	Noddy	67
H.1	Robot in Warehouse example	68
H.2	Finding Loops in Traces	69
H.3	Pushing Ors Down	70
H.4	Position, Force, Tool	71
H.5	Procedure Fill Pallet	72
H.6	Drill Hole	73
H.7	Fill pallet details	74
H.8	Blackboard details	75
I	Table of Contents from the Machine Learning Workshop	76
I.1	Title Page	77
I.2	Contents - Page 1	78
I.3	Contents - Page 2	79

1 What's going on in neural nets and machine learning

1.1 About the neural net conference

Brian Gaines gave a synopsis of the first international conference on neural nets, held in San Diego, in June 1987, organized by IEEE. This, the first fully recursive conference (see A1), almost didn't happen. Response to the call for papers was very slim, but the deadline was extended, and the conference did take place with over 2000 in attendance, and some 300 papers presented.

1.1.1 Best keynote address

Carver Mead presented a paper, "Silicon Models of Neural Computation", which in Brian's opinion was the best of the invited papers. Mead has designed CMOS circuits, operating in the analog region, which simulate visual retinas. This device gives a complete retina on a chip. The paper addressed three aspects of the retina:

1. What does a biological system do?
2. What does a silicon system do?
3. Why did the retina evolve as it did?

1.1.2 Stochastic computing

Brian and John Cleary both presented papers on stochastic computing, an area in which there has been little interest during the past 18-20 years. They noted a renewed interest in this topic, with work underway at a number of centers.

1.1.3 Commercial exhibition

Although there are few fully parallel neural net machines, the commercial exhibit featured numerous simulations for conventional machines: mainframes, micros and lisp-machines. PC's are really too slow, so a number of vendors proposed accelerator cards to be used in conjunction with their simulators. Brian suggested that with such systems, researchers could put together a neural net system "overnight", and speculated that some of the work presented at the conference may have come about through this methodology. One of the simulators, MacBrain©, has been ordered for KSI (see A2).

1.1.4 Conclusions

The conference provided no clear sense of direction, and no front runners. However, some major researchers did not present at the conference.

The study of neural nets seems to be "taking off" again, after a lapse of 20 years. (Now is the time to join the effort - see A3S.) There was some speculation for renewed interest in the area, including the notion of turn-over of those controlling research laboratories and funding agencies. It was suggested that Minsky and Papert's book on perceptrons had left no easy problems to solve, putting a damper on work in the area.

Currently, neural nets are somewhat isolated from other areas of computing. There was very little talk about actual applications; researchers were concerned mainly with learning algorithms and convergence properties.

The real highlight of the conference was Carver Mead's analog retina on a chip, but this was the only analog device discussed. The only really new results to come out of the last 20 years are new learning algorithms.

1.2 Overview of papers presented at the neural net conference

1.2.1 Theory

John Cleary noted that the theoretical papers at the conference involved proofs of convergence and stability properties of neural nets. Many were demonstrations that if the net had a certain property it is guaranteed to converge to a solution. He wondered whether these were even the right questions. Perhaps we don't know enough about neural nets yet to ask the right questions. Unstable systems exist in nature, such as the olfactory system in rabbits, which cannot be stable if the rabbit is to survive.

1.2.2 Hardware

There were two types of hardware presented at the conference; John also included some other possibilities.

1. Conventional floating point.

These were the majority at the conference. They are very expensive and slow for practical tasks, but, unlike integer systems, they do provide the precision often needed for the small weight increments used in learning algorithms. Hinton has looked at the amount of precision required, and found that 3 bits were not enough, but 5 were, provided there was sufficient noise in the system. There was some discussion about why precision could be traded off for noise in such a system. If a system converges to a wrong conclusion, noise may move it away from that point far enough that it can converge to a different point.

2. Analog circuit (Carver Mead)

These circuits operate in the analog region, under 1 volt. No learning is possible on this system, as the weights cannot be stored long enough. Connections are local, which is fine for the retina application, but won't be able to handle all cases, some of which require longer distance connections.

3. Optical computing

This looks promising, but we need better transducers. Those available currently are too slow, and too expensive to manufacture - indeed they cannot be mass produced reliably. Such systems are naturally noisy and parallel, so they may be good for neural nets.

4. CCD was mentioned as another promising possibility.

1.2.3 Software and applications

There were many papers on hand printing recognitions systems, and some on applications in vision, language, and speech. There were no huge breakthroughs. An advance in hardware is needed before

more difficult applications are addressed. Many papers left out the vital parameters which were adjusted to make the systems they described work. This would make it difficult to reproduce the results.

1.2.4 General comments

In this conference, attention was directed to the internal characteristics of nets, rather than directed outward to useful applications. Talk concentrated on convergence and stability, with little talk of time complexity, or of comparisons to traditional digital computers. There is no meta-theory for nets, such as software engineering principles: time required to learn, feasibility of practical applications, etc. Work is still concentrated on small examples that can be done as well or better on standard digital machines.

Carver Mead's system is very different from what has been thought of as nets in the past - it may very well spawn a whole new area of research.

1.3 Overview of the 4th International Workshop on Machine Learning

Peter Andreae gave a synopsis of machine learning from a "mainline AI" perspective, having just come from the workshop held at the University of California at Irvine in June. He began with an overview of the various techniques used in machine learning, then went into more detail on the two major ones, and finished with a list of problems for future research.

1.3.1 Introduction

1. Similarity based learning (SBL)

In similarity based learning (SBL), the system is presented with many examples and learns by finding a similarity the examples share, and uses this to form a generalization.

2. Explanation based learning (EBL)

In explanation based learning, the system has built-in domain knowledge which it uses to generalize from a single example by constructing an explanation of the important properties of the example in terms of its initial knowledge.

3. Analogy

Learning by analogy has been used in the areas of theorem proving, problem solving, planning, design, story understanding, and teaching. Unfortunately, there seems to be no theory on which to base this work.

4. Discovery

The use of such highly connotative words is suspect. In BACON, for example, the work is not well substantiated, and might more accurately be described as "heuristic curve fitting". Peter thought that the work done by Kokar in this field was good (see the Machine Learning journal, volume 1 number 4).

5. World modeling

Ron Rivest presented a paper in the workshop describing a system which could find regularities in an application with a very large number of states, provided the transitions were deterministic. This also included a complexity analysis.

6. Theory

7. Formal induction

8. Soar

This is a very large system from Carnegie Mellon University which claims to be able to do many things.

9. Connectionist architectures

1.3.2 Similarity based learning in detail

There are many established techniques which fit into this area:

1. Quinlan's ID3

This system builds decision trees from examples identified as attribute-value pairs. Recent additions to ID3 include noise handling by tree pruning, the use of soft thresholds in the trees to deal with probabilistic data, and the formation of rules from trees. ID3 is now commercially available for Expert Systems. Furthermore, there are large data bases of examples from which ID3 has successfully created decision trees, which people are sharing for research and experimental purposes.

2. Mitchell's Version Space

This system uses positive and negative examples to determine a concept description, which can later be used to decide if a new instance is in the concept. The algorithm eliminates candidate descriptions as new examples are given. Peter explained its operation using a diagram with two concentric circles: the outer circle representing the boundary between overgeneral descriptions (on the outside) and acceptable descriptions; and the inner circle separating too specific descriptions (inside) from the acceptable ones. A positive example eliminates some of the previously acceptable descriptions as too specific, enlarging the inner circle. On the other hand, a negative example may show that some of the descriptions which were acceptable before the example are too general, causing the outer circle to shrink (so as to exclude the overgeneralizations). So descriptions outside the outer circle are too general, those inside the inner circle are too specific, and we don't know yet about those in the ring between the circles. If this ring ever contains only a single description, then that is the description of the concept.

The conference contained a paper analyzing the complexity of the algorithm as a function of the description language. Attribute-value pairs and their conjunctions are fairly easy to deal with. When some disjunction is added things get more complicated. Fully general languages, relating parts and their attributes, cannot yet be dealt with by the version space algorithm.

3. Michalski's generalization heuristics

Michalski has a list of generalization heuristics, organized into two categories: selective heuristics make use of the descriptions given in the examples, while constructive heuristics require grouping of aspects not present in the descriptions given. Some examples of selective heuristics are: "drop an attribute", and "extend a value range".

4. Prototypes

This is one of the new ideas to come from the conference. For example, consider learning about chairs from particular chairs. The techniques consist of doing "lazy generalization" or "generalization on the fly" by seeing how good a partial match you can get. This requires a more complex representation language.

5. Get a core generalization

Core generalizations are used together with interpretation rules applied "on the fly". For example, the concept "toy chair" would use knowledge about toys and the general description of a chair and combine the two. Knowledge about toys guides the matcher, which can relax constraints as it goes.

1.3.3 Explanation based learning in detail

1. Mitchell's LEXII

This is a problem solving system which learns better rules by working backwards through a trace of rules used to solve a given problem. It recognizes when the use of certain rules was a good idea and forms a new rule stating the reason. This method requires much domain knowledge, and has the ability to operationalize declarative knowledge.

2. Winston's cups

As with LEXII, this work requires a large amount of background knowledge. Given a general description of a cup, and a functional description, Winston's method learns more about cups.

3. Pazzani

Pazzani has used a combination of SBL and EBL in story understanding. The method involves using SBL on a set of instances to get causal relations, which are then used to do EBL in new stories. This is a good first step towards some kind of "sustained learning", where that which has been learned is used to learn more.

1.3.4 Problems for future work

1. Relating structural, functional and behavioral descriptions

Structure and behavior can be observed, but we must infer the goals of a system from our observations.

2. Explanation based learning systems

Require a large amount of complex knowledge to learn some simple things. Is this a good idea?

3. Some problems not addressed at the workshop include:

- (a) retrieval of relevant knowledge from existing knowledge bases
- (b) indexing new knowledge into a knowledge base
- (c) learning of new knowledge representations
- (d) automatically choosing the right knowledge representation for a domain in question.

1.4 Discussion of the current state of the art in neural nets and machine learning

The workshop reconvened for discussion of several questions, prepared by Ian and an ad hoc committee during the lunch break. Two things quickly became evident. First, the discussion generated more questions than answers; and second, the participants seated on the left side of the room took an active, verbal role in the discussion, while those on the right side of the room produced vivid mental images which they declined to articulate.

1.4.1 What is "learning"?

This question began as an attempt to define a number of terms, but only the first of these, "learning", was actually discussed before time ran out. Maurice suggested that learning is "an increase in knowledge over time as a result of some internal processes".

The group quickly generalized this to read "a change in knowledge over time", and this first attempt at a definition turned out also to be the last. Ian added the word "knowledge" to the list of terms, and the group's attention turned to the nature of that which is to be learned.

Several participants suggested some examples of things to be learned in quick succession: tasks, text, multiplication tables, problem solving; the group drew a distinction between rote learning and understanding. (Ian added the word "understanding" to the list.) A suggestion that a machine learner must take an active role in the learning process lead to a discussion of machine learning versus human learning.

Someone remarked that learning appears to be one of those human activities to which many people apply the rule "if you can explain how a machine does X, then it isn't REALLY doing X". Besides the difficulty of answering the "what is" question of learning, it appears that the "how" may lead to a paradox: are we "exploring unexplored territory"?

This, together with time running out, perhaps, lead to the question: "do we really have to define learning?" If not how would we know whether we had achieved learning or not? David Hill wondered if we could accept a change in behavior or performance as being indicative of learning.

1.4.2 What are the goals of Machine Learning?

As the group turned its attention to this new question, the ideas came very quickly. A machine learning system should be able to think for itself, acquire knowledge, in short, do whatever it is people are doing when they say they are learning. Perhaps in frustration someone suggested that it should be able to define learning. Maurice wanted it to improve its explanatory powers without further input from the outside world, to mull over its knowledge as it were. We extended this to include changes in behavior and self-adaptation. And why limit its access to external inputs? It should know enough to know when (and how) to inquire of the outside world.

At this point, Ian interrupted, saying that he was really more interested in the goals of the people studying machine learning.

Peter said his interests included the principles of generalization, and the notion of instructable robots, useable by non-programmers. Such robots would accept goals, instruction, examples, and advice from their users (owners? colleagues? friends?) and apply these to the solution of problems.

Someone else wanted to learn more about human intelligence; another joked "it's a job"; another wanted to become famous - "you mean notorious" rejoined yet another. Others were interested in the transfer of knowledge from one domain to another. This seems a worthwhile aim, given Martin's law that you can't learn something unless you already nearly know it.

Ian admitted that he was working in the area because "it's fun", the most interesting place to be at the moment. He would like to have autonomous systems. Bruce MacDonald joined in at this point with his wish that everyone could have access to the power of computers and robots. Debbie agreed, and introduced the notion of "domestic robots", to do housework. Brian Gaines pointed out the difficulties of getting people to accept such devices, saying that at the very least Asimov's laws would have to be guaranteed, giving "sani-robots".

Speculation ran high, with discussions of how such robots would be given requests, and whether they would be considered "people". Would they respond only to direct orders, or would they understand oblique comments and wishes ("do we want a fairy godmother?" scoffed Bruce M). Brian Gaines broke in with great energy, exclaiming that a serious moral issue was involved. He drew a vertical line, labelling the top "machine", the bottom "people", and asked whether we were dragging computers down to the level of people, or people up towards the level of machines. His interest in machine learning is that it involves interesting things, including the development of autonomous systems.

1.4.3 What functionality should a Machine Learning system have?

We first attempted to characterize machine learning systems, as knowledge acquisition boxes, natural language understanders, knowledge support systems, adaptive components to CAL systems (student models), autonomous entities, instructable robots. The discussion included military computers that help pilots shoot down planes, and the military's interest in autonomous land vehicles. Someone suggested that from a scientific viewpoints, we should be able to elucidate the principles underlying machine learning.

This led to the question "what should a machine learning system be like to use?" We agreed that they should give assistance in their own use - learning machines in the programmer's workbench. Brian G. suggested that what we were after could be subsumed by the notion of "instructable systems". These systems would receive problems, tasks, and goals to work with. They would accept advice, hints, or examples, and solve problems, with their performance improving over time. This is really the teaching versus programming distinction, so that instructable systems would occupy the high end of the continuum which includes fourth generation languages.

1.4.4 Can we hope to understand neural nets (in terms of concepts we already have)?

We distinguished non-distributive neural nets, with localized representations of concepts, which we can understand better than distributed neural nets, where the representation of concepts is distributed across the net.

It is possible to understand at different levels: the node or connection level, the algorithmic level, or at the level of emulating the brain. Bruce M. elaborated an analogy with physics and

understanding the movement of heat from a warm room to a cooler one, which can be understood at the level of Schroedinger's equations, the gas laws, or by common-sense knowledge about warm and cold air masses.

Someone commented that neural nets are good engineering, but not science because we cannot fully understand them. Peter said that he did not care how a neural net is physically implemented, but only how the algorithm is constructed and how it is working.

Brian Gaines showed a slide comparing Information Technology and Neural Nets (see C2), and noted that most of the neural nets conference concentrated on the two lowest boxes on the slide: implementation in VLSI, and neurons, connections, and learning algorithms. In contrast to this, our discussion here has concentrated on the next four higher boxes: training techniques through learning as the fundamental nature of neural nets. Furthermore, it seems that goal-directedness and integration are basic problems with neural nets.

It was felt that work in the field over the next few years should eventually result in the development of theories. This in spite of the fact that currently those working with neural nets can develop abilities that they cannot explain. Someone expressed the concern that the critical nature of representation is outside of the net itself.

1.4.5 Can we understand other learning algorithms in terms of neural nets?

It would be interesting to characterize machine learning algorithms by their input output functional specifications, viewing them as black boxes.

Bruce M. suggested that version space could be done on a marker passing machine, such as Net1, (Fahlman, S. 1982. Three flavors of parallelism. Proc. 4th National Conf. CSCSI. Saskatoon. 230-235) by sending markers to represent generalizations over a lattice from the top and the bottom. Where they met on the lattice would identify the concept learned.

A paper in the neural nets conference showed language syntax learning using a net of constraints. We could view nets in terms of constraint satisfaction.

1.4.6 Methodological issues

Should we work with problems looking for a mechanism (to solve them), or rather study mechanisms by looking for problems they can solve? Hopefully theories will be formulated as researchers become familiar with interesting mechanisms. ("Interesting" was one of the words we never got around to defining in the first question of the afternoon.) One participant suggested that the question was really contrasting goal-directed research with "having fun". Another suggested a third alternative: understanding neural nets as devices. The consensus reached was that some people are working from each viewpoint, and that hopefully they will meet somewhere in the middle.

Bruce M. outlined John Andreae's strategy: work with a promising low-level mechanism, discover tasks it cannot perform, and augment it so that it can perform the new tasks without sacrificing the ability to do the old tasks.

It is very useful to have some grand goal in mind (even if it is an unreachable one) to guide research. The discussion ended for the day at this point, but it is interesting to note that we returned to this idea again at the end of the second afternoon.

