
An Attempt to Generalize AI Part 5: A Completely Probabilistic Hierarchy

By Paul Almond

27 March 2010

Website:

<http://www.paul-almond.com>

E-mail:

info@paul-almond.com

This is the fifth in a series of articles attempting an overview of how minds may work and how similar systems could be implemented in computers. The first article described a hierarchy based on *patterns*. A pattern has a specification describing a set, or population, of *pattern instances*, distributed throughout a hierarchy containing the pattern instances of all the patterns. Each pattern's set of pattern instances is used to obtain statistical information for predictions. Each pattern's population of pattern instances is to be described in a very general way, to provide a very general ontology. The fourth article discussed the need to focus the hierarchy on what is relevant, and how this requires the ability to remove pattern instances from the hierarchy. This is likely to cause some pattern instances to have missing pattern inputs, so that their states, and the states of any pattern instances which depend, directly or indirectly on them, cannot be determined with certainty. This would make the process of "fixing", an important aspect of the system up until now, impractical. The hierarchy needs to be modified so that it does not rely on any special-case "fixing" process, instead allowing pattern instances to be dealt with by purely probabilistic processes. This issue is dealt with in this article. This "fault tolerant" hierarchy will prepare for later articles to discuss how the system can be focused only on the relevant, and made more efficient, by preventing explicit representation of all pattern instances.

Table of Contents

1 Introduction	5
2 Arrangement of this Article	7
3 The Conceptual and Actual Hierarchies.....	8
4 What is Different about the Hierarchy	10
5 The Hierarchy	11
5.1 Pattern instance states are described probabilistically, but never change.....	20
6 How the Hierarchy of Pattern Instances Works	21
6.1 Logic Application	21
6.2 Statistics Generation	24
6.3 Statistics Application	25
6.4 Notes on Logic Application, Statistics Generation and Statistics Application	27
7 Other Issues	30
7.1 Planning and Actions.....	30
7.2 Forgetting.....	30
7.3 Idealized and Real Hierarchies	31
7.4 Convective Delusion.....	31
7.5 A Placeholder for the Construction Specification.....	33
7.6 Generality of the Ontology and Breaking the Strong, Geometrical Analogy	33
8 Conclusion	36
9 Bibliography	39

Table of Figures

Figure 1: Actual and Conceptual Hierarchies.....	9
Figure 2: A Pattern Instance	11
Figure 3: The Hierarchy	13
Figure 4: Patterns and the Hierarchy.....	14
Figure 5: The pattern instances of a pattern are generated and described by the pattern specification.	15
Figure 6: Pattern instances of the same pattern can be connected differently.	17
Figure 7: Roles of the logic specification and construction specification.....	18
Figure 8: Statistics Application.....	27
Figure 9: Convective Delusion.....	32

List of Abbreviations

AI	artificial intelligence
EF	evaluation function
EFS	evaluation function score
P	probability

1 Introduction

This article is the fifth in a series about artificial intelligence (AI) and how our own minds might work. Its main purpose is to set up things so that, in later articles, ways of making the system described so far model only what is *relevant* about the world can be described. The first article, *An Attempt to Generalize AI - Part 1: The Modeling System*, is available at <http://www.paul-almond.com/AI01.pdf>.¹ The second article, *An Attempt to Generalize AI - Part 2: Planning and Actions*, is at <http://www.paul-almond.com/AI02.pdf>.² The third article, *An Attempt to Generalize AI - Part 3: Forgetting*, is at <http://www.paul-almond.com/AI03.pdf>.³ (Reading the second and third articles before this one is suggested, but not absolutely necessary, as the focus here is on the hierarchy itself, rather than actions or forgetting.)

These three articles together described a hierarchy based on *patterns*, which are sets of *pattern instances*, and were intended to give an idea of how humans may model the world, plan actions and discard information from the model when it is no longer useful.

The system as described so far is impractical because there is nothing to focus modeling on what is relevant. All pattern instances of a pattern are explicitly represented, at least until being “forgotten”, whether useful or not. The system will always be finding irrelevant relationships, and building irrelevant layers of abstraction on top of them. To deal with this, methods of limiting the number of pattern instances need specifying. These could involve limiting the rate at which new pattern instances are placed in the hierarchy, but also could involve removing existing pattern instances from it. A problem with this is that the requirement for a pattern instance to have all its pattern inputs specified means that a higher-level pattern instance in the hierarchy is likely to depend on many lower-level pattern instances, and to require their continued existence to exist itself. If a lower-level pattern instance is removed, a large amount of the hierarchy may need removing. Also, requiring fully specified pattern inputs complicates the placing of new pattern instances into the hierarchy, and this has to be done by the patterns in fairly simple ways.

These issues were discussed in the fourth article, *An Attempt to Generalize AI - Part 4: Modeling Efficiency*, which is at <http://www.paul-almond.com/AI04.pdf>.⁴ It was suggested in that article that pattern instances should be allowed to have *incompletely*

¹ Almond, P. (2010). An Attempt to Generalize AI - Part 1: The Modeling System. *paul-almond.com*. <http://www.paul-almond.com/AI01.pdf>. (Also available at <http://www.paul-almond.com/AI01.doc>.)

² Almond, P. (2010). An Attempt to Generalize AI - Part 2: Planning and Actions. *paul-almond.com*. <http://www.paul-almond.com/AI02.pdf>. (Also available at <http://www.paul-almond.com/AI02.doc>.)

³ Almond, P. (2010). An Attempt to Generalize AI - Part 3: Forgetting. *paul-almond.com*. <http://www.paul-almond.com/AI03.pdf>. (Also available at <http://www.paul-almond.com/AI03.doc>.)

⁴ Almond, P. (2010). An Attempt to Generalize AI - Part 4: Modeling Efficiency. *paul-almond.com*. <http://www.paul-almond.com/AI04.pdf>. (Also available at <http://www.paul-almond.com/AI04.doc>.)

specified pattern inputs. The hierarchy might contain information about some of the pattern inputs to a pattern instance, while others, for practical purposes, would be non-existent. This would allow the removal of pattern instances from the hierarchy without having to remove what was “above” them, and it could simplify the connection of new pattern instances into the hierarchy.

This causes a problem, however. The hierarchy, as described so far, relies on a process called *logic application*, in which pattern instances with pattern outputs that are fully dependent on previous inputs/outputs are assigned pattern output values: They are *fixed*. Once they have those values they keep them permanently. The fixed pattern instances are important for statistical use. Probabilities are assigned to pattern instances of a pattern with partially or probabilistically known pattern inputs based on statistics obtained from previously fixed pattern instances of that pattern. If pattern instances are not required to have fully specified pattern instances then few pattern instances will ever have their pattern outputs known with certainty, and relying on them for these statistics will be impractical.

The solution is to modify the description of the hierarchy so that it is not dependent on the fixing of pattern instances. All pattern instances will always be considered in probabilistic terms. The way in which the hierarchy functions will actually be little different from before. There will just be no reliance on the special case of pattern instances which have pattern outputs known with complete certainty. Instead, everything will be done probabilistically. This article will deal with this.

2 Arrangement of this Article

Before we start, I will make a comment about the arrangement of this article.

This article is about a modified hierarchy, based on the one in the first article. I want this to be a description of the new hierarchy in its own right, so that there is at least one document in which the hierarchy is properly described, without readers having to look at this document, and then at the first document to see what this is based on, and so on. For that reason, some of the information from the first document will be reproduced here, modified where needed to take account of the changes to the hierarchy.

On the other hand, some readers will be very familiar with what was said in the first article and will only want to know what is different. I will also explain this briefly, and indicate where such readers may want to skip parts of this article.

Hopefully, this will accommodate everyone reasonably.

3 The Conceptual and Actual Hierarchies

The AI system is based on hierarchies of *pattern instances*. Each pattern instance belongs to a set known as a *pattern*. There are actually two hierarchies to be considered.

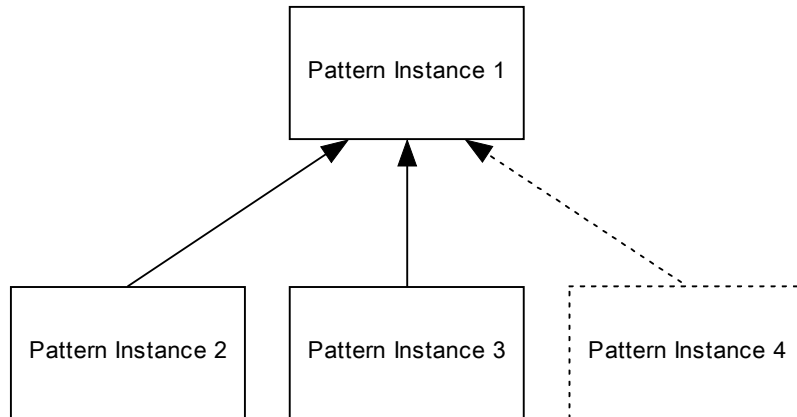
- The **conceptual hierarchy** is the hierarchy of all possible relationships between the inputs/outputs. The conceptual hierarchy contains every possible pattern instance of every possible pattern and is infinite.
- The **actual hierarchy** is the hierarchy implemented in a real AI system. It consists of the pattern instances associated with a subset of possible patterns, and for any given pattern the actual hierarchy only contains some of its pattern instances. The actual hierarchy is a piece of the conceptual hierarchy.

