

# Occam's Razor Part 9: Representation and Planning of Actions in Artificial Intelligence

By Paul Almond, 29 July 2006

**Website:** [www.paul-almond.com](http://www.paul-almond.com)  
**Email:** [info@paul-almond.com](mailto:info@paul-almond.com)

© Copyright Paul Almond, 2006. All Rights Reserved.

# Occam's Razor Part 9: Representation and Planning of Actions in Artificial Intelligence

By Paul Almond, 29 July 2006

## Previous Articles in this Series

This article is part of a series. These are the previous articles:

*Occam's Razor Part 1: What Is Occam's Razor?* <http://www.paul-almond.com/OccamsRazorPart01.htm>. [1]

*Occam's Razor Part 2: Principles of Language* <http://www.paul-almond.com/OccamsRazorPart02.htm>. [2]

*Occam's Razor Part 3: Assumptions About Reality* <http://www.paul-almond.com/OccamsRazorPart03.htm>. [3]

*Occam's Razor Part 4: An Overview of How Occam's Razor Works* <http://www.paul-almond.com/OccamsRazorPart04.htm>. [4]

*Occam's Razor Part 5: How Mapping Can Work* <http://www.paul-almond.com/OccamsRazorPart05.htm>. [5]

*Occam's Razor Part 6: Partial Models as Envelopes* <http://www.paul-almond.com/OccamsRazorPart06.htm>. [6]

*Occam's Razor Part 7: Hierarchy and Ontology* <http://www.paul-almond.com/OccamsRazorPart07.htm>. [7]

*Occam's Razor Part 8: Modelling in Artificial Intelligence* <http://www.paul-almond.com/OccamsRazorPart08.htm>. [8]

In particular, the last three of these, parts 6, 7 and 8, may be helpful in understanding this article: it is in these articles that the artificial intelligence approach being discussed really starts to be developed.

The following article, although not part of this series, is closely related to it:

*What is a Low Level Language?* <http://www.paul-almond.com/WhatIsALowLevelLanguage.htm>. [9]

## Introduction

This series of articles is becoming increasingly inaccurately named. We have gone from thinking about Occam's razor to applying it in artificially intelligent computers. This has

led us into representation of models of reality in computers and the series has now become an attempt to design a general purpose, artificially intelligent computer system. We will now consider the representation and planning of actions in an artificially intelligent system, which may appear to be going still further away from the original subject of Occam's razor. The name of the series may as well stay the same however. There is a sense in which it is still appropriate: automatically generated descriptions of reality are central to the artificial intelligence system that I am describing and Occam's razor is central to making these.

The previous article *Occam's Razor Part 8: Modelling in Artificial Intelligence* [8] described how reality can be represented by a probabilistically expressed hierarchy using meaning extraction algorithms - partial model algorithms. Artificially intelligent systems must not merely observe reality and infer things from it, however: they must *do* things. This article will describe how the hierarchical ontology in previous articles can be used to plan actions. For an artificially intelligent system to plan actions we first need a way of representing its actions, so representation of actions in the hierarchy will be discussed first. We will also discuss how the desirability of outcomes can be determined, as this plays an important role in planning. We will then consider how the hierarchical modelling system, now with provision for the description of actions, can actually be used to plan actions.

This work builds on the model representation and inference system described in the previous article. Readers may wish to compare this with the hierarchical artificial system proposed by Jeff Hawkins [10,11]. There are some differences. I think that the system I am proposing deals with probability in a much more robust way and, as actions are inherently probabilistic (which will be discussed in this article), it will be possible to describe planning of actions more coherently and also deal in a later article with the "carpet texture" problem. If the system described in this article were mapped into a neural system there would not be columns of "closely-coupled" sensory and motor cells, as is the case with the system proposed by Hawkins. In this system, above the bottom level at which the inputs and outputs actually occur, there is no distinction between inputs and outputs.

## **Representation of Actions**

### **Combination of Actions and Inputs**

There is no point in trying to separate the representation of reality from the representation of the system's actions: the system acts in reality and its actions are part of reality. If the system has any reasonable scope to change reality then its own actions will dictate much of its experience of reality. This makes a description of the system's own behaviour unavoidable when we are describing reality. Even if we purposefully ignore the system's behaviour we are still describing other features of reality that are contingent on it. As there is no avoiding this, and as the behaviour of the system is important in planning its own actions, we may as well explicitly incorporate a description of the system's own behaviour into the hierarchical model.

Any description of reality must therefore incorporate a description of the system's own actions. The hierarchical way of representing reality previously proposed needs to be expanded to incorporate representation of the system's actions. This is easy to do and will now be explained.

### **Incorporating Actions into the Hierarchy**

The bottom level of the hierarchy consists of matrices of sensory input bits represented by probabilities. As well as sensory input bits we will now have matrices of *motor output bits*, represented by probabilities, at the bottom level.

The meaning of the probability values in the output matrices is the same as in the matrices of sensory inputs or extracted meanings: each value represents the probability that the relevant bit is “1” and probability values of 0 or 1 correspond to known (I will later be describing these as “fixed”) outputs that have actually been made in the past.

Some bits in the bottom level output matrices correspond to outputs that have not yet been made. These have unknown values, but in keeping with the idea of not qualitatively distinguishing between known and unknown bit values, these bits are assigned probability values of 0.5, although these values can be changed later.

This all means that the bottom level of the hierarchy consists of two kinds of matrices - sensory input matrices and motor output matrices - both containing probability values for the relevant bits – 0 or 1 for inputs or outputs that are absolutely known, 0.5 for inputs or outputs about which nothing is known and other values between 0 and 1 for bits about which there are varying degrees of knowledge.

### **Meaning Extraction and Inference in a Hierarchy Containing Actions**

Partial model algorithms were previously proposed to extract matrices of meanings from matrices of sensory input bits, or from matrices of meanings already derived from such matrices. We now have these extra motor output matrices. How do we deal with them? How does the process of meaning extraction and inference by partial models work with motor output matrices at the bottom level of the hierarchy?

We do not do anything special about them. We treat them in exactly the same way as we have been treating the sensory input matrices: we allow partial model algorithms to extract meaning from the probability values representing bits in the motor output matrices in the same way that they can extract meaning from the sensory input matrices. Meaning extraction from the bottom level of the hierarchy can now involve a partial model algorithm extracting meaning from one or more sensory input matrices, from one or more motor output matrices, or from a number of sensory input and motor output matrices producing a meaning relating both inputs and outputs to each other. Higher level partial model algorithms can extract more meaning from the matrices of meaning produced in this way. As we ascend into higher levels of the hierarchy the distinction between sensory

inputs and motor outputs disappears: we just have matrices of extracted meanings, many of which will have been extracted from both inputs and outputs in various ways.

To summarize: meaning extraction by partial models makes no distinction between sensory input and motor output matrices. With regard to meaning extraction we treat output bits as if they are part of the collection of input bits.

No aspect of the meaning extraction process or the way in which partial models work needs to be altered to allow all this to happen: we just need to represent the bits in the motor output matrices by probabilities and allow them to be processed by partial model algorithms along with the sensory input matrices.

