An Attempt to Generalize AI Part 1: The Modeling System

By Paul Almond

13 February 2010

Website: http://www.paul-almond.com
E-mail: info@paul-almond.com

This is the first in a series of articles that will attempt to give an overview of how minds may work and how similar systems could be implemented in computers. The approach is 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. Later articles will discuss how the system can be made more efficient, by introducing an analogy of "forgetting" and preventing explicit representation of all pattern instances, and how the system can be used to provide intelligent behavior, rather than just modeling.

Table of Contents

Table of Figures

List of Abbreviations

AI	artificial intelligence

1 Introduction

In this series of articles I will present an overview of how minds may work and how artificial intelligence (AI) could be implemented. This is therefore about both human cognition and AI.

The approach needs a modeling system which provides a very general ontology. The purpose of this modeling system is to predict the values of future inputs. This article will describe an idealized version of such a system. In later articles I will describe what can be done to make a real version of it more workable and how it can be used to make a system behave intelligently.[1]

Why I am writing about this now, when I have previously made suggestions about how minds work and how general AI might be implemented? The proposals in previous articles have been limited due to what I think I can describe in a workable way.[2] A consequence of this is that the behavior of AI systems built in those ways is likely to be limited. The main way in which the systems I have proposed so far have been limited is in generality of ontology. The hierarchical modeling systems previously proposed involve sets of elements arranged in neat arrays, forcing the relationships between them to be simple and imposing a "strong, geometrical analogy" on the system.[3] In this series of articles I will try to go beyond that, proposing a much more general ontology. This will not be without problems. I will admit now that I do not know how to do everything that I am proposing. I am hoping at least to present an overview of what should be a system with very general capabilities, and which I am suggesting as a candidate for how minds work.

[1] People who have read my previous articles will know that I think there is a close link between planning and modeling, and there will be no surprises about that here.
[2] Almond, P. (2006). *A Proposal for General AI Modeling*. Retrieved 10 April 2009 from http://www.paul-almond.com/Modeling.pdf.
(Also available at http://www.paul-almond.com/Modeling.doc.)
[3] For now the word "strong" here is unimportant. The idea of a "weak", geometrical analogy in this kind of system will be discussed in a later article.

2 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.[4] 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 1, below.)

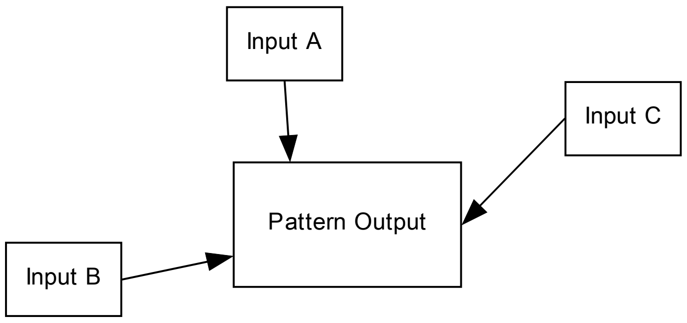
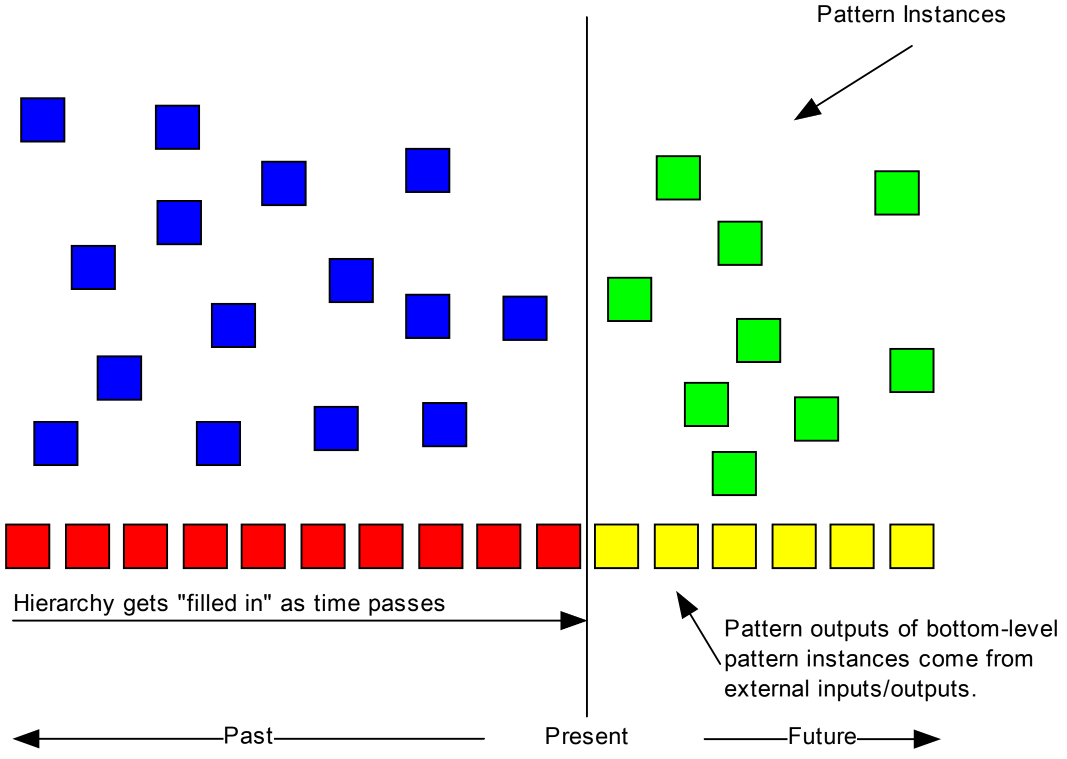


Figure 1: A Pattern Instance

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, 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 2, below. In that diagram, each of the green or blue boxes is a pattern instance, connected to other pattern instances in the hierarchy, which are its inputs,

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

and using some internal logic to produce a pattern output value. Imagine the vertical line labeled “Present” sweeping across the diagram from left to right, with boxes turning blue – being “fixed” with values – when it passes over them.)


