

# Myths for brain modeling

Alan H. Bond

*Center for Cognitive Phenomics, Neuropsychiatric Institute  
University of California at Los Angeles, Los Angeles, California 90095  
email: alan.bond@exso.com*

## Abstract

We give a brief discussion of our underlying assumptions or myths, in the form of a criticism of some widely held false beliefs, namely:

- (i) we need a continuous, not a discrete, concept of computation to describe the brain,
- (ii) there is a class of mechanisms called rule systems which have been tried and found not to work,
- (iii) computer science concepts are not applicable to the brain,
- (iv) systems involving rules describe only symbolic mechanisms, and
- (v) AI failed.

## 1 Introduction

We take myths to be the underlying imponderable, or very difficult to ponder, basic assumptions that we use. They are necessary to allow us to build rafts over the swamp.

A lot can be, and has been, said about these issues. I will give only a brief outline of my own thinking here, but I feel it necessary to say something <sup>1</sup>. These false ideas have been routinely evoked by most people evaluating my research in brain modeling, and are misleading.

Who holds these beliefs? Not computer scientists, but most other people interested in the brain, neuroscientists, psychologists, connectionists, physicists, and others.

## 2 Discrete and continuous processes

Computer science deals with discrete systems of description and computation, so everything is describable with discrete expressions and proceeds by discrete operations on

---

<sup>1</sup>I have a draft book manuscript which will include a more extensive treatment of these issues

these expressions. It can be argued that we need continuous methods to describe the brain, for example the biologist Walter Freeman has made such arguments, basically that biological systems require an unbounded degree of precision and so cannot be discretized. I don't know how one makes a theory of computation based on functions of real variables. It seems to be the same as calculus, but this would get into problems of requiring unbounded accuracy in any computational device. In addition, it will presumably have to deal with underlying logical issues such as self-reference, logical paradoxes, problems of describing sets, and so on.

The brain has a lot of discrete structure, but, on the other hand, neurons are so numerous that it seems reasonable to treat a large neural net as a continuous field of some kind. The struggle to do this has a venerable history starting at least with Rosenblatt's "Neurodynamics" in 1961 and John Griffiths "A view of the brain" in 1965, and it continues today in the vast collection of theoretical papers on neural net theory, adaptive resonance theory of Stephen Grossberg, the Santa Fe Institute's project in complex systems, and so on. I am hoping that all this work will produce something useful.

It is possible to show that a continuous system can represent a Turing machine, however this does not prove that we need it.

When we examine the description of neural nets, we find that we need a vast discrete description of all its geometry and connectivity, as well as boundary conditions and conditions on solutions, so the calculus proceeds within this large descriptive system, the properties of which are not usually examined, but which correspond to the logical structure of the system.

### **3 Computer science and the concept of rule**

Theoretical computer science involves not so much the study of computation as study of the description of computation. It concerns precise, or, if you will, mathematical, languages for describing computer systems of various kinds, and techniques for describing different aspects of these systems and their behavior, performance and properties.

Different ways of describing computation have been developed. Basically, I think of them as falling into three classes:

- (i) State machines including automata, possibly with registers, stacks or tapes, in which

the main organizing principle is that the system is at each moment in a particular named state and where we describe its action by giving an explicit state transition table.

(ii) Functional calculi, including the lambda calculus and formal functional languages. This is the mainstream concept for describing present-day programming languages. Everything is a function, and functions can be applied to functions to produce other functions as values. The main idea is thus application which in most systems becomes the idea of nested evaluation of nested functional expressions. Recursion is a basic descriptive device.

(iii) Logical calculi, including logic programming. Computation concerns logical formulae and the process of inferring formulae from others. It was shown by Goedel that all computation can be described by such logical systems. There are a range of languages, including  $\omega$ -ordered and modal logics, but the mainstream is first-order predicate logic. This approach became highly developed with the invention of Prolog and the theory of logic programming. Here expressions are matched literally, they are not “evaluated”.

Incidentally, an expression is a string of characters, and logical expressions are a particular type of character string.