Given infinite computing resources, the actual hierarchy and the conceptual hierarchy would be the same, but it would be impossible to implement the conceptual hierarchy on any real computer. Instead, we can only implement some version of the actual hierarchy. The relationship between the conceptual hierarchy and the actual hierarchy is a bit like that between an idealized Turing machine and a real computer: One is an abstraction, whereas the other is a real device. The idea is for the actual hierarchy to contain those parts of the conceptual hierarchy that are most relevant in the making of the predictions of future input/output values that the hierarchy will be required to make

Just by restricting the hierarchy to using particular patterns, the actual hierarchy will become different to the conceptual hierarchy. In the previous article, *An Attempt to Generalize AI – Part 4: Modeling Efficiency*, the possibilities of selectively controlling the rate at which different kinds of new pattern instances could be added to the hierarchy, and controlling the local density of the hierarchy – either by selectively limiting the addition of new pattern instances or removing existing ones – were discussed.⁵ This will make the actual hierarchy a still smaller part of the conceptual hierarchy, but this is essential: We will only get an efficient AI system, and human brains can only have the efficiency which they have, by *not* computing things. In that article, the idea was also mentioned of allowing pattern instances in the AI system to have incompletely specified pattern inputs. This would mean that a pattern instance might exist in the AI system which has one or more pattern inputs that are not connected to other pattern instances. This is the main reason that I stated all of the above about actual and conceptual hierarchies. If a pattern instance has “missing” pattern inputs, they can be viewed as being connected to pattern instances that exist in the conceptual hierarchy, but not in the actual hierarchy, so the pattern instances are connected to parts of the hierarchy that are not being represented in the AI system, meaning that for practical

⁵ Almond, P. (2010). An Attempt to Generalize AI - Part 4: Modeling Efficiency. *paul-almond.com*. <http://www.paul-almond.com/AI04.pdf>. (Also available at <http://www.paul-almond.com/AI04.doc>.)

purposes they are left “hanging”. Because of this, it can make sense to assign such pattern instances probabilities based on information we have from the part of the hierarchy that we do know about. (See Figure 1: Actual and Conceptual Hierarchies, below. In this diagram a pattern instance is shown, Pattern Instance 1, with two pattern inputs coming from Pattern Instance 2 and Pattern Instance 3 in the actual hierarchy. A third pattern input is left undefined, but it can be assumed to be coming from another pattern instance, Pattern Instance 4, in the conceptual hierarchy.)



Key

———— Actual hierarchy and conceptual hierarchy

----- Conceptual hierarchy only

Figure 1: Actual and Conceptual Hierarchies

4 What is Different about the Hierarchy

In discussing the hierarchy, there are two main areas to cover:

- How the hierarchy works at a basic level – what patterns are, what pattern instances are, how pattern instances connect together, how inputs/outputs affect probability values in the hierarchy.
- The processes applied to the hierarchy to allow it to make predictions – logic application, statistics generation and statistics application.

The first of these, how the hierarchy works at a basic level, is going to be quite simple. Nothing about this is going to be changed. The only thing that might make things appear different is that we will no longer have any special interest in “fixed” pattern instances. In the old hierarchy, as inputs/outputs occurred, pattern inputs to pattern instances became known and the pattern outputs of pattern outputs became known with less uncertainty. When a pattern instance was completely dependent, directly or indirectly, on previous inputs/outputs, its pattern output value was known with complete certainty and we regarded the pattern instance as “fixed”. This still happens in the new hierarchy, but if we allow pattern instances to have incompletely specified inputs, it may not happen very often and it may be that few pattern instances ever get as far as being “fixed”. We will therefore not be interested in “fixed” pattern instances as a special case. Instead, we will have an interest in the general process that led to fixing: the way that inputs/outputs put information into the hierarchy and reduce the uncertainty in pattern instances, in turn reducing the uncertainty in pattern instances using them as pattern inputs and so on.

If you already understand the previous description of the hierarchy, and all this makes sense, then you may want to go straight to Section 6: How the Hierarchy of Pattern Instances Works, on page 21, where I will discuss how the processes of logic application, statistics generation and statistics application will work now. If you need more details about what I just said then just continue reading.

5 The Hierarchy

The AI system uses a hierarchical model, made of *pattern instances*.

Each pattern instance is a simple, computational unit. It has a set of labeled *pattern inputs* (e.g. A, B, C, etc.). It follows a set of rules to generate a *pattern output* value of 0 or 1.⁶ Each pattern input for a pattern instance is the pattern output from another pattern instance, which can (and usually will) be a pattern instance of a different pattern. (See Figure 2: A Pattern Instance, below.)

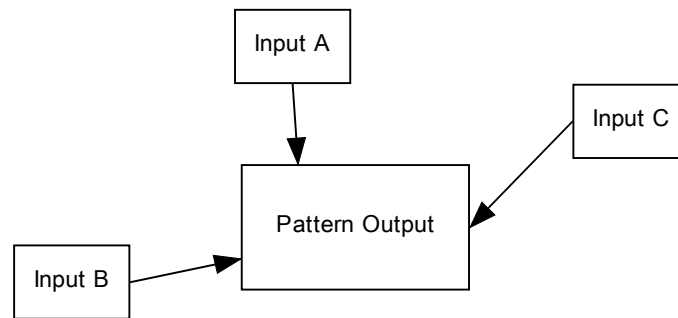


Figure 2: A Pattern Instance

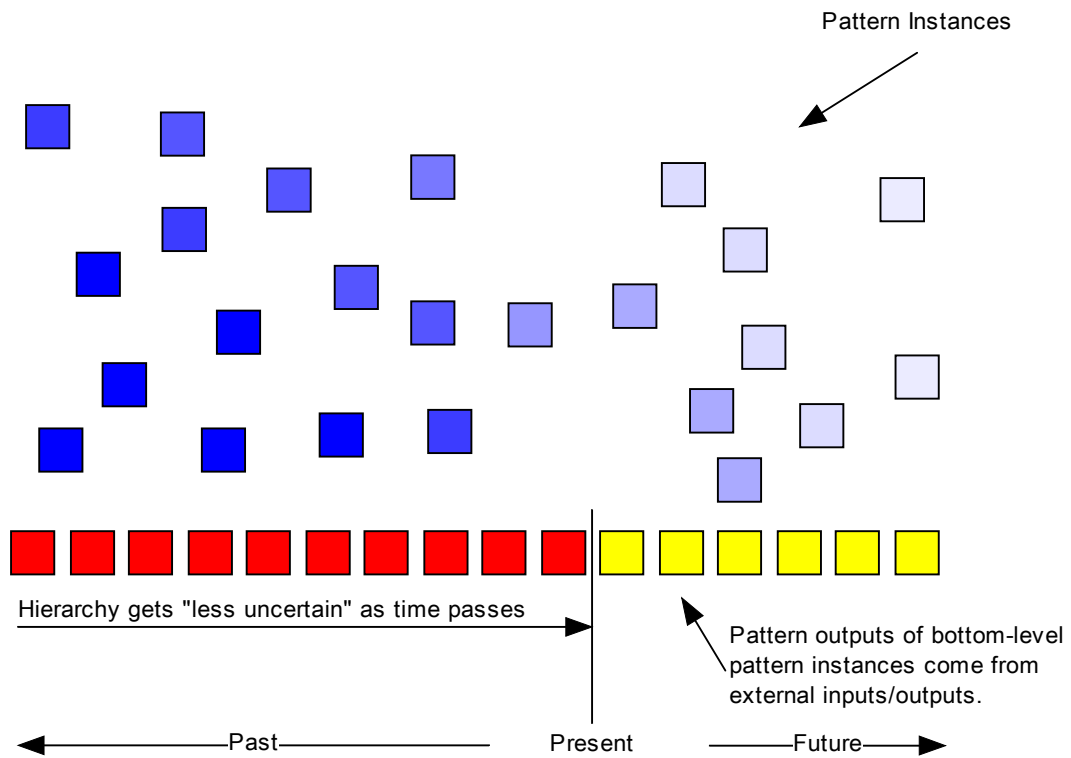
The bottom level of the hierarchical model consists of special pattern instance values corresponding to the history of values of the external inputs and outputs that the AI system has received or made previously. These can be considered to be arranged in arrays containing the input and output values that have occurred over a period of time. For example, if the AI system has a video camera, generating a 2D array of values, this could be represented in the bottom level of the hierarchy as a 3D array containing 2D images captured by the camera at different instants, each pixel state at some instant corresponding to a different pattern instance. External output values are represented as well. For example, if the system has an electrical motor, then output values sent to this motor will be stored in an array. If the system has multiple input/output devices, then there will be multiple arrays representing the history of their values. As well as representing the history of input/output events, these arrays represent the future: They are extended to contain elements corresponding to inputs/outputs that have not yet occurred. Of course, the values of future inputs/outputs can only be known probabilistically, with some uncertainty, and that is what this is all about. In most respects, the inputs/outputs on the bottom level of the hierarchy are treated as if they were pattern instances: The only difference between these and the other pattern instances is that their values are set by external input/output events, rather than by the pattern instance's computation. (See Figure 3: The Hierarchy, on page 13. In that diagram, the shade of blue is used to indicate the uncertainty in a pattern instance:

⁶ This could be generalized for values other than 0 or 1, but I will assume 0 or 1 for now for simplicity.