Pattern Instances
Hierarchy gets "filled in" as time passes
Pattern outputs of bottom-level pattern instances come from external inputs/outputs.
Past
Present
Future

Key

🟥 Bottom-level pattern instances corresponding to external inputs/outputs that have already occurred. Each of these is "fixed": Its pattern output value was assigned, permanently, when the relevant input/output event occurred.

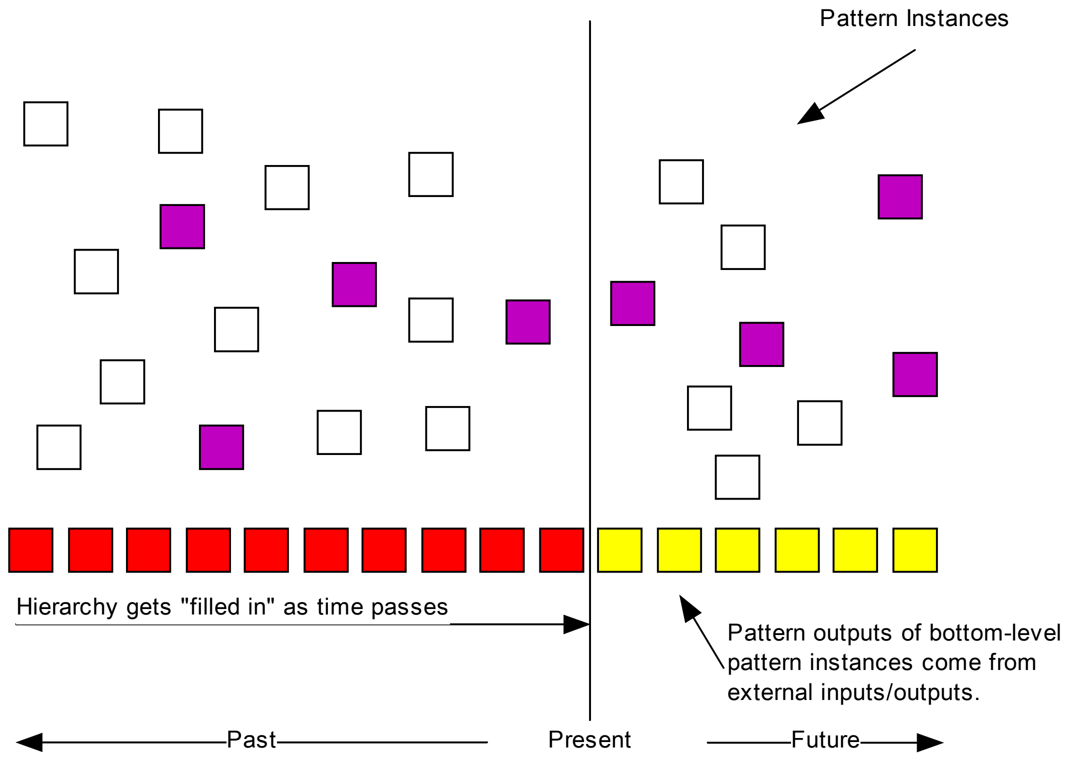
🟦 Pattern instances which depend, directly or indirectly on bottom-level pattern instances for inputs/outputs which have already occurred. Each of these is also "fixed": Its pattern output value was assigned, permanently, when all the pattern instances being used for its inputs were assigned pattern output values.

🟨 Bottom-level pattern instances corresponding to external inputs/outputs that have yet to occur. Each of these has not yet been "fixed": The pattern output value will be assigned when the relevant input/output event occurs.

🟩 Pattern instances which depend, directly or indirectly on bottom-level pattern instances for inputs/outputs that have yet to occur. Each of these has not yet been "fixed": Its pattern output value will be assigned, permanently, when all the pattern instances being used for its inputs have been assigned pattern output values.

Figure 2: The Hierarchy

Pattern instances belong to *patterns*. A pattern is a set of related pattern instances. (See Figure 3, below.)


Pattern Instances
Hierarchy gets "filled in" as time passes
Pattern outputs of bottom-level pattern instances come from external inputs/outputs.
Past
Present
Future

Key

🟪 Pattern instances of the same pattern. All these pattern instances are related in some way.

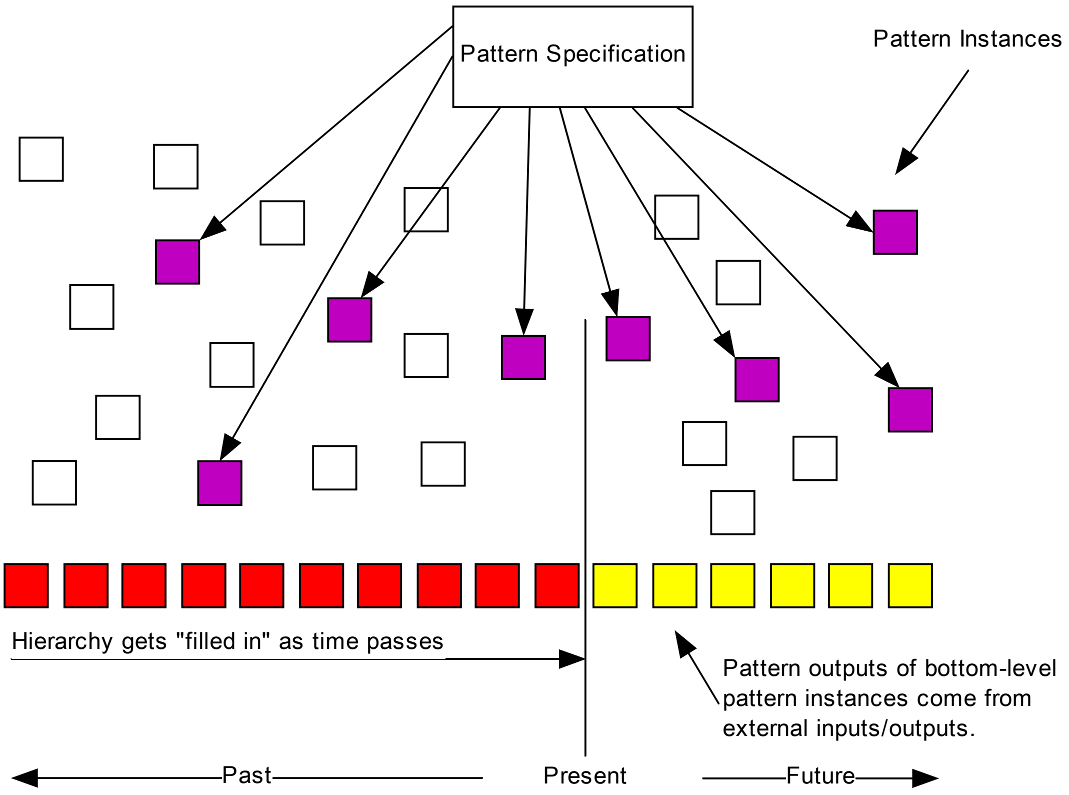
⬜ Pattern instances of other patterns.