### **Downward Transfer of Probabilities and Actions**

A process that I called *downward transfer of probabilities* has still to be properly described. It will allow the probability values associated with higher level meanings to influence the probability values associated with the lower level meanings from which they were derived. Downward transfer of probabilities was previously proposed to work on the meanings extracted from sensory inputs. Given the approach of not distinguishing between sensory input matrices and motor output matrices it naturally follows that downward transfer of probabilities (when defined) will make no such distinction either: it will operate on the matrices of probability values without regard for whether they are extracted from sensory inputs or motor outputs. This means that probability values in matrices of motor outputs on the bottom level of the hierarchy will be affected by the process of downward transfer of probabilities.

### **Uncertainty Caused By Outputs**

There will be a lot of uncertainty in the system's future actions; that is to say in bits in the motor output matrices corresponding to outputs that have not yet been made. This uncertainty will be carried upward into meanings extracted from these, so the uncertainty in the system's future actions will inject further, very large amounts of uncertainty into higher levels of the hierarchy.

This is natural. The system's own undecided future actions must have a lot of uncertainty associated with them because by definition the system does not know what they are going to be until it has made the relevant decisions. Actions are probabilistic in nature and this is why they *must* be represented probabilistically. When the system lacks knowledge about its own future actions then it must also have uncertainty about those aspects of reality which are contingent on those actions, making a statistical approach to the whole modelling issue essential.

Of course, we may imagine contrived situations in which the choice of action by the system has little or no affect on reality, but these would be trivial situations in which the system is not able to achieve anything. If it is worth the system performing any actions then they must be having an effect on reality.

This also gives us a basic idea of what the planning of future actions is - the collapsing of the uncertainty that the system has regarding its own future outputs. This means that planning actions is not necessarily an all or nothing process in which particular actions are selected and *it does not simply involve passing actions down the hierarchy*. The planning process may partially collapse some uncertainty regarding future outputs, with no specific action being definitely selected, but with the uncertainty in the system's future courses of action being diminished at the same time.

It may now be clearer why I have been so concerned with probability in the last article. The very idea of what actions are is probabilistic and actions cannot be represented coherently, nor can a way of planning them be defined coherently, with a system that is not fundamentally probabilistic.

### **How Downward Transfer of Probabilities Relates to Actions**

Downward transfer of probabilities will affect probabilities in both input and output matrices in the bottom level of the hierarchy. Some readers may think that downward transfer of probabilities is the transfer of actions down the hierarchy after meaning from sensory inputs has been passed up it and that downward transfer of probabilities is the planning of actions. This is only partly the case. Downward transfer of probabilities is not contingent on any concept of desirability of actions and simply happens in the system, regardless of what its objectives are supposed to be, and its need was determined before we had introduced actions into the system.

Downward transfer of probabilities to output bits is the same thing that it is for sensory inputs – the making of changes in the probability values in low level matrices that follow on from high level probabilities as an expression of the idea of context dependent frequency that was mentioned in the previous article [8]. What does it mean when downward transfer of probabilities alters the probabilities in the bottom level motor output matrices?

When this happens for the sensory input matrices the system is recognising that some low level information is part of a higher level pattern and then using this to reduce uncertainty at a low level by altering probability values. Using the wallpaper analogy from the previous article this is like recognising that a higher level pattern exists on a piece of wallpaper, which runs off the edge of the sheet, and then filling in the blanks at a low level for the parts of the pattern that cannot be seen. For temporal patterns this is equivalent to making high level predictions and then using these to make low level predictions.

With no distinction between sensory input and motor output matrices then this must also follow for the relationship between downward transfer of probabilities and motor output matrices: when downward transfer of probabilities affects probability values in the bottom level motor output matrices low level *predictions* are being made. *The system is trying to predict its own future behaviour.*

The idea of the system trying to predict its own future behaviour may seem strange, but it is natural that this should happen. The system's own behaviour is part of reality and prediction of future observations of reality is inextricably linked to prediction of its own behaviour. Some readers may object to this by saying that there is no reason for the system to try to guess what it is going to do next when it just needs to *decide* what it is going to do, but there is actually a very close link between the system's actions and its predictions of its own future actions: as we will discuss later, they are essentially the same thing. Even if this were not the case, uncertainty about future events in reality, and prediction of them, which includes prediction of its future behaviour, would provide the context in which any actions by the system are made. Until the system has completely decided on particular outputs then its future behaviour with regard to these outputs is something about which it has some uncertainty – the degree of uncertainty depending on the extent to which it has made a decision.

We should actually be worried if any artificially intelligent system did *not* try to predict its own behaviour: this would imply an arbitrary gap in its view of the world.

### **The Desirability of Actions**

For a planning system to work there must be a way of determining or specifying how desirable various outcomes are so that the system can seek to perform actions that cause desirable outcomes.

The desirability of various outcomes is not directly indicated to the system by any complex analysis of the outside environment. Instead an algorithm, the *evaluation function*, returns a desirability score based on the probabilities of the bottom-level input bits. This is a concept well-known in chess programming [12].

One way in which this could be done is as follows:

Each bit in the sensory input matrices is assigned two scores – one score for the bit value being “0” and another score for it being “1”. A score of 0 is neutral, indicating that no advantage or disadvantage of the input is known. A positive score indicates that the input is desirable and a negative score indicates undesirability. Positive and negative scores are analogous to pleasant and unpleasant stimuli with humans.

This allows the desirability of any situation to be assessed. The evaluation function can look at what sensory inputs have been received recently and add up the scores. As an example, if a particular bit in a sensory input matrix has recently been assigned the value “1” (or more accurately, if it has been assigned the probability of 1) then we would add the score for this particular bit being “1” to the total score.

Consideration could be given to the expected desirability of inputs that will be received in the future by using the probabilities associated with future sensory input bits. The expected desirability in the future is therefore affected by downward transfer of probabilities.

We have some choice about which sensory input bits to consider and how we combine the scores. We should probably consider the expected scores that we expect to be obtained from sensory inputs in the future - using the probabilities that we have on input bits - as these tell us about how the situation is likely to develop. We need to decide how we will treat these bits. Do we just add all the scores up over a period of time or do we ignore scores that are expected to be obtained in the very near future when we can use information about the situation further into the future? We also have to decide what will be regarded as current inputs. How recent does a sensory input have to be to be regarded as contributing to the current situation's desirability?

Such issues are not something we have to worry about much here. They would need to be considered for any artificial intelligence system intended to build complex motivations from simple inputs. The main issue is what the system is intended to achieve. Once this has been decided then a decision can be made about the time scales over which inputs interest us. We would not need to provide too much complexity in doing this: the complexity emerges at higher levels of the hierarchy.

Such a way of determining the desirability of situations could also be used for hypothetical situations, which will be important in the workings of the planning system, when it is specified.

Regardless of how the evaluation function works, the idea is that the desirability of any situation, real or hypothetical, is determined from its basic sensory inputs.