2 Machine learning Projects; System integration

2.1 From MARVIN to ALVIN

2.1.1 What is MARVIN?

Brent Krawchuk presented his work on a reconstruction of MARVIN, a program originally written by Claude Sammut (Sammut, C. and Banerji, R. 1983. Hierarchical memories: an aid to concept learning. Proc. International Machine Learning Workshop. 74-80. Allerton House, Monticello, IL, June 22-24) and an extension of his own design, ALVIN. MARVIN differs from other learning systems in its application of knowledge about concepts it has already learned to the learning of new concepts. Thus it is hoped that MARVIN may be able to exhibit sustained learning. Knowledge about concepts is represented as Prolog clauses.

A general learning system: MARVIN is a general learning system which learns from examples and by experimentation. After receiving a positive example from the trainer, MARVIN uses Similarity-Based Learning techniques to form a hypothesis about the target concept. MARVIN then presents an example consistent with the hypothesis, and relies upon the trainer to verify or refute the hypothesis.

Example: learning about swimmers: As MARVIN is given (positive) examples, it builds its representation of the target concept. Whenever possible, it attempts to generalize. For example, suppose that MARVIN already has the concept fish, including flying fish, cod, and trout (see D4). Upon learning that trout is a swimmer, MARVIN will form a hypothesis by generalizing to all fish being swimmers. To verify this hypothetical generalization, MARVIN will present an example to the trainer, asking, in effect, "is cod a swimmer?". MARVIN will continue generalizing as long as the trainer agrees that the example presented is indeed a swimmer. On the other hand, if the experiment fails, MARVIN will stop generalizing and assume that it has learned the concept, or choose a different generalization, if possible.

The trainer may then give a further positive example. Suppose the trainer asserts that a whale is a swimmer (see D4). MARVIN will then expand the concept by creating the disjuncts: A is a swimmer if A is a fish, or if A is a whale. If MARVIN possessed concepts more general than whale, it would again attempt to generalize.

Crucial objects: When selecting an object to present to test a hypothetical generalization, MARVIN chooses a "crucial" object. For example, suppose the trainer is teaching the concept "whimsy", and has given zero as an example. The currently accepted generalization will be that A is a whimsy if A is equal to zero. Suppose at some later time that MARVIN is considering the possibility that A is a whimsy if A is a digit (digit being a concept it already knows). MARVIN would choose a crucial object to present to the trainer as a possible example of a whimsy.

The choice of a crucial object is constrained by three rules (see D5):

1. A crucial object must be covered by the trial generalization. In our example, any of the digits 0-9 would qualify. The objects dog, cat, etc. would not.
2. A crucial object must not be in the currently accepted generalization. This rule eliminates zero itself as a possible example.

3. A crucial object must not be in danger of being part of the target concept for reasons other than being in the generalization to be tested. If MARVIN already knew the concept bit, for instance, the digit 1 would be eliminated by this rule.

So, in this example, MARVIN would choose one of the digits 2-9 to present to the trainer.

2.1.2 Problems with MARVIN

When MARVIN's knowledge of a domain is rich enough, it may be impossible to find a crucial object. For instance, if MARVIN had concepts of even and odd digits, rule (3) would eliminate all digits from consideration as crucial objects for the digit generalization.

The trainer must have a tree structure of the knowledge in mind so that the choosing of crucial objects will lead MARVIN to the desired concepts. This places too great a load on the trainer, who must present the examples in a bottom-up order.

For example, suppose MARVIN knew about digits and bits, and the trainer wished to teach the concept `prdiv6` (prime divisors of 6), giving 2 as the first positive example. The crucial objects are 3-9 in this case, so MARVIN would choose one of these to test the generalization to digit. Of course, MARVIN should not be allowed to generalize to digit, so if MARVIN happens to choose a digit from 4-9, all is well. On the other hand, if MARVIN chooses 3, the trainer must tell a white lie, saying that 3 is not an example of the concept `prdiv6` (even though it really is!).

It is not so much that the trainer has to lie, but rather that the trainer must answer an implicit question. MARVIN asks "is 3 an example of `prdiv6`", but also shows the generalization from which the crucial object 3 was chosen. The trainer must not answer the question as asked, but must instead answer the implied question: is the generalization valid? Otherwise, MARVIN will overgeneralize (see D6), and since the trainer cannot give negative examples, there will be no way to convince MARVIN to give up the generalization.

People tend to think of concepts in a more general, partially ordered, structure rather than in a strictly hierarchical structure. For example, rather than consider bit to be a subclass of digit, we tend to think of bit as being a separate concept altogether (see D7). If MARVIN could structure its knowledge in this manner and also do away with the need for white lies, then the trainer would require less specific knowledge of the concept to be learned. Unfortunately, the use of a partial ordering structure does not, in itself, preclude the need for "white lies".

As a further example of the increased flexibility of partially ordered structures, consider adding the notion of swimming to a hierarchical concept bird (see D8). If birds has two subclasses, flightless and aerial, then adding swimming requires the introduction of the adhoc subclasses "swimming aerial birds" and "swimming flightless birds". With a more general structure, the notion of swimming can be added to the previously learned concepts without interfering with them in any way.

2.1.3 ALVIN to the rescue

ALVIN is Brent's answer to some of the limitations of MARVIN. It represents concepts in a partially ordered structure, thus simplifying the constraints on the trainer's own representation of the knowledge, and avoiding the "lie".

ALVIN is like MARVIN in the sense that it attempts to generalize from positive examples, and seeks to validate its generalizations by presenting an example to the trainer. The essential difference occurs in the choice of the example to present. If a crucial object does exist in the more general concept that ALVIN is attempting to validate ALVIN will present that object. Otherwise

a significant object will be chosen. A significant object is one which satisfies MARVIN's rule (1) and (2), but not (3); that is, it may be an element of a concept other than the one being validated (see D9).

When a significant object is presented and rejected, ALVIN the hypothesis being tested is rejected. If, on the other hand, the object is accepted, then ALVIN investigates further by a recursive application of its generalization algorithm to the significant object. Supposing that this results in a valid generalization, ALVIN will store it as a disjunct, and continue to explore its original hypothesis.

S

2.1.4 Advantages of ALVIN

ALVIN represents its concept knowledge using a more general data structure, resulting in a reduced burden on the human trainer. ALVIN can work in more complex domains and more complex knowledge structures (see D10). It can learn multiple disjuncts in response to the presentation of a single object by the trainer (see D10 through D13). Finally, Brent has introduced the notion of significant objects for use when there are no crucial objects.

2.2 Some complexity results for the Version Space algorithm

Mathew Ling presented results from his study of version space, an algorithm due to Mitchell (Mitchell, T.M. 1982. Generalization as Search. Artificial Intelligence, 18, 203-226). He began with a brief review of the algorithm, referring back to an earlier presentation which is reconstructed here.

2.2.1 What is Version Space?

Version Space is a concept learning system which applies to areas for which three entities can be clearly defined: a domain, a description language, and a specificity ordering on the possible descriptions. It then defines data structures, and performs algorithms, based on these entities:

1. the domain is a finite collection of distinct objects.

Consider, for example, the set of ordered pairs, or points in the real plane, for instance (3, 5).

2. the description language specifies a set, possibly infinite, of descriptions, each of which represents (describes) a subset of the domain.

A very simple description language over the set of ordered pairs would contain the descriptions " $x = y$ ", " $x < y$ ", " $x > y$ ", " $x \leq y$ ", " $x \geq y$ ", and " $x : y$ ", where x and y are the x -coordinate and the y -coordinate, respectively, and " $x : y$ " means that x and y are comparable, but we don't wish to be more specific. The description " $x : y$ " thus describes all possible points on the plane, while " $x = y$ " describes only those points whose two coordinates are equal.

Then the ordered pair (3, 5) fits all of the descriptions: " $x < y$ ", " $x \leq y$ ", and " $x : y$ ". Notice that the three descriptions of (3, 5) are arranged in order from most specific to most general.

3. the specificity ordering recognizes that some descriptions are more specific than others.

In the example (see E9), the area of the plane described by " $x \leq y$ " is a subset of the area describe by " $x : y$ " (the entire plane). This makes " $x \leq y$ " more specific than " $x : y$ ".

This is actually a partial ordering, since there are some pairs of descriptions which are unrelated in it. For example, " $x < y$ " is neither more nor less specific than " $x > y$ ".

It is possible to show the ordering of all the descriptions in a lattice with each description written above, and connected to, those which are more specific.

The concept to be learned is a subset of the domain of objects. Version Space considers the concept to be learned when it has found a set of descriptions which together describe all of that subset and nothing else. The lattice of descriptions is the space through which Version Space searches to find the descriptions of the concept to be learned. Rather than represent the entire lattice (which may have infinite depth, or breadth, or both), the version space algorithm uses two sets of descriptions: the G set is the set of descriptions which are as general as possible with respect to the examples which have been presented; the S set is the set of descriptions which are as specific as possible, consistent with the examples presented.

When an example is presented, it is matched with the descriptions. A positive instance will result in the removal from G of any description that does not cover the instance; then the descriptions in S will be replaced by a more general set of descriptions - just enough more general to cover the new instance as well. If the instance is negative, then any description in S which covers it is dropped, and the descriptions in G are replaced by more specific ones - just enough more general to exclude the new instance.

2.2.2 Different kinds of search spaces

Mathew distinguished three varieties of search space, for which he has some complexity results.

1. In a hierarchical search space, the descriptions are organized in a tree structure by the specificity relation (see E6).
2. In a Boolean search space, each description is the subset of objects (see E8).
3. In an attributive search space, each object is represented by a list of attribute-value pairs. Mathew restricted the discussion to those spaces for which the attributes form a hierarchy (see E7).

2.2.3 Results

For each of these kinds of search space, Mathew discussed the size of the S and G sets, and the number of training examples required for the version space algorithm to find the concept.

In a tree-structured space, both G and S must contain exactly one description. G will begin by containing the root node only, and S will contain the leaf which describes the first positive example. As each additional example is presented, one of the sets will "move": a positive example will cause the description in S to generalize to include it, and S will move at least one position towards the root of the tree; for a negative example, G will similarly move at least one position towards the leaves. Thus, for a tree with L levels, version space may need as many as L examples.

In attributive search spaces (assuming attribute hierarchies), the S set will always contain exactly one description. The G set may contain more than one, unless Winston's "near miss"

technique is used for negative examples, limiting the expansion of the G set by not causing disjuncts. In this case, the size of G will be one (compare E4 and E5). Given M attributes each with L levels, version space will require ML examples to converge to a single description.

In a Boolean space, again the G and S sets are of size one. But now, version space will require as many examples as there are objects in the domain.

The results are summarized in a table (see E2).

2.2.4 Observations and questions

1. The number of examples needed for convergence does not depend on the number of descriptions in the search space, but rather on its structure.
2. What will happen to the size of G when the size of S is two or larger? For this to happen, version space must be operating in a non-hierarchical space.
3. What will happen to the size of G when the values of an attribute are not in a hierarchy, so that there is no unique generalization?
4. Could some other set be used as a boundary within the search space, other than S and G?
5. Would it be possible to modify the version space algorithm so that it would expand the description language when the Version Space collapses?

2.3 Reconstructing Noddy

David Pauli described his reconstruction of Noddy, originally written by Peter Andreae as a PhD thesis. (Andreae, P.M. Justified Generalization: acquiring procedures from examples. PhD Thesis, Department of Electrical Engineering and Computer Science, MIT.)

2.3.1 Description of Noddy

Noddy learns robot procedures from traces of procedure execution.

Noddy processes traces incrementally to refine a "current generalization procedure", which may be the result of generalizations made from previous traces. As new traces are encountered they are generalized with this current procedure if possible.

2.3.2 How Noddy Works

Noddy makes use of explicit, pre-programmed, generalization hierarchies and information about mathematical and set operators. Events in a procedure trace are represented as nodes in a directed graph. Noddy attempts to generalize by adopting a conservative policy which attacks the problem in three stages: the skeleton matching stage, the propagation stage, and the final stage of functional induction.

Skeleton Matching Stage No generalization takes place in this stage; instead Noddy searches for "key events" in both the procedure and the trace. If a pair of key events are unique within their respective structures, and match exactly, they are considered to match.

Propagation Stage This stage examines events which are different but whose predecessor or successor events have been matched in the skeleton matching stage and attempts to unify the descriptors within these events. Unification of the descriptors is done using the generalization hierarchies and succeeds if two descriptors have a common predecessor in the hierarchy.

Generalizations made in this way are termed first level generalizations because they can have no functional parameters. Any loops or branches present will be recognized and made explicit in the grouping process which completes the propagation stage (see F3 and F4).

Functional Induction Stage This stage constitutes the second level of generalization and is only undertaken when parallel segments have been found between the procedure and the trace. Parallel segments have the following features:

1. they have the same start and end events,
2. they contain the same number of events,
3. the corresponding conditions match, and
4. the corresponding actions are of the same type.

This second level of generalization uses much more powerful generalization techniques - hence more expensive ones - and involves searching for functional dependencies of actions upon earlier patterns. Functional induction involves finding an earlier pattern component upon which the actions may depend and a function that will relate the pattern component to the action. Noddy maintains a list of past values of patterns and action parameters in order to conduct this search.

Candidate functions for relating previous patterns to actions are chosen from a list of pre-defined operators. A single operator is applied to the domain values (from the condition part of the event - ie. input values) and returns range values (from the action part of the event - ie. output values). If the range values do not match those from the action, Noddy iteratively applies any appropriate operators to the domain and range values and searches for a connecting operator which returns the new range values when applied to the new domain values. This has the effect of building up an expression from both ends of the data. The resulting expression will be the composition of the inverses of the operators applied to the range values, the connecting operator and the operators applied to the domain values.

The choice of operators is constrained by the type of the input and output values (eg. positions, angles, etc.).

If the current output values equal their associated input values, then no connecting operator is necessary and the gap is simply removed, but, if there is a binary operator that when applied to each input/output pair produces a constant value, the gap is filled by that operator's inverse applied to the constant and the input value. A constant found in the above way is referred to as a new constant and only one is allowed per expression. Other constants, referred to as known constants, can also exist in the expressions. These constants have predetermined values which may be relevant to the functional dependency. For example:

0.5 in move 0.5@-90
versus
move 0.5@-45 (see F5).

2.3.3 An Implementation in Prolog

While the original Noddy was written in Lisp, David's reconstruction is written in Prolog. One of the major differences is the knowledge representation, as David's Noddy uses explicit typing for the operators and the domain and range values.

Implementing Noddy in Prolog meant that inverse operators did not need to be explicitly defined, rather they are inherent in Prolog since it is bidirectional. Inverses are achieved by permuting the arguments to a functor.

A drawback for the Prolog implementation was that some information about inverse operators was lost, such as the commutative property of addition - thus generating redundant expressions, eg. (plus 2 3) (plus 3 2).

The representation of the built up expression in Prolog required the use of uninstantiated variables to keep track of inputs and outputs to the gap - nothing like this was necessary in the Lisp implementation.

2.4 Architecture of a Knowledge Acquisition System

Maurice Sharp, assisted by Bruce Thompson, presented the architecture of the KITTEN system, which they are developing here on the Apollo systems. This is a descendant of the PLANET suite of programs available on the Apple. The system consists of an integrated knowledge base, analytical tools, and conversational tools (see G1).

2.4.1 Grid elicitation

One of the conversational tools, which has been implemented, elicits a grid of personal constructs from an expert. This program prompts the user for a purpose and a list of elements, entities in the expert's domain. Then it proceeds to find distinctions between elements by presenting them three at a time and requiring the expert to explain how two of these differ from the third.

The expert makes explanations of distinction by naming two poles of a new construct, giving a continuum between two extremes, and by then rating each of the elements according to this construct, typically on a scale of one to five.

The system automatically computes matches: entities which have been similarly rated on all constructs, or constructs which appear to be equivalent, though redundantly named. The expert breaks these matches by adding distinctions or elements.

2.4.2 Grid analysis tools

PRINGRID displays the elements in a two dimensional space, based on the most salient constructs.

SOCIOGRID combines the grids of several experts. This allows the modeling of understanding and agreement. If expert A can, from B's perspective, place the elements on B's constructs the way B did, then A understands B. Placing the elements from his or her own perspective, A can agree with B.

The ENTAIL program analyzes the grid to determine which pole names entail which other pole names, with truth, probability, and uncertainty reduction values. These entailments become rules which can be used in an expert system shell.

TEXAN is a domain organizer. Its purpose is to locate clusters of words in text, called knowledge islands. These words are often usable as element names, to prime the grid elicitation process.

2.4.3 Current work

Maurice and Bruce are currently working on the knowledge base, using an object-oriented approach, and object-attribute-value triples. Peter pointed out that this is a subset of first-order predicate calculus, with only binary relations (the attribute relating object and value).

2.5 Machine Learning in Wellington

Peter Andreae described his work with instructable robots. Such robots would accept at least goals, advice, and examples, and actually execute a task. They will contain domain knowledge, and abilities in geometric reasoning, planning, and generalization.

