Procedural Memory Construction

Akshay Padmanabha, Houston High School, Germantown, TN 38139

1 Introduction

This document describes the procedural memory component of LIDA as it has actually been constructed. This is in contrast to documentation on conceptual models and design documents for planned builds.

2 Notational Conventions

This paper describes an implementation in Java. Noun and noun phrases that have each word starting with a capital letter correspond directly to an implemented Java class e.g. Behavior Scheme.

3 Related Components

Procedural memory does not work in isolation – it relies on a perceptual component that identifies objects in the environment and internal feelings of the agent. The perceptual memory component makes these objects available in the conscious broadcast.

In this implementation of procedural memory, the conscious broadcast is assumed to contain a set of objects together with the properties of these objects. Both objects and their properties are represented as Nodes. The Nodes have "is-a-feature-of" links between them that represents the fact that one Node is a feature of another Node. A Node has many properties, but the only ones used by procedural memory are:

- An id
- The current activation.
- Whether the node is an object or a category
- Whether the node represents an inherent feeling of the agent and how important that feeling is compared to other feelings

Note that relationships between objects such as "cup on a table" are required, but are not supported in the perceptual memory component that is currently available. [SF1]

3.1 Cognitive Cycle

Processing in LIDA happens in cognitive cycles. From the point of view of procedural memory this means that procedural memory will execute once per cycle and on each cycle will receive an updated conscious broadcast.

3.2 Nodes and Node Ids

The contents of the conscious broadcast, may, in different cognitive cycles, use different Nodes to represent the same concept. For example, suppose in two different cognitive cycles the conscious broadcast contains a small, red, pencil. Small, red and pencil will all be represented by Nodes, but in two different cognitive cycles they may be different Nodes. What will be constant is the node id. So the Node for small may be node-1 at cycle-1 and node-2 at cycle-2, but both node-1 and node2 will have the same node id.

3.3 Objects and Categories

To continue the example of the small, red pencil, the representation of this concept will actually require 4 nodes, 3 for small, red and pencil and one for the actual object that has these properties. This fourth object will be an object Node whereas the others will be category Nodes. There will be “is-a-feature-of” relationships between the actual object and the 3 categories that will represent that the actual object has these properties, or, in other words, belongs to these categories.

3.4 Procedural Memory View of Conscious Broadcast

It has been assumed for purposes of procedural memory that the contents of consciousness will contain enough information to be reorganized into a Conscious Broadcast Procedure View that has been found to be most useful for procedural memory. The Conscious Broadcast Procedure View contains a set of Conscious Broadcast Objects. Each Conscious Broadcast Object contains a single Node that is either an object node or a feeling node. The Conscious Broadcast Object also optionally contains a set of nodes that are properties of the object – this set will be empty if the object node is a feeling.

3.5 Object Properties Spread over Many Conscious Broadcast Cycles

The procedural memory component needs to cope with the fact that the presence of objects in the environment and their properties may be spread across multiple cognitive cycles. For example, in cognitive cycle 1, the Conscious Broadcast Procedure View could contain:

Conscious Broadcast Object 1 = {object1 {blue, hot}}
Conscious Broadcast Object 2 = {object2 {red}}

And in cognitive cycle 2

Conscious Broadcast Object 3 = {object3 {small}}
Conscious Broadcast Object 4 = {object4 {pencil}}

As described above, it may be recognized that Conscious Broadcast Object 1 and Conscious Broadcast Object 3 refer to the same object if object1 and object 3 have the same node id.

At this point the procedural memory system may be referencing three objects:

Conscious Broadcast Object 1 = {object1 {blue, hot, small}}
Conscious Broadcast Object 2 = {object2 {red}}
Conscious Broadcast Object 4 = {object4 {pencil}}

i.e. object 2 is assumed to still be in the environment although it is no longer in the conscious broadcast and object 1 is assumed to still be blue and hot even though these properties are no longer in the conscious broadcast.

Procedural memory would reference all objects and properties listed above if they have been “bound” to a Behavior (see sections 4.3.2 and 4.3.3).

3.6 Currently No Recognition of Mutually Exclusive Properties

In the example above, where the Conscious Broadcast Procedure View contains

Conscious Broadcast Object 3 = {object3 {small}}

It would be useful to recognize that Conscious Broadcast Object 3 is never going to have the property of being big, since small and big are mutually exclusive. The current implementation of the conscious broadcast does not represent this concept, so it is not currently used in procedural memory.

4 Major Components/Objects

4.1 Behavior Scheme

A Behavior Scheme is a template used to create Behaviors by a process of instantiation (described below). A Behavior Scheme contains the following components that are described in more detail below:

- A unique identifier
- Context Conditions
- Result Conditions
- Arguments
- Conflictor links
- Action
- Curiosity
- Base activation
- Reliability

4.1.1 Unique Identifier

The unique identifier distinguished this Behavior Scheme from all others. It is mainly used to identify the Behavior Scheme that was used to instantiate a Behavior.

4.1.2 Context

The context represents the conditions in the external world that need to be true for the Behavior Scheme to be able to instantiate a Behavior that can be expected to execute successfully. The context is implemented as a set of Context Conditions. The set of Context Conditions are implicitly ‘anded’ together to give the full context.

A Context Condition contains a node id and an indicator on whether this is a negated condition.

For example, if the context of a Behavior Scheme specified that a Behavior needs a small pencil then it would have two Context Conditions

Context Condition 1 = {pencil_node_id, not negated}
Context Condition 2 = {small_node_id, not negated}

The context conditions do not specify the actual object Node required since any small pencil will do. If, however, a particular pencil were required then there would be a Context Condition whose node id contained the node id of an actual pencil object.

4.1.3 Results

The results represent the conditions in the external world that are expected to be true after a Behavior instantiated from the Behavior Scheme is executed. The results are implemented as a set of Result Conditions. A Result Condition contains a node id of the node that the result affects, a size of the expected effect for each cognitive cycle after execution and a history of the actual observed effects of the most recent executions. This history is used when learning the size of the expected effect and is described further in the section 7.

For example, if the result of a Behavior executing is that an object becomes hot in the next cognitive cycle, then it would need one Result Condition containing:

Result Condition 1 = {hot_node_id, size of effect = [0.5]}

If the object slowly became hot over 3 cognitive cycles, then the Result Condition would be something like:

Result Condition 1 = {hot_node_id, size of effect = [0.1, 0.2, 0.1]}

Where 0.1, 0.2 and 0.3 are the effects in each Cognitive Cycle.

Some results are special in that they represent an inherent feeling of the agent. These results are distinguished by referring to them as goal results. These are special in that they directly affect the chance that a Behavior will be selected for execution.

4.1.4 Arguments

Several different Context Conditions may refer to the same object. And this may be the same object that a Result Condition applies to. It will often be the case that a Behavior Scheme does not need a particular object but it does need to be able to represent that one or more Context Conditions and Result Conditions all apply to the same object.
To achieve this, a Behavior Scheme contains an array of arguments. The order of the array is arbitrary but is fixed once the Behavior Scheme is created. Each argument contains a set of references to the Context Conditions and Result Conditions that apply to the same object.
So, when the procedural memory system is trying to match Behavior Schemes to the contents of the conscious broadcast it uses the arguments to specify which Context Conditions and Result Conditions must apply to one actual object.

4.1.5 Conflictor Links

Two Behavior Schemes may instantiate Behaviors that conflict in the sense that executing one of the behaviors makes it less likely that the other behavior can be executed.

A Behavior Scheme stores both conflictors and conflictees.

A conflictor is another Behavior Scheme whose instantiated Behavior will have reduced chance of execution if this Behavior Scheme's instantiated Behavior executes.
A conflictee is the opposite relationship i.e. a conflictee is another Behavior Scheme such that the execution of its instantiated Behavior will reduce the chance of this Behavior Scheme's instantiated Behavior from executing.

A Behavior Scheme keeps detailed information on conflictors and limited information on conflictees. The difference in the information is that conflictor relationships are directly used in determining which Behavior executes whereas the conflictee relationship is only used to keep the conflictor relationship updated during the instantiation process.

If a Behavior Scheme is a conflictor of this Behavior Scheme then this Behavior Scheme must have a Result Condition that will reduce the likelihood of a Context Condition in the conflictor from being true.

For conflictors a Behavior Scheme contains the conflicting Behavior Schemes, and, for each such Behavior Scheme, the pairs of Result Condition/Context Condition that constitute the conflict.

For conflictees a Behavior Scheme contains a set of the conflictee Behavior Schemes.