### **No Separate Representation of Actions**

The hierarchy represents the description of reality - including the description of the system's actions and intentions. There is no separate representation of the system's actions. This hierarchy is all that there is.

Importantly, this means that the approach being taken is not one of having some system of representing outputs at various levels of abstraction, "closely-coupled" with the extracted meaning bits. Above the bottom level of the hierarchy it is incoherent to talk of any such distinction between inputs or outputs as the meanings are extracted from input and output bits.

The method suggested by Jeff Hawkins for making hierarchical, artificially intelligent neural networks [10,11] is to use a hierarchy of cells producing various levels of abstraction from inputs and a hierarchy of "output" or "action" cells, "closely coupled" with these cells, so that meaning goes up the hierarchy to produce abstraction and actions are produced as abstractions and come down the output side of the system with the details being filled in.

No such approach is being taken with this system. If it were mapped into a neural architecture there would be not be any "close-coupling" between meaning extraction cells and action cells because there would be no such distinction between types of cells: there

would just be the input and output cells at the bottom level and a hierarchy of cells extracting meanings from this bottom level without any distinction being made between input and output cells above the bottom level.

### **Clarification of Previous Matters**

Before we proceed further, I need to make sure that we are clear regarding the relationship bits stored in the hierarchy and the probabilistic representation of it stored in an artificially intelligent computer.

### **Input and Output Matrices and Time**

Individual input and output bits do not change over time, as happens with the bits describing an image on a computer screen or the signal in a specific human sensory or motor nerve. The bits in the input and output matrices are not the same as conventional inputs or outputs in computers. A bit in the output or output matrices corresponds to a single input or output *event* happening only for one instant. This is why the bits in these matrices cannot change. The only value that a bit in an input or output matrix can have is the bit for the particular instant to which it relates.

There is a one-to-many relationship between conventional inputs or outputs and bits in input or output matrices: a conventional input or output has many input or output events associated with it over a period of time, each with a particular value (“0” or “1”) of the input or output at a single instant.

All of the hierarchy of higher level meanings is implied (which for practical purposes means “generated”) from the input and output matrices at the bottom level. As the bits in the bottom level matrices do not change, nor do any of the other bits in the hierarchy change. This applies even for outputs. The making of an output does not imply a change in future events: it *is* one of these events.

This all follows from the hierarchy being atemporal, as was discussed in the earlier, more philosophical articles in this series, but this does not mean that the model stored in an artificially intelligent computer will not change: this would not allow a computer to learn or act. What we have just been discussing is an *idealized* version of the hierarchy in which input events, output events, and extracted meanings are *bits*. The computer only contains a *representation* of this in which all of these bits are represented by *probabilities*. Although the bit values in the idealized, atemporal hierarchy cannot change, the probability values in the computer’s representation *can* change.

### **Fixing of Input and Output Bits**

One way in which the probability values in the hierarchy can change is by the actual occurrence of input and output events. When an input or output event actually occurs then we know what the associated bit value is. If the bit value is “0” then we can assign the bit a probability of 0 and if the bit value is “1” then we can assign the bit value a probability

of 1. Once a bit value has become known like this, and assigned an absolute probability value that expresses certainty, then the probability value cannot change again and must stay as either 0 or 1 permanently. This is because the input or output event that it describes is now in the *past* and is entirely known.

I will refer to this final setting of probability values to 0 or 1 as *fixing*.

### **Propagation of Changes**

While the system is operating, probability values of bits in the input and output matrices are constantly being fixed as the input or output events with which they are associated come into the present. This fixing involves a change in probability values in the bottom level of the hierarchy and it is propagated through the rest of the hierarchy to affect the other probability values.

This is how the fixing of a bit in a bottom level input or output matrix is propagated through the hierarchy:

- Any probability values for extracted meaning bits generated by partial model algorithms that read this bit need to be recomputed, given the new probability value (0 or 1) for the fixed bit.
- Furthermore, any probability values for extracted meaning bits generated by partial model algorithms that read any of the bits that just had their probability values recomputed also need to be recomputed, and so on. Ultimately, any probability value in the hierarchy of extracted meanings that is derived, directly or indirectly, from the probability value of this input or output matrix bit needs to be recomputed.
- The propagation is not just one-way: higher level probabilities in the hierarchy imply low level probabilities and the changes will also be propagated back down the hierarchy by downward transfer of probabilities.
- Changes caused by downward transfer of probabilities will, in turn, cause further changes to higher level probabilities, and this whole process continues until the system stabilizes.

### **Meaning of the Term “Hierarchy”**

In previous articles I have described the system as “hierarchical”. The hierarchy as it has been described so far, however, is not very rigid. Partial model algorithms extract meaning from lower parts of the hierarchy but there is, as yet, no requirement for those lower parts of the hierarchy to be grouped in specific levels; for example, a partial model algorithm could extract meaning from two matrices – one consisting of direct inputs received by the system and another which had been generated by various stages of meaning extraction.

In an actual implementation of the system it may be that the hierarchy would need to be more rigidly divided into layers, so that inputs from the environment would be on the bottom layer, any meanings extracted from this on the next layer, any meanings extracted from these on the next and so on, with restrictions on which layers a partial model algorithm can access.

In this article I do occasionally use the idea of levels when discussing the hierarchy. This is not intended to imply rigid division in layers, but is just using a general idea of levels. Any parts of the hierarchy that are in direct contact with the outside world (inputs and outputs) are considered to be on the bottom level. Any other matrix of extracted meaning bits is considered to be on “higher levels” in the hierarchy if a large number of steps of meaning extraction has been needed to produce it: this does not imply that we can necessarily say accurately on which “level” it is.

### **Planning of Actions**

Now that we have a way of representing actions in the hierarchy we will consider using this to plan actions.

### **What the Planning Process Must Do**

The bottom level of the hierarchy consists only of matrices of input and output bits. The planning process is to determine what values (“0” or “1”) should be assigned to any outputs that are about to be made.

This is done by considering the possible future consequences of different outputs, in terms of the desirability score given to future inputs. The possible future consequences do not just depend on the first output to be made: they depend on subsequent actions. This means that to evaluate a given output that might be made now we have to consider future input and output events.

We run a recursive planning process, rather like that used in chess playing algorithms. This process runs a tree search to evaluate different permutations of outputs, the aim being to choose the best output to make now, by taking into consideration the outputs that are available after it. A similar approach is used in chess algorithms [13,14].

For a reason that I will explain later, the planning process will also be known as the *output vectoring process* and the system which is introduced explicitly to deal with planning will also be known as the *output vectoring system*.

### **Planning Without Higher Level Construction of Actions**

The planning/output vectoring process is restricted to the bottom level of the hierarchy. I know this will surprise some people who would have expected me to make something analogous to a plan, action or some sort of abstract description of future outputs at the top of the hierarchy and pass it down, refining it as we go along.

It may seem that we cannot plan actions at the bottom level. No partial models have extracted meaning here and we have only matrices of input bits and output bits. With no higher level patterns to describe any relationship between input and output bits how can planning be possible at all?