An Attempt to Generalize AI – Part 5: A Completely Probabilistic Hierarchy

Deep blue pattern instances are ones which are significantly dependent, directly or indirectly, on previous inputs/outputs, and about which there is little uncertainty, whereas white pattern instances are ones which are dependent on future inputs/outputs, about which little is known.⁷ As inputs/outputs occur, pattern instances gradually become more blue. In the language of the previous hierarchy, the deep blue pattern instances might be considered to be “almost fixed”. However, we need to be careful about this. There is nothing to tell us where to draw such a line. In reality, pattern instances are known about with varying degrees of confidence, based on how much they depend on previous inputs/outputs.)

⁷ They might also have a lot of dependence on “missing” pattern inputs; that is to say those connected to pattern instances in the conceptual hierarchy only.

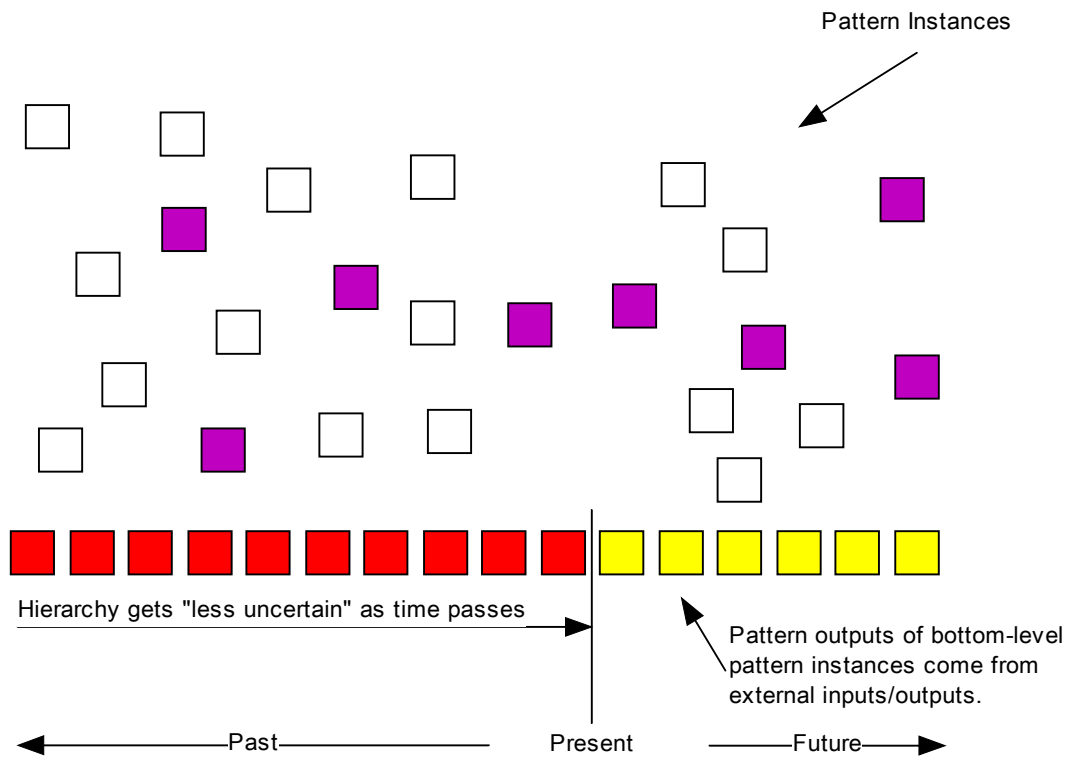


Key

- Bottom-level pattern instances corresponding to external inputs/outputs that have already occurred.
- Bottom-level pattern instances corresponding to external inputs/outputs that have yet to occur.
- Pattern instances which significantly depend, directly or indirectly, on bottom-level pattern instances for inputs/outputs which have already occurred, and which have low uncertainty in their pattern output values.
- Pattern instances which *do not* significantly depend, directly or indirectly, on bottom-level pattern instances for inputs/outputs which have already occurred, and which have *high* uncertainty in their pattern output values.

Figure 3: The Hierarchy

Pattern instances belong to *patterns*. A pattern is a set of related pattern instances. (See Figure 4: Patterns and the Hierarchy, on page 14.)

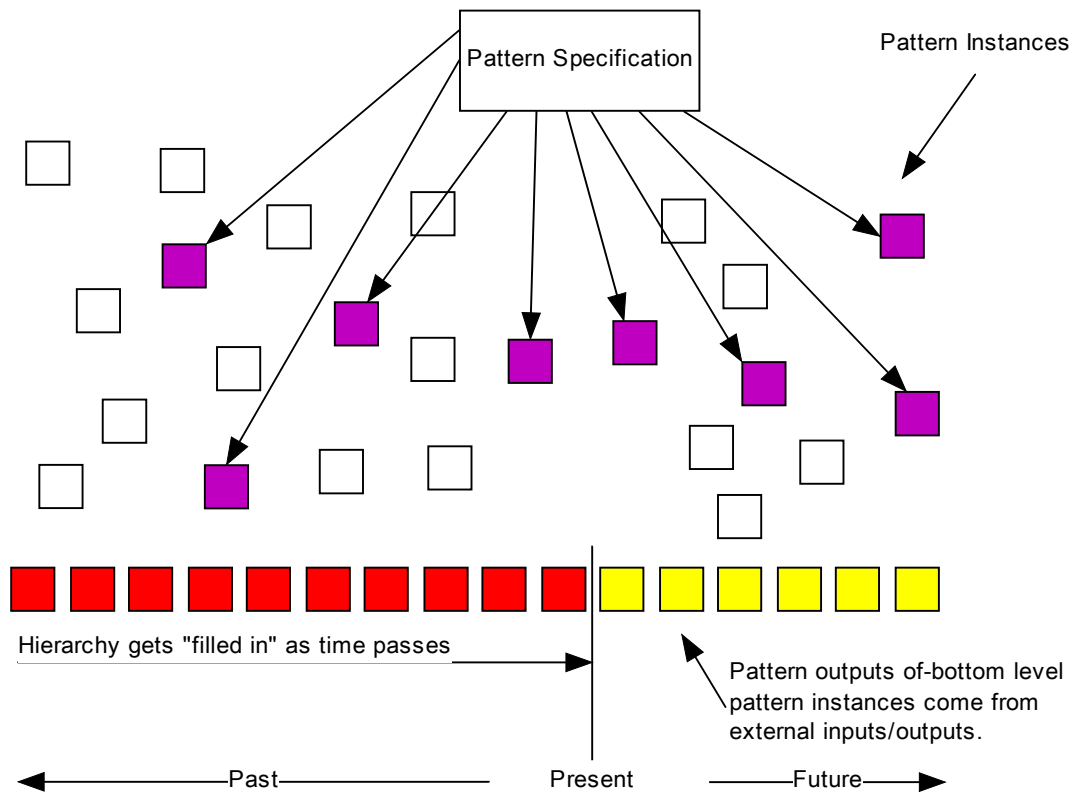


Key

- Pattern instances of the same pattern. All these pattern instances are related in some way.
- Pattern instances of other patterns.
- Bottom-level pattern instances corresponding to external inputs/outputs that have already occurred.
- Bottom-level pattern instances corresponding to external inputs/outputs that have yet to occur.

Figure 4: Patterns and the Hierarchy

Each pattern consists of a *pattern specification* and the associated set of pattern instances. The pattern specification describes the set of pattern instances and gives a set of rules about how they behave. The pattern specification causes the pattern instances to exist. (See Figure 5, on page 15.)



Key

- Pattern instances of the same pattern. All these pattern instances are related in some way: They are generated by the same pattern specification.
- Pattern instances of other patterns.
- Bottom-level pattern instances corresponding to external inputs/outputs that have already occurred.
- Bottom-level pattern instances corresponding to external inputs/outputs that have yet to occur.

Figure 5: The pattern instances of a pattern are generated and described by the pattern specification.

The rules used by a pattern instance to generate its pattern output from its labeled pattern inputs are given in the *logic specification*, which is part of the pattern specification for that pattern. The rules relate pattern inputs, described using their labels, to a pattern output for a pattern instance and they are the same for all the pattern instances of that pattern. What is different for each pattern instance in a pattern is that the labeled inputs are being obtained from different pattern outputs in the hierarchy.

Example

Suppose that for some pattern, the pattern specification states that each pattern instance has three inputs, A, B and C. The logic specification gives rules describing how labelled pattern input values relate to a pattern output value. One of these rules is:

“If Pattern Input A = 0, Pattern Input B = 1 and Pattern Input C = 0 then make the pattern output = 1.”

What this rule does not state is where pattern inputs A, B and C are coming from. Each of these will be obtained from the pattern output of another pattern instance, but it will be different for each pattern instance. Each of the pattern instances of the same pattern will get its pattern inputs A, B and C from the pattern outputs of different pattern instances in the hierarchy.