Each of these classes of description of computation can be put into a rule form. The original rule systems, due to Post, Thue and Markov, involved string matching, so a rule has a left hand side which is a string of characters and it is matched to a working string and then modifies the working string locally. These are very close to automata since each rule is like an entry in a state transition table.

If you introduce variables into rules, so that elements match to bracketed expressions, then you get a system that is close to logic. The variables are essentially universally quantified over. If you have function letters as well, you get the more powerful first order logic which has different properties, and the matching process is then usually taken to be unification. Such systems can also involve self-reference or recursion in that a term can appear on the right as well as the left of the rule. These systems are not close to automata at all.

In the logic programming approach, computation proceeds by a sequence of inference operations, but it is equivalent to the explicit computation of a model of the logic, in fact for predicate logic it constructs the minimal model, that is, the model with the fewest assumptions.

Logic programming also introduced an active procedural view of logical expressions, so that they are not only statements held to be true but also there is a natural control regime which uses these statements to infer their consequences.

It is also possible to represent functional programming by rule systems called term rewriting systems in which the matching elements are bracketed expressions, i.e., terms. Such systems have some advantages over the usual functional evaluation methods, in allowing more control over the evaluation process. Incidentally, it is also possible to represent nested functional evaluation as an automaton with stacks.

Then there is the issue of top-down rules, which leads to another whole class of system. Classically, AI used top-down rules and developed goal trees, etc. In the case of logic programming, it was shown by Maarten van Emden and Robert Kowalski in 1976 that top-down and bottom-up execution of logical rules lead to the same result, the same model.

There is also the issue of uncertainty and various ways to associate strengths with logical expressions. Logic programming has been extended to the fuzzy case by Maarten van Emden. I use a set of strengths associated with each logical statement.

Logic programming also includes real arithmetic, so one can use, and I do, arithmetic expressions as part of logical statements, for example in computing geometric measures in percepts and motor outputs. So, continuous measures are included in this descriptive approach.

Finally, logic programming has been extended to provide fine-grained parallel control, where every atomic statement on the right-hand side of a logical rule is a separate parallel process. This is not so useful for us, however there is other research on larger grained parallelism in logic programming which may be useful in brain modeling. All this work has been done at Imperial College, London.

So, to generalize about rule systems is to generalize about all computational descriptive approaches.

## 4 Computer science concepts

I am mainly interested in developing a computer science approach to the brain, but by this I do not mean so much these theoretical models, which really form the theoretical

basis of my approach, but more general concepts that are used in practical computer system design. These include:

- (i) the concept of data structure, the idea of structures or packages of information containing many components,
- (ii) the concept of process, an entity which develops in time, may change state, may store data, and has various types of interaction with other processes, and
- (iii) parallelism and control structures, processes may proceed in parallel but coordinate their activities to achieve desired system effects.

In addition, however, and most importantly, (iv) there is the underlying approach of using precise description languages, that the language used is an important aspect of the research, we should be conscious of what language we are using, its syntax and semantics and other properties.

The description language becomes the language in which we do natural science.

Also (v) the description of any system, in particular the brain, should be in levels of abstraction. So we have different descriptive languages at each level. This means that each level only describes certain concepts and not others, and only makes postulates, only delivers predictions, and can only use experimental results, in terms of the concepts in its own level. Descriptive levels are used extensively in describing computer systems, as a necessary way of controlling the complexity involved. Each level is its own “reality”.

Descriptive levels are not used in natural sciences such as physics, it doesn't seem to be necessary as most physical phenomena can be well described using real, complex or vector fields of various kinds. One can find some half examples, such as the description of gases with the kinetic theory of gases at one level and Boyle-Charles's law at another, or the theory of solids with atomic level descriptions, then crystal structure and cooperative phenomena at another, and then finally the gross properties of materials and the mechanics of elastic bodies. But in general there has been no need for consciousness of the language of scientific description, calculus has been very successful.

However I believe we will not be able to describe the brain using these methods of physics.