Planning at the bottom level is actually quite possible because, although we do not have direct access to higher level extracted meanings, we do have downward transfer of probabilities. Downward transfer of probabilities is the process by which higher level extracted meaning probabilities affect lower level probabilities. If we make an output then the change in a low level output matrix bit will affect the meaning probabilities in the level above. This in turn will cause low level bits in the input and output matrices to be affected by downward transfer of probabilities.

This means that any change to a bit in an output matrix on the bottom level of the hierarchy causes, by propagation of meaning up the hierarchy and downward transfer of probabilities back down the hierarchy, changes in probability values in both input and output matrices on the bottom level of the hierarchy. This is equivalent to the assignment of a value to an output implying predictions about future inputs and the future behaviour of the computer system itself.

I have already discussed this idea that the system will naturally predict its own behaviour. This is significant: later in this article we will put this to an important use by using predictions to *direct* behaviour.

In the planning process below, the consequences of various outputs being made will be simulated. This means that, as we go down from each node of the search tree, changes are made to the probabilistic hierarchy that only apply below that node. When we come back up the hierarchy to any point above that node – to try a different branch for example – these changes need to be undone. This is a standard method for recursive tree searches. Chess algorithms, for example, work in this way: changes to the state of a chess board are only temporary, simulated changes, which are undone when the search goes back to higher nodes in the tree.

### **How the Bottom-Level Planning/Output Vectoring Process Works**

The planning/output vectoring process is a recursive tree search. It deals with bit probabilities in “absolute” terms: whereas a bit in the hierarchy can have any probability value the planning/output vectoring process works by assigning absolute probability values of 0 or 1 to bits – equivalent to setting bit values to “0” or “1”.

A recursive search tree assigns absolute bit values to a number of bits in the output matrices, or to be pedantic - as the hierarchy is represented only as probabilities it assigns absolute probability values of 0 (meaning that the bit certainly has the value “0”) or 1 (meaning that the bit certainly has the value “1”) to bits. We do the assignment in chronological order, starting with the bits for which output is actually going to occur first; that is to say the output bits corresponding to the earliest output events.

For each bit that is being assigned an absolute probability we assign it both possible absolute probabilities - 0 and 1 (equivalent to assigning the bit the values of “0” and “1” respectively) – each corresponding to a separate branch in the search tree to a new node. For each of these nodes we consider the next output in chronological order and assign the relevant output bit both possible probabilities (0 and 1), each of these corresponding to a branch from the current node to a new node, before considering the next bit at each of these new nodes and so on. We are checking every possible permutation of assignments of probabilities of 0 or 1 to output bits.

Whenever we assign a value to an output bit we allow this to propagate through the hierarchy. A bit formerly represented by a probability value with some uncertainty is now represented by an absolute probability of 0 or 1 which allows no uncertainty and we allow the change to propagate through the hierarchy. This involves propagation of meaning up the hierarchy, but downward transfer of probabilities will mean changes are also propagated down to probabilities in the input and output matrices on the bottom level, changing prediction of the future expectation of the system’s experiences and actions.

We cannot continue indefinitely: the number of possible paths through the search tree is  $2^n$ , where  $n$  is the number of output bits being considered. When the search tree has reached a certain depth (when we have considered a certain number of bits) we therefore terminate it. This fixed-depth method of termination typifies the search tree as a flat search tree and is used in basic chess algorithms.

This means that we have a number of possible terminations of the search tree, each involving a specific permutation of assignment of probabilities of 0 or 1 to bits in output matrices. At each such termination we determine the desirability of the situation based on the scores associated with the probabilities in the input matrices for bits corresponding to future inputs.

We want to select outputs which have high desirability scores, but we are not really choosing an entire sequence of actions: we only need to decide what to do *now*. The scores obtained at the terminal nodes of the search tree are “backed-up”: they are passed back up to the start of the search tree, in a process somewhat similar (but not identical) to the search tree collapse used with the meaning extraction algorithm decision trees. Where we have two branches from a node (for 0 and 1 as probabilities on outputs), each leading to a sub-node with a score, we place the higher of the two sub-node scores on that node. This node, with its score, is then considered as one of the two sub-nodes of whichever output it followed on from, and so on. Eventually we end up with scores for assigning probabilities of 0 and 1 to the very first output bit considered in the search tree. We make the output – “0” or “1” corresponding to the higher of these two scores – and fix the probability of the relevant output matrix bit at 0 or 1.

The computation is then repeated for the next output that is required.

“Backing-up” involves using the higher of two scores below a node as the score which should be assigned to it, while in chess algorithms the lowest score for the positions branching off from a node is used in that node when we are dealing with the moves of the opponent. This is not a mistake. A chess algorithm has to be pessimistic when dealing with its opponent’s moves – assuming its opponent always has the ability to play optimally. This is equivalent to assuming that the algorithm’s own actions will have the worse possible result in the environment. In this situation there is no equivalent of an opponent. This does not mean that a system like this could not learn to play a game against an opponent: the opponent and its actions would merely be dealt with as part of the environment modelled by the hierarchy. As far as the selection of outputs is concerned, when the system has a choice of two outputs there are two possible futures but we can select the better of these two futures because the system can do this merely by choosing to make the better output later. This does not mean that we are assuming that the best possible consequences will result from making such an output: the resulting probability values on bottom-level inputs imply acceptance that there is uncertainty about the result.

### **Using Predictions to Direct Behaviour**

The planning/output vectoring process described above is computationally intensive, but we can make it much more efficient. We can use the system’s predictions of its own behaviour to make the search for optimum behaviour easier and I will now discuss this.

The hierarchy contains predictions about the system’s own future outputs. This is a necessary result of the incorporation of the system’s output bits into the hierarchy of extracted meanings and it is automatic: the system makes these predictions without the need of any special mechanism. These predictions are expressed as probability values in the bottom level output matrices.

The search process described above did not give any consideration to the likelihood of particular sequences of actions being made: it just analysed all possible sequences of outputs for a certain number of future output events.

We can use these probabilities of future outputs to prioritize the search. We can consider the same number of possible sequences of outputs, but instead of including sequences in our consideration purely on the basis of number of output events (required search depth) we can include sequences in the consideration on the basis of *probability*.

To do this we run a planning tree search as described above, but changed as follows:

Before starting the tree search we define a value that I will call the *cut-off probability*. Sequences of future outputs that have a probability of future occurrence of at least the cut-off probability will be included in the tree search and all other sequences of future outputs will be omitted. The choice of cut-off probability value is based on how much computing we can do in the tree search. If we have a lot of computing power or a lot of time to do the computation then we choose a high cut-off probability, but if we have little

computing power or there is little time available then we choose a low cut-off probability. The minimum value of the cut-off probability is 0 – specifying consideration of an infinite number of sequences of outputs – and the maximum value is 1 – resulting in no sequences being considered at all.

We maintain a *running probability* value representing the probability of the particular sequence of outputs under consideration happening. The running probability starts with a value of 1.

