Perception and Experience in Problem Solving

Edmund Furse
Department of Computer Studies
University of Glamorgan
Pontypridd, Mid Glamorgan
CD37 1DL
UK
efurse@uk.ac.glam.genvax

Rod Nicolson
Department of Psychology
University of Sheffield
Sheffield
S10 2TN
UK
pclrin@uk.ac.sheffield.sunc

Abstract

Whilst much emphasis in AI has been placed on the use of goals in problem solving, less emphasis has been placed on the role of perception and experience. In this paper we show that in the domain that may be considered the most abstract, namely mathematics, that perception and experience play an important role. The mathematician has a vast amount of mathematical knowledge, and yet is able to utilise the appropriate knowledge without difficulty. We argue that it is essential to model how well the knowledge is grasped, so that mathematical knowledge can grow from partial knowledge to important results that are easily accessed. Not all knowledge is equal in its importance, and we argue that perception and experience play a key role in ordering our knowledge. Features play a role in both representing the information from the environment, and indexing the knowledge of our memories, but a key requirement is that the features should be dynamic and not be built in. This research is implemented in the program MU, the Mathematics Understander, which utilises the CMS, Contextual Memory System. MU has sucessfully "read" university level texts in pure mathematics, checking the proofs and solving the simple problems.

1. Introduction

Problem Solving has been thought of as the primary examplar of intelligence, and has been central to work in artificial intelligence from the early work of Newell and Simon's GPS, to expert systems and theorem proving. Laird, Rosenbloom and Newell's SOAR (1987) has problem solving as the cornerstone of its architecture. However, whilst expert systems, and in particular Lenat and Feigenbaum (1991), have shown the importance of a large amount of knowledge to give systems power, theorem proving in contrast has tended to concentrate on general methods such as resolution, tempered by meta-level reasoning (eg Bundy 1983). Given the evidence that expert problem solving tends to have access to large amounts of knowledge and tends to use shallow search methods, it is surprising that much of the AI community continues to pursue methods which are knowledge thin rather than knowledge rich.

It can be argued that if one is concerned with discovering powerful machine problem solving methods, then these do not necessarily have to be similar to human problem solving methods. Furthermore, it is clearly more elegant to have neat general problem solving methods, rather than a collection of special purpose methods. However, Schank (1981) has argued that even if one's goal is to build intelligent machines, it is a good first step to model how humans perform the task. We argue that the neat approaches to problem solving do not sufficiently model genuine ecological tasks, are of insufficient power, and since they take little account of learning are inadequate accounts of human cognition.

Instead we present an alternative thesis on cognition which places learning at the centre of what it is to be intelligent, rather than placing problem solving at the centre. Interestingly, this is a view much nearer to Alan Turing's thought (1953), than his Turing machine conception of computation. Turing wanted to know not only how machines could solve problems, but how they could learn. Certainly SOAR places great impotance on learning, but it is in our view an impoverished view of learning to see it only as search within a problem space.

For too long perception has been seen as almost a separate faculty from the mainstream of AI and cognition, and yet deGroot (1966) and Chase and Simon (1973) have convincingly showed that expert knowledge in chess is very largely a matter of 50,000 perceptual features. One could argue that board games are naturally prone to perceptual processing, but we show that the same arguments apply in the domain of pure mathematics. Given the highly abstract nature of pure mathematics, if perceptual features play an important part in problem solving in this domain, it may well be the case that they play an important role in many other types of problem solving.

It is easy to demonstrate that perception plays an important part in problem solving. Consider the following proposition:

$$\forall x > 0 \exists y \forall z > y \ |\phi_n - \phi| < x \qquad [1)$$

[1) is difficult to recognise, but by using the usual letter names as in [21:

$\forall \varepsilon > 0\ \exists N\ \forall n > N\ |x_n - x| < \varepsilon$ [2]

we have a proposition that is easily recognised by mathematicians as the definition of the limit of the sequence x_n as n tends to infinity is x. However, since [1] is obtained from [2] simply by a change of letters, it is logically equivalent to [1]. Thus there is more to problem solving than logic. What we are arguing is that expressons such as $\forall \varepsilon > 0$ act as perceptual features which enable the expert to rapidly recognise the mathematical result.