Peter's main interests in this project are the generalization of structured descriptions, particularly procedures with parameterized commands, iteration; and syntactic justification for generalization.

The overall approach is one of incremental learning, so that a usable generalization (procedure) is available at each stage. Only positive examples of procedure execution traces are provided to the system, which doesn't remember each of them explicitly, but incorporates them into the current general procedure. Minimal use is made of backtracking, so that possible mistakes must be avoided. Peter hopes to avoid exhaustive searching, and to use minimal domain knowledge.

2.5.1 Old Noddy

This was Peter's PhD project (MIT 1985). The program acquires procedures for a simple robot, which works in a two dimensional space, is blind, but can sense position and contact with other objects. Examples are traces of primitive commands with sensor values. This system can learn procedures with loops, conditional branching, and generalized actions (see H1). To do this, it must be able to infer branch points and conditions, and find functional dependencies.

A generalized procedure is formed by matching two procedure traces. This requires the inference of branch points, with loops "falling out" (see H2). Another way of looking at this process is to consider the two execution traces as alternatives, and then "push the OR's" down (and up) merging the traces at steps which match (see H3).

Noddy generalizes structure descriptions by:

1. finding correspondences between parts,
2. generalizing corresponding parts, either through attributes or relations with other parts, and
3. finding groups and substructures.

Why Noddy appears to work. Noddy exploits structure to find correspondences. It includes builtin assumptions about the nature of procedures to constrain generalization (ex. procedures must be executable, and hence deterministic). Noddy is efficient through the use of different levels of generalization: it uses the simplest methods first, and more powerful (and expensive) methods only when they are well justified. The context of the match justifies the use of stronger methods.

Limitations of the approach

1. Noddy has built in heuristics for generalized actions and conditions, so that it doesn't do as much learning as it might seem at first. The context measures are untested; he never checked to see if spurious generalizations would be produced when using more powerful generalization methods.
2. The procedures generated were unstructured, and included only a limited class of loops. For example, in counted loops, the loop increment could only be one.
3. Examples of execution traces are lists of commands rather than observed behavior.
4. The program contains "hacks" for finding seed events to begin the matching process.
5. It was only used to induce four simple procedures. Peter suggested that before starting work on a problem, a researcher should have ten interesting instances of the problem in mind.

2.5.2 New Noddy: Big Ears and Mr. Plod

New Noddy is a robot moving in three dimensional space with compliant guarded motion (the robot will move by applying force in a given direction, and act like a spring at right angles to that motion). Examples are descriptions of behavior: segmented streams of sensor values and timing information (see H4). It generates structured procedures, including tail and full recursion, implemented as and/or trees. Parallel commands are allowed.

An example of a procedure that Noddy might learn is one to fill a pallet (see H5). Part of this procedure is another one to drill holes (see H6). Once the drill holes procedure is learned, it would become a part of the overall, fill pallet, procedure (see H7).

New Noddy will use a blackboard architecture, with explicit loop finding experts (see H8). It will integrate many generalization experts, including correspondence proposers and critics, generalizers, and installation experts and critics.

2.5.3 Other projects at Wellington

1. Inference of specifications from procedures, the inverse of program synthesis.
2. Investigation of qualitative geometric reasoning in two and three dimensions.
3. Generalization of visual descriptions, relating function to structure, and handling descriptions with optional parts.

2.5.4 Discussion

What should Noddy do if it over-generalizes? Peter had avoided backtracking by making Noddy's decisions irrevocable, not because it was infeasible to store all examples and reconsider previous decisions, but because it was not clear if backtracking could be done without exploding the search space. Peter speculated that an interactive component added to Noddy might help. Brian Gaines suggested that keeping interesting examples which caused structural changes would help the problem. Both of the above would allow backtracking to take place if Noddy did over-generalize.

2.6 Discussion

2.6.1 MARVIN

What is MARVIN that we must lie to it? MARVIN makes a generalization on the basis of one example, and it has no backtracking capability. It presents an example of the generalization it would like to make, but along with the example, it shows a representation of the generalization. When the trainer rejects the generalization, he may have to lie about the example.

Brian Gaines suggested that perhaps lying might be necessary to optimize training time. This raised issues of "trust", and "disillusionment". Brian pointed out that all text books lie, because they can't enumerate all the exceptions.

What kind of generalization does MARVIN do? In Michalski's terms, MARVIN uses the "climb hierarchy" and "find common substructures" heuristics.

What kind of problems is it good for? Tiny. Brent sketched the steps necessary to teach MARVIN the append function: teach letter(A), eqletterlist(A,B), concat, etc.

The group agreed that numerous improvements were needed. MARVIN's strong point is the synthesis of examples to test hypotheses.

2.6.2 Version Space

What kinds of problems is it good for? Version Space induces class descriptions for objects describable with attribute-value pairs. It cannot deal with structural descriptions. Peter said that he used some of the ideas from Version Space in Noddy, and that some knowledge acquisition systems also incorporate ideas from Version Space.

What is induction, and is it used by Version Space? Brian Gaines gave an example of the differences between induction, deduction and abduction from C.S. Peirce, the 19th century philosopher.

```
World ---abduction--->
  |                     |
  |                     | induction(introduce universal
  |                     | quantifiers)
  |                     V
  <-----deduction---
                (introduce existential quantifiers)
```

Bruce MacDonald thought that Version space is doing induction because it sees some examples of a concept and derives a generalization for all instances in the concept. Ian Witten thought that Version space is not doing induction since it is forming a unique generalization. Brian Gaines felt Version Space was not doing Induction because it does not search a complexity ordering.

A remark was then made that in reality conjunctions and disjunctions are used in Version space to get away from complexity orderings. Another remark was then made that any inductive system has a complexity ordering in it so that Version space is partially inductive and partially deductive. At this point Bruce Macdonald said that maybe Version space is just inducing parameters.

Peter Andreae told us that Noddy deals with layers of complexity and when one layer collapses Noddy goes to the next layer of complexity - this can be thought of as having layers of Version spaces.

2.6.3 Noddy

What kinds of functions can it induce? Since Noddy allows the introduction of at most one new constant per input variable, it cannot induce general polynomials. It was suggested that combinators might be used to replicate arguments. This would give a larger search space, but if something in this space were found to be profitable several times, it could be "remembered" by making it an operator. As an alternative one could allow "oracles".

What kinds of functions can Bacon induce? Bacon looks for proportional relations and monotonic relations. The group agreed that it was difficult to characterize the precise set of functions inducible by Bacon. Someone pointed out that there is a danger of getting more information in the function induced than is actually present in the original data. Methods of searching through a space of polynomials, including linear regression, were briefly discussed.

What kinds of domains is it good for? Since Noddy induces procedures from examples, it should be good for many things done everyday on a computer system, ex. updating a database, etc. On the other hand, it would not be good at inducing an algorithm like quicksort, where it is difficult to present the example traces.

2.6.4 Knowledge representation

This was subtitled "a diatribe by Peter Andreae". Peter warned of the dangers of getting caught up in notation (ex. semantic nets, frames, logic, etc.). Ontology is more important: one should begin by identifying the entities one wishes to represent. There are many possible kinds of entities which we might wish to distinguish: physical objects, actions, ideas, statements, events, relations, and so on. We need to distinguish between atomic entities and classes of entities. We must also consider the issue of indexing: how to relate or connect two separate facts.

Peter was just warming up when we ran out of time for this question.

2.6.5 CONCLUSION: What are we trying to do, and would we know when we achieved it?

We decided that we needed a big project, not necessarily doable, with a name and a catchy phrase. Someone, building on Brian Gaines' suggestion that humans are really a collection of hacks, proposed HACK as an acronym for the big project. Maurice tried his hand at coming up with a phrase for HACK, starting with "Hopefully, use AI...", and everyone liked the first word, at least. Brian jokingly suggested the C might stand for Canadian, and that we might use our expertise with robot arms to build a giant one to catch and dismantle missiles. Perhaps HACK might stand for "Hopefully, use AI to Create a Knowledge-based system".

The notion of instructable systems came up again. Peter expressed it as the use of knowledge acquisition techniques to capture an expert's knowledge, then convert it into executable procedures.

A Neural Net Conference

Features of MacBrain version 1.00

Runs on Macintosh Plus, Macintosh Se, and
Macintosh II computers.

Speed	25,000 COPS (Macintosh II)
Size	< 80K
Maximum Processing Nodes	200
Maximum Connections	40000
Hebbian/Hopfield Learning Rule	
Boolean Activation	
Linear Activation	
Linear Drop-off Activation	
Asymptote Activation	

Input Parsing

Text editor/notepad

**Graphically oriented
Animation**

Color support

7 ways (instruments) to display units

4 ways to display weights/links

Strip charts

Averaging strip charts

Meters

Phase plane diagrams

Completely Interactive

Instant unit creation/editing/moving

Instantly change unit activation

Instantly change activation function

Instant weight creation/editing

**Advance network one cycle at a time, or
for a given number of cycles.**

**Someday,
everybody will use
MacBrain™ for
Neural Net
Research. But if
you prefer leading a
field to catching up
to others, you can
buy MacBrain now.**

IEEE First Annual International Conference on Neural Networks

San Diego, June 1987

First fully recursive conference—
neural nets communicating with neural nets about neural nets

Almost didn't take place—
IEEE didn't believe there would be 200 registrants
initially short of papers and participants

Finally massive—
some 300 papers, 1500-2000 registrants
Proceedings out in September, some 1800 pages

Best keynote address—
Carver Mead, "Silicon Models of Neural Computation"
analog CMOS circuits as visual retina

Stochastic computing—
Gaines & Cleary papers
renewed interest, work at number of centers

Commercial exhibition—
Range of net simulators on mainframes, micros & lisp-machines
accelerator cards for PCs

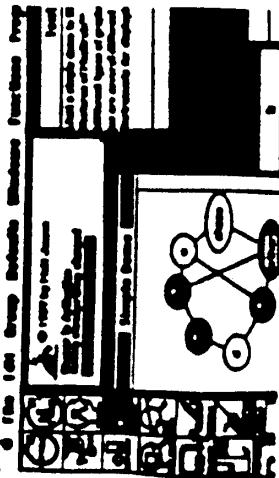
Overall—
no clear sense of direction, no front runners
many major researchers missing, Hinton, Smolensky

Conclusions—
neural net technology and applications taking off
it's open season, grab some action
how does this technology integrate with standard computing?
role of analog and digital?, role of stochastic computing?
applications techniques—equivalent of software engineering?

\$99 TODAY!

MacBrain™

Workstation Power for Simulation



MacBrain runs on Macintosh Plus, Macintosh Se, and Macintosh II computers. Obviously, we suggest you use a Macintosh II if you have one. The advantages are:

- **Color.** Colors break up those tangled plots and also indicate how excitation and inhibition are flowing across the network.
- **Speed.** A Macintosh II can run MacBrain about 20 times faster than a Macintosh Plus by using the 68020 and 68881 chips. That adds up to about 25,000 COPS (connections per second). This puts MacBrain at the same speed as HNC's ANZAS system for less than 2% of their price.
- **Hardware.** The Macintosh II has six self-configuring NuBus slots that you will be able to fill with parallel processing boards and other accessories. For example, a future version of MacBrain will support boards from Mechanical Intelligence, Inc., of Cardiff, CA.

Setting Standards and Breaking Limits

Neuronics™, Inc. of Cambridge, MA, introduces MacBrain™, a software system for simulating neural networks.

Research which until now could only be done by a few well-funded universities and corporations can now be done by anyone with a Macintosh™ computer and an interest in exploring the new field known variously as neural network theory, connectionism, parallel distributed processing, and adaptive systems theory.

MacBrain is fast, powerful, easy to use, and quite inexpensive. In fact, MacBrain provides as much processing power as other systems that cost ten to a hundred times more! It is also the only neural net simulation environment to sell for under \$10,000. It sells for \$250. Quite a difference, eh? That's only the beginning.

Customer Support

Upgrades

Neuronics will provide, free of charge, an upgrade from version 1.00 of MacBrain to version 1.10, when it becomes available, to all registered buyers of version 1.00 who made their purchase before August 1, 1987. All others may upgrade by paying the difference between the price of 1.10 and the price they paid for 1.00.

Telephone Support

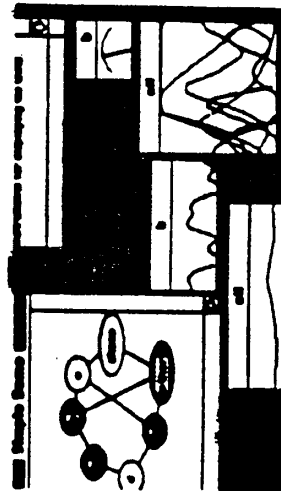
Registered buyers of MacBrain have free access to phone support. This support is unlimited.

NeiNet™ Database Access

Registered buyers of MacBrain have free, unlimited access to NeiNet, Neuronics' online database, news, and message system for neural network researchers. By the end of this year, NeiNet will be completely tied into our research library, allowing MacBrain users to do bibliographic/abstract searches.

Warranty

The physical materials of MacBrain are guaranteed to be in good physical order and free of physical defect. Defective or damaged materials will be replaced free of charge. Unfortunately, the complexity of computer software, particularly in a very new field, prohibits us from making any warranty about software performance.



Neuronics, Inc. Box 738, Cambridge, MA 02142

617/367-9254 617/367-9254

24

INTERNATIONAL NEURAL NETWORK SOCIETY

Membership Application

Dr. Harold Szu
Naval Research Lab
Code 5756
Washington, D.C. 20375-5000, USA

The International Neural Network Society (INNS) is an association of scientists, engineers, students and others seeking to learn about and advance our understanding of the modelling of behavioral and brain processes, and the application of neural modelling concepts to technological problems. INNS will sponsor its first international meeting in 1988. INNS membership includes a subscription to NEURAL NETWORKS, the official journal of the society.

MEMBERSHIP FEES (1987-88)

☐ Regular (\$45.00 enclosed)

☐ Student (\$35.00 enclosed)

Payments cover 1987 and 1988.

Payment enclosed: ☐ Check ☐ Money Order

Amount \$ _____

Please charge my: ☐ American Express ☐ MasterCard ☒ VISA ☐ Diners Club

Account Number: _____ Expires _____

Signature: _____

Name _____
Last First M.I. Title

Department _____ Institution _____

Employment: ☐ University ☐ Government ☐ Industry ☐ Other _____

— over please —

INTERNATIONAL NEURAL NETWORK SOCIETY

Membership Application

Mailing address _____

City _____ State _____ ZIP/Postal Code _____ Country _____

Electronic mail address _____

Telephone(s) _____

EDUCATION: Highest Degree _____ Date _____

University _____ Department _____

Check your principal areas of interest in neural networks.

- ☐ Vision and image processing
- ☐ Speech and language understanding
- ☐ Pattern recognition
- ☐ Associative learning and long-term memory
- ☐ Self-organization
- ☐ Cognitive information processing
- ☐ Cooperative and competitive network dynamics and short-term memory
- ☐ Sensory-motor control and robotics
- ☐ Parallel distributed processing
- ☐ Local circuit and systems analyses of brain-behavior relationships
- ☐ Combinatorial optimization
- ☐ Electronic hardware
- ☐ Optical hardware
- ☐ Hybrid hardware
- ☐ Virtual devices
- ☐ Neurocomputers
- ☐ Other _____

Signature _____ Date _____

Mail application with payment made payable to INNS to: Dr. Harold Szu,
Naval Research Lab, Code 5756,
Washington, D.C., 20375-5000, USA.
Telephone: (202) 767-1493, FAX: 202-767-4277,
E-Mail: ARPNET-Szu @ NRL3

B ID3 Counter-Example

From Brian Gaines
To Peter Andreae

8th July 1987

**Counter-Example to Show that ID3 Information Reduction Heuristic
Does Not Generate Shortest Expected Decisions**

I believe Ross has never claimed the information reduction criterion of ID3 to be more than a heuristic.

Intuitively it should have simple counter-examples because it is a search heuristic that commits too early. Think of the problem as a search tree over possible decision trees—the heuristic does not take account of the later stages of the decision tree it is selecting.

My guess at a counter-example is to take a 'nasty' case that ID3 is pretty helpless at and tack on a misleading 'carrot' that sends it down the wrong path.

Let's start with the exclusive-or function:-

Entity	Att1	Att2	Class
A	1	0	P
B	0	1	P
C	1	1	N
D	0	0	N

The entropy is 1.0. Neither attribute reduces it! ID3 choses either first and then the other second, landing up with a correct and best solution:

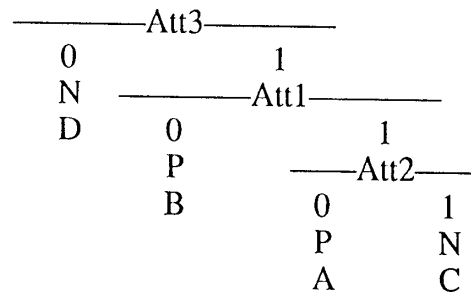
———Att1———			
0		1	
—Att2—		—Att2—	
0	1	0	1
N	P	P	N
D	B	A	C

The average number of decisions is 2.0

Now give ID3 a misleading attribute that is of no help at all to the problem.