Each time we go to a new node in the search tree (that is to say, each time we add a new output to the sequence), we multiply the running probability by the probability of the corresponding output value. This means that if the output probability at this node is 1 then we multiply the running probability by the probability value of this output bit in the hierarchy and if the probability chosen for this node is 0 then we multiply the running probability by 1 minus the probability value for this output in the hierarchy. The resultant value becomes the new running probability.

After updating the running probability by the above operation at each node its value is examined. If the running probability is less than the cut-off probability then no further output bits are added to the sequence and the tree is terminated. This means that the search tree will be deeper when more likely paths through it are being considered and very unlikely paths through the search tree will quickly reach the cut-off probability and be terminated.

It should be noted that this means that a sequence will be considered when its running probability is slightly less than the cut-off probability because the tree is terminated when this happens, rather than before it, but this does not matter much: when the running probability is less than the cut-off probability the search tree is not extended any further and this adequately approximates what was specified earlier.

The tree search previously involved any sequence of outputs terminating when a certain number of outputs (or search depth) was reached. This feature of the tree search is removed. There is no fixed upper limit on the length of sequence that can be considered. All of the limitation on the search tree is now dependent on the cut-off probability.

This search strategy is similar to the Shannon type-B search strategy [15] in chess programming. There is one difference: the Shannon type-B strategy is more about extending the search until quiescent positions occur, whereas this strategy is about extending them based on probability, but the same idea of selective extension of searches is in use.

What I have described here is the minimum needed for such a search to work, together with the most important constraint method. Other, more conventional ways of increasing the efficiency of the search will be available.

As I stated earlier, the search tree is a simulation of the effects of possible actions. This means that any changes to the running probability are also simulated and only apply below the node in which they are made. When we go back up the search tree to higher nodes they must be undone.

### **An Example of the Use of Predictions to Direct Behaviour**

We will now look at an example of the use of probabilities on bottom-level output bits to constrain the planning/output vectoring search tree.

Suppose we choose a *cut-off probability* of 0.017.

We start with a *running probability* of 1.

We look details for the next output event to happen. This will correspond to a single output bit on the bottom level. Suppose its probability is 0.8. This means that there is a 0.8 chance that this bit has the value “1”.

We have two branches in the search tree. If we go down the “1” branch we will fix the probability of this output bit at 1 and multiply the running probability by 0.8. If we go down the “0” branch we will fix the probability of this output bit at 0 and multiply the running probability by  $1-0.8=0.2$ . We follow both branches, each with a different recursive call to the search process. We will just look at what happens with the recursive call that follows the “1” branch.

We follow the “1” branch of the search tree. The output bit under consideration is fixed with a probability of 1. The running probability is multiplied by 0.8.  $1 \times 0.8 = 0.8$  so the running probability is now 0.8. 0.8 is greater than the cut-off probability of 0.017 so the search can continue down this branch.

The next output bit to be considered has a probability of 0.72. There are two branches on the search tree. If we fix this bit with a probability of 1 we multiply the cut-off probability by 0.72 and if we fix it with a probability of 0 we multiply the cut-off probability by  $1-0.72=0.28$ . We follow both branches, each with a different recursive call to the search process. We will just look at what happens with the recursive call that follows the “0” branch.

We follow the “0” branch of the search tree. The output bit under consideration is fixed with a probability of 0. The running probability is multiplied by 0.28.  $0.8 \times 0.28 = 0.224$  so the running probability is now 0.224. 0.224 is greater than the cut-off probability of 0.017 so the search can continue down this branch.

The next output bit to be considered has a probability of 0.05. There are two branches on the search tree. If we fix this bit with a probability of 1 we multiply the cut-off probability by 0.05 and if we fix it with a probability of 0 we multiply the cut-off probability by  $1-0.05=0.95$ . We follow both branches, each with a different recursive call

to the search process. We will just look at what happens with the recursive call that follows the “0” branch.

We follow the “1” branch of the search tree. The output bit under consideration is fixed with a probability of 1. The running probability is multiplied by 0.05.  $0.224 \times 0.05 = 0.0112$  so the running probability is now 0.0112. 0.0112 is less than the cut-off probability of 0.017 so the search can no longer continue down this branch.

This is now a terminal node of this search tree. The evaluation function examines the bottom-level input probabilities and assigns the search tree node a score which can be “backed-up” to earlier nodes as described earlier.

### **Why Prediction Driven Behaviour Works**

The system can start with no previous behaviour and learn to behave competently:

Initially there is no guidance from previous behaviour. The probability values in the bottom level are all 0.5 and will not effectively constrain the search tree. The search process initially finds behaviour that is not very good, because of this lack of constraint, but it will still be slightly better than running no search process at all.

In later searches, this previous behaviour is used to constrain the search. The constraint works by means of the meaning extraction system. Meanings are extracted from previous inputs and outputs and passed up the hierarchy. Downward transfer of probabilities causes changes in probability values to propagate back down through the hierarchy, allowing these higher level extracted meanings to affect low level probabilities. Some of these low level probability values affected are the ones associated with outputs on the bottom level of the hierarchy and these are equivalent to predictions of the system’s own future behaviour. It is these predictions, resulting from the *previous* behaviour, that are used to constrain the tree search finding the *current* behaviour. When the tree search was previously constructed there was no constraint at all, but the search would still have worked to some small degree. Now that this previous behaviour is constraining the search the constraint is better and the available computational resources can be used more efficiently to find behaviour better than what was previously found. This new behaviour will in turn form part of the constraint on future behaviour, allowing computational resources to be used to be used more efficiently to improve on it still further.

There is a deeper explanation for the effectiveness of this approach that will be discussed shortly.

### **How Planning of Actions Uses Existing Parts of the System**

Much of the planning/output vectoring process relies on the existing processes of meaning extraction and downward transfer of probabilities: this is the only real reason for such processes. All that is added is a planning/output vectoring system running a tree search at the bottom level of the hierarchy to examine the future consequences of making

various outputs in terms of expected desirability of inputs, the tree search being constrained by probabilities of future outputs produced by downward extraction of probabilities.

### **The planning/output vectoring system only operates on the bottom level**

The planning/output vectoring system only operates on the bottom level of the hierarchy. The system is *not* dependent on any higher level planning of actions which are then passed down to lower levels, though the possibility of higher level planning of actions will be exploited later.

*Nothing* comes down from the higher levels of the hierarchy apart from changes in probability values, by downward transfer of probabilities.

If the top half of the hierarchy were cut off, or even more levels removed, it would not stop the system planning. Some efficiency may be lost due to downward transfer of probabilities from the missing levels not occurring, but this would just mean that the constraint on the bottom level planning system was not as good.

### **How All of the Hierarchy Is Used**

Although the planning/output vectoring process only occurs on the bottom level of the hierarchy, this does not mean that the planning/output vectoring process does not involve higher levels in the hierarchy. The bottom-level planning/output vectoring process uses information from *all* levels of the hierarchy. The meanings extracted by higher levels of the hierarchy result in information being passed down to lower levels by downward transfer of probabilities and this is used to constrain the low level planning/output vectoring process.