It is obvious that experience plays a role in an expert's ability to solve problems. However, apart from our own work, we believe there is no convincing account of how this might work. Too many models of expertise are relatively static, and yet all expertise must have been learned at some time, and experts continue to grow in expertise. Through the study of education, and in particular mathematics text books, one can see how knowledge is slowly acquired. We argue that in text books, any particular mathematical result goes through three stages of usage in problem solving:

1. The result is stated explicitly, for example:
$(3x - 2y)(3x + 2y) = (3x)^2 - (2y)^2$ since $(a-b)(a+b) = a^2 - b^2$
2. The result is used implicitly, for example:
$(3x - 2y)(3x + 2y) = (3x)^2 - (2y)^2$
(ie the same as (1) but without the explanation).
3. The result is used in compound inferences, e.g.:
$(3x - 2y)(3x + 2y) = 9x^2 - 4y^2$

Not all mathematical results get beyond stage (1), some results may only be used a few times, and may then be forgotten. In contrast, results which are frequently used become so well known through their stage (2) usage, that eventually they get used in stage (3). Thus, ironically the most important results are the ones which are not mentioned at all, precisely because they are so well known.

If all of an expert's knowledge was of equal importance, and therefore equally easy/difficult to access, most experts could not function at all. Yet most truth maintenance systems work on this basis. In contrast we argue that the expert has a perceptual system and memory so organised that important results are easily recognised and retrieved. Furthermore, we will show how the Contextual Memory System, CMS, (Furse 1992, Furse and Nicolson 1992) allows a continuing change of the perceptual and memory systems, so that the novice can become the expert through sufficient learning experience.

The CMS is a network architecture of features and items. The features are used to both encode the external object in the environment, and to index items in memory. Both features and items have an energy level, and the links between features and items have a strength. The novel characteristic of this architecture is that the features are generated dynamically from the environment, and the configuration of connections is frequently changed during memory processes.

2. Pure Mathematics

Pure Mathematics is a good domain in which to study problem solving since although it is a genuinely ecologically valid task, it uses little common sense knowledge, and all the mathematical knowledge has been acquired through learning. Indeed, some branches of pure mathematics, such as group theory, can be learned by arts undergraduates, illustrating the point that the subject can be learned with little prior knowledge. Within a course a large body of knowledge is built up which is used to understand proofs and solve problems.

This research has concentrated on modelling the understanding of mathematics texts. Since about the beginning of this century pure mathematics texts have taken the form of definitions, theorems and lemmas (a lemma is a little theorem), their proofs and exercises. Clearly it would be a large scale exercise to develop a program to read verbatim mathematics texts, and so the natural language component has been factored out by rewriting the texts by hand in the language FEL (Formal Expression Language), Furse (1990). FEL has a formal syntax and is very expressive, as the example in Figure 1 shows.

Definition 2.4 of converges
«s_n» converges to α
iff $\forall \varepsilon > 0\ \exists m\ \forall n > m\ |s_n - \alpha| < \varepsilon$

Lemma 2.1.1
not («$(-1)^n$» converges to 1)
Proof
1 RTP not («$(-1)^n$» converges to α)
2 Suppose to the contrary «$(-1)^n$» converges to 1
3 $\Rightarrow \forall \varepsilon > 0\ \exists m\ \forall n > m\ |(-1)^n - 1| < \varepsilon$ by definition of converges
4 $\Rightarrow \exists m\ \forall n > m\ |(-1)^n - 1| < 1$ by substituting $\varepsilon = 1$
5 let $n = 2m + 1$
6 $\Rightarrow$ n is odd
7 $\Rightarrow (-1)^n = -1$
8 $\Rightarrow |(-1)^n - 1| = |-2|$ since $a = b \Rightarrow |a-1| = |b-1|$
9 $= 2$
10 $\Rightarrow 2 < 1$ by combining steps 4 and 9
11 $\Rightarrow$ contradiction
12 QED

Figure 1

There are many levels of understanding a mathematics text, but this research has focused on the ability to check proofs by giving explanations of the steps and solving the simple problems. The problem solver uses a number of built-in general heuristics and uses the CMS to filter the mathematical results to ensure there is no combinatorial explosion. Many theorem provers model an artificial task by carefull feeding of only the results needed to solve the problem. In contrast, MU has access to the whole body of mathematical knowledge it has learned at the time, and uses its experience to focus on the problem in hand.