Entity	Att1	Att2	Att3	Class
A	1	0	1	P
B	0	1	1	P
C	1	1	1	N
D	0	0	0	N

Att3 now reduces the entropy to $-3/4 (1/3 \log_2(1/3) + 2/3 \log_2(2/3)) - 1/4 (0) < 3/4 < 1$, so ID3 choses it, but still has to use the old decision tree thereafter, landing up with the correct but not as good solution:



The average number of decisions is 2.25

This is the smallest counter-example. There is clearly none for two entities. With three, it is not possible to create an asymmetry for a carrot.

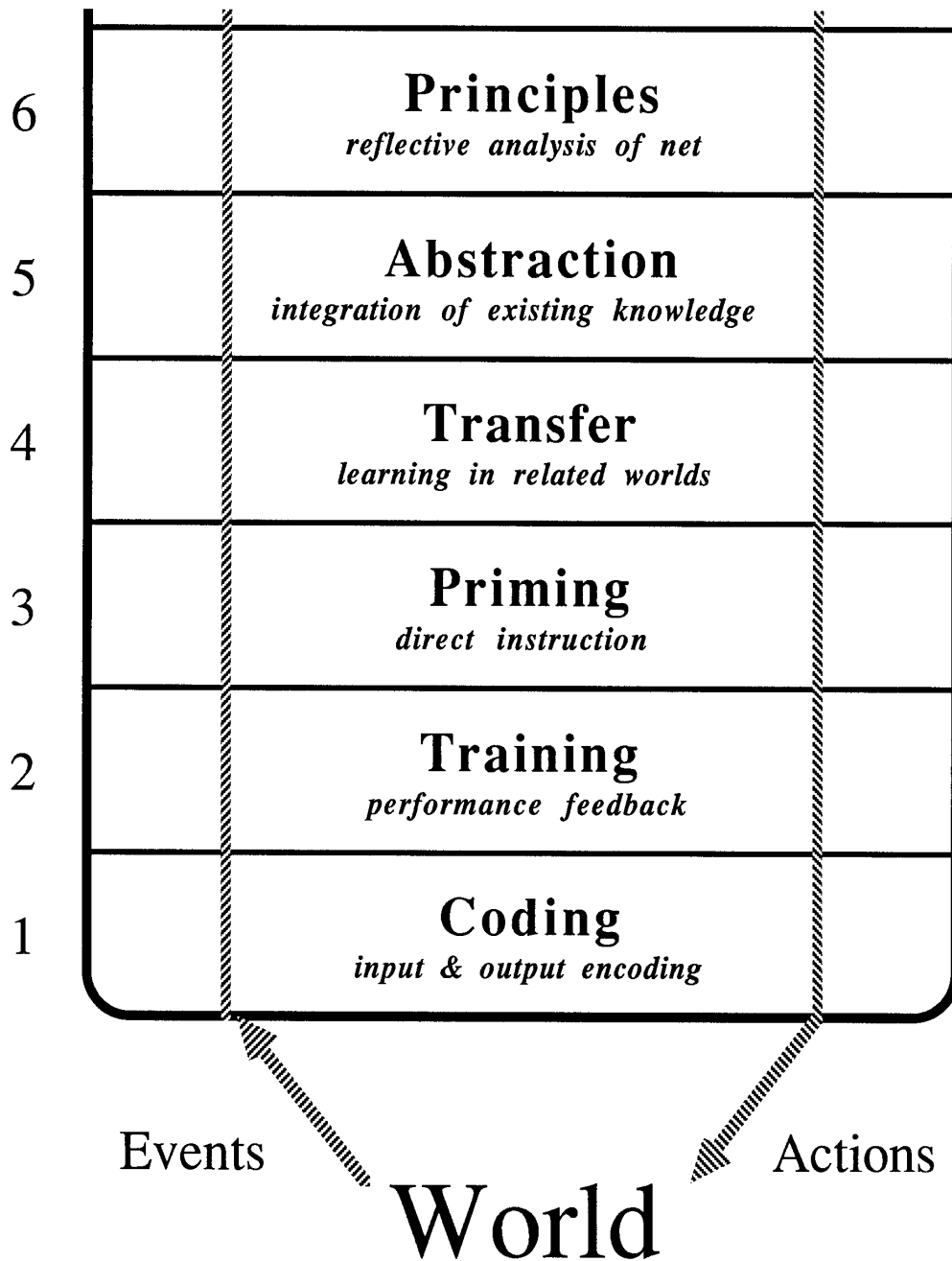
This should also be a counter-example for any tree-pruning algorithms derived from an ID3 first pass, since it is the root of the tree that has to be removed.

The example is very dry and perhaps it should be distributed in a more interesting context:

It came to pass that God offered Adam and Eve the opportunity to redeem their sin and return to the Garden of Eden. "Thou shalt truly tell me the features of those who walk upright in the land", saith the Lord. "Adam, the man, has a long left leg and a long right leg, and walketh upright in my sight. Eve, the woman, has a short left leg and a short right leg, and walketh upright in my sight. Cain, the man, has a long left leg and a short right leg, and walketh crookedly in my sight. Abel, the man, has a short left leg and a long right leg, and walketh crookedly in my sight even while he is sober. Now, telleth thou me, how one may decide who walketh upright such that the Devil might not surpass thee in the timeliness of the truth." Eve pondered and Adam wandered the breadth of the land searching for the truth, or so he said. In the fullness of time they gathered together and Eve said, "Adam darling, the left leg telleth nothing. The right leg telleth no more. A woman walketh upright. A man with a long left leg and a long right leg walketh upright." Adam replied, "So be it sweetie pie, I asked far and wide but none had the knowledge until I consulted the Three Ides of Quin Lan, and they answered what thou has truly spoken." This they communicated to the Lord who sighed and saith, "Truly, hath the Apple blinded thee, even the prunings of MacTwo, the sage of the West, could not open thine eyes." Go forth and multiply for the tree of decision is beyond thee.

It is a sad note to this tale that the French Revolution, ungodly as it was, with its proclamation of, "Equality, fraternity and liberty!", first offered the solution to this ancient problem—equality of legs leads to uprighteous walk. This answer would have led to the fraternity of the Lord and the liberty to walk in the Garden of Eden, now known as Palo Alto, unfortunately blighted by semiconductor wastes but where the Apple of life still blooms.

C Hierarchies



Socially Organized Systems	<i>Integration a basic problem</i>
Autonomous Activity Systems	<i>Goal-direction a basic problem</i>
Inductive Inference Systems	<i>Learning is fundamental nature of nets</i>
Knowledge-Based Systems	<i>Structural & functional differentiation</i>
Human-Computer Interaction	<i>Interactive training, feedback from performance</i>
Problem-Orientated Languages	<i>Training techniques</i>
Virtual Machine Architecture	<i>Neurons, connections, learning algorithms</i>
Electronic Device Technology	<i>VLSI allowing large parallel systems</i>
Information Technology	<i>Neural Nets</i>

D From Marvin to Alvin

FROM MARVIN TO ALVIN

Presentation Outline For ML Symposium

B.J. Krawchuk

1. MARVIN

- (a) general learning System
 - i. learning by example
 - ii. learning by experimentation
 - iii. relationship between student and teacher
 - iv. Cohen(1978); Banerji, Sammut (1981)
 - v. already-learned concepts can be used in future learning
 - vi. stores concepts as Prolog clauses
- (b) Example: Learning Swimming
- (c) Crucial objects in Marvin

2. PROBLEMS

- (a) Claim it works well for tree structures. (Hierarchical Concept Memory)
- (b) hierarchical memory unnatural
 - i. awkward to use
 - ii. sub classes not really thought of as contained in the superclass (*bit* \prec *digit*)
 - iii. attributes as subclasses
 - iv. learning in frame systems
- (c) cannot maintain tree structure
 - i. tree maintenance not inherent in the algorithm
 - ii. relies on trainer to teach in proper order
 - A. trainer must know structure of a concept beforehand (bad order screws things up)
 - B. trainer must know everything that is to be taught
- (d) Even if procedures to convert back into tree structure not even limiting to tree structure works: overgeneralization and overspecialization

- (e) Suppose we let Marvin work on partial orders: lying on the part of the trainer must be done in order to learn.

→ also with learning problem

3. ALVIN

- (a) Krawchuk (1987)
- (b) created to handle
 - i. partial orders, not just trees
 - ii. without lying
 - iii. side-effect: multiple-disjunct concepts can be learned without problems
- (c) Significant vs Crucial Objects
- (d) Learning Two-Leggedness
- (e) Applications
 - i. horn clauses re ordering
 - ii. nets and frame systems with multiple inheritance

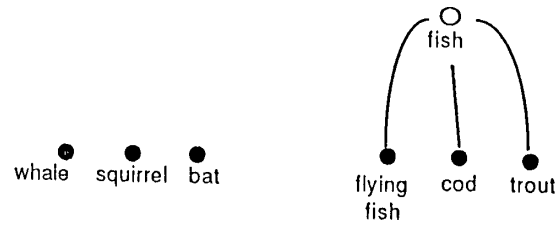
FROM MARVIN TO ALVIN

MARVIN

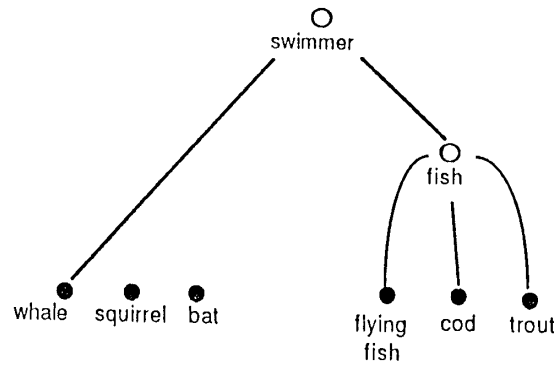
- general learning system
- learning by example
- learning by experimentation
- relationship between student and teacher
- Cohen(1978); Banerji, Sammut (1981)
- already-learned concepts can be used in future learning
- stores concepts as Prolog clauses

MARVIN: LEARNING SWIMMING

BEFORE



AFTER



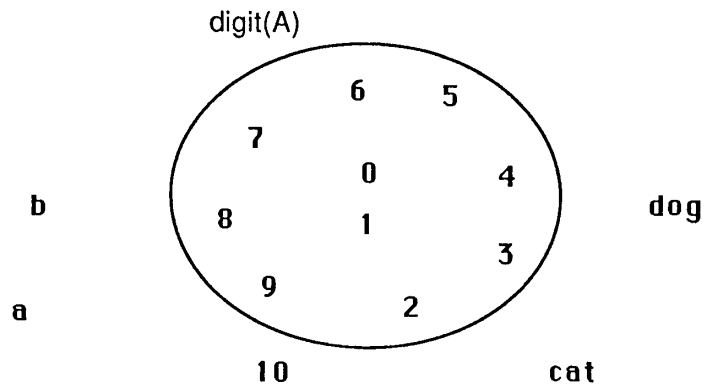
TWO DISJUNCTS CREATED:

swimmer(A) :- eq(A,whale).

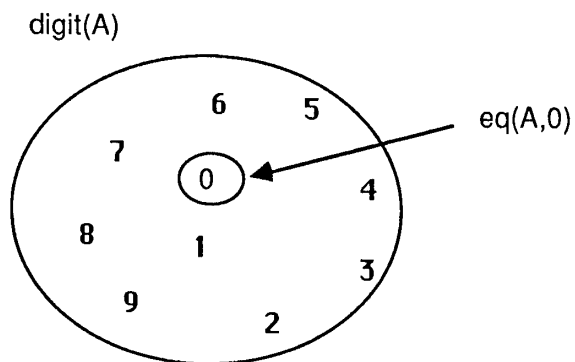
swimmer(A):- fish(A).

CONSTRUCTING CRUCIAL OBJECTS IN MARVIN

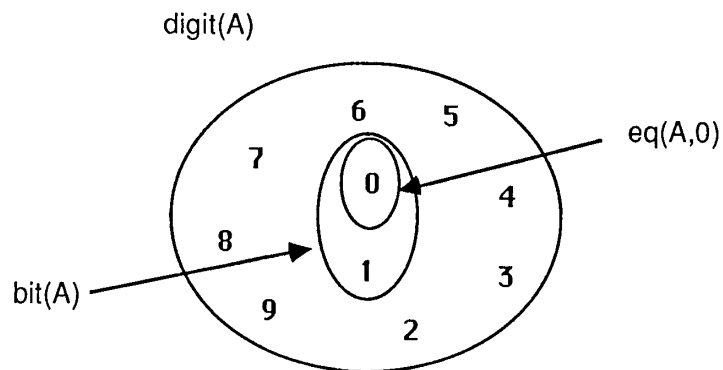
I. the object must be covered by the trial hypothesis



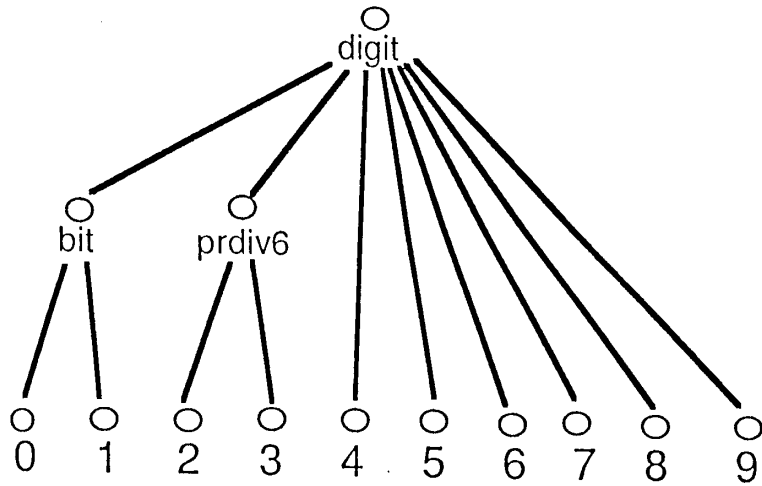
II. the object must not be in the currently accepted generalization



III. the object must not be in danger of being part of the target concept for a reason that differs from the one currently being tested.