Allen Newell led the introduction of rule system ideas into AI in 1967. He had been working with GPS for ten years which had a powerful recursive and backtracking control structure, and he wanted to get back to basics in order to conceptually break out from this method. So he started looking at Markov algorithms, with very simple matching elements which were associations between atomic features and atomic values. He also

wanted to use the working string to correspond to short term memory in psychological modeling. This program of research lasted until about 1980, after which he returned to the issues from GPS and synthesized the SOAR system.

John Anderson developed his ACT rule system approach in 1976 or so and it is similar to Newell's ideas at that time, and also to Forgy's OPS-5 system being developed at the same time as an efficiently engineered and very fast expert system language. Anderson's basic idea has not changed substantially over the years, but he has done a very large number of models which fit data well. He does admit that the logical content of his rules has very little effect, and that the main expressive and modeling power comes from arithmetic expressions evaluated by the rules, which he calls the subsymbolic level.

Finally, for interest, I believe we can express more neurophysiological ideas in logic programming, but I have yet to develop this. As a start, here is one formulation of the McCulloch-Pitts neuron:

```
neuron(neuron1).
neuron(neuron2).
neuron(neuron3).
connects(neuron1,neuron3).
connects(neuron2,neuron3).

firing_rate(Neuron,Rate):-
    setof(Contribution(Neuroni,Ratei),
        (connects(Neuroni,Neuron),firing_rate(Neuroni,Ratei),Rates),
        sum_rates(Rates,Sum),
        threshold(Neuron,Theshold),
        Sum > Threshold,
        Rate is Sum - Threshold.
```

## 5 Symbols

It is often said that rule systems describe symbolic processing, and that since the brain doesn't use symbols for a lot of its activity, therefore rule systems cannot be successful. This is actually a little bit difficult to grapple with but here are my thoughts.

First, the fact that a theory uses symbols to describe something does not mean that the

models of that theory use symbols. Every theory uses symbols, this is presumably so that humans can communicate it to each other. Even neural nets are written as functional expressions.

To me the clearest statement of what a symbol is comes from Allen Newell. He was steadfast in his ideas from 1956 until 1990. There are three aspects: a symbol is an arbitrary token, symbols can form structures such as lists, and a symbol can have a value which is a structure.

If we write a logical theory of a system then are we automatically assuming that this system uses symbols? Patently not. For example, I can describe a group with the axioms of group theory, where are the symbols, the structures, and the value relation?

However of course we can include in our theory some axioms which assert Newell's idea of symbols in which case we are making a scientific postulate that there are symbols in the system in some role determined by the rest of the axioms. This is as it should be, and is a consequence of the descriptive approach; things only exist in the model if they are postulated to do so, and are explicitly described.

But this brings us to what we are describing in the first place, presumably processes and structures, so having symbols means that there are processes for constructing symbols and structures from them, and for assigning values and for determining and using such values. All of this would have to be described before a system can be called symbolic.

My own alternative idea, to symbol, is that the brain can be described, at some level of description, by chunks of information, and that these chunks contain several different kinds of information, are stored and processed, i.e., used as inputs and computed as outputs by processes.

To me, the components of chunks are not necessarily tokens, but I think there will be arbitrary tokens resulting from generalization and learning. I think it was Emerson who said that every word was once a poem. Events with explicit descriptions may become generalized and used in other descriptions, and eventually become arbitrary.

Chunks are data structures and they can be described by logical terms, i.e., bracketed expressions. Actually, in the future we may extend our notion of structure beyond bracketed expressions, in particular to two dimensional structures like images; after all, the cortex is essentially two-dimensional and sets of neurons connect one two-dimensional surface to another.

It is I think helpful to regard chunks as neural codes, so my postulate is that, at some level of description, there are discrete neural codes; neural areas can store them, can generate new codes and can transmit these codes to other areas.

To what extent data types need to be strictly and explicitly separated I am not sure, it may also not be necessary to make explicit postulates about this as it will simply be a property of the processing within neural areas.