A typical problem and its solution by MU is shown in Figure 2. The output shows a solution only slightly more verbose than a human protocol.

Problem 2.7.2
Prove (G isa group and $g \in G$ and f isa function from G to G and $f(x) = gxg^{-1}$) $\Rightarrow$ f isa isomorphism from G to G

Solution
Suppose G isa group and $g \in G$ and f isa function from G to G and $f(x) = g(xg^{-1})$
RTP f isa isomorphism from G to G
RTP f isa homomorphism from G to G and f isa one-to-one from G to G by definition of isomorphism
Part 1
RTP f isa homomorphism from G to G
RTP f isa function from G to G and $\forall a,b$ $a \in G$ and $b \in G \Rightarrow f(ab) = f(a)f(b)$ by definition of homomorphism
RTP $\forall a,b$ $a \in G$ and $b \in G \Rightarrow f(ab) = f(a)f(b)$
Suppose $a \in G$ and $b \in G$
RTP $f(ab) = f(a)f(b)$
Now $f(x) = g(xg^{-1})$
RTP $g((ab)g^{-1}) = (g(ag^{-1}))(g(bg^{-1}))$
$(g(ag^{-1}))(g(bg^{-1}))$
$= gaebg^{-1}$ since $a^{-1}a = e$
$= gabg^{-1}$ since $ae = a$
$= g((ab)g^{-1})$ since associative-law
QED Part 1
Part 2
RTP f isa one-to-one from G to G
RTP f isa function from G to G and $f(a) = f(b) \Rightarrow a = b$ by definition of one-to-one
RTP $f(a) = f(b) \Rightarrow a = b$
Now $f(x) = g(xg^{-1})$
RTP $g(ag^{-1}) = g(bg^{-1}) \Rightarrow a = b$
$g(ag^{-1}) = g(bg^{-1})$
$\Rightarrow ag^{-1} = bg^{-1}$ since $ab = ac \Rightarrow b = c$
$\Rightarrow a = b$ since $ba = ca \Rightarrow b = c$
QED Part 2

Figure 2

3. Perception and Features

Many models of perception and memory use a fixed set of features, but such models have difficulty with pure mathematics as it is a large and completely open-ended domain. A pure mathematician can at any point introduce a new concept by means of a definition. Most models tend to be reductionist, seeking to form representations at some lowest common demoninator, or primitives. However, it seems unlikely that a mathematican is thinking in terms of concepts at this sort of level. For example, consider the following lemma:
Lemma. If ψ is a homomorphism of G_1 onto G_2 with kernel J, then J is a normal subgroup of G_1.
This lemma can be unpacked futher by use of the definitions of homomorphism, kernel and normal subgroup, so that we obtain a representation like:
Lemma. If ψ is a function from G_1 onto G_2 and $\psi(a*b) = \psi(a)*\psi(b)$, and $\psi^{-1}(G_2) = J$, then J is a subgroup of G_1 and $a \in G_1$ and $j \in J \Rightarrow aja^{-1} \in G_1$.
But the mathematician does not re-represent in his or her head the definition in this lower level. Rather the representation is at the original level, and can even be utilised at this level without further unpacking, for instance the lemma can be used in a step such as:
ϕ is a homomorphism of G onto H with kernel K
=> K is a normal subgroup of G
simply by pattern matching.

Rather, we represent the proposition in terms of its component concepts, namely "homomorphism", "kernel" and "normal-subgroup". If the student on encountering this lemma was very familiar with normal subgroups then this should be easy to encode, and the student should already have developed features for this purpose.

In the CMS, features are generated dynamically from the environment using built in feature generating mechanisms. There are no built in features (see Furse and Nicolson 1992, Furse 1993b). The program MU utilises a dynamic parser which converts the FEL propositions into a parse tree which is then fed to the feature generating processes. Thus, e.g., the above proposition is represented as the tree shown in Fig. 3.