4.1.6 Action

Since a Behavior Scheme is a template used to create Behaviors and a Behavior is a rule for controlling actions (see below), then the "action" component represents this action.
The action is an intrinsic ability of the agent. For example, the action might be "move to the north".
Behavior Scheme and Behaviors are basically rules for when to execute an action. Currently the action is held as a name (e.g. "MOVE_N") which will be returned by the Action Selection as the action to execute.

4.1.7 Base Activation

The base activation represents how useful the Behavior Scheme is in affecting the internal feelings of the agent. For example, a Behavior Scheme that represents the action of eating when food is available would have an affect on the internal feelings of the agent (it would reduce hunger). On the other hand a Behavior Scheme that represents the action of moving towards perceived food would not have an affect on the internal feelings of the agent.
Base activation is used to set the initial value of the activation of a Behavior when it is instantiated (see below).
Base activation is implemented as a sigmoid function as described in section 6.3 of "A Cognitive Architecture Capable of Human Like Learning".

4.1.8 Curiosity

The curiosity represents how curious the agent is about a Behavior Scheme. It is used to drive Behavior selection during learning. When a new Behavior Scheme is first

proposed it has a fixed curiosity assigned to it. As the agent tries out the Behavior Scheme (by selecting a Behavior instantiated from the Behavior Scheme) the curiosity is reduced (more sophisticated updating of curiosity has also been tried – see section 7.8.2). Eventually the curiosity will drop to the point where it will no longer influence Behavior selection. By this time the Behavior Scheme will either have proved to reliably be able to produce its expected results or else it will be forgotten.
See section 7 for more details.
Curiosity is implemented as a sigmoid function, similar to base activation.

4.1.9 Reliability

The reliability represents a measure of whether a Behavior Scheme can produce its expected results often enough to be useful and therefore whether it should be used in creating Behavior Stream Schemes or in creating other, more reliable, Behavior Schemes.
The reliability of a Behavior Scheme does not have to be very high for the Behavior Scheme to potentially be useful but it does need to be high enough so that it is very unlikely that the results are produced by chance.
The concept of reliability as used here is similar to that used by Drescher in Made-Up Minds, 1991. However, the concept here is much simpler and involves far less statistic gathering. Drescher's mechanism required keeping statistics for each result, both when the action has been taken and when the action has not been taken. But here an assessment is made on whether the whole Behavior Scheme (more accurately the instantiated Behavior) has been successful. If so the reliability is increased, if not, it is decreased.
Reliability is implemented as the ratio of the number of successful executions to the total number of executions. To be treated as reliable this ratio must exceed a reliability threshold. Also a Behavior Scheme or Behavior Stream Scheme is not treated as reliable unless it has executed some number of times – this is to avoid spurious results after a very low number of executions.

4.2 Behavior Stream Scheme

A Behavior Stream Scheme is a template used to create a set of Behaviors that have successor links by a process of instantiation (described below). A Behavior Stream Scheme contains the following components that are described in more detail below:

- A unique identifier
- Context Conditions
- Result Conditions
- Arguments
- Successor links

Except for successor links, Context Conditions and Result Conditions, these components are the same as those in a Behavior Scheme and are not described further.

4.2.1 Successor Links

Two Behaviors can be related by a successor link, which indicates that executing one of the Behaviors makes it more likely that the other Behavior can be executed. Such links can extend across several Behaviors – for example given three Behaviors b1, b2 and b3, executing b1 may make b2 more likely which in turn makes b3 more likely.

Once such an interaction between Behaviors has been identified it can be recorded in a Behavior Stream Scheme.

Two Behavior Stream Schemes may also be related by a successor link, and this can also be recorded in a "higher level" Behavior Stream Scheme. So, to continue the previous example, suppose the relationship between Behaviors b1, b2 and b3 has been recorded in Behavior Stream Scheme bss1. Suppose further that there are also Behaviors b4 and b5 such that b4 makes b5 more likely and this has been recorded in Behavior Stream Scheme bss2. If b3 makes b4 more likely, then the set of Behaviors in bss1 makes the set of Behaviors in bss2 more likely and this can be recorded in a new Behavior Stream Scheme bss3, which records the relationship between bss1 and bss2. This process can continue to any depth.

To record such relationships, a Behavior Stream Scheme contains a set of nodes, each of which is a Behavior Scheme or a Behavior Stream Scheme and a set of links between the nodes. Each node is a successor or successee of one or more other nodes.

Node 2 is a successor of node 1 if and only if executing the instantiated Behavior(s) of node 1 will increase the chance of node 2's instantiated Behavior(s) executing.

A successee is the opposite relationship i.e. node 1 is a successee of node 2 if and only if node 2 is a successor of node 1. Consequently one relationship is derivable from the other, both are maintained for efficiency.

If node 2 is a successor of node 1, then node 1 must have a Result Condition that will increase the likelihood of a Context Condition in node 2 being true. Each successor and successee link records references to the relevant Result Condition and Context Condition. Note that node 1 may have more than one Result Condition that will increase the likelihood of a Context Condition in node 2. Consequently, there may be more than one successor or successee link between any two given nodes.

4.2.2 Context and Result Conditions

Any Context Condition or Result Condition in a Behavior Stream Scheme will have been derived from a Context Condition or Result Condition in one or more of the Behavior (Stream) Scheme nodes. For example, suppose a Behavior Stream Scheme has a Result Condition of making an object hot. Then some node within the Behavior Stream Scheme must also have a Result Condition of making the object hot. Similarly, if a Context Condition in a Behavior Stream Scheme requires that an object be hot, then some node within the Behavior Stream Scheme must also have a Context Condition that requires the object is hot.
Also, there may be many such nodes – consider again a Context Condition that requires an object to be hot. Although this is a single Context Condition in the Behavior Stream Scheme there may be many nodes within the Behavior Stream Scheme that have this as a Context Condition.

This information is required when instantiating a Behavior Stream Scheme as described below.

To record this information, Context Conditions and Result Conditions used in a Behavior Stream Scheme are enhanced to hold references to the set of nodes and Context Conditions and Result Conditions in the nodes that they were derived from.

The enhanced Context Condition and Result Conditions allow successor and successee links in the Behavior Stream Scheme to be converted to successor and successee links between the lowest level nodes i.e. between Behavior Schemes (not Behavior Stream Schemes). This is used when instantiating a Behavior Stream Scheme.

4.3 Behavior

A Behavior is a rule for controlling actions. Unlike a Behavior Scheme a Behavior can actually be executed, but for that to happen it must be selected for execution. Behaviors compete for execution on the basis on how well their PreConditions and PostConditions match the conscious broadcast. The process of selecting a Behavior for execution is described in more detail below, but it is worth noting that most of the components in a Behavior are to support that selection process.

A Behavior contains the following components that are described in more detail below:

- A behavior scheme identifier
- Preconditions
- Post-Conditions
- Arguments
- Conflictor links
- Activations
- Action

4.3.1 Behavior Identifier

The Behavior Scheme identifier identifies the Behavior Scheme that (directly or indirectly) instantiated this Behavior. It is mainly used to identify those Behaviors that have been instantiated from the same Behavior Scheme, which is useful when checking that a potential new Behavior instantiation is not a duplicate.

4.3.2 Preconditions

Preconditions in a Behavior directly correspond to the context in the Behavior Scheme that instantiated the Behavior and are created from them.

The preconditions represent the conditions in the external world that make successful execution of the Behavior more likely. The preconditions are implemented as a set of PreConditions. The set of PreConditions are implicitly ‘anded’ together to give the full precondition.

A PreCondition contains two NodeBindings and an indicator on whether this is a negated condition.

A NodeBinding represents the relationship between a node id and the actual Node that is the current representation of that node id in the conscious broadcast.

One of the NodeBindings in the PreCondition corresponds to the condition in the Context Condition from which this PreCondition was created. The other NodeBinding corresponds to the actual object in the conscious broadcast that has been found to have that property.

For example, suppose that the context of a Behavior Scheme from which this Behavior was instantiated contained the two Context Conditions

> Context Condition 1 = {pencil_node_id, not negated}
> Context Condition 2 = {small_node_id, not negated}

Suppose further that the conscious broadcast contained three nodes representing that the perception system had noticed my blue pencil. Suppose the nodes are node1, node2 and node3 with ids pencil_node_id, small_node_id and my_blue_pencil_node_id, where node1 and node2 are category nodes and node3 is an object node. The conscious broadcast would also contain links between node1 and node3 and between node2 and node3. Then the Behavior would have two PreConditions:

> PreCondition 1 = {Node Binding 1, Node Binding 3, not negated}
> PreCondition 2 = {Node Binding 2, Node Binding 3, not negated}

where

> NodeBinding 1 = {pencil_node_id, node1}
> NodeBinding 2 = {small_node_id, node2}
> NodeBinding 3 = {my_blue_pencil_node_id, node3}

4.3.3 Post-conditions

Post-conditions in a Behavior directly correspond to the results in the Behavior Scheme that instantiated the Behavior and are created from them.