One way of thinking of this is as follows:

A pattern is a set of pattern instances. Each pattern instance is like a microchip. All the microchips (pattern instances) for a pattern are the same. Each microchip has a number of inputs (A, B, C, etc.) and uses some logic (such as a truth table) to generate an output. The logic is the same for all the microchips of a pattern. Each microchip has its inputs connected to the outputs of other microchips (which can be those of different patterns). Although the microchips for a pattern all contain the same logic (such as the same truth table), what is different is the way their inputs are connected to the hierarchy: Each microchip has its inputs connected to different outputs of other microchips, so although any two microchips use the same logic to determine what effect their pattern inputs A and B have on their outputs, their inputs A and B are wired to the outputs of completely different microchips.

The pattern consists then of a set of pattern instances, all following the same logic to generate a pattern output value from pattern inputs, but each having its pattern inputs connected differently to other pattern output values in the hierarchy. (See Figure 6, on page 17.)

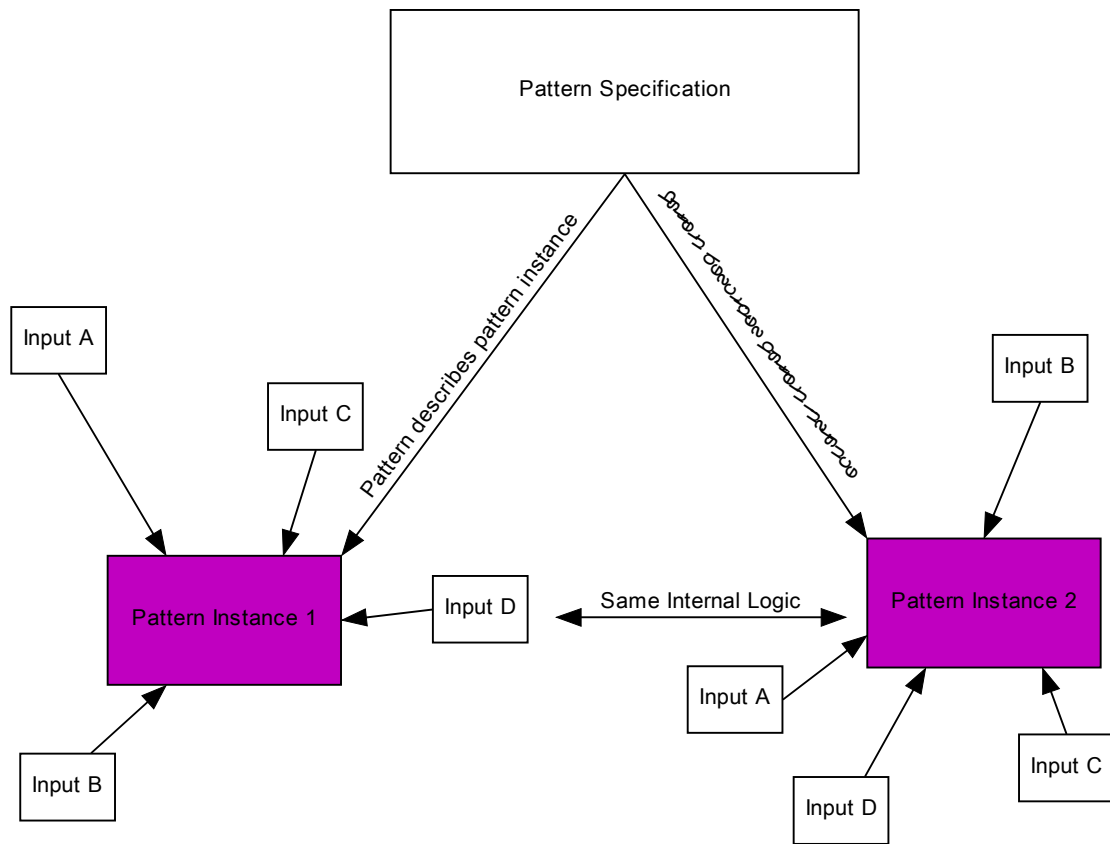


Figure 6: Pattern instances of the same pattern can be connected differently.

What needs to be dealt with now is how this set of pattern instances is generated for a pattern. How is it decided what pattern instances connect their inputs to? This is also controlled by the pattern specification. The pattern specification contains a *construction specification*. The construction specification for a pattern generates the set of pattern instances. For each pattern instance, the construction specification determines the pattern outputs to which its labeled pattern inputs correspond. (See Figure 7: Roles of the logic specification and construction specification, on page 18.)

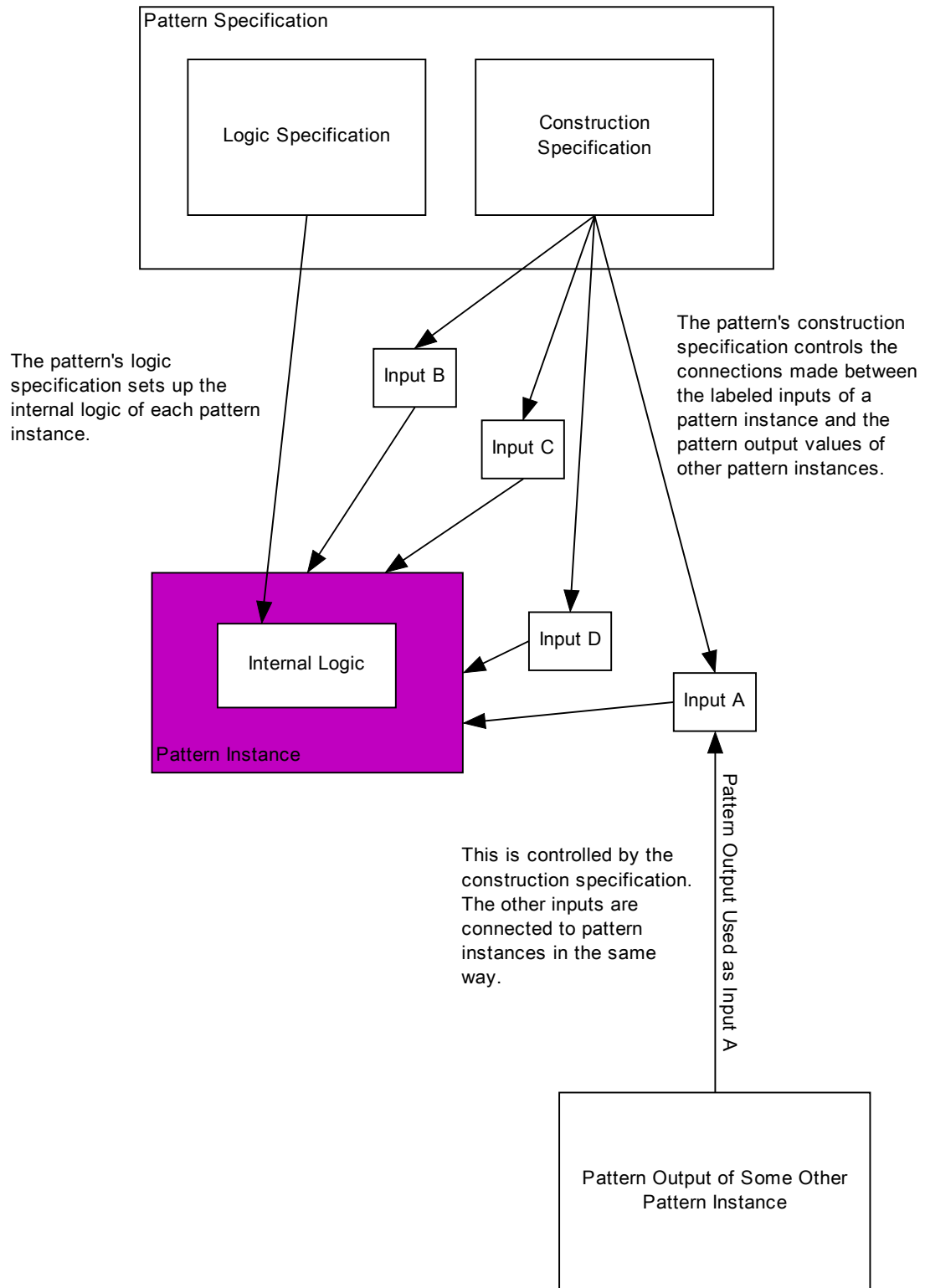


Figure 7: Roles of the logic specification and construction specification

To get an idea of how the construction specification works, imagine it “moving around” in the hierarchy, saying, “Make a pattern instance. Connect Input A to here. Connect Input B to here. Connect Input C to here. Make another pattern instance. Connect Input A to here. Connect Input B to here. Connect Input C to here...” and so on. Although the pattern instances are all connected in different ways, there is a statistical link between them all because the same thing is doing the connecting.

The whole point of this proposal is that the ontology should be as general as possible. The construction specification therefore needs encoding in a very general way. It needs to be able to distribute the pattern instances for a pattern throughout the hierarchy in very general ways. The only connection between the different pattern instances of a pattern is the logic specification and the fact they have all been set up by the same construction specification, but if the construction specification can be defined very generally, this connection could be very abstract.