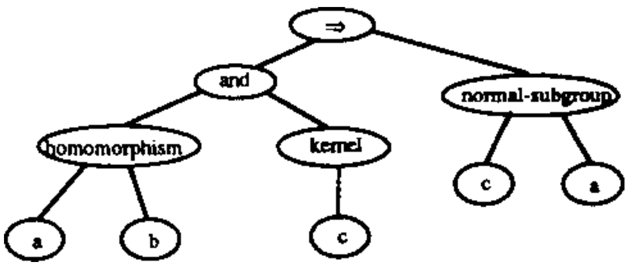


Figure 3, Extracting features from a tree

Here, the leaves of the tree have been replaced by a canonical representation using the letters a,b,c,... By using mechanisms which construct different parts of this tree a very large number of features can be built. In the following we use LHS to represent the left hand side of the tree, RHS the right hand side, and RHS-LHS to represent the RHS of the LHS. Thus in this case the RHS-LHS is the proposition kernel(c). It is also possible to abstract a node, so for example if we abstract the whole of the left hand side we obtain the proposition a => normal-subgroup(b,c), where again the letters have been replaced by their canonical form. Using such methods one obtains features such as:
has-form-[homomorphism_a_b]
has-form-[kernel_a 1
rhs-has-form-[normal-subgroup_a_b]
rhs-lhs-is-form-[kernel_al
is-form-[=>Jand_a_b]_[normal-subgroup_c_d])
is-form-[=>_[and_[homomorphism_a_bLc]L[normal-subgroup_d__a]]
The last feature captures the notion that the student can remember that the lemma was something about being a

homomorphism implying that there was a normal-subgroup. Thus these features enable a rich representation of partial knowledge. With sufficient features the original propostion can be reconstructed, and given the development of sufficiently specialised features, it can be represented exactly and compactly.

4. Learning and Experience

Not only does the expert mathematician learn a large body of mathematical results, but also a large number of features of these results. This then ensures that results are easily recongised and retrieved. Given a mathematical step such as:

$\log[(x - y)(x + y)] + 2\log(y) = 2\log(x)$

the mathematician has little difficulty in noticing that the result:

$(x - y)(x + y) = x^2 - y^2$

has been used, or at least in checking the forward reasoning on applying this inference rule to produce the intermediate step:

$\log(x^2) - \log(y^2) + 2\log(y)$

Given that the mathematician has hundreds or even thousands of results that might be relevant at any particular step, it is an important computational problem how the appropriate result is retrieved without difficulty. We argue that it is patterns or features that the mathematician learns to recognise. Thus, in the above example, the mathematician has no difficulty in seeing the pattern (x - y)(x + y) as being the left hand side of a well known result.

If this result has been used sufficiently often then a specialised feature such as
has-form-[*_[-_a_b][+_a_b]]
may have been stored with which the result is indexed and retrieved. It remains to explain how items are first stored in memory in terms of features, and how these features change through experience.

As explained in the previous section, when a result is first processed by the CMS it is broken up into a large number of features using knowledge free methods. Some of these features may already have been stored, others will be completely novel. The CMS stores a mixture of old and new features, where the old ones are selected from the ones with highest energy, all features being given an energy value which is adjusted with utilisation. If only old features were used, then we would soon be in a closed box representation, but at the time of storage one does not know which of the new features may be useful. For example, consider the definition of a normal subgroup:
Definition. N isa normal-subgroup of G
iff N isa subgroup of G and $\forall g \in G, \forall n \in N\, gng^{-1} \in N$
Here, on initial encoding features that might be used could include:
has-form-[*_a_b]
has-form-[subgroup_a_b]
but the crucial feature is:
has-form-[*_a_[*_bJinv_a]]]

but it is only through experience that the mathematician learns of the importance of recognising the feature gng-[1].