🟥 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 3: 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 4, below.)


Pattern Specification
Pattern Instances
Hierarchy gets "filled in" as time passes
Pattern outputs of bottom-level pattern instances come from external inputs/outputs.
Past
Present
Future

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 4: 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 outputs 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 inputs connected differently to other pattern output values in the hierarchy.[5] (See Figure 5, below.)

[5] It may be tempting to think of a pattern instance as corresponding to a neuron, but I would caution against this. For reasons that will be explained later, the structure of the hierarchy is not fixed. Only a small number of pattern instances that could be represented will be explicitly represented in the hierarchy at any time, suggesting that the relationship between pattern instances and neurons will be more complex.

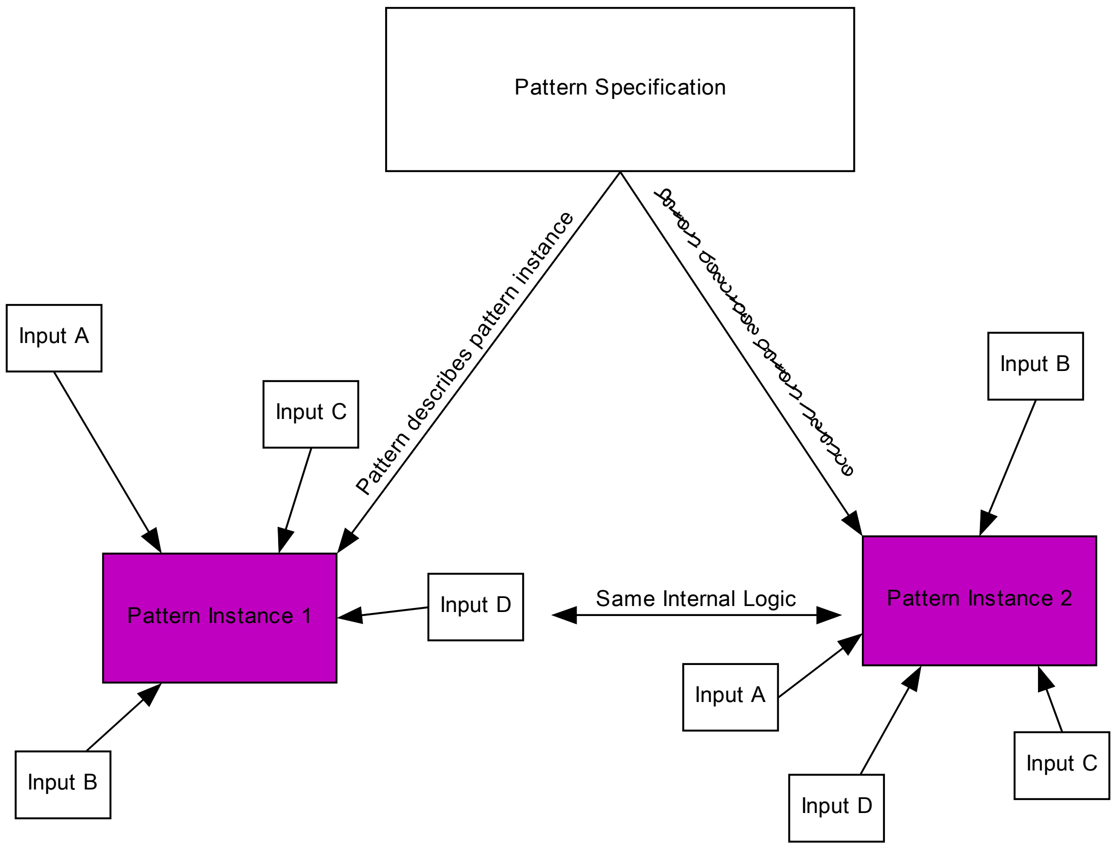


Figure 5: Pattern instances of the same pattern can be connected differently.

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 6, below.)