The construction specification must be able to examine the “wiring” of the hierarchy when it is setting up a pattern instance. For example, when it has determined that some pattern instance that it is setting up is going to use a particular pattern output as Input A, it might follow the “wiring” from this pattern output to see what inputs that pattern instance uses, and it may look at one of these pattern instances to see what pattern instances use its output and so on.

The construction specification for a pattern might be considered being applied locally in the hierarchy, setting up pattern instances dependent on the local “wiring” of the hierarchy. An approach like this would have some similarities with the way in which the DNA of a cell controls how a cell is made, taking account of other nearby cells and signaling.⁸ This idea might be extended if the construction specification is applied “locally” in the parts of a hierarchy near an existing pattern instance, so that a new pattern instance is generated from an existing one, and “wired” to the hierarchy in a way that takes account of the local “wiring” of the hierarchy.⁹

At this stage, I am unsure about how to implement the construction specification, but we need some way of thinking about it for now. The need for generality suggests that we think of the construction specification for a pattern as being a computer program, in some general purpose programming language. The language in which the construction specification is expressed should make it possible for the construction specification

⁸ I am not suggesting a strong association here though. This is just an analogy. I am not suggesting that DNA serves as the construction specification in humans.

⁹ My use of the word “local” might be questioned, given that I have said that we are trying to get away from geometry. However, we are trying to get away from a “strong” geometrical analogy, in which things have rigidly defined coordinates. There is a sense in which a very “weak” kind of geometry can be useful and that was discussed in the previous article when I mentioned “logical distance”.

program to “read” the structure of the hierarchy, determine what pattern instances are connected to what, and make decisions based on this.

5.1 Pattern instance states are described probabilistically, but never change.

In any implementation of this, we will be assigning probabilities to the pattern output values of pattern instances. It is important to realize that the *conceptual value* of a pattern instance never changes. A pattern instance is not like a switch which changes states, for example. It represents some logic applied to specific, external inputs/outputs of the AI system occurring at specific instants in time, or some logic applied to other pattern instances (which merely means it is being applied indirectly to specific, external inputs/outputs of the AI system occurring at specific instants in time). A pattern instance’s pattern output value is just the single value determined by the specific inputs/outputs on which it depends. In a real hierarchy, we will often not know this value, either because some or all of the external inputs/outputs on which it depends have not yet occurred, or because the value is affected by some logic which is in the conceptual hierarchy, but not the actual hierarchy. We will therefore use probability values to describe pattern instances. The probability for a pattern output value may change over time, but this does not mean any change in the conceptual pattern output value: All that is changing is our state of knowledge about it. The conceptual hierarchy, which is being represented by the actual hierarchy, is atemporal, although it contains temporal information and patterns.

6 How the Hierarchy of Pattern Instances Works

The following operations are performed on the hierarchy of pattern instances.

1. Logic application
2. Statistics generation
3. Statistics application

I will now describe each of these operations.

6.1 Logic Application

This is the most obvious process. Each pattern has a logic specification describing how the pattern output value of any its pattern instances is generated, based on its pattern inputs. Logic application is the process of using what we know about the pattern inputs of each pattern instance, together with the logic specification (of the pattern to which that pattern instance belongs) to generate a probabilistically described pattern output for that pattern instance. Logic application can be thought of as applying truth tables with probabilistically described inputs to obtain probabilistically described outputs.¹⁰

If nothing is known about the pattern inputs of a pattern instance, then logic application will not add any information about it; however there are the external inputs/outputs that have already occurred. Logic application will therefore start at the bottom level of the hierarchy. Each element in the arrays of inputs/outputs to/from the AI system is treated as a pattern instance. The pattern instances corresponding directly to past inputs/outputs to/from the AI system *are* known about with certainty. This allows pattern outputs of pattern instances which receive their pattern inputs solely from the past inputs/outputs to be computed. This in turn makes the pattern outputs of more pattern instances known, allowing more pattern outputs to be calculated, and so on. Pattern outputs are calculated for pattern instances working “up” the hierarchy, increasing abstraction. So far, this is the same as logic application in the previous description of the system. The difference, now, is that *the process continues even if there is not full information about the pattern instances of patterns*. If are only known about probabilistically, this information is still used to generate a probabilistic pattern output value for that pattern instance. This can work even if there are some pattern inputs of a pattern instance about which nothing is known – if they have probabilities of 0.5.

¹⁰ It should be noted, however, that logic specifications do not *have* to use truth tables.

Example

To see how pattern logic can be applied probabilistically, suppose we knew the values of the pattern inputs to a pattern instance with complete certainty, and they were as follows.

Pattern Input A=1, Pattern Input B=0, Pattern Input C=1

The logic specification for this pattern tells us that the pattern output for this set of pattern inputs is 1, so we would assign a pattern output value of 1 to this pattern instance.

Now, suppose we did not have complete certainty, but were *almost* sure that the pattern outputs were as above. We have a probability of each pattern input being 1, as follows.

$P(A=1)=0.97$, $P(B=1)=0.03$, $P(C=1)=0.98$

This situation is scarcely different from the first one, in which we determined that the pattern output was 1. We can still tell, from this, that there is a *high probability* of the pattern output being 1. Furthermore, there is no cut-off point beyond which we cannot do this. If the probability values contained less information – if they were closer to 0.5 – they would still give us some probabilistic information about the pattern output.

Assigning a probability to the pattern output of a pattern instance, based on the pattern's logic and probabilities for the inputs, should be quite simple. We might do it just by determining the probability of each possible combination of inputs, and looking at the associated output.¹¹

¹¹ If a pattern instance has many pattern inputs, we may need to cut corners here.

Example

Suppose a pattern instance has two inputs, A and B, and the pattern's logic takes the form of the following truth table.

Pattern Input A	Pattern Input B	Pattern Output
0	0	1
0	1	0
1	0	1
1	1	0

We have probabilities for the pattern inputs as follows:

$$P(A=1)=0.37, P(B=1)=0.65$$

We can use this to calculate that:

$$P(A=0,B=0) = (1-0.37)(1-0.65) = 0.4095 \text{ (and Pattern Output=1)}$$

$$P(A=0,B=1) = (1-0.37)(0.65) = 0.2205 \text{ (and Pattern Output=0)}$$

$$P(A=1,B=0) = (0.37)(1-0.65) = 0.1295 \text{ (and Pattern Output=1)}$$

$$P(A=1,B=1) = (0.37)(0.65) = 0.2405 \text{ (and Pattern Output=0)}$$

$$\text{So, } P(\text{Pattern Output}=1) = 0.4095 + 0.1295 = \mathbf{0.539}$$

The previously described process of logic application only occurred once, ever, for any given pattern instance, assigning a pattern output value to a pattern instance (when it become dependent, directly or indirectly on previous inputs/outputs) which it kept permanently. This new process of logic application is very similar to this. If a pattern instance is ever assigned a probability of 0 or 1, the probability will not change again.

Logic application has no predictive ability. It does not, in itself, allow pattern instances dependent on future events to be computed.

Logic application is not the main operation: It is a means to an end and its main purpose is to prepare data to be analyzed in *statistics generation* (Section 6.2: Statistics Generation, on page 24).

6.2 Statistics Generation

Statistics generation involves acquiring statistical information about the frequency with which each of the possible combinations of labeled inputs is expected to occur with the pattern inputs for each pattern. Importantly, reality, and its effects on the AI system, has an influence here. For the pattern instances of a particular pattern, some combinations of pattern inputs may be more common than others, because of the particular patterns of external inputs/outputs which occur and their interaction with the way that the pattern instances of that pattern are distributed throughout the hierarchy.

Statistics generation treats all the pattern instances of a pattern as belonging to the same statistical set and generates statistics for that pattern, based on the information about pattern inputs and pattern outputs which has previously been generated by logic application (Section 6.1: Logic Application, on page 21).

For each pattern, the frequency with which each particular combination of labeled pattern inputs is expected to occur is determined, based on the combinations of probabilities which have occurred so far for pattern instances of that pattern in logic application.

One way in which we can do this is to keep a running count of the relative frequency with which each combination of pattern inputs for the pattern instances of a particular is expected to occur, and to keep this updated based on the probability values which result during logic application.

Example

Suppose we had the particular pattern instance and combination of pattern input probabilities being considered in the previous example.

$P(A=0,B=0)$	$=(1-0.37)(1-0.65)$	$=0.4095$
$P(A=0,B=1)$	$=(1-0.37)(0.65)$	$=0.2205$
$P(A=1,B=0)$	$=(1-0.37)(1-0.65)$	$=0.1295$
$P(A=1,B=1)$	$=(1-0.37)(1-0.65)$	$=0.2405$

This is suggesting that $A=0,B=0$ might occur often for pattern instances of this pattern, whereas $A=1,B=0$ might occur rarely. Of course, this is only one pattern instance. We must take account of many pattern instances of this pattern, with pattern input probabilities that have been set by logic application, to obtain a full picture.