Within the CMS, when a result is retrieved (for example in proof checking or problem solving), the features are adjusted to ensure that retrieval is more efficient in future. Recall involves first computing the features of the probe. For example, in trying to prove that:
log[(x - y)(x + y)l + 21og(y) = 21og(x)
the system first searches for a whole matching result before just trying to reason from the left hand side. In reasoning from the left hand side, the LHS acts as the probe for the CMS, generating features such as:
lhs-has-form-[log_a]
has-form-[_*_ a_b]
lhs-has-form-|-_a_bl
lhs-lhs~has-form-[*_a_b]
All these features must be existing ones, for clearly it is pointless to use a novel feature as an index to old knowledge. The feature generating mechanisms generate many features, and a subset is chosen of those of highest energy.

In the second stage of recall this set of features is used to index a number of mathematical results. Results which do not match the probe are considered failures. Ideally the features should only index relevant results, which can be applied, but this only comes through experience as the feature space changes. The matching results are then applied and the outcome which has the highest plausibility in terms of being nearer to the goal and a simpler expression is chosen.

In the third stage of recall learning takes place to ensure that the found item is more easy to retrieve in future. This means ensuring that the same set of input features should be more likely to retrieve the found item than the failures. This is achieved by four processes:

- storing the uncomputed features
- encouraging the useful features
- discouraging the unuseful features
- creating new distinguishing features

Most of the features used in the probe will already index the found item, but some of them may have arisen from storing other items, and may not be connected to the found item. These are known as uncomputed features, and provided the found item possesses the feature, they are now stored. By this means new connnections between features and items are regularly being made during retrieval.

Useful features are features which index the found item but not the failures. These features are encouraged by increasing both their energy and the link strength between feature and found item. Conversely, unuseful features index the failures but not the found item, and have their energy decreased.

The final method of adjusting the CMS involves the dynamic creation of novel features which index the found item but not the failures, and do not already exist. This is achieved by means of comparing the large feature sets of the found item and the failures generated

by bottom-up methods. By such a mechanism, specialised features such as
has-form-[*_[-_a_b][+_a_b]] can be generated.

By these means the CMS ensures that results which are important have a number of high energy specialised features so that such results are recognised easily. Thus as a result is used through the stages (l)-(3) described above, so its representation changes from initially being encoded only partially in terms of its features, and likely to be retrieved with other similar results, to ultimately being a result with specialised features easy to recall and use. See Furse (1992) for more details of the CMS.

5. The Mathematics Understander

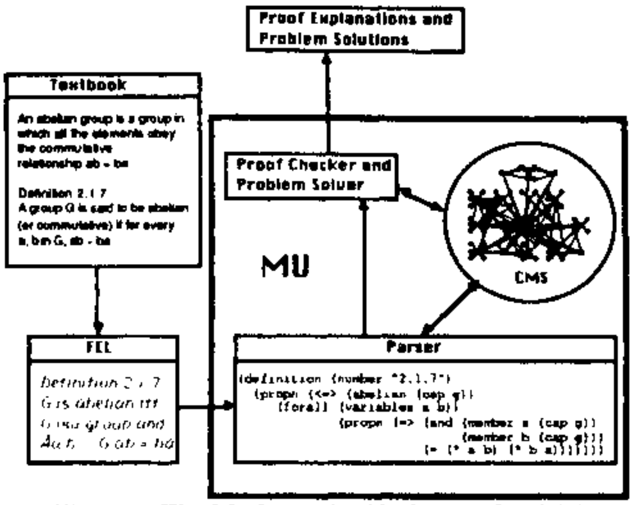
The Mathematics Understander (MU), (Furse 1993a), is a computer program which utilises the CMS to "understand" mathematics texts expressed in FEL. The original text is translated by hand into FEL, which is then input to the program. MU parses the input using a parser whose syntactic knowledge is built up dynamically from definitions and stored in the CMS. The parsed input is f*ed* to the proof checker and problem solver which utilises the acquired knowledge in the CMS to produce explanations of the steps and solutions to the problems (see Fig. 4).