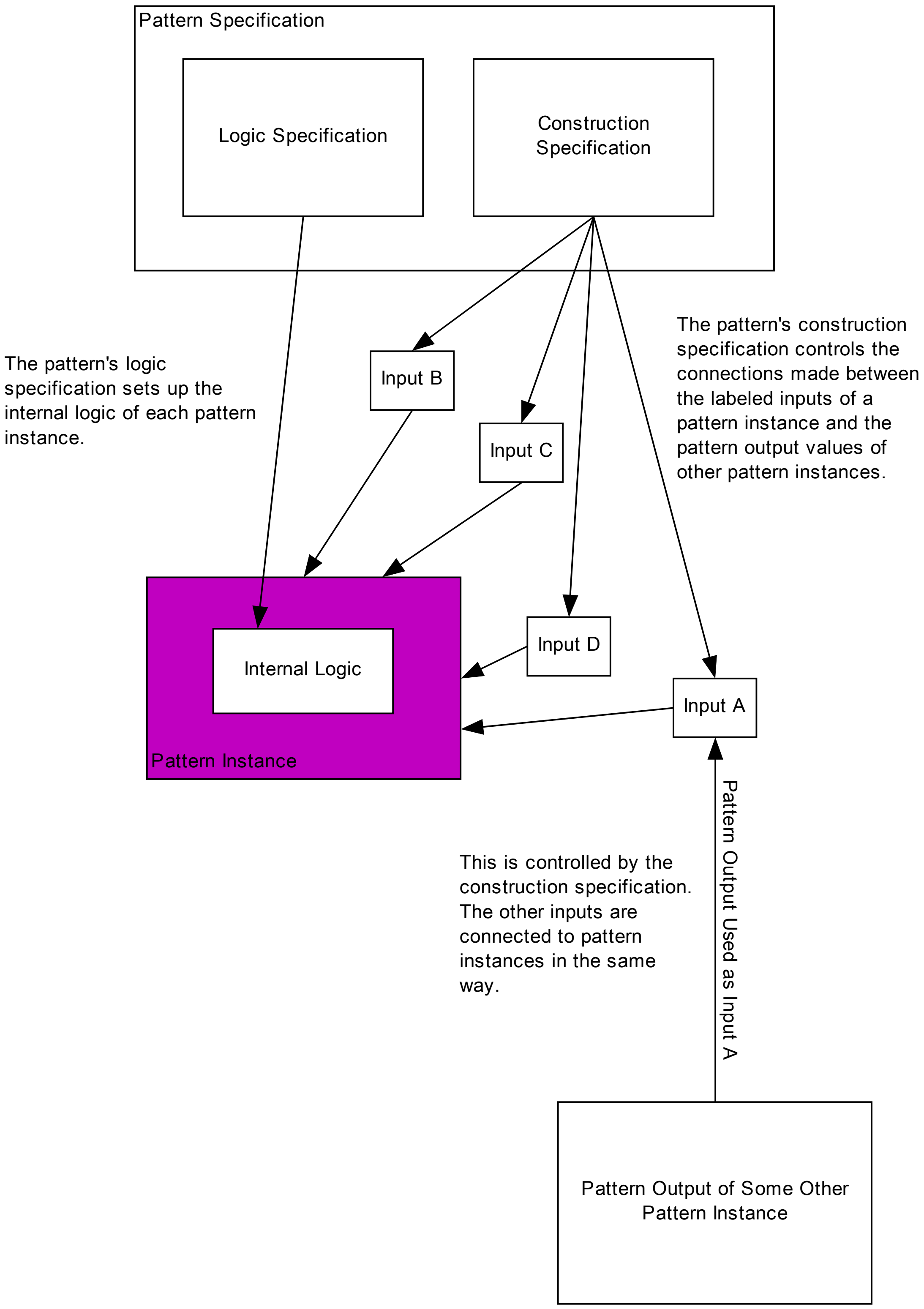

Pattern Specification
Logic Specification
Construction Specification
The pattern's logic specification sets up the internal logic of each pattern instance.
The pattern's construction specification controls the connections made between the labeled inputs of a pattern instance and the pattern output values of other pattern instances.
Input B
Input C
Input D
Input A
Internal Logic
Pattern Instance
This is controlled by the construction specification. The other inputs are connected to pattern instances in the same way.
Pattern Output Used as Input A
Pattern Output of Some Other Pattern Instance

Figure 6: 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.[6] 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.[7]

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

[6] 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.

[7] 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 I will discuss this in a later article.

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

3 How the Hierarchy of Pattern Instances Works

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

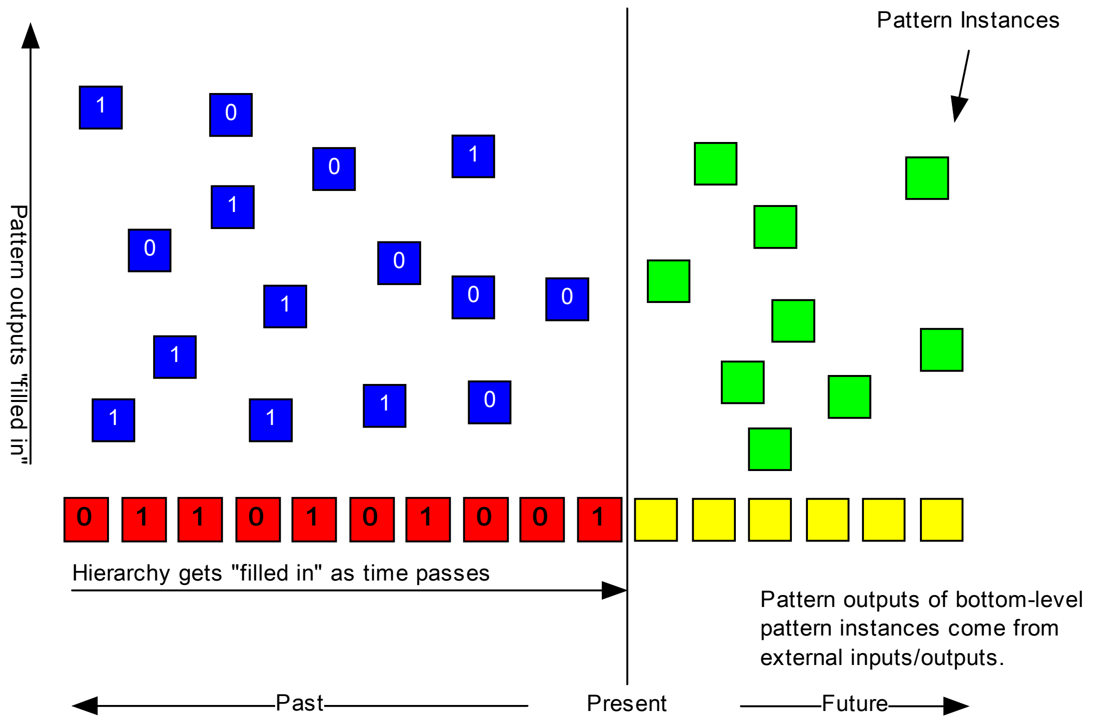
1. Logic application
2. Statistics generation
3. Statistics application

I will now describe each of these operations:

3.1 Logic Application

This is the most obvious process. It is the process of setting up known pattern outputs in the hierarchy. For every pattern instance that has all its pattern inputs known, the pattern output is computed. This will 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 instances. For past inputs/outputs to/from the AI system, these pattern outputs are known. This allows pattern outputs of pattern instances which get 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.