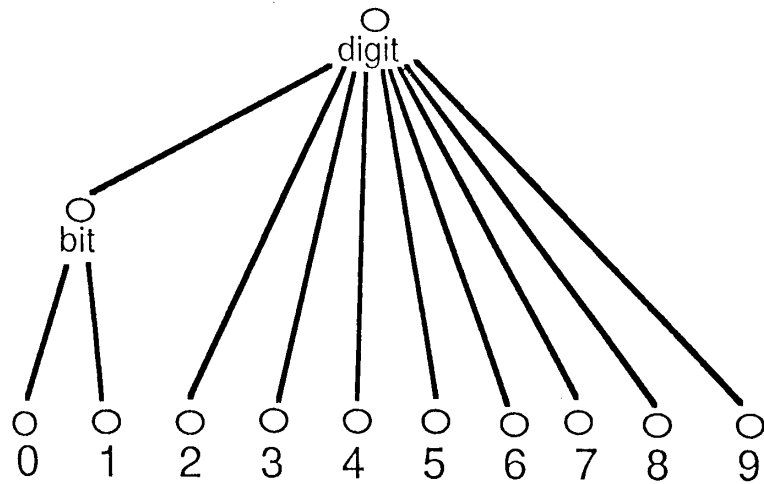


MARVIN OVERGENERALIZES ON THIS TREE

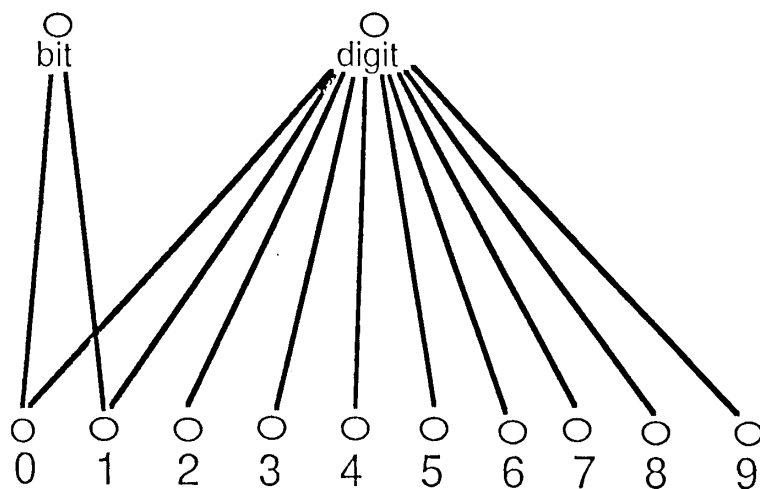


D6

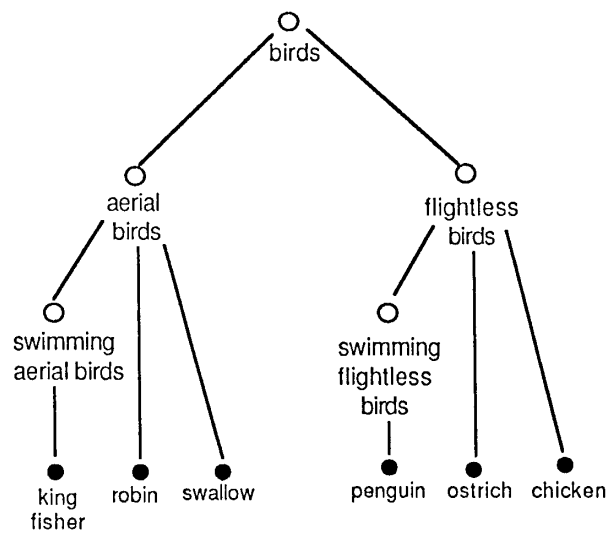
DOMAIN WITH TREE STRUCTURE



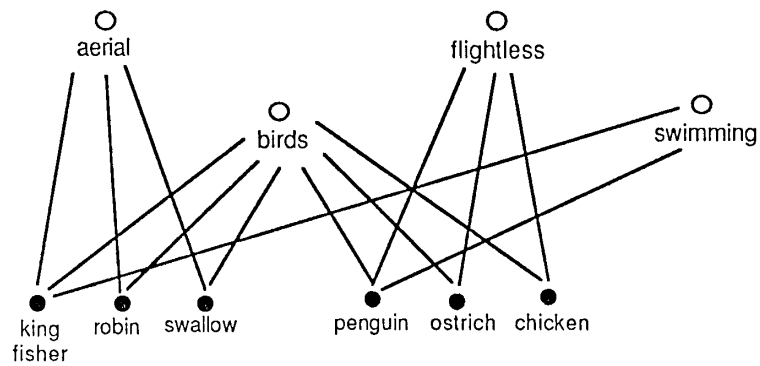
DOMAIN WITH PARTIAL ORDER STRUCTURE



DOMAIN WITH TREE STRUCTURE

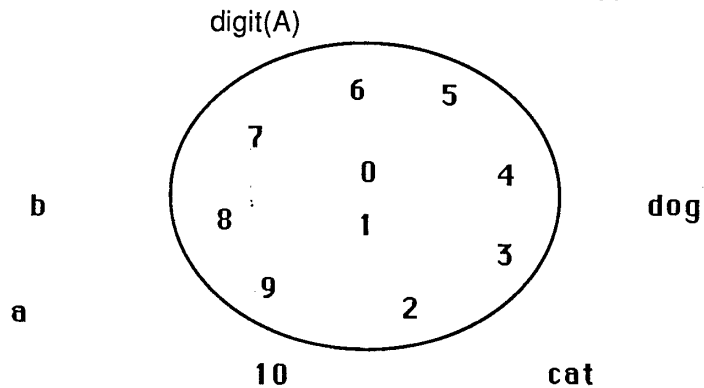


DOMAIN WITH PARTIAL ORDER STRUCTURE

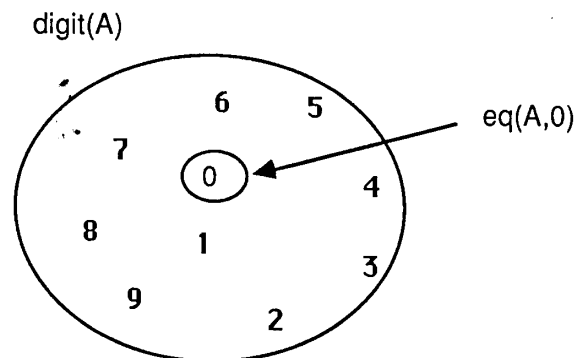


CHOOSING CRUCIAL OR SIGNIFICANT OBJECTS IN ALVIN

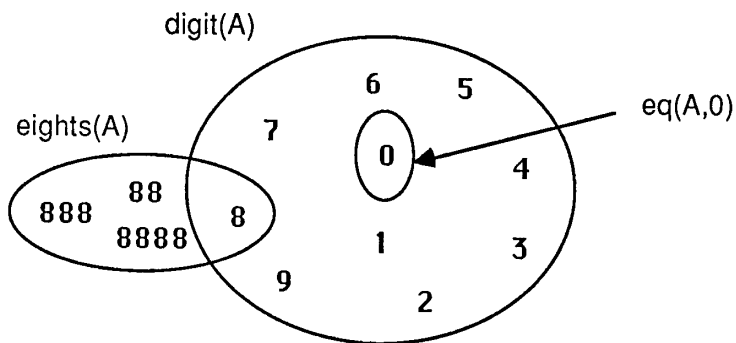
- I. the object must be covered by the trial hypothesis



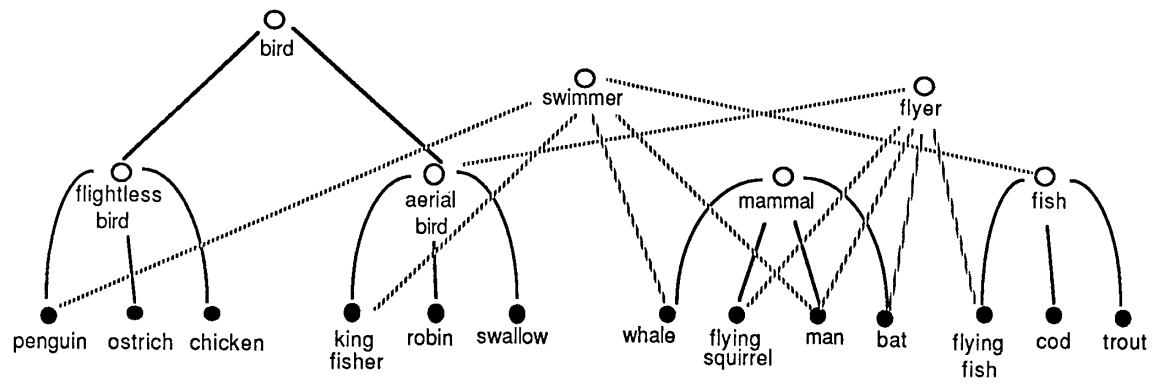
- II. the object must not be in the currently accepted generalization



- III. the object should not be in danger of being part of the target concept for a reason that differs from the one currently being tested. If it is in danger, the object is called a SIGNIFICANT object. Objects which are not in danger are called CRUCIAL objects.



ANIMAL KINGDOM (complex)



LEARNING TWO-LEGGEDNESS

An animal is two legged if

- 1) it is a man
- 2) it is a bat
- 3) it is an example of a bird

```
twolegged( $\Lambda$ ) :- eq( $\Lambda$ ,man).  
twolegged( $\Lambda$ ) :- eq( $\Lambda$ ,bat).  
twolegged( $\Lambda$ ) :- bird( $\Lambda$ ).
```

POSSIBLE REASONS FOR MAN'S TWO-LEGGEDNESS

it is a mammal
it is a swimmer
it is a flyer
it is a swimming mammal
it is a flying mammal
it is a swimming and flying mammal
it is a flying and swimming animal
just because it is a man

Learning Two-Leggedness: Summary

EXAMPLE	CURRENT GENERALIZATION	REFUTATION	DISJUNCT
man	MAMMAL		
whale X		whale	
	Swimming Mammal	whale	
	Swimming and Flying Mammal	only one object	
squirrel X	Flying Mammal		
	Flying and Swimming Mammal	squirrel	
	SWIMMER	only one object	
	Swimming Mammal	whale	
	Swimming and Flying Mammal	whale	
king fisher	Swimming and Flying Animal	only one object	
robin	AERIAL BIRD		
	verified		
ostrich	BIRD		
	verified		
flying fish X	no further gens possible		bird(Λ)
	FLYER		
	Flying Mammal	squirrel	
	Flying and Swimming Mammal	squirrel	
	Flyer and Swimmer	only one object	
	Flying and Swimming Mammal	flying fish	
	no more possible generalizations	only one object	
			eq(Λ , man)
bat	MAMMAL		
	Flying Mammal	whale	
	FLYER	squirrel	
	Flying Mammal	squirrel	
	no more possible generalizations	squirrel	
			eq(Λ , bat)

E Version Space

Reminder

A reminder of what is version space:

- a means of exploring the space of possible descriptions
- characteristics:
 - Description (or generalization) language
 - A partial ordering on descriptions
 - more-general-than-or-equal-to
 - S set – set of most specific descriptions
 - G set – set of most general descriptions

Different kinds of search spaces

- Boolean Search Space
 - Any subset of objects has a corresponding description
- Hierarchical Search Space
 - descriptions forms a hierarchy
- Attributive Search Space
 - an object is represented by attributes
 - values of each attribute forms a hierarchy

Focus

- sizes of S and G sets
- number of examples for convergence to a single description

Search Space	$ S $	$ G $	examples needed
Hierarchical	1	1	l
Attributive	1	1	$l \times m$
Boolean	1	1	$ \Omega $

Observations

- If it is able to describe any combinations of the objects in the given universe. Teaching any concept will require the teaching of the *characteristic function* of the concept, i.e., *all* objects in the universe must be given as examples in learning the concept.
- Winston's near-miss method is good because it limits the expansion of the G set and it works best in the hierarchical environment.
- In the case of Hierarchical and Attribute Search Space, we can see the 'depth' of search space and 'width' (number of attributes rather than branch factor) gives an approximation of the number of examples needed for convergence into a concept.
- Number of concepts in the search space does not necessarily tell the number of searching needed. Searching depends entirely on the structure of the space.
- As the complexity of the search space increases,

the number of examples needed for convergence increases.

Questions

- What will happen to $|G|$ when $|S| \geq 2$?
- What will happen to $|G|$ when the values of an attribute are not in a hierarchy ?
- Is there a possible of a set other than S and G , say M ?
- What about expansion of description language ?

Script started on Mon Jul 6 09:33:33 1987

% prolog eg

C-Prolog version 1.5

[Restoring file eg]

yes

1 ?- do([1,2,3,4,5]).

Version Space -- Released Version 2 (1987/06/05)

1 Positive instance: [large,blue,triangle]

S: [large,blue,triangle]

G: [size,colour,shape]

2 negative instance: [small,red,triangle]

S: [large,blue,triangle]

G: [size,blue,shape]
[large,colour,shape]

3 Positive instance: [large,blue,circle]

S: [large,blue,shape]

G: [size,blue,shape]
[large,colour,shape]

4 negative instance: [small,blue,triangle]

S: [large,blue,shape]

G: [large,colour,shape]

5 Positive instance: [large,red,triangle]

S: [large,colour,shape]

G: [large,colour,shape]

yes

| ?- do([1,4,5,2,3]).

Version Space -- Released Version 2 (1987/06/05)

1 Positive instance: [large,blue,triangle]


S: [large,blue,triangle]

G: [size,colour,shape]

4 negative instance: [small,blue,triangle]


S: [large,blue,triangle]

G: [large,colour,shape]

5 Positive instance: [large,red,triangle] 

S: [large,colour,triangle]

G: [large,colour,shape]

2 negative instance: [small,red,triangle] 

S: [large,colour,triangle]

G: [large,colour,shape]

3 Positive instance: [large,blue,circle]

S: [large,colour,shape]

G: [large,colour,shape]

yes

| ?-

[Prolog execution halted]

%

script done on Mon Jul 6 09:34:56 1987

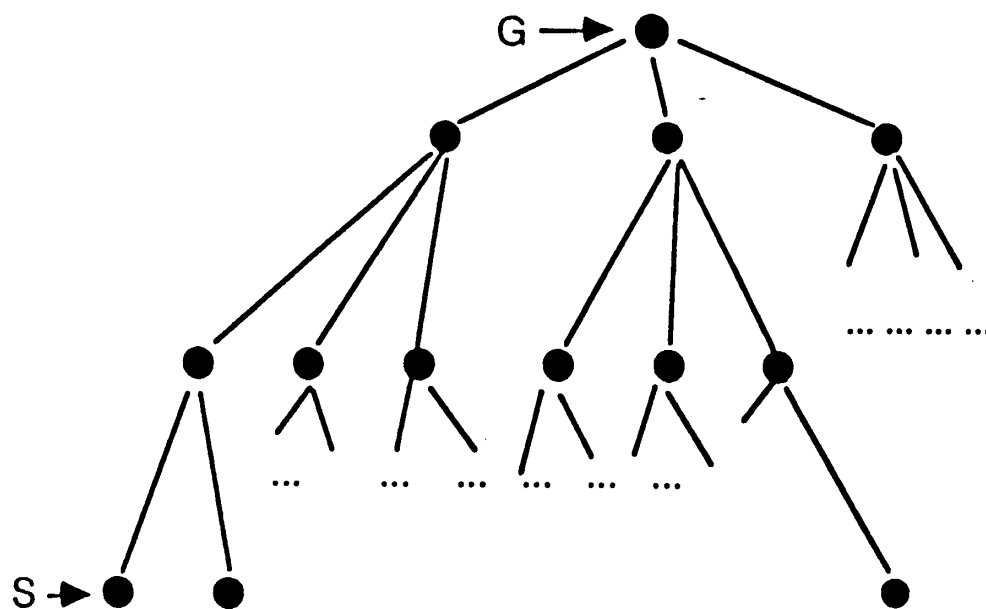


Fig. A hierarchical search space

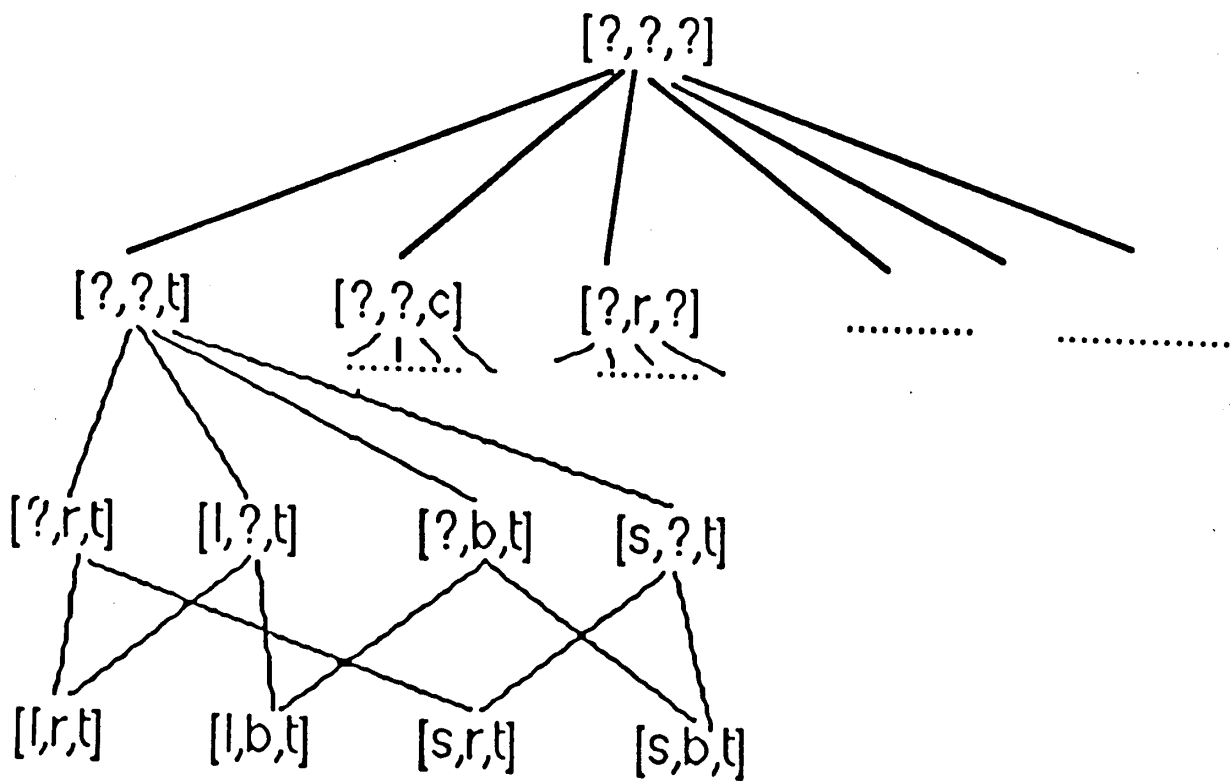


Fig. A attributive search space
 object : [size, colour, shape]