I said there are nevertheless difficulties with the notion of symbol. One problem is that every consistent first-order logic has a denumerable model, this is the Lowenheim-Skolem theorem, and in fact it usually has a specific kind of denumerable model called a Herbrand model. The elements of this model are discrete bracketed expressions, so even if the intended model of our logical description is continuous, and there is such a continuous, and non-denumerable, model of our logic, there is always also a discrete model.

Another possible problem is the issue of describing real numbers using predicate logic. It has actually been shown that this can be done, by Abraham Robinson in 1963. However in order to do this he had to use a non-denumerable set of constants and a special form of model theory which allowed him to model the basic property of the set of reals, which is that it contains all the limits of all denumerably infinite sequences of reals. It is this property that allows calculus to be developed and accounts for all of its powerful properties. Robinson had to axiomatize the infinitesimals and the infinite numbers and include them with the reals, and having done that he was able to logically derive all the results of calculus, by using infinitesimals instead of limit arguments. So we can incorporate continuous ideas into discrete logic descriptive methods, but at some technical cost!

## 6 AI failed

I have never identified with AI actually, I have always said I am interested in the human condition and the human brain, and I am not particularly interested in other kinds of “intelligent” system or mechanism. I taught AI for 15 years at London University, then at USC, and even at Caltech, and although I have found some AI ideas to be useful, most AI mechanisms are not useful, since I need everything to work within the constraints and style of processing in the brain.

I don't see that AI failed at all. It has produced many important and seminal ideas which have propagated into computer science, psychology and linguistics. This was achieved by a very small number of people incidentally, probably a few thousand only.

The funding to AI has been greatly reduced, and most of its classical subareas have been fractionated off and become more conservative engineering subjects, so general AI has been much reduced by the allocation of funding and by the need for clear evaluations for journal refereeing and for tenure. Nevertheless there is still some activity, particularly in multiagent systems which is a very fertile and creative area concerning cooperation and social groups of agents.

In the late seventies, the AI community became controlled by a clique of people from MIT, Stanford and CMU and this caused problems for others trying to do AI. However the main problem was that AI started to suffer from lack of funding after the end of the Vietnamese war and the oil crisis of 73. They tried to legitimize AI by getting Lisp accepted as a standard language for military contracts, and they tried to push simple logical rule systems as practical for engineering purposes. These "expert" systems caught on as they had some value, but of course their value was limited and applications hit against various research problems that had not been solved. During this time, again to justify funding, different aspects of AI started to be separated out. For example, the funding of computer vision was polarized into 5 centers of excellence by DARPA and made to work on satellite imagery recognition; this was reported in the Image Understanding workshops. These groups were funded in the millions of dollars per year and overshadowed other more general work in computer vision. In the UK, incidentally, things were much worse, the British government commissioned a report on AI by a control theorist, James Lighthill, and he reported in 1974 that it was confused nonsense and should not be funded. So the AI funding committee of the SRC was disbanded in 1976, and almost all AI researchers left the UK and went to the US in the late 70's. There remained me in London and Robin Poppleston in Edinburgh, but we emigrated also in the early 80s.

This is part of another dimension to this historical question. There was a conflict between conventional engineering and control theory on the one hand and computer science and AI on the other. The dichotomy is that calculus has limited applicability but can deliver strong results, whereas computational models are of much more universal applicability but cannot provide analyses of the same detail and precision as calculus. This plays into the natural tendency of bureaucracies toward conservatism, to emphasize what can be described accurately, rather than what we actually need to describe. AI has been

hounded out of essentially all computer science departments, and survives only in AI departments. Computer science was also hounded out of engineering departments, it has actually been so successful that it is much hated by engineers. Computer science today attracts the brightest students.

My own idea is that AI is really just the creative aspects of computer science and engineering, and most people have limited need for creativity, they have “real” problems to attend to, and they are in a hurry. AI is like a child which asks embarrassing questions and comes up with unusual solutions. The child expresses its ignorance and tells the truth, the truth we cannot handle. Our reaction to it is to want to control it, to limit the embarrassment, but it is however the key to your future.

After all, everybody is excited by the idea of understanding how the brain works.