### **Where the Planning Is *Really* Happening**

I stated earlier that there would be a deeper explanation of the effectiveness of this approach to planning and this is it:

It may seem to some readers that the artificially intelligent system has a reasonably good ontology, compromised by a weak, shallow planning system. This would be a misconception because what I call the “planning system” actually plays a very limited part in the system’s planning!

The planning system is really just the final stage in planning. This is why I also gave it the alternative name of *output vectoring system*. Although it performs a tree search for the best outputs to make, this search is constrained by the probability values fed down by the hierarchy. If the probability values sent down to the bottom level constrain the planning system a lot then it has little freedom – although this does translate into longer range planning - meaning that most of what we would conventionally regard as “planning” is

coming from somewhere else: almost all of the planning is already done before the “planning system” gets involved.

The real process of planning is happening in the hierarchy of extracted meaning. The work of the “planning system” is necessary, but mainly to give the system a direction and relate its actions to desirability.

### **Planning is prediction**

If the hierarchy contains predictions, and actually does most of the system’s planning, then what I am saying here is that planning is really prediction. The modelling system is doing the planning.

This may seem strange to some readers, but it is actually quite natural. We often see people doing things and have a good idea – from modelling – what they are going to do next. Now, if we can apply modelling to work out what a person is going to do next, it follows that the person could also know what they are going to do next if they had access to the same kind of modelling – and it makes a good case that this is how people determine what to do next.

This gives what may be a strange picture of the process of “making our minds up” about some decision. We are used to thinking of “making our minds up” as meaning the determination of the optimum actions to make, given some model. In the context of this article it means something different. When we have not “made our mind up” about something we do not have enough information to predict our actions with regard to this matter. When we experience the “making up of our mind” it means that we now have enough information in our hierarchy to *predict* what we are going to do.

### **Planning need not be simple**

It may seem to some readers that using previous behaviour as a guide means that the hierarchy is merely expected to “copy” previous behaviour and generate future behaviour in a simplistic way. This is not the case. Basing the system’s future behaviour on its previous behaviour means that the future behaviour is based on a *model* generated from its past behaviour. The relationship between past and future outputs is merely that they are part of the same model: this model can have any degree of sophistication allowed by the available processing power.

Far from demanding that the system simply copy old behaviour, this actually allows the system to improve its behaviour. If the past behaviour shows a history of the system’s performance in some task improving, and if there is enough of a history of inputs and outputs, then the most obvious model is one which would predict further improvement for the system!

This may seem to some readers to be an attempt to get a “free lunch”. There is no “free lunch” in reality. Even though the system has an inbuilt tendency to improve its own

behaviour, it is still dependent on how much abstraction the modelling system is managing to provide. There will still be limitations on this, one of which is the available computing power.

Some readers may still be sceptical of how the system is supposed to “know” to improve itself, given that its behaviour is based on modelling from its previous behaviour. This “direction” to the system’s behaviour is given by the planning system at the bottom level of the hierarchy which does test possible outputs according to desirability. In a way, the planning/output vectoring system can be considered a link between the hierarchy (which is doing the real planning) and the way that desirability of inputs is defined. I will now give a better idea of how this is supposed to work.

### **The Planning/Improvement Cycle**

The way in which the system improves its behaviour can be thought of as the *planning/improvement cycle*. This cycle is not something that we need to do explicitly: it is just a simplified way of describing how the system’s outputs can feed back into the model to control later outputs and the role that the planning system plays in this.

The system does not really get into the planning/improvement immediately, so the description below will include the leading up to the start of the planning/improvement cycle.

1. The system initially has no previous inputs or outputs. All of the probabilities in the hierarchy, including those at the bottom level, are 0.5. The planning/output vectoring system achieves nothing and the system’s behaviour is arbitrary.
2. Input and output events have now occurred and the hierarchy contains patterns relating inputs to outputs. Any projection of future behaviour is based on arbitrary previous behaviour and is still useless. When the planning/output vectoring system makes outputs the hierarchy can at least now alter the bottom level probabilities because enough previous data exists to do this. This means that the planning/output vectoring system should at least be able to assess the consequences of actions in terms of desirability. This is better than nothing and a slight improvement in behaviour should occur. The planning (in the planning/output vectoring system itself), however, will be very short range, due to the lack of any effective constraint that is not arbitrary.
3. The system now has a hierarchy containing better behaviour from Step 3. The hierarchy makes a projection based on this previous behaviour. This projection, expressed in the bottom level of the hierarchy as output probability values, provides constraint on the bottom level planning process. The hierarchy itself is now having some influence on behaviour. There is no tendency in the hierarchy to improve the behaviour (yet) but the planning system will attempt to find the optimum behaviour, given the bottom level probability values. It manages to

slightly improve on the previous behaviour in Step 3, because it has the same computational resources but now has the benefit of this constraint.

4. The system has previous behaviour that has been improving. This means that the model of its future behaviour in the hierarchy will be a model that describes an improving system. The probability values passed down to the outputs in the bottom level will therefore constrain the planning system in a way that describes improved behaviour – even before the planning system starts working. By performing a tree search the planning system is able to improve on this behaviour slightly, meaning that the improvement achieved now is better than the previous improvement.
5. The system's behaviour has not only improved in the past. Its rate of improvement has increased. The hierarchy therefore describes a system which is functioning in this way and the constraint at the bottom level already describes a system which is improved more than last time. This constraint allows the system to improve more than last time, even without the planning system doing any work. The planning system does perform a search tree, however, and makes a very slight improvement on this. The *rate* of improvement has now been increased, and this will be reflected in future modelling of the system's behaviour by the hierarchy without the planning system having to provide it.

The planning/improvement cycle can be thought of as Step 5 repeating. There is a runaway aspect to this. Where I left it, the hierarchy had not only acquired the tendency to improve the system's behaviour, but had the tendency to increase the rate of improvement of the system's behaviour.

### **Graphical View of the Planning/Improvement Cycle**

The improvement need not stop with what was just described. If we imagine the desirability of the system's behaviour being plotted on the vertical axis of a graph, with time on the horizontal axis, then the curve is gradually extended over time and the modelling system will automatically include any previous properties of the curve in the predictions of future behaviour which constrain the planning/output vectoring system. This means that when the planning system/output vectoring makes an improvement to the behaviour or the rate of improvement of behaviour, the hierarchy's model of the system's behaviour accounts for the improvement that has occurred so far. This has a good chance of meaning that the planning system itself does not have to do anything to maintain the gradient of this curve and is free to try to optimize the system's behaviour to make the curve steeper.

### **Qualifications**

It should be noted that none of this automatically means that any improvement in behaviour recently attained by the planning system immediately becomes a permanent feature of the system. The hierarchical model of the system's behaviour, which guides its

future behaviour, is that model which most economically accounts for the system's overall behaviour.

Although a runaway process was described above, there will be limitations on it. Any unlimited runaway behaviour is dependent on a hierarchical modelling system with unlimited capability to construct models and this is not the case. An obvious limitation on the hierarchy is that imposed by the finiteness of the computing resources available.