Logic application only occurs once, ever, for any given pattern instance. When the values of a pattern instance's pattern inputs are known, it is assigned a pattern output value by logic application, and keeps that value permanently. This means that a pattern instance's pattern output value is "fixed" when it becomes dependent, directly or indirectly, on previous inputs/outputs. (See Figure 7, below.)


Pattern Instances
Pattern outputs "filled in"
0 1 1 0 1 0 1 0 0 1
Hierarchy gets "filled in" as time passes
Pattern outputs of bottom-level pattern instances come from external inputs/outputs.
Past
Present
Future

Key

🟥 Bottom-level pattern instances corresponding to external inputs/outputs that have already occurred. Each of these is "fixed": Its pattern output value was assigned, permanently, when the relevant input/output event occurred.

🟦 Pattern instances which depend, directly or indirectly on bottom-level pattern instances for inputs/outputs which have already occurred. Each of these is "fixed" in this process: Its pattern output value is set by applying its internal logic to its inputs from other pattern instances.

🟨 Bottom-level pattern instances corresponding to external inputs/outputs that have yet to occur. Each of these has not yet been "fixed": The pattern output value will be assigned when the relevant input/output event occurs.

🟩 Pattern instances which depend, directly or indirectly on bottom-level pattern instances for inputs/outputs that have yet to occur. Each of these has not yet been "fixed": Its pattern output value will be assigned, permanently, when all the pattern instances being used for its inputs have been assigned pattern output values.

Figure 7: Logic Application

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* (see below).

3.2 Statistics Generation

Statistics generation involves acquiring statistical information about the frequency with which each of the possible combinations of labeled inputs occurs with the pattern inputs for each pattern.

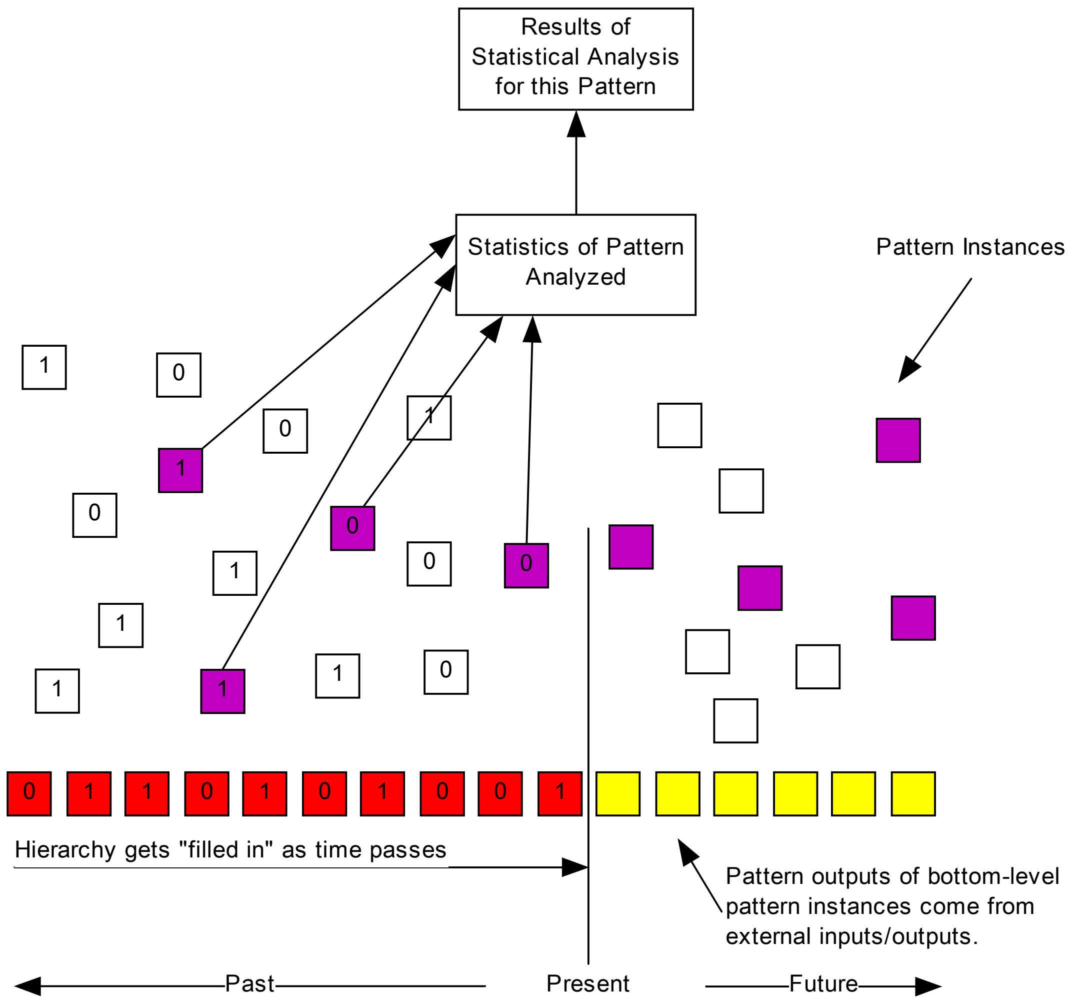
Statistics generation treats all the pattern instances of a pattern as belonging to the same statistical set and generates statistics for that pattern, based on all of its pattern instances. It involves looking only at pattern instances with pattern outputs which have already been set by logic application (see above).

For each pattern, the frequency with which each particular combination of labeled pattern inputs occurs is counted.

Example

Suppose a particular pattern has three labeled pattern inputs: A, B and C. One combination of labeled pattern inputs is A=1, B=1, C=0. The number of times that A=1, B=1, C=0 has occurred with pattern instances for that pattern is recorded. This is done for all combinations of pattern inputs for that pattern, using the combined data for all pattern instances for that pattern which have known pattern output values; that is to say, looking at every pattern instance which is dependent, directly or indirectly on past inputs and outputs, and which has already been assigned a pattern output value in logic application (see above).