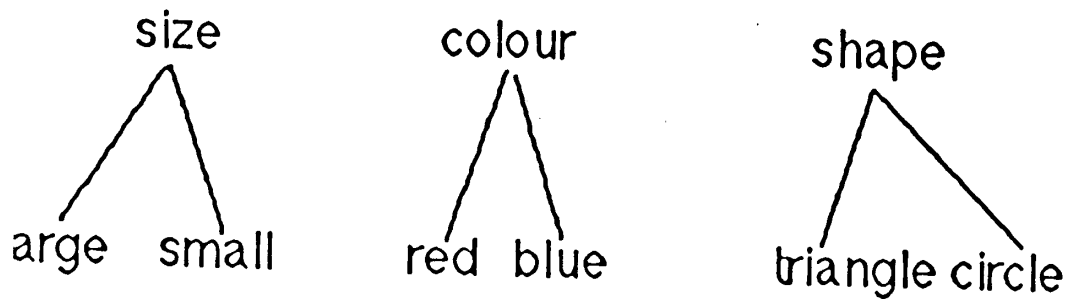


Fig. Attributive hierarchies

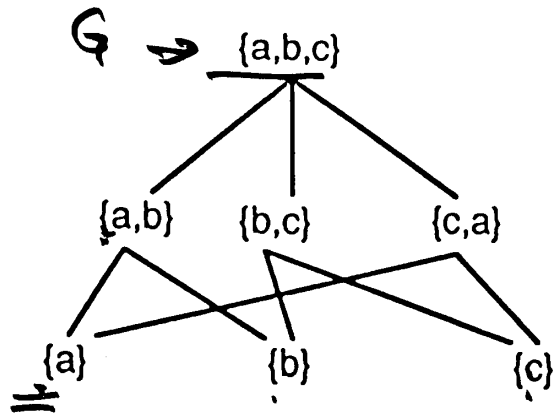
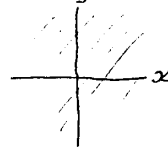


Fig. A Boolean algebra search space

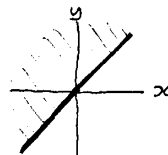
Description

Domain_y subset

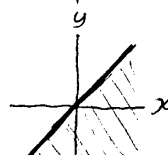
$x:y$



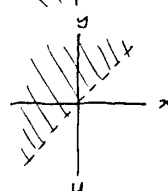
$x \leq y$



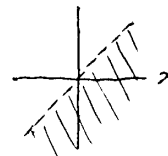
$x \geq y$



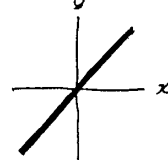
$x < y$



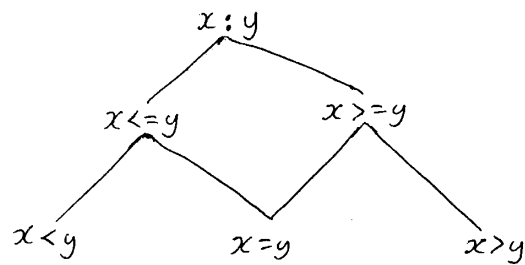
$x > y$



$x = y$



Specificity partial ordering



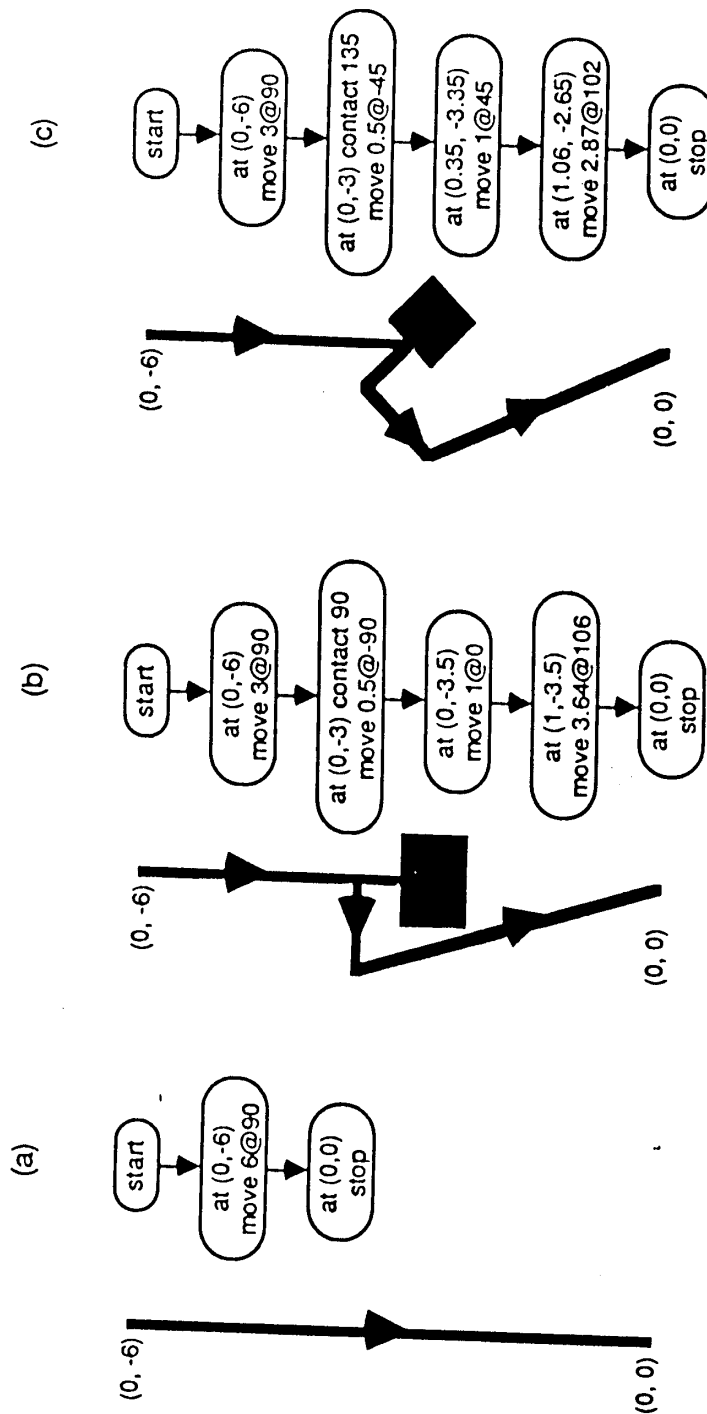
F Reconstructing Noddy

RECONSTRUCTING NODDY

- DESCRIPTION OF NODDY
- FUNCTION INDUCTION ALGORITHM
 - How Expressions are Built
 - The Problem with Constants
 - Composition of Operators
- AN IMPLEMENTATION IN PROLOG
 - Representation of Knowledge
 - inverse operators
 - argument typing
 - Representation of the Final Expression

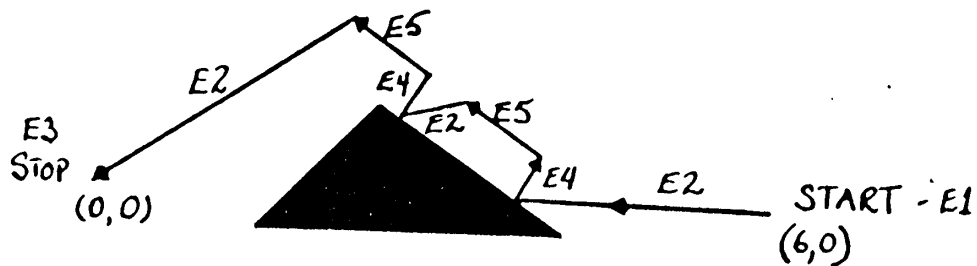
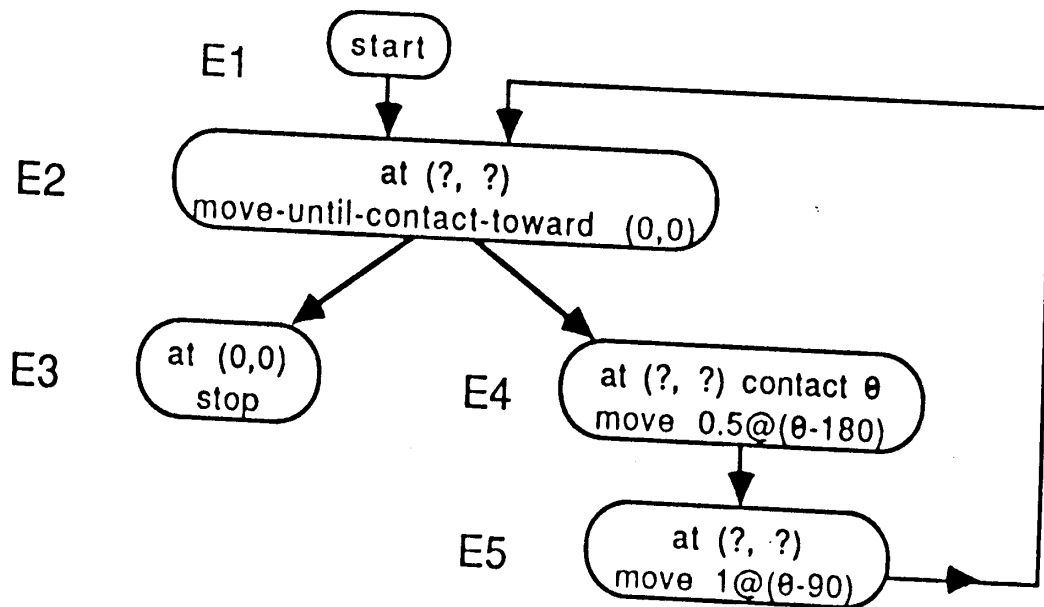
SLIDE 2

EXECUTION TRACES AS TRAINING EXAMPLES



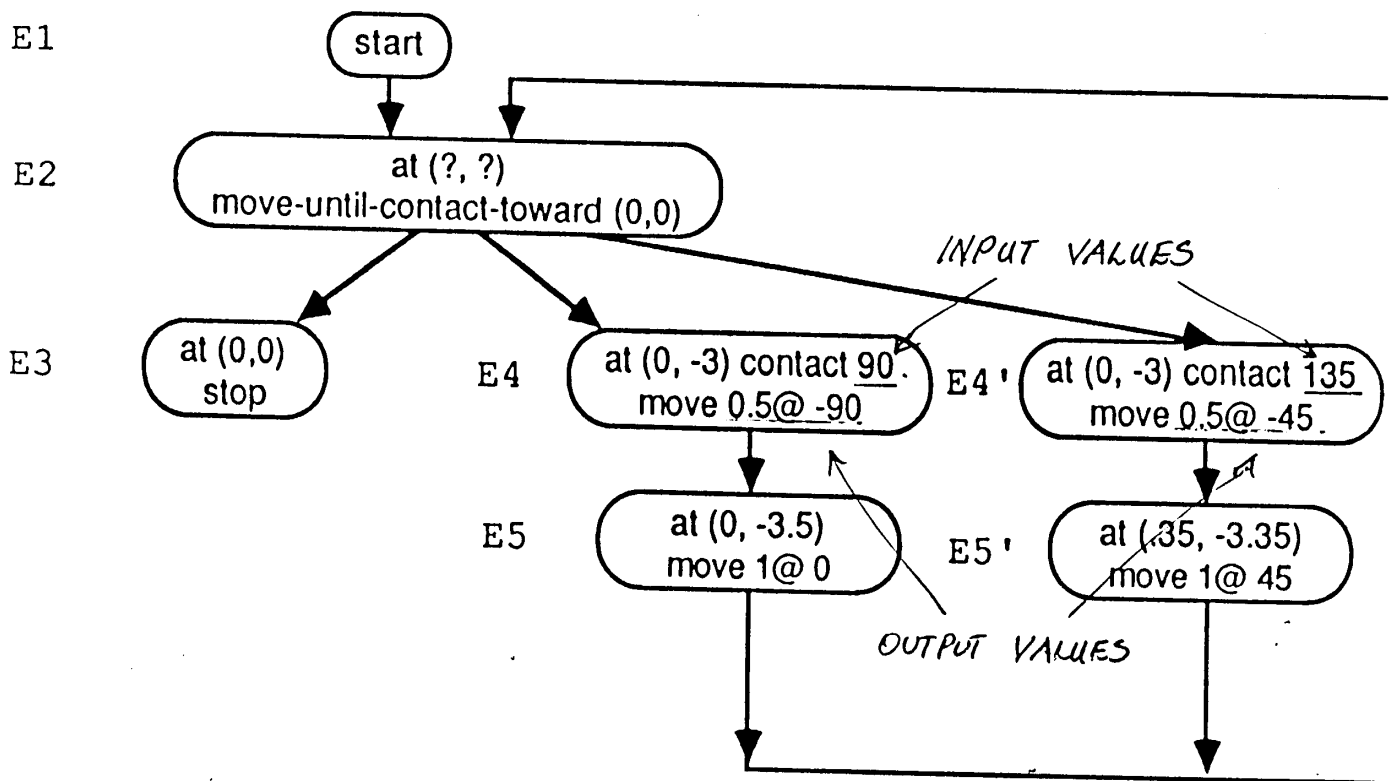
THE TURTLE PROCEDURE

1st two traces generate the structure



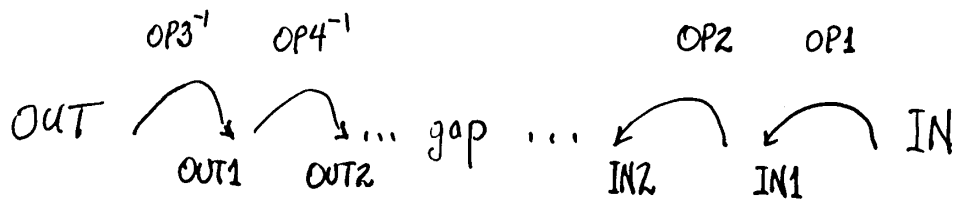
F4

FUNCTION INDUCTION



Partially accommodated third trace before function induction

BUILDING THE EXPRESSION



$$\text{expression} = (OP3 (OP4 (gap (OP2 (OP1))))))$$

CLOSING THE GAP:

- 1) If each of the current set of output values equals its associated input value, then remove the gap.
- 2) If there is a binary operator that, when applied to each input/output pair, produces a constant value, then replace the gap with that operator's inverse applied to the constant and the input value.

i.e. If $(OP \text{ } In_i \text{ } Out_i) = \text{Constant}$ then

$$(OP^{-1} \text{ Constant } In_i) = Out_i$$

CONSTANTS

Known Constants - appear as components in compound data types

e.g. move 0.5 @ -90
 move 0.5 @ -45

New Constants - must be derived from data
- one per expression

COMPOSITION OF OPERATORS

(OUT	(gap	IN)
$\{0.5 \angle -90, 0.5 \angle -45\}$		$\{90, 135\}$

(OUT \rightarrow (vect cons 0.5 (gap IN)

New Out = $\{-90, -45\}$	$\{90, 135\}$
--------------------------	---------------

(vect cons 0.5 (- IN 180))

REPRESENTATION OF KNOWLEDGE

Lisp:

Operator	Domain	Range	Inverse1	Inverse2
vect-cons	(num, dir)	vector	mag	direction
mag	(vector)	num	vect-cons	
direction	(vector)	dir	vect-cons	

Prolog:

operator(vect-cons(mag(Mag), dir(Dir), Mag@Dir)).

operator(pos-cons(x(X), y(Y), pos(X,Y))).