The post-conditions represent the conditions in the external world that are expected to be true after a Behavior is executed. The post-conditions are implemented as a set of PostConditions.

A PostCondition contains two NodeBindings and a size of the expected effect in each cognitive cycle

As for a PreCondition, the two NodeBindings correspond to a property and an actual object.

For example, suppose that the results of a Behavior Scheme from which this Behavior was instantiated contained the one Result Condition:

> Result Condition 1 = {hot_node_id, size of effect = [0.5]}

Suppose further that the conscious broadcast contained two nodes, one representing an actual object and the other its degree of 'hotness'. Suppose the nodes are node1 and node2 with object_node_id, hot_node_id. Then if the Behavior has been instantiated so that this is the object that it will affect, then the Behavior would have one PostCondition:

> PostCondition 1 = {Node Binding 1, Node Binding 2, size of effect = [0.5]}

where

> NodeBinding 1 = {hot_node_id, node2}
> NodeBinding 2 = {object_small_node_id, node1}

Some PostConditions are special in that they represent an inherent feeling of the agent. These PostConditions are distinguished by referring to them as goal PostConditions. These are special in that they directly affect the chance that a Behavior will be selected for execution.

4.3.4 Arguments

Several different PreConditions may refer to the same object. And this may be the same object that a PostCondition applies to. It will often be the case that a Behavior is only partially bound to actual objects and while in this state it needs to be able to represent that one or more PreConditions and PostConditions all apply to the same object.
To achieve this, a Behavior contains an array of arguments. The order of the array is arbitrary but is the same as the order of the arguments in the Behavior Scheme from which it was instantiated. Each argument contains a NodeBinding for an actual object and a set of NodeBindings for the PreConditions and PostConditions that apply to that object. The position of an argument in this array is called the argument index. Basically the ith argument contains information about the ith object the Behavior needs. A NodeBinding may not yet be actually bound, in which case it will contain a node id but the actual node will be null.

So, when the procedural memory system is trying to match Behaviors to the contents of the conscious broadcast it uses the arguments to specify which PreConditions and PostConditions must apply to one actual object.

For example, suppose a Behavior has PreConditions that require a small pencil and an eraser.

It may have

> PreCondition 1 = {Node Binding 1, Node Binding 3, not negated}
> PreCondition 2 = {Node Binding 2, Node Binding 3, not negated}
> PreCondition 3 = {Node Binding 4, Node Binding 5, not negated}

where

NodeBinding 1 = {pencil_node_id, node1}
NodeBinding 2 = {small_node_id, node2}
NodeBinding 3 = {my_blue_pencil_node_id, node3}
NodeBinding 4 = {eraser_node_id, node4}
NodeBinding 5 = {my_white_eraser_node_id, node5}

Now NodeBindings 1 and 2 are bindings for two PreConditions that must apply to the same object. NodeBinding 3 is the NodeBinding for an actual object that has properties specified by NodeBindings 1 and 2. NodeBinding 4 is the binding for a PreCondition that must be a different object to that used for NodeBindings 1 and 2. Finally, NodeBinding 5 is the NodeBinding for an actual object that has properties specified by NodeBindings 4.

In this case, the behavior would have two arguments:

argument 1 = {object = NodeBinding 3,
categories = { NodeBinding 1, NodeBinding 2}}

argument 2 = {object = NodeBinding 5, categories = { NodeBinding 4}}

These arguments would exist in this form even if some of the NodeBindings have not yet actually been bound to an actual object. So, for example, if the pencil has been bound but the eraser has not, the situation would be:

NodeBinding 1 = {pencil_node_id, node1}
NodeBinding 2 = {small_node_id, node2}
NodeBinding 3 = {my_blue_pencil_node_id, node3}
NodeBinding 4 = {eraser_node_id, null}

argument 1 = {object = NodeBinding 3,
categories = { NodeBinding 1, NodeBinding 2}}

argument 2 = {object = null, categories = { NodeBinding 4}}

4.3.5 Conflictor Links

Two Behaviors may conflict in the sense that executing one of the behaviors makes it less likely that the other behavior can be executed.
A Behavior stores both conflictors and conflictees.

A conflictor is another Behavior that will have reduced chance of execution if this Behavior executes.
A conflictee is the opposite relationship i.e. a conflictee is another Behavior such that its execution will reduce the chance of this Behavior from executing.

A Behavior keeps detailed information on conflictors and limited information on conflictees. The difference in the information is that conflictor relationships are directly used in determining which Behavior executes whereas the conflictee relationship is only used to keep the conflictor relationship updated during the instantiation process.

If a Behavior is a conflictor of this Behavior then this Behavior must have a PostCondition that will reduce the likelihood of a PreCondition in the conflictor from being true.

For conflictors a Behavior contains the conflicting Behavior, and, for each such Behavior, the pairs of PostCondition/PreCondition that constitute the conflict.

When a Behavior is instantiated from a Behavior Scheme, the conflictor information in a Behavior cannot be directly created from the conflictor information if a Behavior Scheme because:

- Firstly, if two Behaviors b1 and b2 have been instantiated from two Behavior Schemes bs1 and bs2 and bs2 is a conflictor of bs1, then b2 will not necessarily be a conflictor of b1. This may happen because the PostCondition in b1 refers to a different object than the conflicting PreCondition in b2. This is not counted as a conflict.
- Secondly, as will be described below, a Behavior may be partially instantiated i.e. some of its PreConditions and PostConditions have been instantiated and some have not. So if the PostCondition in b1 or the PreCondition in b2 has not yet been instantiated, there is not a conflict between b1 and b2.
- Thirdly, b1 may be instantiated before b2 is, i.e. b1 may exist when b2 does not exist. Again in this case there is clearly not a conflict between b1 and b2

The conflictor information that a Behavior can obtain from its instantiating Behavior Scheme still needs to be recorded, as, in the second and third case above, a conflict may arise when either b1 or b2 are more fully instantiated or b2 is created.

To handle this, a Behavior also holds potential conflictor information. This is similar to the actual conflictor information except the behavior scheme id of the conflicting Behavior is recorded instead of the actual Behavior (since this may not exist yet) and a NodeBindingLocator is used instead of a PreCondition. A NodeBindingLocator is enough information to find a NodeBinding and consequently find a PreCondition that uses the NodeBinding (as its condition binding not its actual object binding). This information is the node id and argument index. Potential conflictor information does not affect the chance of a behavior being selected for execution – they are used to build actual conflictor information as new Behaviors and new instantiations take place.

For example, suppose a Behavior b1 has a PostCondition that reduces the "hotness" of an object and this is a conflictor with Behavior b2 that requires a hot object. If Behaviors b1 and b2 both exist and have been instantiated, then the situations could be:

b1 has

PostCondition 1 = {Node Binding 1, Node Binding 2, size of effect = 0.5}

where

NodeBinding 1 = {hot_node_id, node2}
NodeBinding 2 = {object_node_id, node1}

and
node1 is an object node and node2 is a “hotness” node related to node1

b2 has
PreCondition 1 = {Node Binding 3, Node Binding 4, not negated}
where

NodeBinding 3 = {hot_node_id, node2}
NodeBinding 4 = {object_node_id, node1}

In this case there is a conflict between b1 and b2 and b1 will hold conflictor information as follows:

Conflictor 1 = {b2, {PostCondition 1, PreCondition 2}}

But now suppose that b2 has been created but has not been instantiated with node1 (how this can happen will be described below). This is the second situation above. In this case b1 does not have b2 as an actual conflictor, but Behaviors created from b2 by further instantiation may be. In this case b1 will hold the following potential conflictor information:

Potential Conflictor 1 = {b2_id, {hot_node_id, NodeBindingLocator 1}}

where

NodeBindingLocator 1 = {hot_node_id, i}

where i is the argument index of PreCondition 1 in b2’s argument array. This NodeBindingLocator is valid even if b2 does not yet exist (i.e. the third situation described above).

For conflictees a Behavior contains a set of the ids of conflictee Behaviors. Holding the ids means that a Behavior can hold this information even if actual Behaviors with these ids do not actually exist yet.