(See Figure 8, below.)


Results of Statistical Analysis for this Pattern
Statistics of Pattern Analyzed
Pattern Instances
1
0
0
1
1
0
0
1
0
0
1
1
0
1
0
1
1
0
0 1 1 0 1 0 1 0 0 1
Hierarchy gets "filled in" as time passes
Pattern outputs of bottom-level pattern instances come from external inputs/outputs.
Past
Present
Future

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 8: Statistics Generation

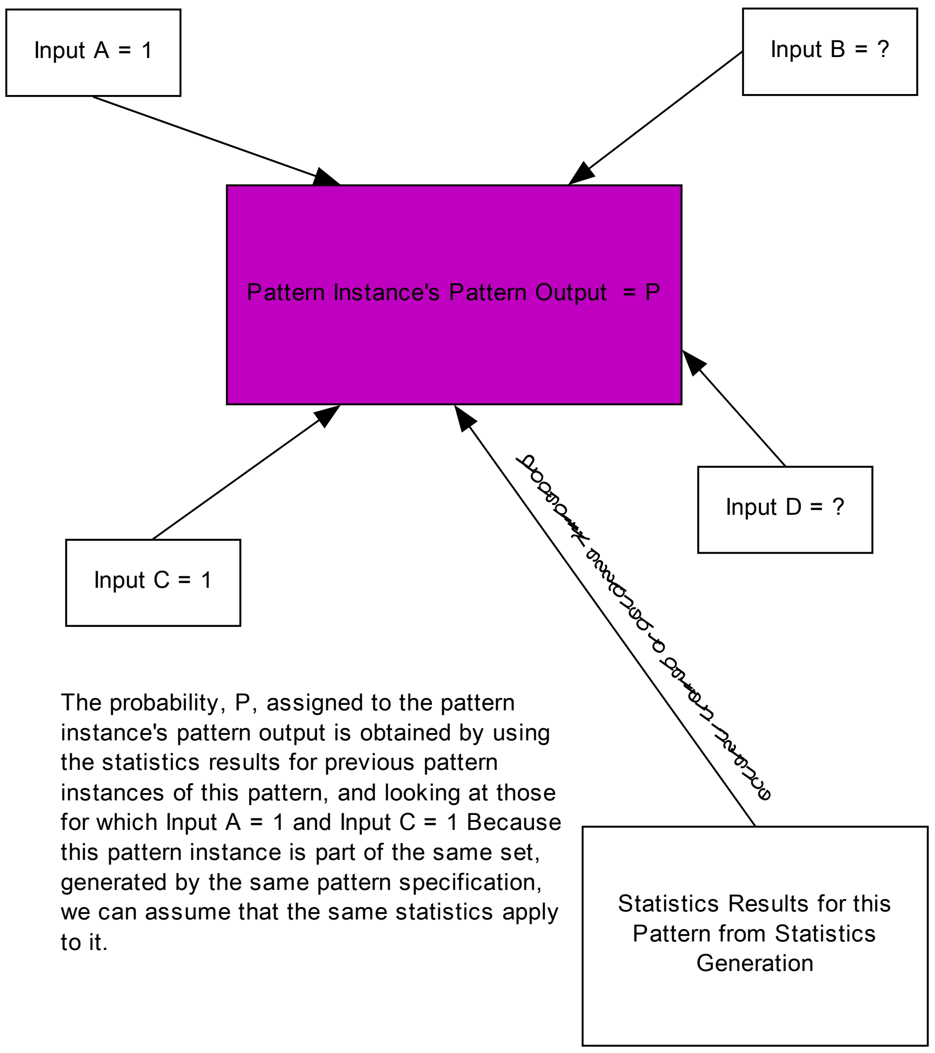
The purpose of statistics generation is to generate statistical data for extending the hierarchy probabilistically in *statistics application* (see below).

3.3 Statistics Application

Statistics application involves using the pattern instances in the hierarchy, together with the statistics from statistics generation (see above), to fill in the unknown parts of the hierarchy – pattern outputs for those pattern instances which depend, directly or indirectly, on future inputs/outputs - with probability values.

Logic application (see above) will have filled in the hierarchy with 0/1 values for pattern instances that depend only on previous inputs/outputs with known values. Some pattern instances will depend partly on previous, known inputs/outputs and partly on future inputs/outputs. Statistics application allows these pattern instances to have probabilities assigned to their outputs. This in turn allows probabilities to be assigned to "higher up" pattern instances that are dependent only on future inputs/outputs, and so on. The process works upwards and downwards, 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.

Statistics application starts with pattern instances for which some of the pattern inputs are known, and will be easier to understand if we start by looking at how we can deal with these pattern instances. For such a pattern instance, the fact that we know some of the pattern inputs means that the statistics produced in statistics generation for the pattern to which that pattern instance belongs can be used to obtain a probability that that pattern instance will have a pattern output of 1. This can be done by looking at pattern instances which have already been assigned pattern outputs and which have pattern inputs matching the ones that we know with the pattern instance being considered. (We do not know all of the pattern inputs of the pattern instance being considered, so we are only looking at pattern instances for which some of the pattern inputs are the same.) We can look at the total number of such pattern instances and the number with a pattern output of 1, finding the proportion with a pattern output of 1. This is the probability that the pattern instance being considered has a pattern output of 1. (See Figure 9, below.)


Input A = 1
Input B = ?
Pattern Instance's Pattern Output = P
Input D = ?
Input C = 1
Statistics Results for this Pattern from Statistics Generation

The probability, P, assigned to the pattern instance's pattern output is obtained by using the statistics results for previous pattern instances of this pattern, and looking at those for which Input A = 1 and Input C = 1 Because this pattern instance is part of the same set, generated by the same pattern specification, we can assume that the same statistics apply to it.

Figure 9: Assigning a Probability to a Pattern Instance

Example

Suppose we are considering a pattern instance with pattern inputs A, B, C and D. We know that A = 1, B = 0. We do not know pattern inputs C and D. We want to know the probability that this pattern instance will have a pattern output of 1. We can do this by looking at the numbers of pattern instances with known pattern outputs and with pattern inputs A = 1, B= 0 and any values for C and D.