⋮

known constant list = {1,4}

known constant list = {y(1), mag(4)}

REPRESENTATION OF THE EXPRESSION

Lisp: $(\lambda \text{ In } (\langle \text{gap} \rangle \text{ In}))$
 $(\lambda \text{ In } (\text{pos-cons } 2 (\langle \text{gap} \rangle \text{ In})))$
 $(\lambda \text{ In } (\text{pos-cons } 2 (\langle \text{gap} \rangle (\gamma\text{-part } \text{In}))))$
 $(\lambda \text{ In } (\text{pos-cons } 2 (\text{minus } (\gamma\text{-part } \text{In}))))$

Prolog:

$\text{exp}((\text{Gap}, \text{pos-cons}(2, \text{In}, \text{Out})), \text{Gap}, \text{GI}, \text{GO}, \text{In}, \text{Out}).$

\downarrow
 $\text{Gap} = (\text{pos-cons}(-, \text{GI}, \text{In1}), \text{Gap1})$

\downarrow
 $\text{exp}(((\text{pos-cons}(-, \text{GI}, \text{In1}), \text{Gap1}), \text{pos-cons}(2, \text{GO}, \text{Out})), \text{Gap1}, \text{GI}, \text{GO}, \text{In1}, \text{Out})$

\downarrow
 $\text{Gap1} = \text{minus}(\text{GI}, \text{GO})$

\downarrow
 $\text{exp}(((\text{pos-cons}(-, \text{GI}, \text{In1}), \text{minus}(\text{GI}, \text{GO})), \text{pos-cons}(2, \text{GO}, \text{Out})), \text{In1}, \text{Out})$

G Kitten

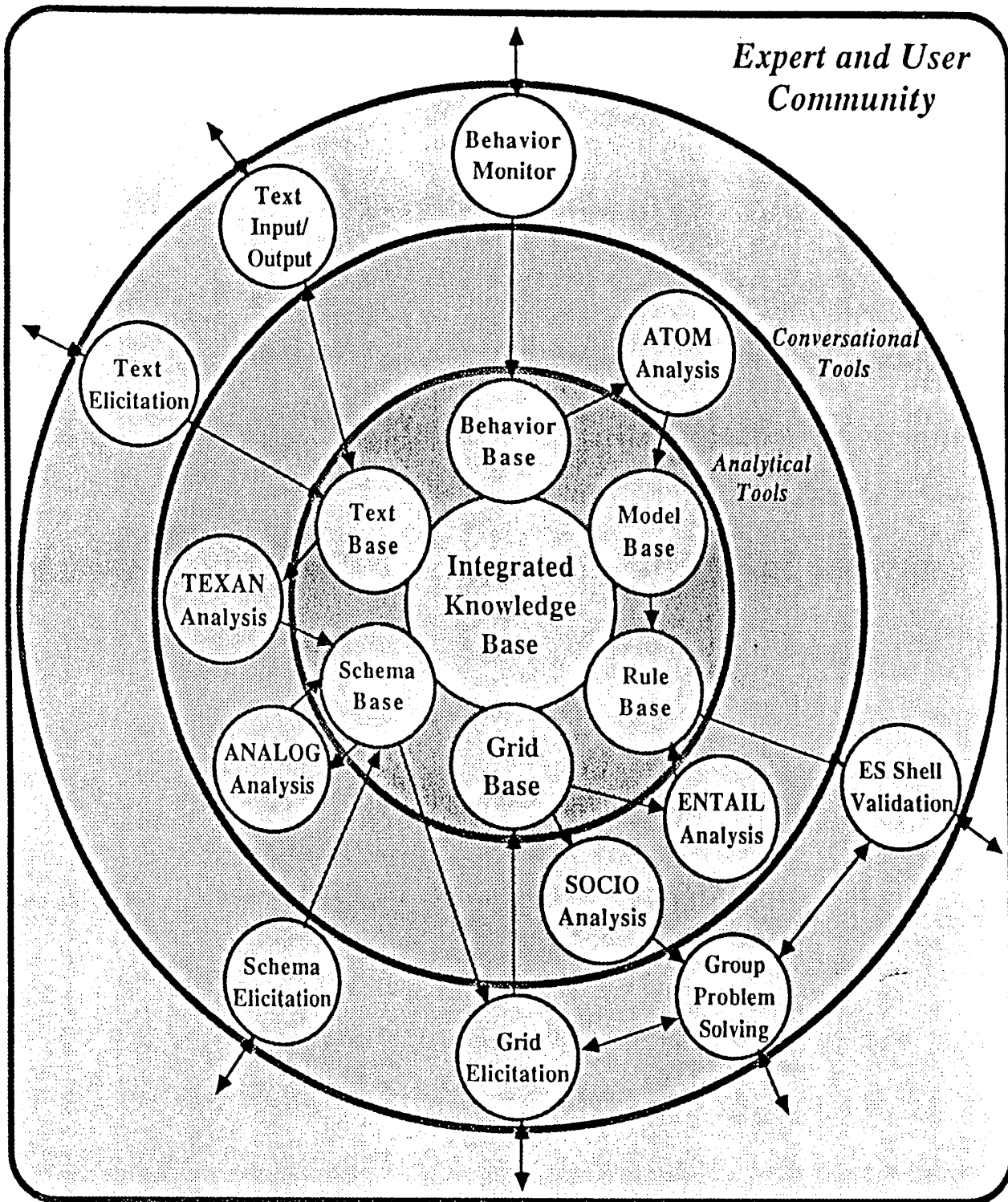
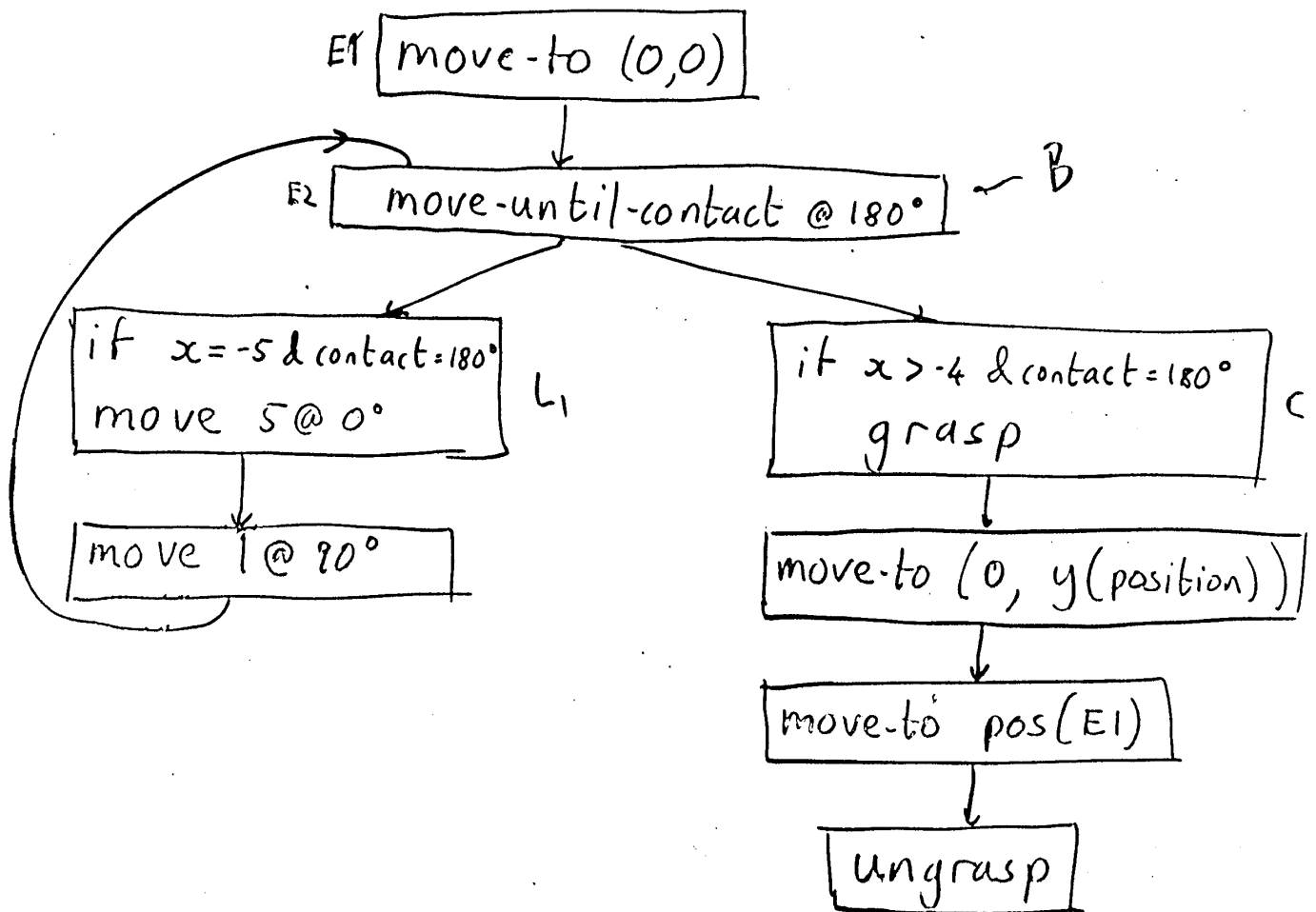
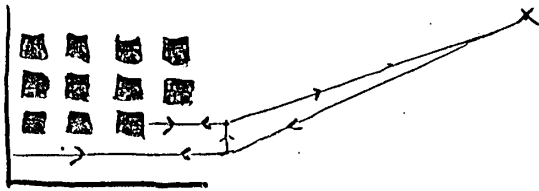
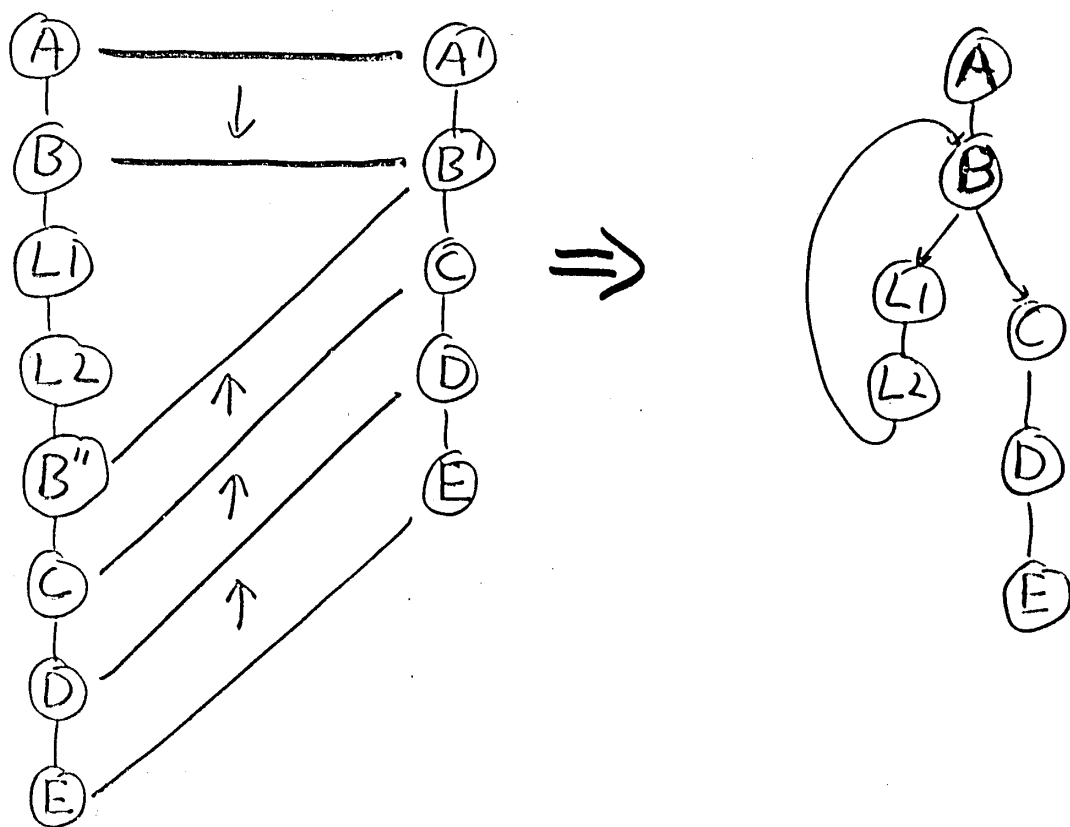
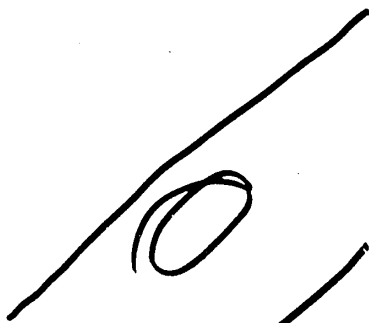


Fig.1 KSS1—KITTEN—
Knowledge Initiation & Transfer Tools for Experts & Novices

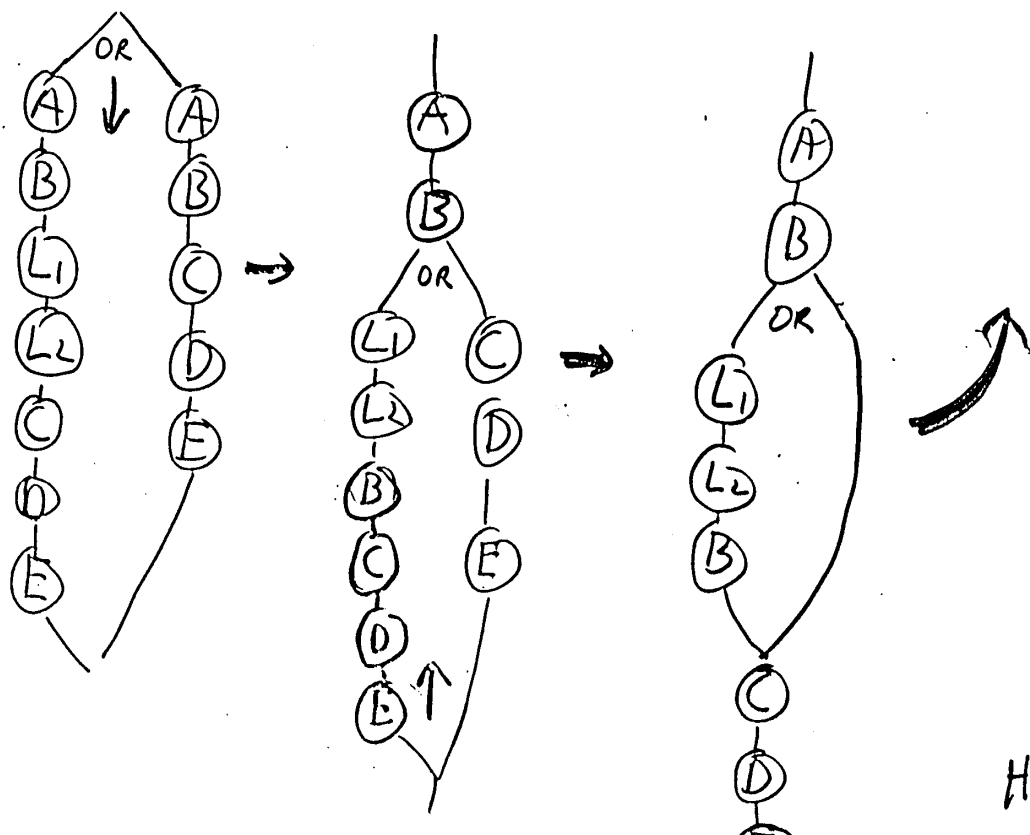
H Noddy



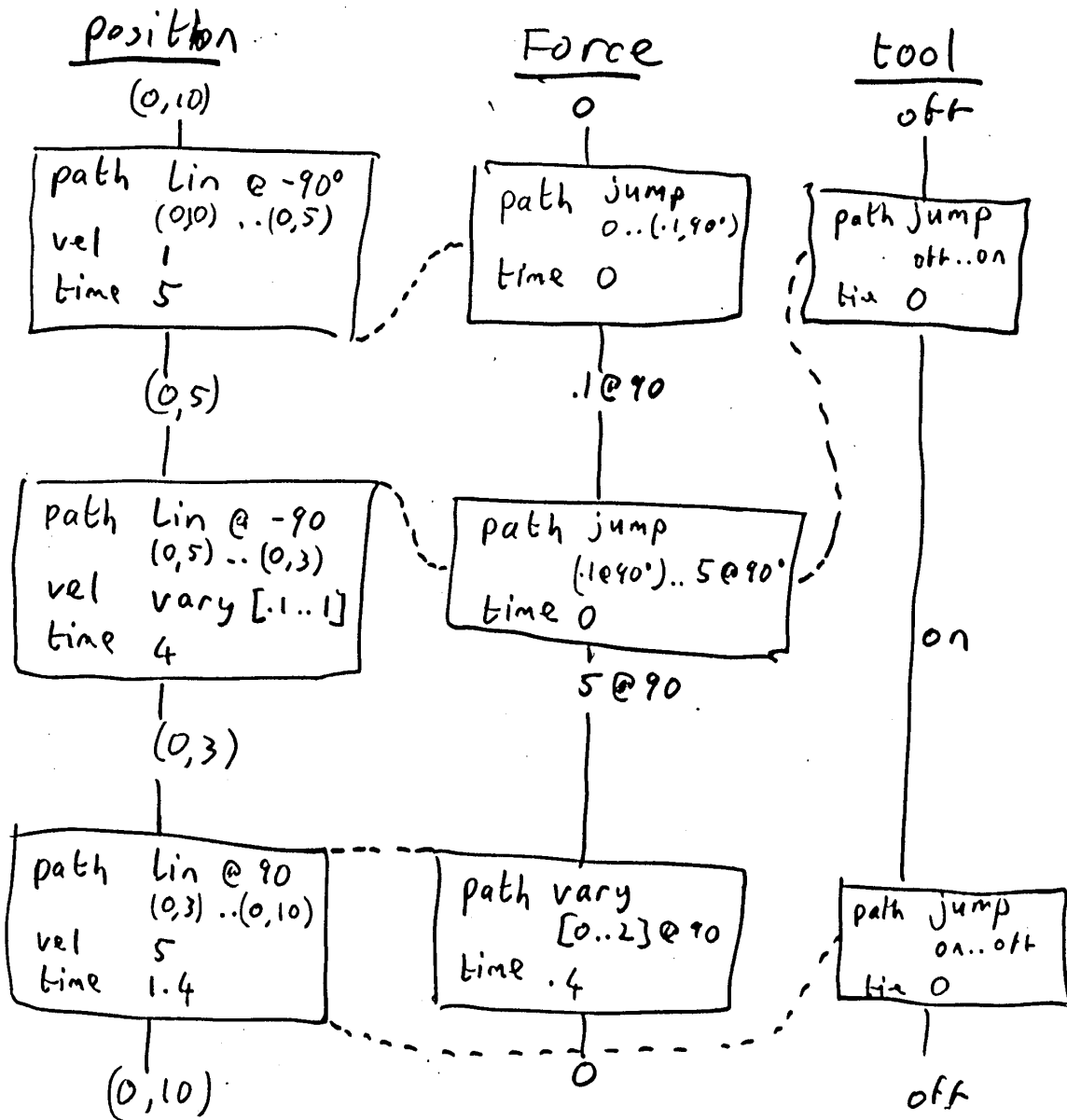




"Pushing OR's down"



H3



"Fill pallet."