To continue the example above, b2 would hold conflictee information as follows:

Conflictee 1 = {b2i_d}

4.3.6 Successor Links

A Behavior may have a successor in the sense that executing this Behavior makes it more likely that the other Behavior can be executed.
A Behavior stores both successors and successees.

A successee is the opposite relationship i.e. a successee is another Behavior such that its execution will increase the chance of this Behavior executing.

The successor and successee information is an exact copy of the conflictor and conflictee information and consequently is not described separately.

4.3.7 Activations

The activation of a Behavior measures the extent to which the Behavior can contribute (directly or indirectly) to the current goals of the agent as presented in the conscious broadcast. The activation does not include how executable a Behavior is (i.e. how well its PreConditions match the conscious broadcast) – this is not stored but calculated when required in the Action Selection process described below. A Behavior does not hold a single activation, but the following:

- Current goal activations
- Previous goal activations
- Current activation
- Previous activation
- Activation

These various activations are required by Behavior selection, described in section 6.

4.3.7.1 Current Goal Activations

This records the amount of activation that the Behavior has from each feeling node that has recently been in the conscious broadcast.

For example if the conscious broadcast has recently contained nodes for the agent's battery level and temperature, goal activations could be

{{temperature_node_id, 0.5}, {battery_node_id, 0.3}}

4.3.7.2 Previous Goal Activations

This is the contents of the Current Goal Activations in the previous cognitive cycle.

4.3.7.3 Current Activation

This is the sum of the current goal activations.

4.3.7.4 Previous Activation

This is the current activation in the previous cognitive cycle.

4.3.7.5 Activation

This is the sum of the current activation and the decayed value of the previous activation. So even if the current activation is zero, a Behavior still keeps a decaying amount of activation.

4.3.8 Action

Since a Behavior is a rule for controlling actions, then the "action" component represents this action.
The action is an intrinsic ability of the agent. For example, the action might be "move to the north".

Behaviors are basically rules for when to execute an action.
Currently the action is held as a name (e.g. “MOVE_N”) which will be returned by the Action Selection as the action to execute.

4.4 Behavior Stream

As discussed below, whenever a Behavior Scheme matches, however partially, a Conscious Broadcast Object, then the Behavior Scheme is instantiated to produce a Behavior.

A similar process could be used to instantiate a Behavior Stream Scheme whenever it partially matched a Conscious Broadcast Object. However, in this case instantiation may involve the creation of many, possibly irrelevant Behaviors which would significantly increase the administration costs of managing Behaviors (for example Behavior Selection).

So, instead of directly instantiating Behaviors, a partially matched Behavior Stream Scheme results in the creation of a Behavior Stream. A Behavior Stream is similar to a Behavior except that it doesn’t have any successor or conflictor links and consequently has a simpler algorithm to calculate activation.

The activation of the Behavior Stream is used to calculate the relevance of the Behavior Stream Scheme from which it was instantiated over several cognitive cycles. A Behavior Stream Scheme cannot do this directly as it has no data structures to record matches between its arguments and Conscious Broadcast Objects.

Just like a Behavior, a Behavior Stream can become more fully instantiated over time (as different objects enter the Conscious Broadcast Procedure View) and this may increase the activation of the Behavior Stream. If a Behavior Stream’s activation reaches a threshold, then the Behavior Stream is discarded and the Behavior Stream Scheme from which it was instantiated is used to instantiate all Behavior Schemes in the Behavior Stream Scheme.

Note that this process of monitoring the activation of a Behavior Stream Scheme until it reaches a threshold was always part of the LIDA conceptual model.

A Behavior Stream Scheme contains the following components:

- A behavior stream scheme identifier
- Preconditions
- Post-Conditions
- Arguments
- Activations
- Action

All of these components are the same as those in a Behavior and are not described further.

5 Major Processes

In one cognitive cycle, procedure memory does the following:

1. Update Behaviors including Behavior Streams
2. Instantiate Behaviors and Behavior Streams from Behavior Schemes and Behavior Stream Schemes
3. Create new Behavior Streams from existing Behavior Streams
4. Instantiate Behavior Stream Schemes which have Behavior Streams above threshold
5. Create new Behaviors from existing Behaviors
6. Behavior Selection

The reasons for choosing this order are:

- Step 1 should be done before step 2 because otherwise the sets of instantiated Behaviors and Behavior Streams will have new members that do not need to be updated
- Step 3 should be done after step 2 as otherwise the set of instantiated Behavior Streams will be missing those created in step 2
- Step 4 should be done after step 3 as otherwise the set of instantiated Behavior Streams will be missing those created in step 3
- Step 5 should be done after step 4 as otherwise the set of instantiated Behavior will be missing those created in step 4

Note that this does not include learning and is referred to as “normal” mode in section 7.3.

5.1 Update Behaviors including Behavior Streams

As mentioned in section 3.2, Nodes that represent a concept change. Behaviors and Behavior Streams are updated with the latest Nodes so that Node properties such as current activation can be given their current value. The update is performed as follows:

```
for every Behavior or Behavior Stream b
    for every argument arg of b
        for every Conscious Broadcast Object cbo
            if node id of arg’s NodeBinding for actual object
            = node id cbo object
                for every Context Condition/Result Condition
                con in arg
                    for every property of the cbo prop
                        if (node id of con = node id of
                        prop)
                            update NodeBinding for
                            con to be bound to prop
```

Note that the NodeBinding for con may have not previously been bound to any Node, so this creates new property bindings as well as updating existing ones.

5.2 *Instantiate Behaviors and Behavior Streams from Behavior Schemes and Behavior Stream Schemes*

It was mentioned above that a Behavior Scheme is a template for creating Behaviors. This section describes how the instantiation happens. It first describes the process of instantiating Behaviors and then describes the differences for instantiating Behavior Stream Schemes.

5.2.1 Overview of Instantiation

Instantiation is the process of identifying objects in the conscious broadcast that are a match for the arguments of the Behavior Scheme. When a match is found, a Behavior can be created from the Behavior Scheme. As described above the arguments of a Behavior (as distinct from those of a Behavior Scheme) contain NodeBindings – binding an actual Node to one that, in the Behavior Scheme, is only identified by a node id.

5.2.2 Matching Process

The matching part of the process can be described by the following algorithm:

```
for every Behavior Scheme bs
    for every argument of bs
        for every Context Condition/Result Condition con in argument
            for every Conscious Broadcast Object cbo
                for every property of the cbo prop
                    if (node id of con = node id of prop)
                        //we have a match on a property
                if (node id of con = node id cbo object)
                    //we have a match on cbo object
```

Some explanation is required for the comments "we have a match on a property" and "we have a match on a cbo object".
As described above, the conscious broadcast is presented as a Conscious Broadcast Procedure View containing Conscious Broadcast Objects. Each Conscious Broadcast Object contains one object or feeling Node and zero or many category Nodes. If a Behavior Scheme has a Context Condition that requires an object in a certain category (e.g. a pencil) then it will match to a category Node in the Conscious Broadcast Object. But if a Behavior Scheme has a Context Condition that requires a specific object (e.g. my blue pencil) then it will match to the object Node in the Conscious Broadcast Object.

Once a match is found a check is made to see if there is already a Behavior, instantiated from the same Behavior Scheme (i.e. having the same behavior scheme identifier) with the same Conscious Broadcast Object bound to the same argument. If not then a new Behavior will be created from the Behavior Scheme by the process of instantiation described below. This Behavior will have only one argument bound as is described in the next section.

Note that the matching algorithm only requires one property of the Conscious Broadcast Object to match one Context Condition or Result Condition of the Behavior Schemes argument.

For example if an argument refers to Context Conditions:

Context Condition 1 = {pencil_node_id, not negated}
Context Condition 2 = {small_node_id, not negated}

Then, if the Conscious Broadcast Procedure View contains an object that has the property of being a pencil, a match will be found regardless of whether the object has been identified as small or not. The reason for this is that a subsequent cognitive cycle may contain the property that this object is small but no longer contains the property of being a pencil. So the procedural memory system needs to remember the match on pencil, and it does this by creating a Behavior bound to the pencil property.

5.2.3 Reason for Initially Binding Only One Argument

A given Behavior Scheme may have many arguments and there may be many Conscious Broadcast Objects in the Conscious Broadcast Procedure View.
So, once the first match has been found it seems desirable to keep trying to match other arguments to other Conscious Broadcast Objects, and then create a Behavior with as many arguments as possible matched. This is not done, each Behavior that is created by the instantiation process only has a single argument bound. This is explained in this section.