Previous statistics application for the pattern of which this pattern instance is a part has told us that:

A=1, B=0, C=?, D=? (where ? means “0 or 1”) occurred 1,946 times, some of these involving a pattern output of 0 and some of them involving an output of 1.

A=1, B=0, C=?, D=? with a pattern output of 0 occurred 1,226 times.

A=1, B=0, C=?, D=? with a pattern output of 1 occurred 720 times.

(1,226 + 720 = 1,946)

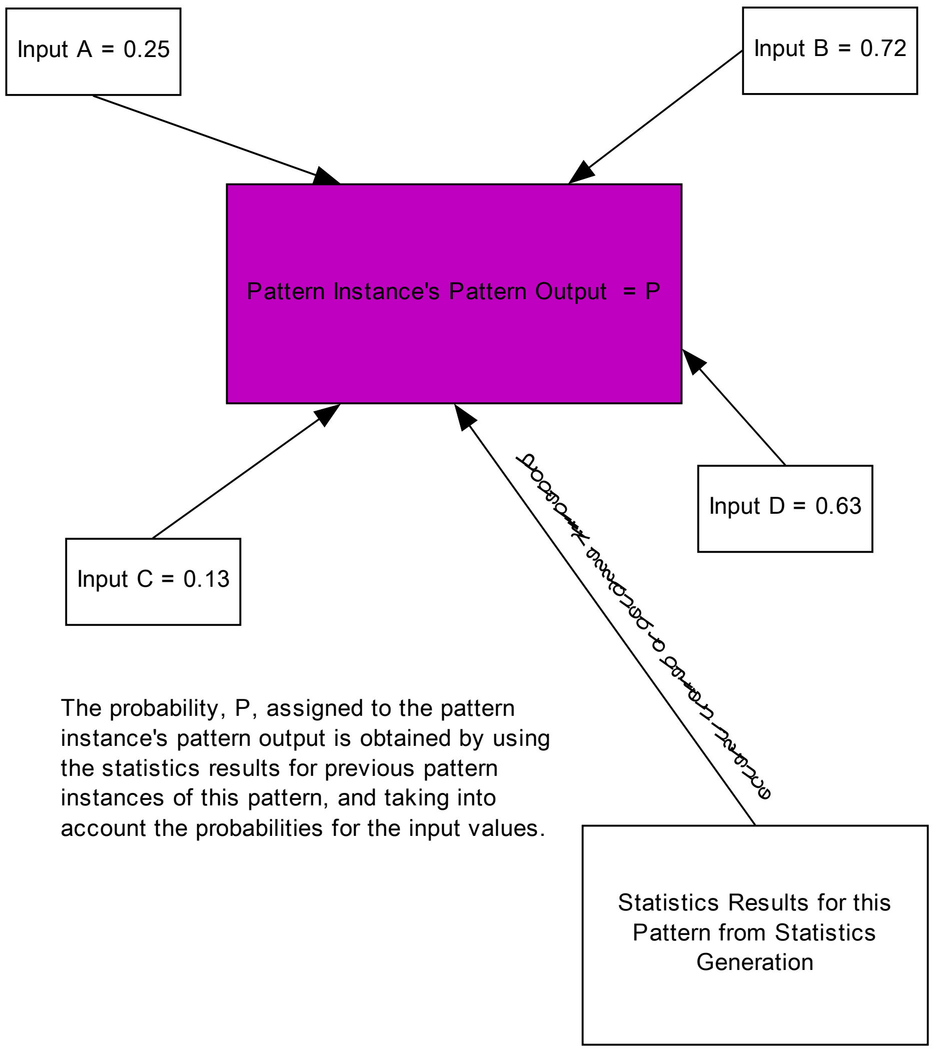
Probability that pattern output will be 1 = $^{720}/_{1,946}$ = 0.3700.

The mathematics here is fairly simple: If we have previous pattern instances with known pattern inputs to look at, we can get a probability of an output of 1 for a pattern instance when we only know some of its pattern inputs.

The process is not limited to obtaining probabilities for pattern instances with partially known pattern inputs. We can use a similar kind of process, using the statistics for a pattern to obtain a probability for one of its pattern instances, even if we only have probabilities for its inputs. (See Figure 10, below.) In fact, the procedure just discussed is merely a special case of this. If we know the value of one of the inputs to a pattern instance, from the previous logic application, we can assign a probability of 0 or 1 to it[8], depending on its value, and if we know nothing at all about one of the inputs, we can assign a probability of 0.5 to it. We can use this probabilistic approach, so that we do not need a separate process.

The probabilistic information that we have propagates through the hierarchy: When we have obtained probabilities for pattern outputs in this way, any pattern instances which use these pattern outputs as their pattern inputs can be assigned probabilities in a similar way.

[8] If a pattern output is known to be 0 we can regard it as having a probability of 0, and if it is known to be 1 we can regard it as having a probability of 1.


Input A = 0.25
Input B = 0.72
Pattern Instance's Pattern Output = P
Input D = 0.63
Input C = 0.13
Statistics Results for this Pattern from Statistics Generation