MU thus has parsing, proof checking and problem solving knowledge built in, but no knowledge of mathematical results. All the mathematical results arc acquired through the reading of texts. Fig. 2 shows an example of a problem solution generated by MU. The built in heuristics used in problem solving are:

• expand a definition
• break a compound step into parts
• suppose the left hand side
• simplify

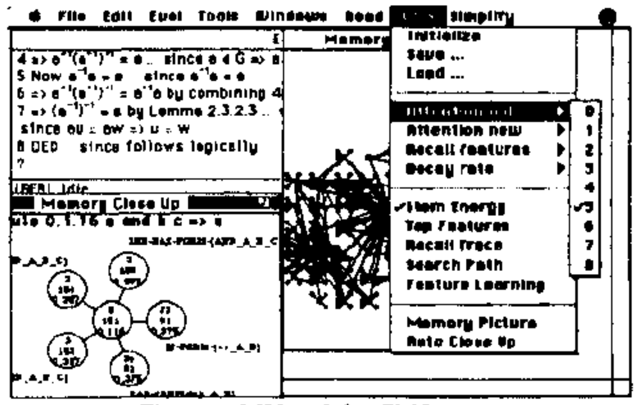
Simplification involves the application of known results using the CMS to only retrieve relevant results rather than all possibly applicable results. The results retrieved are applied and their plausibility measured in terms of nearness to the goal and simplicity. The result which achieves the highest plausibility is the one chosen, and if necessary, MU will backrack, but this is very rarely found to be necessary.

MU is implemented in Macintosh Common LISP and runs on Apple Macintosh computers. The CMS can be displayed in a graphical format, and navigated through. It is also possible to set parameters for individual modelling. See Fig. 5. MU has successfully read texts in group theory and classical analysis, and solved simple problems in group theory.


Proof Explanations and Problem Solutions
Textbook
An abelian group is a group in which all the elements obey the commutative relationship ab = ba
Definition 2.1.7
A group G is said to be abelian (or commutative) if for every a, b in G, ab = ba
Proof Checker and Problem Solver
MU
CMS
FEL
Definition 2.1.7
G is abelian iff
G is a group and
Aa,b ∈ G ab = ba
Parser

```
(definition (number "2.1.7")
  (propn (<=> (abelian (cap g))
     (forall (variables a b))
             (propn (=> (and (member a (cap g))
                             (member b (cap g)))
                        (= (* a b) (* b a))))))
```

Figure 4. The Mathematics Understander (MU)



```
4 => e⁻¹(e⁻¹)⁻¹ = e .. since e ∈ G => e
5 Now e⁻¹e = e     since e⁻¹e = e
6 => e⁻¹(e⁻¹)⁻¹ = e⁻¹e by combining 4
7 => (e⁻¹)⁻¹ = e by Lemma 2.3.2.3 ..
 since eu = ew => u = w
8 QED    since follows logically
?
```

Figure 5, MU and the CMS menu

Education in pure mathematics is often a process of the student understanding the proofs of theorems and solving the problems. This experience enables the student to develop appropriate features so that relevant results can easily be retrieved when necessary. The lazy student who skips the proofs and the simple problems runs the risk of not being able to solve the harder problems. If MU "reads" only the definitions and theorems, and skips the proofs and simple problems, it too cannot retrieve the results needed when trying to solve the problem in Figure 2 and gives up.

6. Discussion

In some sense an expert "sees" that in solving a problem a particular step should be taken. It seems implausible that a large scale search takes place of knowledge which might be useful. Rather the knowledge required almost springs out from the problem. It is this notion which places perception and learning centre stage in cogntion which this research attempts to address. We have shown how in the domain of pure mathematics expertise can be captured with a large body of knowledge indexed by features. Further the CMS allows the modelling of how these features are learned through experience.

Naturally the CMS uses several ideas which are well-known in the literature. For instance, the feature recognition mechanism has similarities with EPAM (Feigenbaum and Simon, 1986). The energy mechanism resembles the activation used by Anderson in ACT-R and ACT* (1983), though the there is no spreading activation in the CMS. We believe, however, that the CMS is distinctive in combining all these ideas within an integrated environment.

The CMS can, in principle, be applied to other domains than mathematics, since there is no built in mathematics in its construction. The major requirement is the development of appropriate feature generating mechanisms. Work is in hand in applying the CMS to the learning of board games.