For this pattern's statistics (which will affect any pattern instances of this pattern), we would:

add 0.4095 to the relative frequency with which $A=0,B=0$ is expected to occur.

add 0.2205 to the relative frequency with which $A=0,B=1$ is expected to occur.

add 0.1295 to the relative frequency with which $A=1,B=0$ is expected to occur.

add 0.2405 to the relative frequency with which $A=1,B=1$ is expected to occur.

The purpose of statistics generation is to generate statistical data for further extending the hierarchy probabilistically in *statistics application* (Section 6.3: Statistics Application, below).

6.3 Statistics Application

Statistics application involves using the pattern instances in the hierarchy, together with the statistics from statistics generation (Section 6.2: Statistics Generation, on page 24), to fill in the hierarchy with probability values that reduce the uncertainty in the states of its pattern instances. The idea of statistics application is that it can fill in probability values in parts of the hierarchy about which relatively little is known, and on which logic application will not have had much effect: *those pattern instances with pattern outputs which are mainly determined by inputs/outputs that have yet to occur.*

Logic application (Section 6.1: Logic Application, on page 21) will have partially filled in the hierarchy with probability values, but logic application is not a predictive process: It will only have made any significant difference to pattern instances that depend mainly on previous inputs/outputs with known values. Pattern instances that are mainly dependent, directly or indirectly, on future inputs/outputs will be mostly unaffected by logic application.

Some pattern instances will depend significantly on previous, known inputs/outputs and partly on future inputs/outputs. Statistics application allows these pattern instances to have probabilities assigned to their outputs, because the partial information we have about their pattern inputs from logic application, can be used, with the statistics for that pattern, to determine the probability of each possible combination of pattern inputs.

This in turn allows probabilities to be assigned to “higher up” pattern instances that are more dependent on future inputs/outputs, and so on.¹² The process works both upwards, generating probability values for high-level, “abstract” pattern instances in the hierarchy, and downwards, ultimately “filling in” probability values on the bottom level of the hierarchy, for the pattern instances corresponding to the arrays of future inputs/outputs. (See Figure 8: Statistics Application, on page 27.)

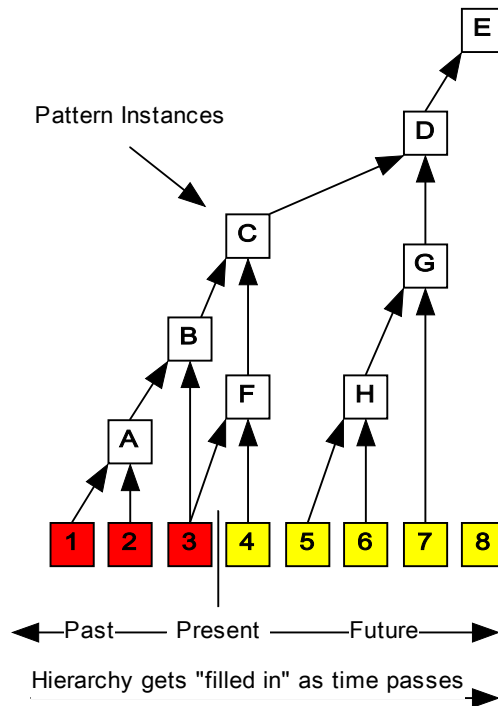
In Figure 8, pattern instances 1, 2 and 3 correspond to inputs/outputs that have already occurred, while 4, 5, 6, 7 and 8 have yet to occur, so the diagram represents a “snapshot” of part of the hierarchy, just after the input/output corresponding to Pattern Instance 3 has occurred. Pattern Instance A depends on pattern inputs 1 and 2, which are known, so its pattern output can be determined with a high degree of confidence: In fact, it could be determined with complete confidence, going off this, but this diagram is ignoring the possibility of pattern instances with missing pattern inputs. The output for B can similarly be determined by logic application with high confidence.

We have high confidence about B, but logic application will not tell us much about C or F, because they depend more on inputs/outputs that have yet to occur. However, we at least have some information about the pattern input being provided to C by B, and this means that we can use statistics application to make a reasonable, probabilistic prediction for C, based on the combinations of inputs that we expect, from experience of previous logic application, for pattern instances of that pattern. We can also use statistics application with our partial information about the pattern input to F to make a reasonable, probabilistic prediction for F, which in turn improves our ability to use statistics application with C.

When we have better probabilistic information about B, this in turn is telling us something about one of the pattern inputs for D, so statistics application enables us to make a prediction about D, which in turn allows us to make a prediction for E. Importantly, we are now making predictions for pattern instances which depend on inputs/outputs further in the future. Statistics application does not just work upwards: It

¹² In the first article, I mentioned (page 9) assigning probabilities this way to pattern instances that are *only* dependent on future inputs/outputs. I should have actually said “more dependent”, “almost completely dependent” or something similar. The idea is that as we work up the hierarchy we will encounter pattern instances with direct or indirect dependency that “spans” an increasing chronological range of inputs/outputs. When we work down the hierarchy, however, we can assign probabilities to pattern instances that are dependent only on future inputs/outputs.

works downwards, telling us about D, F, G and H, ultimately allowing us to make predictions about bottom-level pattern inputs 4, 5, 6 and 7, meaning that we are predicting external inputs/outputs.



Key

- Bottom-level pattern instances corresponding to external inputs/outputs that have already occurred.
- Bottom-level pattern instances corresponding to external inputs/outputs that have yet to occur.
- Other pattern instances.

Figure 8: Statistics Application

6.4 Notes on Logic Application, Statistics Generation and Statistics Application

Logic application, statistics generation and statistics application are ongoing processes; however they should not be mixed together incorrectly. Logic application is only intended to propagate information through the hierarchy from inputs/outputs that have already occurred. Statistics generation is only intended to use information that has been generated in logic application.

Logic application will only tend to work well with pattern instances that are strongly dependent on previous inputs/outputs. Statistics application will only tend to work better than logic application with pattern instances that are more dependent on future, external inputs/outputs, or which are dependent on missing pattern inputs. This does not mean, however, that each operation needs to be restricted to a specific part of the hierarchy. Logic application can be performed, in principle, on the entire hierarchy; however for pattern instances which are not strongly dependent on previous inputs/outputs, it will not generate much more information than is already known about them.

Likewise statistics generation does not need to be applied to any particular group of pattern instances: Information for statistics generation can be obtained from any pattern instance in the hierarchy. Only pattern instances about which logic application has generated a significant amount of information will have any significant effect on statistics generation.

Statistics application can occur with any pattern instances, but when applied to ones which are strongly dependent on previous inputs/outputs it will be ineffective: Logic application will have already told us more in such cases. When either process is applied, it is only allowed to affect the information stored about a pattern instance if it increases that information.

There is the need to ensure that logic application and statistics application work as separate processes, even though they can be applied to the same pattern instances, so that logic application only involves information from external inputs/outputs and probabilities produced in logic application itself, and to ensure that statistics generation only uses information created by logic application.

One way of separating the information produced by logic application and statistics application is to store two types of information for each pattern instance: the information generated by logic application, and the information that represents our state of knowledge about the pattern instance after statistics application. For example, a pattern instance may have a probability for its pattern output which has been generated by logic application, and a second probability for the pattern output which represents what is known about the pattern output after any statistics application process. Before statistics application, or if statistics application has occurred, but has not succeed in generating a probability with less uncertainty, the two probability values would be the same.¹³

¹³ With regard to statistics generation, it should be noted that the information from a pattern instance could be obtained once for any single process of logic application, as soon as logic application generates it, anyway.

Example

Suppose logic application indicates a probability of 0.35 that a particular pattern instance will have a pattern output of 1, and statistics application has not been applied yet. The following information might be stored for the pattern instance.

Logic Application Probability =0.35

Resultant Probability =0.35

If statistics application indicates that the probability is 0.87 then this indicates less uncertainty than the existing resultant probability of 0.35. The probabilities would be set as follows.

Logic Application Probability =0.35

Resultant Probability =0.87

and the pattern instance would be regarded as having a probability of 0.87. The probability generated by logic application, 0.35, is still retained, however, for use in any logic application or statistics generation process.

(There are other ways of doing this.)

7 Other Issues

7.1 Planning and Actions

What has been described here is how modeling is performed. A way is needed for the AI system to use this to decide what to do. Planning of actions has already been discussed in the second article of this series, *An Attempt to Generalize AI - Part 2: Planning and Actions*.¹⁴ An evaluation function (EF) is continually used to compute an evaluation function score (EFS) from recent, external inputs. Each time the EFS is computed, it is treated as an external input itself, so that it is encoded in the hierarchy's bottom-level pattern instances. When an external output is to be made, the output is tried with each possible value, updating the hierarchy accordingly, and in each case a prediction of a future input of the EFS is obtained from the hierarchy. This information is used to select the output value. As explained in the previous article, this is not the real planning process but is only a method of driving it in the right direction. The real planning process occurs within the system's model of its own behavior, which will occur in the same way as its modeling of any other aspect of reality, in the hierarchy.

What has been discussed in this article does not change any of this. This does not necessarily mean that the method of planning actions will never change at all in future, but it is not changed as a result of this article.