Suppose a Behavior Scheme contains 3 arguments arg1, arg2 and arg3. Also suppose that the Conscious Broadcast Procedure contains 3 objects o1, o2 and o3. Suppose that there is a match between arg1 and o1 and arg2 and o2, and we create a Behavior, b1, with its first two arguments bound to o1 and o2. arg3 is unbound.

In the next cognitive cycle the Conscious Broadcast Procedure contains 2 new objects, objects o4 and o5 – o1, o2 and o3 are no longer in the conscious broadcast.

It may well be the case that a Behavior that binds its three arguments to o1, o4 and o5 is a better match than any binding using o1 and o2. However a binding to o1, o4 and o5 cannot be created from a Behavior Scheme since a Behavior Scheme never gets bound and has no record of objects. We cannot create a Behavior binding to o1, o4 and o5 from a Behavior already bound to o1 and o2 as the second argument is already bound.

Basically, by only creating a Behavior bound to o1 and o2, the procedural memory system has over committed too early.

So, what actually happens is that, given the match between arg1 and o1 and arg2 and o2, two Behaviors are created, b1 with arg1 bound to o1 and b2 with arg2 bound to o2. As will be described soon, partly bound Behaviors are used to create new Behaviors that are more fully bound. So either b1 or b2 could be used be create b3 that has arg1 bound to o1 and arg2 bound to o2. More importantly, b1 can be used, in the next cognitive cycle, to create b4 that has arg1 bound to o1 and arg2 bound to o4 and from b4 can be created b5 that has arg1 bound to o1, arg2 bound to o4 and arg3 bound to o5. So, whenever a match is found, either between a Behavior Scheme and the conscious broadcast or between a partly bound Behavior and the conscious broadcast, new Behaviors are created that have only one additional argument bound.

These arguments will themselves be matched and further Behaviors created that are more fully bound.

5.2.4 Behavior Instantiation

Behavior instantiation is the process of creating a Behavior from a Behavior Scheme. Since there is such a close correspondence between Behavior Scheme components and Behavior components, this is a straight forward task. The only points to note are:

- Only one argument is bound as described above
- This single argument is bound as much as possible i.e. all NodeBindings in one argument are bound to an actual Node if a matching one exists in the Conscious Broadcast Object
- Only the potential conflictors are created and any other Behaviors that need to add this Behavior as a conflictor are not updated in this process. Instead, once the Behavior is created it is told to set up actual conflictors and to notify conflictees – this is described in the next sections
- The initial value of the Behavior's activation is set from the base activation of the Behavior Scheme

5.2.5 Set Up Actual Conflictors

The actual conflictors are set up as follows:

```
for all behavior scheme ids in the potential conflictors
    for all actual Behaviors with this behavior scheme id
        for each conflicting PostCondition post of this Behavior
            find the PreCondition pre in the potential conflictor
            Behavior using the NodeBindingLocator
            if the pre and post are both bound to the same actual
            object
                create a conflictor between post and pre
```

5.2.6 Notify Conflictees

All Behaviors that have behavior scheme ids specified in the conflictees are notified. They use the same check as described in the previous section to see if the conflict is real. If so, they update their conflictors.

5.2.7 Instantiating Behavior Stream Schemes

This is the same as instantiating Behaviors except that no conflictor/conflictee actions are taken, as Behavior Stream Schemes do not use these relationships.

5.3 Create new Behavior Streams from existing Behavior Streams

This process is the same as Create New Behaviors from Existing Behaviors described below but applied to instantiated Behavior Streams instead of instantiated Behaviors.

5.4 Instantiate Behavior Stream Schemes which have Behavior Streams above threshold

It was mentioned above that a Behavior Stream Scheme is a template used to create a set of Behaviors. This section describes how the instantiation happens.

5.4.1 Overview of Instantiation

First the set of Behavior Streams that are above threshold is determined. Then, for each such Behavior Stream, the Behavior Stream Scheme from which it was instantiated is determined. The Behavior Stream plays no further part in the instantiation. Each node in the Behavior Stream Scheme is then examined. If it is a Behavior Scheme it is instantiated in a similar way to instantiating a “normal” Behavior Scheme. If it is a Behavior Stream Scheme then it is processed recursively. The main complication in this procedure is generating all the successor and successee links for each instantiated Behavior, since these must include any relevant successor/successee information recorded between Behavior Stream Schemes. To find these links the links between any nodes in the Behavior Stream Scheme are converted to links between Behavior Schemes (i.e. not Behavior Stream Schemes). Initially, sets which will contain these links between Behavior Schemes, are initialised as empty sets.

5.4.2 Find Set of Behavior Streams above Threshold

The activation of Behavior Streams is calculated in the same way as Behaviors except that no successor or conflictor links are taken into account.

5.4.3 Process Each Node in Behavior Stream Scheme

Each node in the Behavior Stream Scheme is examined to see if it is a Behavior Scheme or another Behavior Stream Scheme. These two cases are handled separately as described in following sections.

5.4.4 Node is Behavior Stream Scheme

First convert all successor and successee links between this node and any other node into successor and successee links between Behavior Schemes.
Each successor and success link is actually a link between a Result Condition at one end and a Context Condition at the other end. Process each link involving this Behavior Stream Scheme as follows:

- If the Context Condition is an enhanced Context Condition, as described in section 4.2.2, then this Context Condition holds references to a set of nodes and Context Conditions where the nodes are the nodes in the Behavior Stream Scheme being processed (i.e. nodes one layer lower). Replace this link with a set of links derived from this link. Each link in the set has the same Result Condition as the link being replaced and has a Context Condition from the set referenced by the original Context Condition. So now the original links have been replaced by ones that have the same Result Conditions but Context Conditions are from the nodes one layer down.
- Process Result Conditions similarly so that now the original links have been replaced by ones in which both the Result Conditions and Context Conditions have been replaced by ones that are from nodes one layer down

- continue processing in this way until all links are between Result Conditions and Context Conditions on Behavior Schemes
- add the set of successor and successee links generated from this Behavior Stream Scheme node to the set of links between Behavior Schemes for the Behavior Stream Scheme being instantiated

5.4.5 Node is Behavior Scheme

This node is a Behavior Scheme, but it may have successor and successee links to other nodes that are Behavior Stream Schemes. Replace these with links only involving Behavior Schemes as described above for Behavior Stream Schemes and add them to the set of links between Behavior Schemes for the Behavior Stream Scheme being instantiated.

Extract from the set of links between Behavior Schemes for the Behavior Stream Scheme being instantiated all links that involve this Behavior Scheme.

The Behavior Scheme is then instantiated (i.e. a Behavior is created) in the same way as described above except that:

- the Behavior may already exist since it may have been instantiated from a Behavior Scheme. In this case the successor and successee links are added to the Behavior
- successor and successee links are added, in the same way as conflictor and conflictee links as described previously (including that initially only potential successors are created and the Behavior is told to set up actual successors and to notify successes)
- arguments are not bound to any ConsciousBroadcastObject – this will happen in the next step of the Cognitive Cycle. Note that some of the Behaviors may not yet have any match to the Conscious Broadcast – but they still need to be instantiated so that they can pass activation along the successor links

5.5 Create New Behaviors from Existing Behaviors

As mentioned above, Behaviors are initially created from Behavior Schemes with only one argument bound. Behaviors are matched to the conscious broadcast to create new, more fully bound, Behaviors.

The process of matching a Behavior to the ConsciousBroadcastObjects and creating new Behaviors from existing ones is very similar to the processes described above for matching Behavior Scheme and creating Behaviors from the Behavior Scheme, so does not need to be described in detail. The only points to note are:

- Each time a new Behavior is created it has just one new argument bound
- As soon as a new Behavior is created, it also becomes a candidate for further matching
- New Behaviors are created from the existing ones by cloning the existing Behavior and then adding the new binding
- As for Behavior Scheme matching, a check is made to ensure the Behavior being created is not a duplicate

- Once the new Behavior is created it both updates its actual conflictors and notifies conflictees.

6 Behavior Selection

Behavior selection is the process of selecting a single Behavior for execution. Behavior selection is based on Dorer's version of Maes's action selection mechanism. This is described in "Behavior Networks for Continuous Domains using Situation-Dependent Motivations" IJCAI 1999.

This has been trivially modified to work with the LIDA representation of the environment and Behaviors. But it has also been modified in some more fundamental ways as is described next.