Research in neural networks has brought learning back to the centre of work in artificial intelligence, and there is a clear emphasis in their application to perceptual classification problems. However, to our knowledge there has not been much success in applying such architectures to problem solving in domains as complex as pure mathematics. Furthermore, it could be argued that a neural network is restricted in its internal representations by the features used as the input to the network. In contrast, the CMS generates its features dynamically from the environment.

The other main competitors to the CMS model are Anderson's ACT* model and Laird, Rosenbloom and Newell's SOAR. Whilst both architectures model problem solving, and in particular model goal based reasoning, which the CMS does not model, the focus is quite different. Both models are essentially based on a production systems architecture, and this ultimately limits their scope for adaptation. Although both ACT* and SOAR model learning, they do not appear to model declarative learning which the CMS is designed for. Anderson has concentrated on modelling how knowledge becomes proceduralised, but does not address the question of how the initial knowledge is learned in the first place. Newell defined learning as search within a problem space, but as both Norman (1991), and Boden (1988), have remarked, this seems an impoverished view of learning. Pure mathematics cannot be represented as a problem space since new constructs are continually being introduced in a relatively ad hoc manner. SOAR appears to want the domain to be pre-characterised as a problem space in advance, before learning can take place. A similar problem occurs with explanation based generalisation whereby a domain theory is required in advance. However, human learning is much more piecemeal, and teachers almost never map out in advance a characterisation of the domain to be learned.

In conclusion, perception and experience play an essential part in problem solving. It is not sufficient to be an expert to know a lot of facts. The expert also has to be able to recognise when they are relevant and be able to retrieve them.

7. References

Anderson J.R. (1983), The Architecture of Cognition, Harvard University Press.

Boden, M. (1988) Computer models of mind. Cambridge: Cambridge University Press.

Bundy A, (1983), The Computer Modelling of Mathematical Reasoning, Academic Press.

Chase W.G. & Simon H.A. (1973) Perception in Chess, Cognitive Psychology A pp55-81.

DeGroot A.D. (1966) Perception and Memory versus thought: Some old ideas and recent findings, In B.Kleinmuntz (ed) Problem Solving: Research, Method and Theory, Wiley.

Feigenbaum, E.A. and Simon, H.A. (1986), EPAM-like models of recognition and learning. Cognitive Science, 8: 305-336.

Furse E., (1990), A Formal Expression Lanuage for Pure Mathematics, Technical Report CS-90-2, Department of Computer Studies, The University of Glamorgan.

Furse E. (1992) The Contextual Memory System: A Cognitive Architecture for Learning Without Prior Knowledge, In Cognitive Systems 3-3, September 1992 pp305-330.

Furse E. and Nicolson R.I. (1992) Declarative Learning: Cognition without Primitives, Proceedings of the 14th Annual Conference of the Cognitive Science Society, pp. 832-7.

Furse E, (1993a) The Mathematics Understander. In 'Artificial Intelligence in Mathematics', (eds) J H Johnson, S. McKee, A. Vella, Clarendon Press, Oxford (in press).

Furse E, (1993b), Escaping from the Box. In Prospects for Intelligence: Proceedings of AISB93, (Eds) Aaron Sloman, David Hogg, Glynn Humphreys, Allan Ramsey, Derek Partridge, IOS Press, Amsterdam.

Laird J.E., Newell A. & Rosenbloom P.S.. (1987) SOAR : An Architecture for General Intelligence, In Artificial Intelligence 33 No.I, Elsevier Science Publishers.

Laird J.E., Rosenbloom P.S. & Newell A. (1986) Chunking in SOAR the anatomy of a General Learning Mechanism, In Machine Learning 1 ppll-46.

Lenat, D.B. and Feigenbaum, E..A. (1991). On the thresholds of knowledge. Artificial Intelligence, 47: 185-250.

Norman D.A, (1991), Approaches to the Study of intelligence, Artificial Intelligence 47 (1991), 327-346.

Schank R.C. and Riesbeck C.K., (1981), Inside Computer Understanding, Lawrence Erlbaum 1981.

Turing A.M. (1953), Digital Computers applied to games. In B.V. Bowden (Ed.), Faster than Thought, London. Pitman, 286-310.