- go to item feed
- count items by putting into a temporary store
- choose appropriate pallet
- put on bench
- get drill
- drill rows of holes, up to number of items
- put drill down
- put items from temporary store into holes
- place filled pallet on conveyor belt

Drill-hole

State-1

when: t

move-search @-90°
until: done

state 2

push 5 @-90°
until y(pos)=3

turn-tool on

state 3

move-to pos(state 1)

state 4

turn-tool off

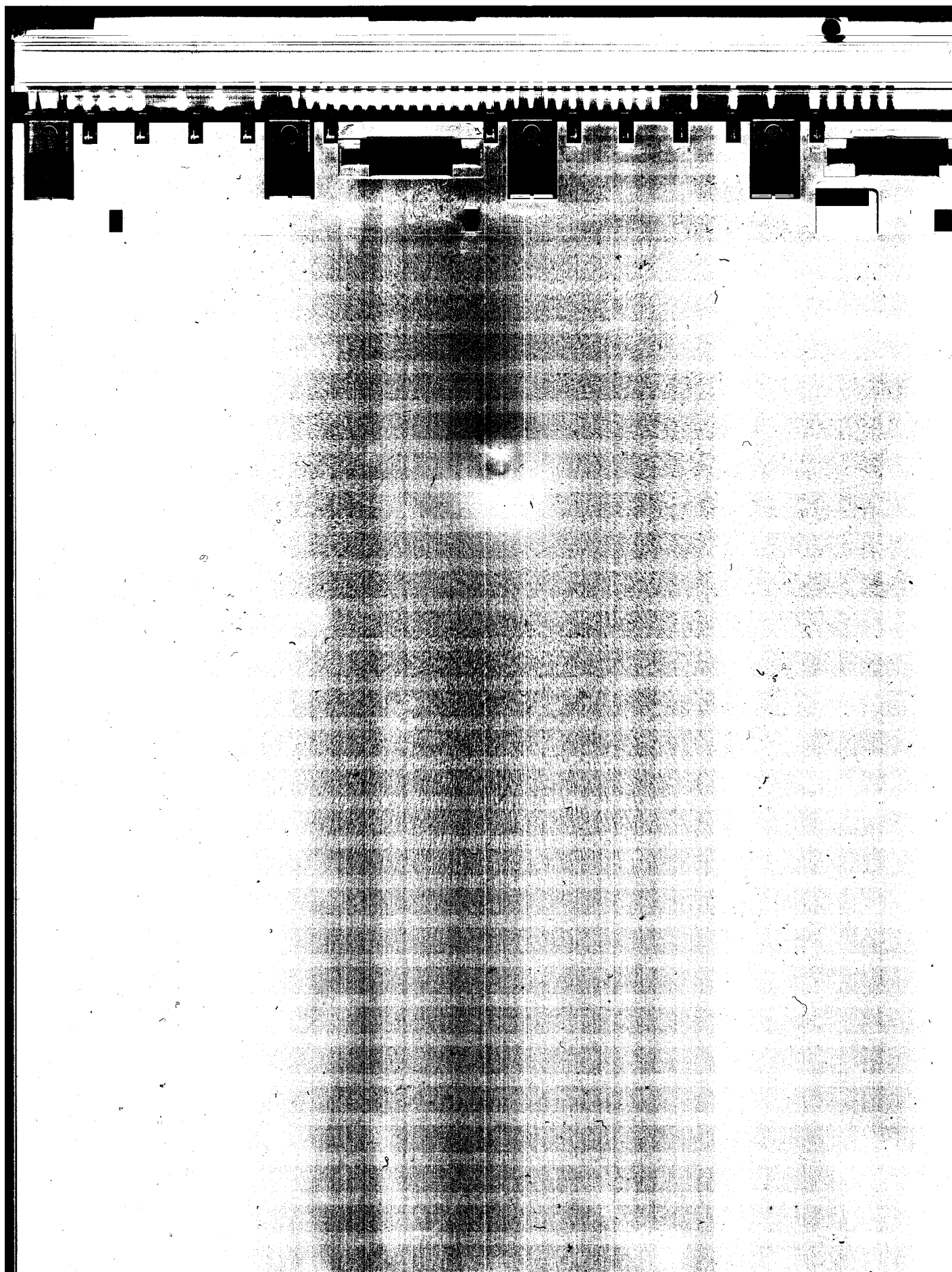
move-search (angle)

move path: Lin @ angle
position .. any

velocity: 1

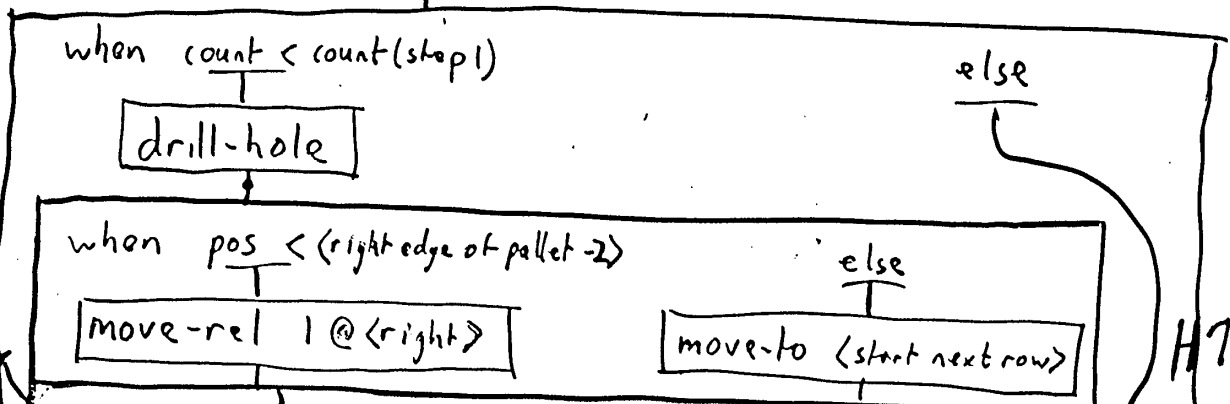
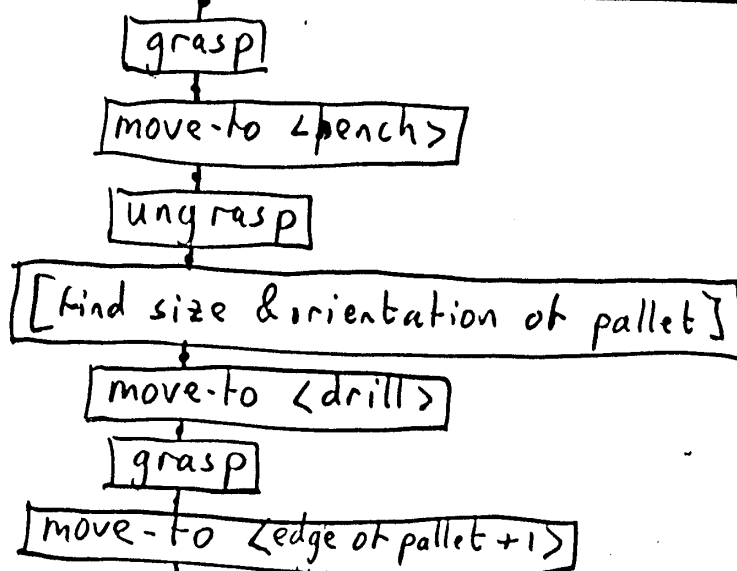
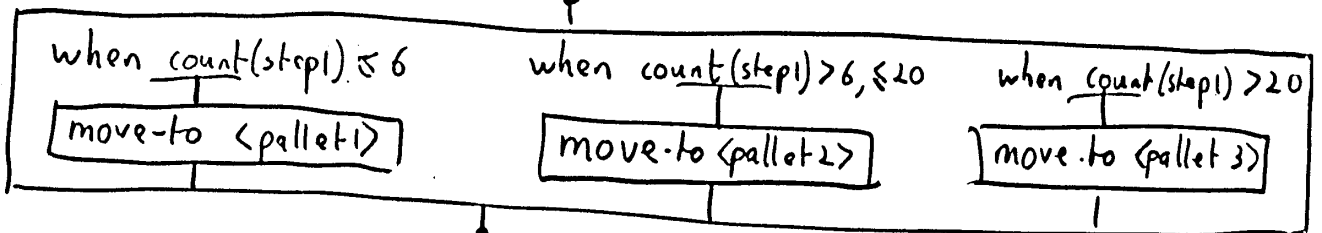
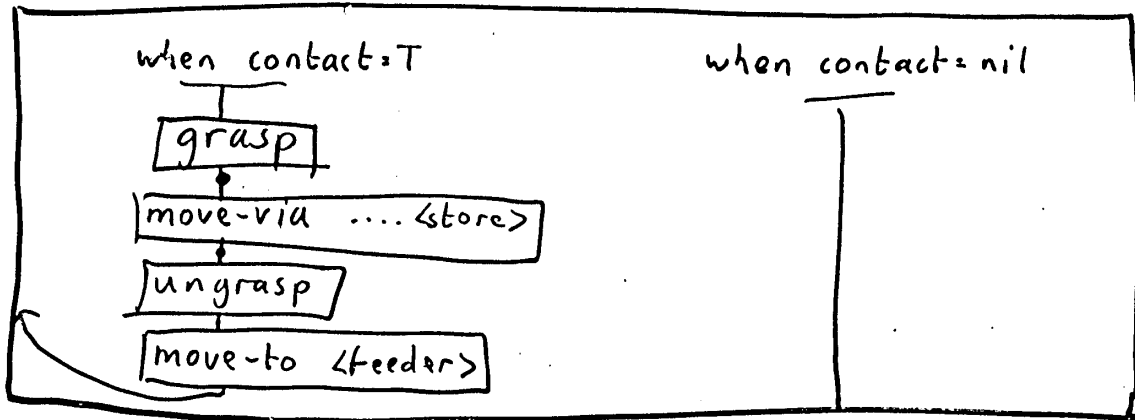
Force: $F_n = .1$

H6



when T

move-to <feeder>



Black board

Evolving Description	proposed pairs & groups	
	new generalized items	
	critics' messages	
correspondence proposers & critics □ □ □ □ □	generalizers □ □ □ □	Installation experts and critics □ □ □ □ □

- Add new example as a disjunct to old description
- simplify

I Table of Contents from the Machine Learning Workshop

P. Andreae

Proceedings of the Fourth International Workshop on MACHINE LEARNING

June 22-25, 1987
University of California, Irvine

Editor/Program Chair: Pat Langley

Organizing Committee:

Jaime G. Carbonell, Carnegie-Mellon University
Richard H. Granger, University of California, Irvine
Dennis F. Kibler, University of California, Irvine
Pat Langley, University of California, Irvine
Ryszard S. Michalski, University of Illinois, Urbana
Tom M. Mitchell, Carnegie-Mellon University

Administrative Organizer: Caroline Ehrlich

Sponsored by:

American Association for Artificial Intelligence
Defense Advanced Research Projects Agency
Program in Computation and Learning, University of California, Irvine
Honeywell
National Science Foundation, Information Science Program (IST-8615825)
Office of Naval Research, Information Sciences Division (N00014-87-G-0107)

MORGAN KAUFMANN PUBLISHERS, INC.

95 First Street, Los Altos, CA 94022

CONTENTS

Preface: The Emerging Science of Machine Learning	v
<i>P. Langley</i>	
LEARNING AND CLASSIFICATION	
EXEMPLAR-BASED APPROACHES	
Learning About Speech Sounds: The NEXUS Project	1
<i>G. Bradshaw</i>	
PROTOS: An Exemplar-Based Learning Apprentice	12
<i>E. R. Bareiss and B. W. Porter</i>	
Learning Representative Exemplars of Concepts: An Initial Case Study	24
<i>D. Kibler and D. W. Aha</i>	
PROBABILISTIC APPROACHES	
Decision Trees as Probabilistic Classifiers	31
<i>J. R. Quinlan</i>	
Conceptual Clustering, Learning from Examples, and Inference	38
<i>D. H. Fisher</i>	
How to Learn Imprecise Concepts: A Method for Employing a Two-Tiered Knowledge Representation in Learning	50
<i>R. S. Michalski</i>	
Quasi-Darwinian Learning in a Classifier System	59
<i>S. W. Wilson</i>	
CONCEPT LEARNING AND BIAS	
More Robust Concept Learning Using Dynamically-Variable Bias	66
<i>L. Rendell, R. Seshu, and D. Tcheng</i>	
Incremental Adjustment of Representations for Learning	79
<i>J. C. Schlimmer</i>	
LEARNING, PROBLEM SOLVING, AND PLANNING	
HEURISTIC SEARCH APPROACHES	
Concept Learning in Context	91
<i>R. M. Keller</i>	
Strategy Learning with Multilayer Connectionist Representations	103
<i>C. W. Anderson</i>	
Learning a Preference Predicate	115
<i>P. E. Utgoff and S. Saxena</i>	
PLANNING APPROACHES	
Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System	122
<i>S. Minton, J. G. Carbonell, O. Etzioni et al.</i>	
The Anatomy of a Weak Learning Method for Use in Goal Directed Search	134
<i>T. L. McCluskey</i>	
Learning and Reusing Explanations	141
<i>K. J. Hammond</i>	
PROBLEM REDUCTION APPROACHES	
LT Revisited: Experimental Results of Applying Explanation-Based Learning to the Logic of Principia Mathematica	148
<i>P. O'Rorke</i>	
What Is an Explanation in DISCIPLE?	160
<i>Y. Kodratoff and G. Tecuci</i>	
Extending Problem Solver Capabilities Through Case-Based Inference	167
<i>J. L. Kolodner</i>	

LEARNING AND NATURAL LANGUAGE	
Learning to Integrate Syntax and Semantics	179
<i>W. G. Lehnert</i>	
How Do Machine-Learning Paradigms Fare in Language Acquisition?	191
<i>U. Zernik</i>	
The Acquisition of Polysemy	198
<i>J. H. Martin</i>	
MACHINE DISCOVERY	
OBSERVATIONAL DISCOVERY	
Cirrus: An Automated Protocol Analysis Tool	205
<i>K. VanLehn and S. Garlick</i>	
Scientific Theory Formation Through Analogical Inference	218
<i>B. Falkenhainer</i>	
Inducing Causal and Social Theories: A Prerequisite for Explanation-based Learning	230
<i>M. J. Pazdani</i>	
The Role of Abstractions in Learning Qualitative Models	242
<i>I. Mozetic</i>	
DISCOVERY AND EXPERIMENTATION	
Learning by Experimentation	256
<i>J. G. Carbonell and Y. Gil</i>	
Observation and Generalisation in a Simulated Robot World	267
<i>C. Sammut and D. Hume</i>	
Empirical and Analytic Discovery in IL	274
<i>M. H. Sims</i>	
Combining Many Searches in the FAHRENHEIT Discovery System	281
<i>J. M. Zytkow</i>	
COGNITIVE ARCHITECTURES FOR LEARNING	
Causal Analysis and Inductive Learning	288
<i>J. R. Anderson</i>	
Varieties of Learning in Soar: 1987	300
<i>D. M. Steier, J. E. Laird, A. Newell, P. S. Rosenbloom et al.</i>	
Hill-Climbing Theories of Learning	312
<i>P. Langley, J. H. Gennari, and W. Iba</i>	
OVERVIEWS	
Bias, Version Spaces and Valiant's Learning Framework	324
<i>D. Haussler</i>	
Recent Results on Boolean Concept Learning	337
<i>M. Kearns, M. Li, L. Pitt, and L. Valiant</i>	
Machine Learning from Structured Objects	353
<i>R. E. Stepp</i>	
A New Approach to Unsupervised Learning in Deterministic Environments	364
<i>R. L. Rivest and R. E. Schapire</i>	
Searching for Operational Concept Descriptions in BAR, MetaLEX, and EBG	376
<i>J. Mostow</i>	
Explanation-Based Generalization as Resolution Theorem Proving	383
<i>S. T. Kedar-Cabelli and L. T. McCarty</i>	
Analogy and Single-Instance Generalization	390
<i>S. J. Russell</i>	
Index	403