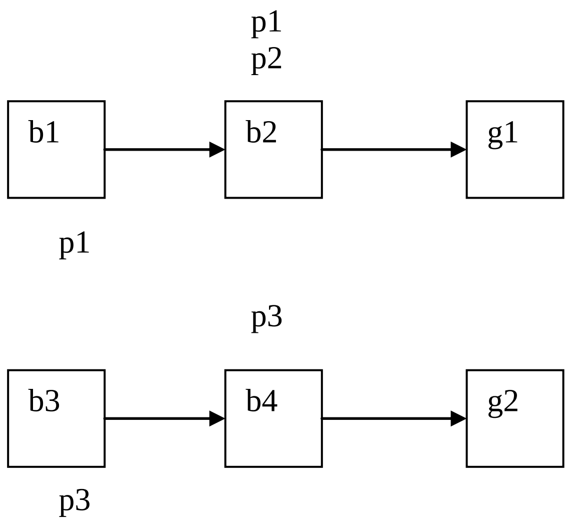
6.1 Modifications to Dorer's Action Selection

6.1.1 Discount on Time

A minor change to Dorer's action selection mechanism is the addition of a discount on time. This discount is used as a new term in Dorer's equations 3 and 4. Such a discount is commonly used in action selection mechanisms to take into account that if two Behaviors are expected to give equal reward but one is expected to take longer to execute, then the one with the shorter expected execution time should be chosen.

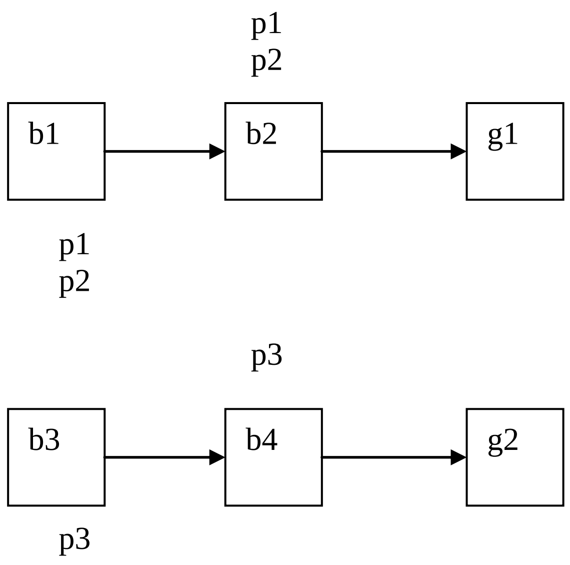
6.1.2 Handling Multiple Successor Links Between Two Behaviors

Suppose we have four Behaviors b1, b2, b3 and b4. b2 is a successor of b1 and b2 achieves a goal. Similarly, b4 is a successor of b3 and b4 achieves a goal. b2 has PreConditions p1 and p2 and b1 has p1 as a PostCondition. b4 has PreConditions p3 and b3 has p3 as a PostCondition. This is shown in the following diagram, which has PreConditions above a Behavior and PostConditions below.



If goals have equal value and all other factors are the same, b3 should be preferred for execution over b1, since it leads directly to b4 being executable, whereas b1 only partially satisfies b2's PreConditions. However, in Dorer's implementation b1 will get the same activation from b2 that b3 gets from b4. This is not fully described in Dorer's paper, but a private communication has confirmed that each successor link between Behaviors passes back activation and then the maximum is chosen.

Now consider a revised situation:



Now b1 can fully satisfy b2's PreConditions so b1 and b3 should be considered equally good candidates for execution.

Further, consider the situation in both scenarios above where p2 is already true. Now b1 and b3 should be considered equally good candidates for execution in both scenarios.

To handle the above situations correctly the following algorithm is used.

Given two Behaviors b1 and b2, b2 a successor of b1 with one or more Post Conditions of b1 satisfying PreConditions of b2, the activation passed back from b2 to b1 is calculated as follows:

1. Add up the maximum increase that could be made to the truth-value of b2's PreConditions. PreConditions that are already true will obviously not contribute
2. Now calculate the effect that each PostCondition/PreCondition link between b1 and b2 could have on the truth-value of b2's PreConditions. This step follows Dorer except that a discount on time is applied as described above
3. Instead of using the maximum from step 2 as the final result, instead sum the results from step 2 and divide by the result of step 1

6.1.3 Choice of Dorer's Optional Functions

Dorer's paper leaves some functions unspecified, sometimes with examples. The following choices have been used:

Function name	**Chosen implementation**
And	p1 and p2 = min(p1, p2)
Or	p1 or p2 = Calculated as not(not(p1) and not(p2))
Transfer function	Equality function i.e. not used – private communication from Dorer confirmed that mechanism worked perfectly ok without a transfer function

It should also be noted that the truth of a PreCondition is currently set to true or false (1 or 0), not a value between 0 and 1 as originally intended.
This was done because of problems with interpreting negated PreConditions. For example, suppose a PreCondition represented that an object should not be hot. Then it would be expressed as a negated PreCondition on an object in Conscious Broadcast with "hot" as a property node. If this node had current activation 0.3 (a measure of the hotness of the object) then the negated PreCondition would base its activation on 1.0 – 0.3 = 0.7.
Now suppose further that another PreCondition represented that an object should be hot. This PreCondition would base its activation on 0.3. Consequently the negated PreCondition is far more "true" than the non negated PreCondition.

In worlds where activation of 0.3 is actually quite high this results in Behaviors that have the negated PreCondition always being preferred to Behaviors with the non-negated PreCondition.

To simplify this situation, any activation that is significant (above a "noise threshold") is considered "true" and activation below that threshold is considered "false".

7 Learning

7.1 Introduction

The learning described in this section has been based on that described in "A Cognitive Architecture Capable of Human Like Learning" by Franklin and D'Mello. However, experience with various implementations has led to some significant differences between the actually constructed implementation and the description in that paper. The most significant such difference is the use of three statistics instead of one, as described in the next section.

7.2 The Need for Three Statistics

7.2.1 Reliability

Section 6.3.1 of "A Cognitive Architecture Capable of Human Like Learning" states that "[t]he base-level activation measures the reliability of the scheme, that is, the likelihood of its result occurring when the action is taken within its context".

Similarly, section 6.2 states “base-level activation that estimates the likelihood of that result occurring as a result of taking the action in its context. “

Thus this paper clearly states the need for a statistic that measure reliability of a Behavior Scheme and this has been confirmed by the implementations. In the implementation, this statistic is referred to as “reliability”, rather than “base activation”. The implementation does also use a statistic called “base activation” for a different purpose as described below.

The reliability statistic is used extensively in the implementation to distinguish between proposed Behavior Schemes that usually achieve something and those that are ineffective and so can be “forgotten”. In the implementation, reliable Behavior Schemes are not only kept but are the basis of creating further Behavior Schemes and Behavior Stream Schemes. This has been an essential ingredient in making the learning task tractable in worlds as complex as Tyrrell’s world using limited computer resources.

Note that the use of reliability here is a much simpler form of the positive-correlation and negative-correlation in section 4.1.2 of Drescher’s Made-Up Minds.

7.2.2 Curiosity

A major difference between “A Cognitive Architecture Capable of Human Like Learning” and the implementation is the use of the “curiosity” statistic. In the implementation this is used to drive exploration. Experience has shown that using reliability to select Behavior Schemes (or rather their instantiated Behaviors) is not effective at making the agent try out new, untried Behavior Schemes. Since these schemes have not been tried, nothing is known about their reliability. They can be given an initial reliability as suggested in “A Cognitive Architecture Capable of Human Like Learning” but this still makes them unable to compete (on the basis of reliability) with existing highly reliable Behavior Schemes. For example, if the agent already has a highly reliable Behavior Scheme:

Move north east: Mate to north → Mate to east

It may never try (using selection on reliability) a new, more useful

Move north: Mate to north → Mate in animal square

So, in the implementation, new behavior schemes are given an initial curiosity factor which decays away as the behavior scheme is tried. After it has been tried a few times, either it will have significant reliability (in which case it will be kept for that reason) or it will have low reliability and low curiosity and it will be forgotten.

Note that the use of curiosity here is intended to achieve the same result as hysteresis and habituation in section 3.4.2 of Drescher’s Made-Up Minds.

7.2.3 Base Activation

Although “A Cognitive Architecture Capable of Human Like Learning” states that “we use only a single statistic, the base-level activation”, it actually seems to use this statistic in two different ways. One way is to record reliability as discussed above.