The planning/output vectoring system will have a lot of influence on the improvement of the system in the early stages, but may eventually be marginalized. This is because as the hierarchical model of the system's own behaviour increases in scope it will be making longer range predictions of the system's possible behaviour, while the planning/output vectoring system only deals in short term predictions involving small numbers of input/output events. If this occurs however, it is not really a problem. The very fact that it occurred at all would mean that the system had outgrown the planning/output vectoring system and its own model of itself was one which incorporated a significant capability of self-improvement.

The system may not generate a model that contains some type future improvement immediately in response to some recent behaviour that contains that type of improvement. For this to happen, the model that features the future improvement has to be the one that naturally emerges in the hierarchy and the improvement may need to occur for a reasonable period of time first.

### **Stability and the Planning/Output Vectoring System**

When the system has been running long enough, its model of itself will include a description of future improvement in capability: the system's learning of more effective behaviour will be coming from its own hierarchy. At this stage we may wonder if the planning/output vectoring system is needed at all.

In one sense, we do need the planning/output vectoring system because the hierarchy is represented by probability values and these need converting into actual bit values of "0" and "1", but we do not need all of the planning/output vectoring system to do this: a simple process to convert probabilities to absolute bit values (for example rounding them up to "0" or down to "1") would suffice. Apart from this, is there any real reason why we need the planning/output vectoring system in the long term, especially if the progress from development of the system's hierarchical model marginalizes it?

One benefit of the planning system, even in the long term, is stability. The system's behaviour is generated by its hierarchical model, but the hierarchical model is partly dependent on its previous behaviour. This means that future states of the hierarchical model have some dependence on previous states. Small inaccuracies in the hierarchical model may occasionally cause an error in the value at which a bit is fixed and get propagated through to previous states and as such inaccuracies accumulate the model will

start to “drift away” from its desired behaviour. Even when it is marginalized the planning system can prevent this happening by selecting the optimum outputs to make.

### **The “System X” Analogy**

One way of thinking about how the approach described in this article works is as follows:

Imagine that you observe the behaviour of an artificially intelligent machine. You imagine that the machine is running some software called “System X” and you make a model to predict System X’s behaviour. The machine itself, however, is doing the same thing. It is also making a model to predict the behaviour of “System X” and using it to generate its behaviour.

In a way, this could be taken as meaning that System X does not really exist! Even the computer which would be running System X is just modelling it and the mind, already arguably a virtual thing, can be viewed as a virtual shadow of itself. The artist, Rene Magritte’s, work “La trahison des images” (“The treachery of images”) comes to mind here.

People should not get carried away by what I have just said: it is intended more to aid understanding of what the system is doing than as a serious philosophical claim and it could be argued that System X exists by virtue of the very attempt to model its behaviour – that System X and the model of it are equivalent.

If you are worried by this point just remember that your brain probably works in the same way, which will probably not make you feel better.

### **Outstanding Issues**

There are issues still to be resolved with the artificial intelligence system design. The main issues are as follows:

#### **Representation of the Hierarchy**

The hierarchy uses matrices of bits. A meaning extraction algorithm can only extract meaning from a number of matrices that all have the same number of dimensions and can only produce a matrix of extracted meaning with that number of dimensions. Also, unless downward transfer of probabilities works in a different way it is likely that this process will have the same problem, meaning that all the matrices have the same number of dimensions. This is inflexible.

The ontology needs to be hierarchical and to allow the sort of processes that I have described, but the way that I have put it together should only be considered a starting point. A more flexible way of representing the hierarchy is needed.

## **Downward Transfer of Probabilities**

Although I have stated the purpose of the downward transfer of probabilities process I have not yet described in detail how it will work. In comparison with other problems relating to this system it should be reasonably easy to solve, being just straightforward mathematics.

## **Generation of Partial Model Algorithms**

I have described how the artificially intelligent system will use partial model algorithms (meaning extraction models) but I have not said where they come from. There are an infinite number of possible partial model algorithms. We will probably restrict the length of these algorithms – otherwise there would be no point in using a hierarchical system – but the search for useful partial model algorithms still presents a problem.

## **The Carpet Texture Problem**

If every possible act of meaning extraction is formed for every input or output bit or extracted meaning bit then a large amount of processing could be involved, most of it irrelevant. The problem is that different acts of meaning extraction would be relevant at different times. This is already an issue in other contexts and is known as the *carpet texture problem*. It may be closely related to the issue of generation of partial model algorithms that was just discussed.

## **Abstraction of Stored Low Level Information**

In theory, the entire matrix of meaning is generated from the fixed bits corresponding to the system's history of input and output events. In practice this would mean storing every input and output that that the system has received or made and using it in generation of the hierarchy. In practice, this does not seem practical and I doubt that human brains work in this way.

It is likely that some parts of the hierarchy would need abstracting; for example, some extracted meanings derived from data which is far enough back in the past could be stored permanently as probability values and the low level data in the hierarchy beneath them erased. If we use an approach like this we would have to be careful not to compromise the integrity of the system and some restrictions on the types of meaning extraction permitted may be needed.

## **Psychological and Neurological Implications**

The artificially intelligent system being discussed in these articles is not copied from what we know about how neural systems function in organisms. It may be inspired by knowledge of human neurology and psychology in some areas, however. An artificial intelligence proposal that is intended to be reasonably strong is also really a theory of human intelligence. This does not mean that this system is proposed as an exact model of

human intelligence, but human brains should function according to the same basic principles.

I am not an expert in neurology or psychology but some obvious observations about this system in relation to human behaviour are as follows:

### **Wish-Fulfilment**

Humans seem to have a propensity for mixing up what they want to be true with what they think is true. Given the very close association between planned intentions and information about reality – they are both represented in the same structure – maybe this should not be surprising.

### **Habit Formation**

Humans find it easy to get into patterns of behaviour, and hard to get out of established patterns of behaviour. This should not be a surprise in a system that actually gets its future behaviour from modelling of its previous behaviour. This does not mean that habits cannot be broken, but for a habit to be broken a model of a mind that breaks the habit, derived from previous behaviour, would need to be a more plausible model than a model of a mind, derived from previous behaviour, that continues the habit. Maybe consideration of the sort of system I describe could give rise to ideas for dealing with issues like this in humans?

### **Delusional Beliefs**

Humans are often vulnerable to delusional belief. One way that a delusional belief could be set up in a hierarchy of the type that I describe here is by what I will call *convictional delusion*. Convictional delusional is a positive feedback situation where an increase in a high level meaning extraction probability causes downward transfer of probabilities to cause changes in lower level probability values that then cause the meaning extraction process to reinforce the original high level probability.

A simple, and rather extreme, example of this may be as follows:

John is wandering around a dark place. He does not know where he is. He can only see some shapes dimly. He estimates various probabilities of him being in various places.

Included in this is his estimate of a 1% chance that he is in a zoo.

He sees a shape and thinks there is a 5% of it being a tiger.

He sees another shape and thinks there is a 3% chance of it being a penguin.

Higher level meaning extraction (though John does not realise it he is using that), as a result of these probabilities of animals, slightly increases the chance that he is in a zoo to 5%.