7.2 Forgetting

The hierarchy described here, and in the first article, is idealized, consisting of the bottom-level pattern instance values corresponding to external inputs/outputs extending without limit into the past and the future, together with the pattern instances based on them. This is impractical: In any real system there must be a limit on how much of the past and the future the hierarchy represents.

It is unrealistic to think that a human brain permanently stores the value of every input/output that has ever occurred, or every equivalent of a pattern instance that has been derived from these, and it would be impractical to do this with an AI system. The need to limit the amount of the past that the hierarchy represents requires a process that fulfils the role of forgetting in humans, removing "old" pattern instances from the hierarchy – actually, reducing the level of detail with which the hierarchy represents reality as we go further into the past.

¹⁴ Almond, P. (2010). An Attempt to Generalize AI - Part 2: Planning and Actions. *paul-almond.com*. <http://www.paul-almond.com/AI02.pdf>. (Also available at <http://www.paul-almond.com/AI02.doc>.)

Such a process was suggested in the third article of this series, *An Attempt to Generalize AI - Part 3: Forgetting*.¹⁵ This process was intended to work with the previously described hierarchy from the first article, in which logic application only worked with “fixed” pattern instances. In that process, a pattern instance would be erased when it was not having any effect on the rest of the hierarchy. The new version of the hierarchy would require some changes to this process, but the same general idea would still apply: A pattern instance would be erased when it was not having much effect on the rest of the hierarchy. This would just need to be some probabilistic version of the previous forgetting process, which does not assume that we can rely on the existence of many pattern instances about which there is complete certainty. This is not a major issue, and I will return to this later, in the meantime leaving the previously described forgetting process as an indication of the *sort of* thing that will be happening.

7.3 Idealized and Real Hierarchies

Even allowing for the use of “forgetting” to reduce the amount of information in the hierarchy, as previously discussed (Section 7.2: Forgetting, on page 30), the idealized hierarchy described here would still contain far too many pattern instances. Pattern instances at higher levels of the hierarchy would contain information even if they were based on abstraction that had been made irrelevant at a low level of the hierarchy.¹⁶ Even the kind of forgetting process already described would only remove pattern instances that were irrelevant to the rest of the *irrelevant* pattern instances in the hierarchy. In a real system, explicit computation or representation of most pattern instances needs to be avoided, and ways of “focusing” the hierarchy on what it is supposed to be predicting are needed.

Ways of focusing the hierarchy on relevant things will be discussed in later articles: This article is intended just as preparation for this. Focusing the hierarchy on relevant things requires the ability to remove pattern instances from the actual representation of the hierarchy. This could create pattern instances with incompletely specified pattern inputs, which means that the hierarchy needs to be able to work with them. This is the main reason for the changes to the hierarchy discussed in this article.

7.4 Convective Delusion

Logic application and statistics application will together propagate probabilities through the hierarchy. In any real system, we must ensure that this is not done in a way that

¹⁵ Almond, P. (2010). *An Attempt to Generalize AI - Part 3: Forgetting*. *paul-almond.com*.

<http://www.paul-almond.com/AI03.pdf>. (Also available at <http://www.paul-almond.com/AI03.doc>.)

¹⁶ One way in which this could occur would be like storing lots of information about how car parts do not exist inside your house, storing information about how these non-existent car parts are not combined into cars and storing more information about how these non-existent cars are not arranged in circles or other geometrical patterns.

causes what I will call “convective delusion”. Convective delusion occurs when some low-level belief about reality reinforces a high-level belief, which in turn reinforces the original, low-level belief, and so on, so that there is a positive feedback loop, going up and down the hierarchy, in which beliefs are reinforced tautologically.

In simple, human terms, we might imagine convective delusion working as follows.

Suppose you walk into some poorly lit, outdoor place at night. You cannot see it clearly. You see some shapes that are hard to recognize, but you think they might be animals, though you are far from certain about this. This suggests, just a bit, that you may be in a zoo. The slight possibility that you may be in a zoo makes it just a bit more likely that the unknown shapes are animals. The increased chance that you are looking at animals makes it more likely that you are in a zoo, making it more likely that you are looking at animals, in turn making it more likely that you are in a zoo, and so on. The beliefs that you are seeing animals and that you are in a zoo are reinforcing each other. If this continues, you will eventually be sure that you are seeing animals and that you are in a zoo, and this conviction will have come from nowhere. (See Figure 9: Convective Delusion, below.)

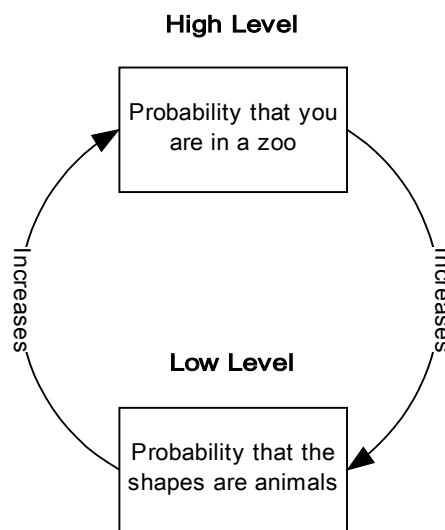


Figure 9: Convective Delusion

With the hierarchy discussed here, the equivalent of this would be a low-level probability contributing to the increase of a high-level probability which in turn increases the low-level probability, which increases the high-level probability, etc. We should not want this to happen.

The next two sections (7.5 and 7.6) are content from the first article.¹⁷ I have reproduced them here because this is a description of the way that the new hierarchy works, and I wanted to keep things together in one place. Readers already familiar with this material may wish to go straight to the conclusion, on page 36.

7.5 A Placeholder for the Construction Specification

I have not given full details about how the pattern specification is encoded. The construction specification needs to be expressed in a very general way, to provide the general ontology that is sought here. At this stage, I am not claiming to know exactly how the construction specification should be expressed, and some consideration should also be given to how the logic specification should be expressed, although this is not as much of a problem.

For now, as a placeholder idea, to allow meaningful discussion of the concept, I suggest that the logic specification is thought of as a truth table or something similar, relating labeled pattern inputs to a pattern output, and the construction specification is thought of as a small computer program, in some Turing equivalent language which constructs a set of pattern instances for the pattern. When this construction specification program constructs a pattern instance it will need to associate its labeled pattern inputs with the specific pattern outputs of actual pattern instances. To do this, it needs to be able to examine the hierarchy – the network of pattern instances that already exists. For example, the language in which it is expressed needs to allow it to “read” the type of pattern to which a particular pattern instance belongs, to determine the pattern instances serving as its inputs, to examine the inputs of these pattern instances and so on. The language needs to allow a program to “examine” the hierarchy and wire a new pattern instance into it. I am not claiming that this is the best way of doing things, but it will do for now.

As mentioned previously, this is analogous with the way in which DNA controls construction of cells, taking account of local conditions, and we might extend this idea, using an approach in which a pattern instance can give rise to another pattern instance by “running” the pattern’s construction specification locally.

7.6 Generality of the Ontology and Breaking the Strong, Geometrical Analogy

The hierarchy is intended to provide a general ontology which can have real-world relationships mapped onto it. The ontology is more general than a hierarchy I described

¹⁷ Almond, P. (2010). An Attempt to Generalize AI - Part 1: The Modeling System. *paul-almond.com*. <http://www.paul-almond.com/AI01.pdf>. (Also available at <http://www.paul-almond.com/AI01.doc>.) p25, pp27-28.

in a previous proposal, before starting this series of articles.¹⁸ In that system, each pattern consisted of a set of pattern instances arranged in an array, each input of a pattern instance being the pattern instance of another pattern at some coordinate offset.¹⁹ This required all the pattern instances of a pattern to be at the same level of the hierarchy, looking at the same data. Reality does not work like this. An abstract concept such as “circle” might be relevant at multiple levels of the hierarchy and describe a relationship between different kinds of object. The basic system could not, in itself, represent this kind of abstraction. The approach limited the relationships between the pattern instances in a pattern to being ones based on a strong, geometrical analogy. In reality, the inputs/outputs at the bottom level of the hierarchy may be related according to some simple, strong, geometrical analogy, but this might not be the case for more abstract pattern instances, higher in the hierarchy. The previously proposed system would impose this geometrical analogy all the way up though, with the geometry used closely matching that of the input/output data. As well as limiting the generality of the hierarchy’s ontology, this could cause problems when the input/output data on the bottom level corresponds to pattern instances in different arrays. This will be particularly obvious if the arrays of input/output pattern instances have different dimensionality for different input/output devices.

I knew this was an issue at the time, and to try to deal with it I proposed that “meta-patterns” should be used. A meta-pattern defines a set of patterns combining all of their statistics. A single meta-pattern could cause many patterns to exist at many levels in the hierarchy, allowing more abstract relationships to be represented. This solution is a crude attempt to provide generality. Any generality it provided would be limited according to whatever system was used for representing meta-patterns and the basis of it all would still be simple, geometrically based patterns. While they may give some capability for abstraction, the meta-patterns themselves would still impose a strong, geometrical analogy. The proposal in this article is intended to provide more generality, although with the cost of me being unable to say, right now, how some parts of the system, in particular, the construction specification, should work.