The other way is to record the affect that a behavior scheme has on the feelings/emotions of the agent. For example, section 6.3 states “In procedural learning, the base level activation of a scheme is updated on a bi-directional basis based on the valence of the affect” and section 2.3, point 6 states that
“The affective content (feelings/emotions) together with the cognitive content, help to attract relevant resources (schemes, processors, neural assemblies) with which to deal with the current situation.” Finally, section 3 states that “This is achieved by each instantiated behavior codelet maintaining the affective response that has traditionally followed its execution as part of its post-conditions. The agent would choose those actions which wouldn’t give too much pain, or which gives the agent satisfaction. Thus a feeling would drive a LIDA controlled software agent or robot in whatever action selection it takes.”

Significant attempts were made during the implementation to record reliability and affect in a single statistic. This one combined statistic was increased when a behavior scheme was successful, decreased when a behavior scheme was unsuccessful and also increased or decreased based on the size and valence of the affect. These attempts achieved some success but using a single statistic makes it impossible to distinguish between a Behavior Scheme that succeeds very rarely but has a large affect and one that succeeds much more often and has a smaller affect. This resulted in significant computer resource being spent exploring Behavior Schemes that succeeded very rarely but have a large affect when it is much more effective to only study those that have shown significant reliability.

In the end it was found very much easier to implement a system that had two statistics, one for reliability and one to record the affect. This later statistic is referred to in the implementation as base activation.

The base activation is very useful in biasing the learning of behavior stream schemes towards those that will not only be reliable but will also achieve a positive affect.

Unlike reliability and curiosity, base activation is not used just during learning. It is passed from a Behavior Scheme to a Behavior during instantiation and so biases the behavior selection towards Behaviors that have high base activation. These Behaviors will also likely get extra activation from goal activation as described above.

7.3 The Need for Learning Mode

The behavior selection process described in section 6 can be considered the “normal” mode of operation. It is intended to achieve the best outcome for the agent, given the Behavior Schemes and Behavior Stream Schemes that it knows. Here, “best outcome” refers to keeping the activation of feeling nodes (i.e. hungers) as low as possible.

Operating in this mode is not effective for learning. As usual in learning systems, there is a tradeoff between maximizing rewards in the short time and exploring other possibilities that may allow better rewards in the future.

The procedural memory implementation describes here has a separate Learning Mode, which has a different action selection algorithm to that used in normal mode. The action selection is aimed as “best learning” rather than “best outcome”. The learning mode has the following differences to normal mode that are described in more detail below:

- Behaviors are chosen by curiosity and executability rather than by goal activation and executability
- A Behavior Stream Scheme can be explicitly chosen (rather than just chosen for instantiation), in which case only Behaviors from that Behavior Stream are instantiated so that the results of executing the Behavior Stream can be seen in isolation
- The agent remains inactive for a period after each action selection so as to see the full, possibly delayed, results of the action
- After the inactive period, actual and expected results are compared and expected results updated
- New Behavior Schemes are created by specialization
- New Behavior Schemes are created by generalization
- New Behavior Stream Schemes are created
- New conflictor links are created

A separate mode for learning is not mentioned in “A Cognitive Architecture Capable of Human Like Learning”. Instead this paper seems to assume that learning be part of the normal behavior of the agent. This would be desirable, but was found difficult to achieve in the implementation because:

- It was found difficult to come up with an effective action selection algorithm that was a compromise between “best outcome” and “best learning”. Some investigation was taken into allowing the agent to switch back and forward between normal mode and learning mode based on the current activation of feeling nodes (i.e. how serious were the various “hungers”) but this idea has not been included in the current implementation
- Learning is difficult when running in normal mode because it is difficult to assign effects to a single behavior. For example, suppose Behaviors b1 and b2 are chosen in succession. If b1 has an effect 3 cognitive cycles later, then this will be after b2 has run, so it is difficult to decide whether to assign the effect to b1 or b2. In contrast, when humans are learning new Behaviors they will often try an action and then “see what happens” (i.e. wait a while before trying anything else). The learning mode simulates this “see what happens” approach.
- Similarly, when running in normal mode, it is difficult to determine whether a behavior has been successful. Returning to b1 and b2 run in succession, if b1 is expected to have an effect but b2 reverses that effect, then it is difficult to determine whether b1 has been successful or not
- The situations for Behavior Stream Schemes are even harder in normal mode. In this case two Behavior Stream Schemes bss1 and bss2 may have enough activation to instantiate a behavior b (i.e. b is the instantiated Behavior of a Behavior Scheme bs that is in both bss1 and bss2). Even if the effects could be attributed to b, it is not clear whether the credit should go to bss1 or bss2. Various ideas were considered for how to determine that a Behavior Stream Scheme is running successfully (for example by the execution of the terminal nodes in the successors graph or by the execution of all nodes in a path

through the successor graph) but in the end this problem was solved by only instantiating the Behaviors from a single Behavior Stream Scheme when in learning mode. Again, this seems more similar to what humans would do.

Note that Dresher in Made-Up Minds also required a learning mode in which “the mechanism alternates between emphasizing goal-pursuit criterion for a time, then emphasizing exploration-criterion”.

7.4 Behavior Selection in Learning Mode

When running in learning mode, Behaviors are chosen by a combination of curiosity and executability.

Behavior Schemes and Behavior Stream Schemes are matched against the Conscious Broadcast Procedure View (i.e. the Conscious Broadcast Objects) and instantiated as described in sections 5.1, 5.2, 5.3 and 5.5 above. Section 5.4 (which consists of instantiating all Behavior Stream Schemes above a threshold) does not happen unless the Behavior Stream Scheme is chosen for execution (see below).

Then, for each instantiated Behaviors and Behavior Stream, instead of combining the activation with executability (as described in Dorer’s work) the executability is combined with the curiosity. The Behavior or Behavior Stream with the highest such combination is chosen to run.

If a Behavior Stream is chosen to run then it is instantiated as described in section 5.4 above, and only Behaviors from that Behavior Stream are instantiated so that the results of executing the Behavior Stream can be seen in isolation. One of the instantiated Behaviors from the Behavior Stream Scheme is then chosen for execution in the normal way. Once this first Behavior is executed, Behaviors from the Behavior Stream Scheme continue to be selected in the normal way until either non are executable or else enough cognitive cycles have elapsed to run every Behavior instantiated from the Behavior Stream Scheme.

Thus, the selection process is very similar to that in normal mode, but by basing it on curiosity all Behavior Schemes and Behavior Stream Schemes get “tried out” at least a few times (the curiosity is reduced each time a Behavior Scheme or Behavior Stream Scheme is selected).

If no Behaviors or Behavior Streams have significant combined curiosity and executability then a fallback mechanism is required. Two have been tried: to revert to normal selection and to choose one of the intrinsic Behavior Schemes that will help exploration at random (in Tyrrell’s world this is the various MOVE Behavior Schemes). This situation seldom occurs unless the learning is run for a very long time (several days on current computer resources) and is not a significant part of this research. .

This simple scheme has been enhanced in three ways:

- Biasing selection to simpler Behavior Schemes
- Biasing selection to Behavior Schemes that have not yet had enough executions to establish reliability

- Ensuring all intrinsic Behaviors are tried in new situations

7.4.1 Biasing selection to simpler Behavior Schemes

In some cases, instead of using the curiosity straight from the Behavior Scheme, it is modified before use.
One such case is to bias the selection towards more simple Behavior Schemes. A "simplicity factor" is calculated from the number of Context Conditions and Result Conditions that the Behavior Scheme has, whether the Context Conditions are negated and how many intervening Behavior Schemes were required to create this one from the original intrinsic Behavior Scheme (referred to as the "distance" from the intrinsic). This simplicity factor is used to modify the curiosity before calculating the combined curiosity and executability.

This ensures that the agent tries out simple Behavior Schemes in preference to more complex variations.

7.4.2 Biasing selection to Behaviors yet to achieve reliability

As mentioned in section 4.1.9 a Behavior is not treated as reliable unless it has executed some number of times.
The curiosity is also modified to give preference to Behaviors yet to achieve this number of executions.
This is important so that their reliability can be established (which is needed for further learning as described below)

7.4.3 Ensuring all intrinsic Behaviors are tried

The curiosity is also modified to ensure all intrinsic Behaviors are tried when new situations are encountered.

To understand the need for this, consider the agent encountering a relatively rare event such as perceiving a potential mate in close proximity (represented as a Conscious Broadcast Object in the Conscious Broadcast Procedure View).
By this time the agent may already have learned many new Behavior Schemes as well as the initial intrinsic Behavior Schemes that it starts with.

All the intrinsic Behavior Schemes will already have likely been chosen many times and will now have low curiosity, so it is unlikely that any of them is chosen. But even if an intrinsic Behavior Scheme were somehow chosen at random each time a potential mate in close proximity is encountered, it may be a very long time before the agent tries out the "courting" intrinsic Behavior Scheme. Consequently, it may be a very long time before the agent realises the significance of some objects, as it has never tried the appropriate Behavior Scheme.