Downward transfer of probabilities, due to the increase in the chance that he is in a zoo, causes the probability that the first shape is a tiger to increase to 8% and the probability that the second shape is a penguin to increase to 7%.

The increase in the tiger and penguin probabilities causes John to increase, as a result of meaning extraction, the probability that he is in a zoo to 10%.

The increase in the chance that John is in a zoo causes, through downward transfer of probabilities, the probabilities for the tiger and penguin to increase still further.

This process continues, with the low level and high level probabilities driving each other alternately by meaning extraction and downward transfer of probabilities. When this process concludes John is convinced that he is in a zoo.

John has probably just made a mistake. His opinion that he is in a zoo has mostly come from nowhere.

It is easy to imagine similar real-world scenarios; for example, someone thinks that a flying saucer has landed near his/her town, so he/she starts to think that people in suits that he/she sees are the Men in Black, increasing his/her conviction that a flying saucer has landed, which in turn feeds the Men in Black belief and so on. A lot of extreme beliefs that people have may be supported by the interaction of their own components in a way that leaves few doubts for the believer.

I am not saying that we should expect this to happen in artificially intelligent computer systems. The downward transfer of probabilities process needs designing in such a way that this does not happen. Downward transfer of probabilities should never allow a high level probability to reinforce itself by changing low level probabilities. It would be wrong for this to happen because any change in low level probabilities caused by a high level probability should, by definition, be consistent with that high level probability – which would hardly be the situation if it required a further change in it.

A human brain, however, is likely to work in a messier way than this kind of ideal system and whatever measures are used prevent convective delusion may occasionally fail in humans.

### **False Memories of the Past**

Human memories of past events often seem to be compromised. This may not be too surprising in system in which any memories of events in the past are just part of a temporal model generated from the low level inputs and outputs. There is scope for memories of the past to change, given later experiences which make a different model plausible. The extent to which this could happen would depend on the degree to which abstraction of stored low level information, previously discussed, occurs.

### **Learning by Repetition**

A model like this makes it clear why learning by repetition is useful to humans. We do not need to rely on some physically specific model, nor on some concept like “it tells your brain that the information is important”. Repeating a task would simply establish a history of inputs and outputs in which the most plausible model derived from that history

is one in which you are good at the task. This also suggests that educational ideas may be obtained from the model in this article.

## Conclusion

This article has described how actions can be represented in the hierarchy of meaning extraction described in the previous article [8]. Nothing special needs to be done: matrices of motor output bits are simply added to the bottom level of the hierarchy, with bit values represented by probabilities, and treated as part of the set of all the other matrices from which partial model algorithms can extract meaning.

Partial model algorithms can extract meaning from both sensory input and motor output matrices and this means that there is a tight association between inputs and output right from the start of meaning extraction: above the bottom level there is no distinction between information derived from inputs and information derived from outputs.

Motor outputs which have not been made yet will initially be represented by probability values of 0.5 to express uncertainty about the system's own future actions. Such probability values may be changed by downward transfer of probabilities or by the system's planning mechanism.

Downward transfer of probabilities, a process still remaining to be described in detail, will deal with context dependent frequency by allowing probabilities at higher levels to influence those at lower levels. When this causes probabilities corresponding to future outputs to be changed this may appear to be *decisions* about actions being propagated down the hierarchy, but it is in fact the system's *predictions* about its own actions. Decisions and predictions of action are the same thing.

Now that actions can be represented in the hierarchy, a method of planning actions has been described. This uses a planning system, working at the bottom level of the hierarchy, which performs a look-ahead tree search. The search is constrained by the bottom-level output probability values, meaning that the predictions of behaviour generated by the hierarchical modelling system influence the system's behaviour. The hierarchical model is actually where most of the planning really occurs.

An important concept in this article is the unification of modelling and planning. A model of reality can also be used to plan a system's actions by incorporating its previous actions into the model. An output vectoring system – a system that gives the outputs a slight tendency towards being desirable – is needed to start the system's improvement and to maintain stability. The output vectoring system relates the model, in which the planning occurs, to the concept of desirability.

## References

[1] Web Reference: Almond, P. (2005). *Occam's Razor Part 1: What Is Occam's Razor?* Retrieved 22 August 2005 from <http://www.paul-almond.com/OccamsRazorPart01.htm>.

- [2] Web Reference: Almond, P. (2005). *Occam's Razor Part 2: Principles of Language*. Retrieved 9 October 2005 from <http://www.paul-almond.com/OccamsRazorPart02.htm>.
- [3] Web Reference: Almond, P. (2005). *Occam's Razor Part 3: Assumptions About Reality*. Retrieved 13 November 2005 from <http://www.paul-almond.com/OccamsRazorPart03.htm>.
- [4] Web Reference: Almond, P. (2005). *Occam's Razor Part 4: An Overview of How Occam's Razor Works*. Retrieved 24 December 2005 from <http://www.paul-almond.com/OccamsRazorPart04.htm>.
- [5] Web Reference: Almond, P. (2006). *Occam's Razor Part 5: How Mapping Can Work*. Retrieved 14 January 2006 from <http://www.paul-almond.com/OccamsRazorPart05.htm>.
- [6] Web Reference: Almond, P. (2006). *Occam's Razor Part 6: Partial Models as "Envelopes"*. Retrieved 1 March 2006 from <http://www.paul-almond.com/OccamsRazorPart06.htm>.
- [7] Web Reference: Almond, P. (2006). *Occam's Razor Part 7: Hierarchy and Ontology*. Retrieved 30 April 2006 from <http://www.paul-almond.com/OccamsRazorPart07.htm>.
- [8] Web Reference: Almond, P. (2006). *Occam's Razor Part 8: Modelling in Artificial Intelligence*. Retrieved 9 June 2006 from <http://www.paul-almond.com/OccamsRazorPart08.htm>.
- [9] Web Reference: Almond, P. (2005). *What is a Low Level Language?* Retrieved 17 July 2005 from <http://www.paul-almond.com/WhatIsALowLevelLanguage.htm>.
- [10] Hawkins, J., Blakeslee, S. (2004). *On Intelligence*. New York: Henry Holt.
- [11] Web Reference: George, D., Hawkins, J. (?). *Belief Propagation and Wiring Length Optimization as Organizing Principles for Cortical Microcircuits*. Retrieved 24 April 2006 from <http://www.stanford.edu/~dil/invariance/Download/CorticalCircuits.pdf>.
- [12] Levy, D.N.L. (1984). *The Chess Computer Handbook*. London: Batsford. Chapter 2, pp7-37.
- [13] Levy, D.N.L. (1984). *The Chess Computer Handbook*. London: Batsford. Chapter 3, pp38-52.
- [14] Heinz, E, A. (2000). *Scalable Search in Computer Chess: Algorithmic Enhancements and Experiments at High Search Depths*. Vieweg Verlag. Chapter 0, pp11-18.

[15] Levy, D.N.L. (1984). *The Chess Computer Handbook*. London: Batsford. Chapter 4, pp58-59.