This issue of geometrical analogy, where the higher-level features of the model are expected to map onto the same kind of geometry as the inputs/outputs is also present in the hierarchical system proposed by Hawkins.²⁰ The system being proposed here is an

¹⁸ Almond, P. (2006). A Proposal for General AI Modeling. *paul-almond.com*. <http://www.paul-almond.com/Modeling.pdf>. (Also available at <http://www.paul-almond.com/Modeling.doc>.)

¹⁹ For example, if Pattern Instance (100,100) of Pattern 1 obtained Input A from Pattern Instance (103,104) of Pattern 2, then Pattern Instance (101,101) of Pattern 1 would obtain Input A from (104,105) of Pattern 2.

²⁰ 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>.

Hawkins, J., Blakeslee, S. (2004). *On Intelligence*. New York: Henry Holt.

attempt to get beyond this by breaking the dependence of relationships between elements of the same kind in the hierarchy on an analogy with geometry and the way that the inputs/outputs are arranged.

With the system proposed here, “hierarchy” does not mean a system arranged in neat layers. All it really means is that some pattern instances get their inputs from other pattern instances and so can be considered to be on a “higher level” than them.

In the proposed system, the construction specifications of patterns would probably tend to be simple. This means that any individual pattern is not likely to use relationships vastly different from those of the pattern instances from which its own pattern instances tend to get pattern inputs. The bottom level of the hierarchy consists of arrays of pattern instances corresponding to external inputs/outputs. These pattern instances *are* arranged in a regular, geometrical way, so it is likely that pattern instances which obtain all their inputs from here will also tend to be related with a geometrical analogy, but to a slightly lesser degree. As we go higher in the hierarchy, the relationships between pattern instances will become progressively less like those between the bottom-level pattern instances. Things will become less describable in simple, geometric terms, or in some cases different geometries will start to apply. At the bottom level of the hierarchy, and with pattern instances not too far away from it, it will make sense to use coordinates to describe the “position” of a pattern instance, but as we go higher in the hierarchy this will become less meaningful and it will only make sense to describe “where” a pattern instance is in terms of the pattern instances it uses for inputs.

This “breaking of the strong, geometrical analogy” is an important part of what I am trying to do here.

8 Conclusion

In the first article of this series, an overview was given of how minds may work and how a hierarchical modeling system might work in an AI system.²¹

The proposed system is based on *patterns*. A pattern is a set, or population, of *pattern instances*. A pattern instance is a simple, computational unit that produces a pattern output based on applying some computation to labeled inputs, which are obtained from the pattern outputs of other pattern instances. Each pattern has a *pattern specification* consisting of a *logic specification* and a *construction specification*. The logic specification describes how the pattern output of each pattern instance is determined from the labeled inputs. The construction specification describes how the pattern instances for the pattern are distributed throughout the hierarchy, with the labeled pattern inputs for each pattern instance corresponding to actual pattern outputs of other patterns.

The earlier description of the hierarchy was based on the idea that any pattern instances would eventually become completely dependent on inputs/outputs that had already occurred. Patterns in such a situation were referred to as “fixed” and their pattern output values would be known with certainty. The previously fixed pattern instances of a particular pattern were to be used to generate its statistics, by counting different combinations of pattern inputs, so that these statistics could be used in *statistics application* to make probabilistic predictions about parts of the hierarchy with unknown pattern inputs and probabilities could be propagated through the hierarchy.

In the fourth article of this series, the issue of efficiency in the hierarchy was discussed. It is important to be able control the numbers of pattern instances added to the hierarchy, and to remove them when they do not appear relevant to the prediction of the particular, external inputs/outputs required of the hierarchy.²² A problem with this is that it could cause pattern instances to lose some of their pattern inputs, leaving them with incompletely specified pattern inputs. That, and the issue of connecting pattern instances into the hierarchy in the first place being more complex if fully specified pattern instances are required, suggest that the hierarchy should be able to function with pattern instances that have missing pattern inputs.

A problem with pattern instances with missing pattern inputs is that the pattern outputs of such pattern instances could never be known with certainty: Such pattern instances could never be fixed. This does not just affect pattern instances with missing pattern inputs: Any pattern instances using them as pattern instances would also always have some uncertainty associated with them. Unless the removal of a single low-level pattern

²¹ Almond, P. (2010). An Attempt to Generalize AI - Part 1: The Modeling System. *paul-almond.com*. <http://www.paul-almond.com/AI01.pdf>. (Also available at <http://www.paul-almond.com/AI01.doc>.)

²² Almond, P. (2010). An Attempt to Generalize AI - Part 4: Modeling Efficiency. *paul-almond.com*. <http://www.paul-almond.com/AI04.pdf>. (Also available at <http://www.paul-almond.com/AI04.doc>.)

instance is typically accompanied by the removal of a huge number of higher-level pattern instances from the hierarchy, few pattern instances would be fixed in such a system, and the process of logic application, which is needed to obtain statistics about patterns, would be impractical.

This article has been aimed at addressing this. Instead of relying on the special-case process of fixing, the hierarchy is now working completely probabilistically. Logic application can now occur with pattern instances even when they are not known about with certainty. This allows the hierarchy to be used even if some pattern instances have lost their pattern inputs, and it sets things up for later articles in which methods for focusing the hierarchy on what is relevant will be described.

The issue of pattern instances with “missing” pattern inputs can be viewed with reference to the ideas of the *conceptual hierarchy* and the *actual hierarchy*. The conceptual hierarchy is a theoretical structure containing all possible relationships between the system’s external inputs/outputs. The conceptual hierarchy contains every possible pattern instance of every possible pattern. Representing the conceptual hierarchy in a computer is impractical, so instead the actual hierarchy, a part of the conceptual hierarchy, is represented. The actual hierarchy contains only a subset of the pattern instances in the conceptual hierarchy, and the idea is that those pattern instances will be selected to represent the parts of the conceptual hierarchy most relevant to making predictions about the particular, future, external inputs/outputs that are of interest. If a pattern instance in the AI system has a “missing” pattern input this can be viewed as a pattern instance in the actual hierarchy having a pattern input that is connected to another pattern instance in the conceptual hierarchy: It is still meaningful to assign a probability to such a pattern instance, based on what we do know about it.

The revised hierarchy, with its ability to work with incompletely specified pattern instances, is more fault tolerant than the previous version. As well as being useful in allowing the hierarchy to be “edited”, this fault tolerance makes it possibly a better candidate for the system produced by the human brain, in which the correct functioning of every equivalent of a pattern instance cannot be assured.

As with the earlier description of the hierarchy in the first article, I am not, at this stage, claiming to know exactly how the construction specification should be expressed, and some consideration should also be given to how the logic specification should be expressed. The construction specification needs to be expressed in a very general way, to provide the general ontology that is sought here. For now, as placeholders, I suggest that the logic specification is thought of as a truth table or something similar, and the construction specification is thought of as a small computer program which constructs a set of pattern instances for the pattern, and that the computer language used for this is thought of as being Turing equivalent and one which allows programs to “read” information about the structure of the hierarchy.

An Attempt to Generalize AI – Part 5: A Completely Probabilistic Hierarchy

Like the first article, this article has not discussed how the pattern specifications are generated. From where do the logic specifications and construction specifications come? This will be discussed later, but for now I will say that trial and error will play a big part in this. This may seem to be asking a lot of a trial and error process, given that the patterns are supposed to provide a system with intelligence. It is important to realize, however, that patterns are not expected to be very intelligent by themselves. A pattern is not required to solve any complex problem. All a pattern is required to do is expose a statistically interesting relationship within its set of pattern instances.

As with the first article, I ask readers to note that I am not claiming to be able to describe every detail, but am merely attempting to present an overview of what may be going on in a mind, and what might be done to achieve the same in a computer.

9 Bibliography

Almond, P. (2006). *A Proposal for General AI Modeling*. *paul-almond.com*. Retrieved 27 March 2010 from <http://www.paul-almond.com/Modeling.pdf>. (Also available at <http://www.paul-almond.com/Modeling.doc>.)

Almond, P. (2010). An Attempt to Generalize AI - Part 1: The Modeling System. *paul-almond.com*. Retrieved 27 March 2010 from <http://www.paul-almond.com/AI01.pdf>. (Also available at <http://www.paul-almond.com/AI01.doc>.)

Ibid. p25, pp27-28.

Almond, P. (2010). An Attempt to Generalize AI - Part 2: Planning and Actions. *paul-almond.com*. Retrieved 27 March 2010 from <http://www.paul-almond.com/AI02.pdf>. (Also available at <http://www.paul-almond.com/AI02.doc>.)

Almond, P. (2010). An Attempt to Generalize AI - Part 3: Forgetting. *paul-almond.com*. Retrieved 27 March 2010 from <http://www.paul-almond.com/AI03.pdf>. (Also available at <http://www.paul-almond.com/AI03.doc>.)

Almond, P. (2010). An Attempt to Generalize AI - Part 4: Modeling Efficiency. *paul-almond.com*. Retrieved 27 March 2010 from <http://www.paul-almond.com/AI04.pdf>. (Also available at <http://www.paul-almond.com/AI04.doc>.)

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>.

Hawkins, J., Blakeslee, S. (2004). *On Intelligence*. New York: Henry Holt.