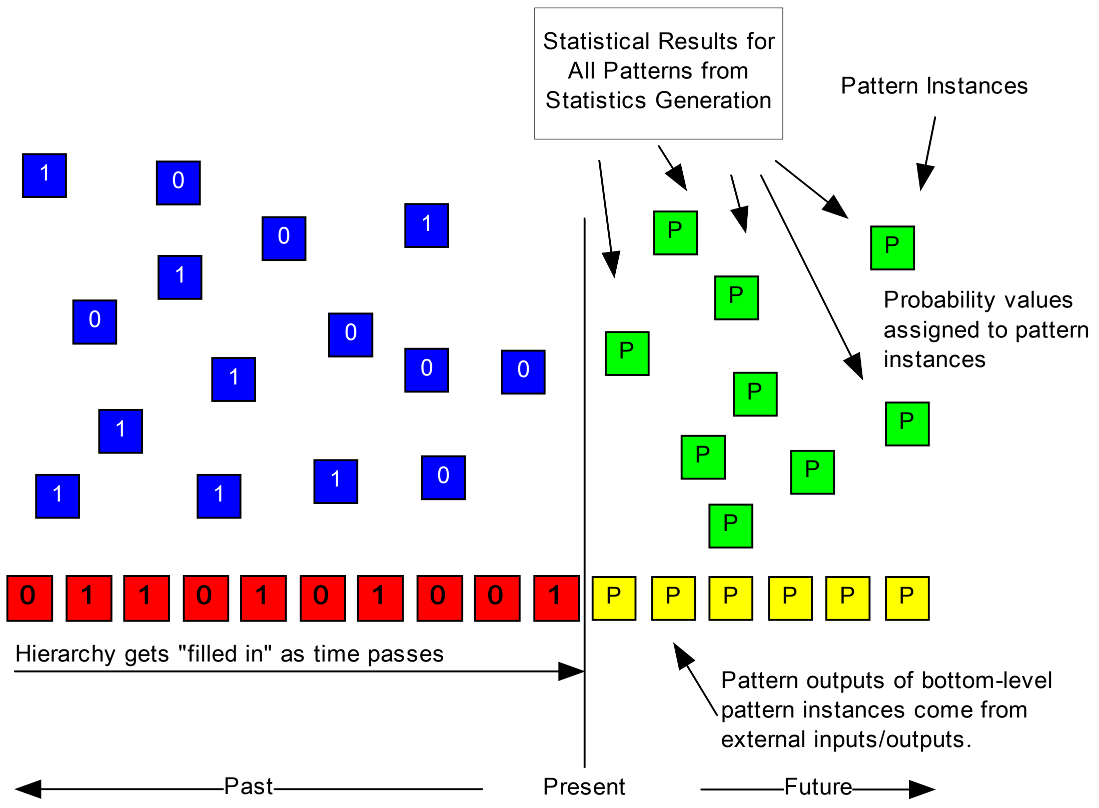
The probability, P, assigned to the pattern instance's pattern output is obtained by using the statistics results for previous pattern instances of this pattern, and taking into account the probabilities for the input values.

Figure 10: Assigning a Probability to a Pattern Instance from Probabilities for its Inputs

Example

If a pattern instance has pattern inputs A and B, and we know that the pattern output corresponding to Pattern Input A has a probability of 0.75 of being 1, and the pattern output corresponding to Pattern Input B has a probability of 0.27 of being 1, we can use the information obtained in statistics generation for the pattern of which this pattern instance is a member to obtain a probability of the this pattern instance having an output of 1.

Statistics application does not just work upwards. If a pattern instance is used as a pattern input for one or more pattern instances, and probabilities are known for the pattern outputs of these pattern instances, a probability can be assigned for that pattern instance. Probabilities can be transmitted downwards, as well as upwards, in the hierarchy. In this way, all pattern instance outputs in the hierarchy can be assigned probabilities – including those on the bottom level of the hierarchy corresponding directly to future inputs/outputs. (See Figure 11, below.)


Statistical Results for All Patterns from Statistics Generation
Pattern Instances
Probability values assigned to pattern instances
1 0 0 1 1 0 0 1 0 0 1 1 1 1 0
P P P P P P P P P
0 1 1 0 1 0 1 0 0 1 P P P P P P
Hierarchy gets "filled in" as time passes
Pattern outputs of bottom-level pattern instances come from external inputs/outputs.
Past
Present
Future

Key

🟥 Bottom-level pattern instances corresponding to external inputs/outputs that have already occurred. Each of these is "fixed": Its pattern output value was assigned, permanently, when the relevant input/output event occurred.

🟦 Pattern instances which depend, directly or indirectly on bottom-level pattern instances for inputs/outputs which have already occurred. Each of these was "fixed" in logic application: Its pattern output was assigned a value permanently.

🟨 Bottom-level pattern instances corresponding to external inputs/outputs that have yet to occur. Each of these has not yet been "fixed": The pattern output value will be assigned when the relevant input/output event occurs. A probability is assigned in this process.

🟩 Pattern instances which depend, directly or indirectly on bottom level pattern instances for inputs/outputs that have yet to occur. Each of these has not yet been "fixed": Its pattern output value will be assigned, permanently, when all the pattern instances being used for its inputs have been assigned pattern output values. A probability is assigned in this process.

P Probability value

Figure 11: Statistics Application

4 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.

5 Idealized and Real Hierarchies

The hierarchy described here 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, of course, is as impractical as an idealized Turing machine with its infinitely long instruction tape. 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 with 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.

Even allowing for the use of "forgetting" to reduce the amount of information in the hierarchy, 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.[9] 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.

The purpose of this article is only to describe the basic, idealized system. Ways of achieving these things will be discussed in later articles.

[9] For example, this is 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.

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 that in my previous proposal. In the previously proposed 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.[10] 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

[10] 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.

in the hierarchical system proposed by Hawkins.[11,12] The system being proposed here is an 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.

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

7 Conclusion

An overview has been given of how minds may work and how a hierarchical modeling system might work in an AI system. This article is part of a series, so a number of concepts have been left out of this article, to be discussed later.

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.

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. 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.

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.

A number of issues have not been discussed in any detail. An idealized hierarchy of pattern instances has been described, with no limit on how far pattern instances may extend into the past and the future. A real system would need a way of limiting the number of pattern instances, and a process to deal with this, analogous to “forgetting” in human brains, will be discussed later. It is inefficient to represent all pattern instances explicitly, and further ways of dealing with this issue will be described later. Also to be discussed later is the way that the hierarchy is used to plan actions and provide intelligent behavior, rather than just prediction.

I also 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. This has just been the start. With the later articles, this overview of a system providing a very general ontology should become apparent.

8 Acknowledgements

I wish to thank Norm Nason (Machines Like Us, http://www.machineslikeus.com/) for reviewing this article. Any errors are my own.

9 Bibliography

Almond, P. (2006). *A Proposal for General AI Modeling*. Retrieved 10 April 2009 from http://www.paul-almond.com/Modeling.pdf. (Also available at http://www.paul-almond.com/Modeling.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.