To reduce learning time, the agent counts the number of times it has selected each intrinsic Behavior Scheme for each distinct Conscious Broadcast Object. Conscious Broadcast Objects are considered distinct if they have different objects or have different properties (see section 3.4). If this count is lower than a threshold then the curiosity for the intrinsic Behavior Scheme is modified to be the initial curiosity. This will make the intrinsic Behavior Scheme more likely to be selected than Behavior Schemes that have been tried before.

In the current testing that has been done, this enhancement has proven extremely effective at reducing learning time and takes up only a trivial amount of memory (compared to the Behavior Schemes). It may not work so well in environments where there are a very large number of distinct Conscious Broadcast Objects. But it does seem that some similar mechanism will be required to ensure that all intrinsic Behavior Schemes are tried when the agent encounters some specific situation.

Note that the mechanism described in this section is similar to that described in section 3.4.2 of Drescher's Made-Up Minds where "schemas with underrepresented actions receive enhanced exploration value".

7.5 "Wait and See" Period

After the instantiated Behaviors from a Behavior Scheme or Behavior Stream Scheme have executed, the agent waits to see what the results will be for some given number of cognitive cycles. This allows delayed results to be observed before another Behavior is selected for execution. This greatly simplifies the analysis of results since all results can be attributed to a single Behavior Scheme or Behavior Stream Scheme.

Note that this "wait and see" period is only required in environments where actions may have delayed results. This is not the case in Tyrrell's world for example.

More sophisticated analysis methods not requiring this "wait and see" period were considered but not implemented because of the complexity required and the increased learning time expected.

Also note that during this period the instantiated Behavior or Behavior Stream may be able to become more bound (for example, suppose a Behavior results in an object becoming hot. A Node for hot may not appear in the Conscious Broadcast till after the Behavior has executed). To ensure that the Behavior is as bound as possible before the analysis of results begins, the process described in section 5.1 is used to update the selected Behavior or Behavior Stream while in the "wait and see" period.

7.6 Actual Results

7.6.1 Detecting Actual Results

During the "wait and see" period actual changes in the environment are detected and are assumed to be the actual results of the Behavior. Of course, some unrelated changes will also occur, and may result in spurious Behavior Schemes being created, but it is assumed these will be very unreliable and soon be dropped (there is nothing here as sophisticated, or as resource consuming, as Drescher's positive-correlation and negative-correlation statistics in section 4.1.2 of Made-Up Minds).

It is not currently clear to this author how these changes will be detected in a full LIDA implementation. Will the actual change be available in the Conscious Broadcast? (i.e. not just the fact that "thirst is low" but that "thirst has been reduced from high to low"). If not, will Perceptual Memory be able to provide the change?

Procedural Memory is only intended to "remember" "objects and properties … if they have been "bound" to a Behavior (see section 3.5). So, if, during the "wait and see" period, the Conscious Broadcast contains a node such as “thirst is low”, Procedural Memory may not be able to detect this as a change using its normal memory mechanism.

To allow Procedural Memory to be tested as a stand alone process a Conscious Broadcast Merged History has been implemented. This represents the last known state of any object. If an object exists in the Conscious Broadcast in one cognitive cycle but is not in the Conscious Broadcast in the next cognitive cycle it is assumed to still exist in the same state (specifically with the same current activation). An object is only updated or removed from this history if it appears again in the Conscious Broadcast. Removal requires that this object appear in the Conscious Broadcast with current activation of zero (i.e. the agent is noticing that something has been removed from the environment).

When a Behavior Scheme or Behavior Stream Scheme is selected in learning mode, the Conscious Broadcast Merged History at the time the selection is made is saved. Then, during the “wait and see” period, any objects in the Conscious Broadcast are compared with the objects in the Conscious Broadcast Merged History at selection time and changes are detected. All such changes are recorded in the appropriate cognitive cycle in which they happened, relative to when the Behavior Scheme was selected.

The implementation of the Conscious Broadcast Merged History is an interim work around and is not a significant part of this research.

7.6.2 Recording Actual Results

One way to record actual results over several cognitive cycles would be to simply sum the changes over all cycles in the “wait and see” period, with or without a discount for changes that occur later. This approach could be used even in relatively complex situations. So, for example, if an object exists in the Conscious Broadcast Merged History with current activation 0.5 before a Behavior Scheme is selected, has its activation reduced in next cycle to .2, is removed in the next cycle, is reinstated in the next cycle with current activation 0.7, then the undiscounted change is an increase of 0.2, whereas the discounted change is

$$-0.3 + -0.2 * \text{discount} + 0.7 * \text{discount}^2$$

This actual change could then be compared with the expected change in the Result Conditions, which could record a single figure being the expected change (discounted or not) over all cognitive cycles in the “wait and see” period.

An even simpler approach would be to simply record that activation has increased or decreased and Result Conditions could simply record whether increase or decrease is expected.

These approaches are appealing in their simplicity but they do not enable the agent to construct Behavior Schemes that predict the actual change expected in each cognitive cycle.

This seemed important to the author at the time the current implementation was constructed and so Result Conditions actually hold an expected change for each cognitive cycle. Consequently, analyzing actual results becomes more complex and the comparison between objects in the Conscious Broadcast and objects in the Conscious Broadcast Merged History at selection time categorizes the change as follows:

- The object itself or one or more of its properties do not exist in the Conscious Broadcast Merged History at selection time but are now in the Conscious Broadcast – this is categorized as an ADDITION
- The object itself or one or more of its properties do exist in the Conscious Broadcast Merged History at selection time but are explicitly removed in the Conscious Broadcast (by having a current activation of zero) – this is categorized as a REMOVAL
- The object itself or one or more of its properties do exist in the Conscious Broadcast Merged History at selection time and are now in the Conscious Broadcast with increased current activation– this is categorized as an INCREASE. For reasons explained later, if the INCREASE results in full activation, that fact is also recorded
- The object itself or one or more of its properties do exist in the Conscious Broadcast Merged History at selection time and are now in the Conscious Broadcast with decreased current activation– this is categorized as a DECREASE

When only one cognitive cycle is considered this categorization is straightforward. But complications arise when more than one cycle is analyzed. For example it could happen that an object is removed in one cognitive cycle only to be reinstated in the next, or the current activation could increase in one cycle and then decrease in the next.

The current implementation is intended to handle actual results that may take place over several cognitive cycles and result in:

- An ADDITION (which may be delayed), possibly followed by further INCREASES, possibly resulting in full activation (categorized as an ADDITION)
- One or more INCREASES possibly resulting in full activation (categorized as an INCREASE)
- One or more DECREASES not followed by a REMOVAL (categorized as a DECREASE)
- One or more DECREASES followed by a REMOVAL (categorized as a REMOVAL)
- A REMOVAL (which may be delayed, categorized as a REMOVAL)

So, to reduce the overall complexity of the result analysis, only the above sequence of events is fully analysed. If more complex situations occur (for example an ADDITION followed by a REMOVAL) the categorization is only based on those cognitive cycles before this complex situation is detected.

Once the actual changes have been recorded, they are examined to determine the feeling effect of running the Behavior Scheme or Behavior Stream Scheme. The feeling effect is simply that actual discounted change in activation for all changes that have been recorded and which affect feeling nodes.

7.7 Analyzing Results

Although a Behavior Scheme or Behavior Stream Scheme may have many Result Conditions, the three statistics described above are held on the Behavior Scheme, not on the Result Condition. So, to update these statistics, it is necessary to determine the overall effect of the Behavior Scheme or Behavior Stream Scheme.

So, once the actual results have been categorized as describes above, they are compared with the expected results to determine the overall affect of selecting the Behavior Scheme or Behavior Stream Scheme. The expected results are categorized as INCREASE or DECREASE. Since actual results do not handle mixtures of DECREASES and INCREASES there is no point in the actual results doing so, so complex situations such as increase in one cognitive cycle followed by a decrease in the next are not handled.

The overall effect could be as simple as SUCCESS or FAILURE. This is appealingly simple but limits updates to curiosity, base activation or reliability to be the same regardless or whether the effect was above expectation, below expectation, as expected, no result or mixed result (i.e. the Behavior Scheme or Behavior Stream Scheme has more than one result and some were above and some below expectation).

To allow more sophisticated updates to the three statistics, the overall effect is categorized as:

- ABOVE: every expected result was INCREASE (DECREASE) and the actual result was ADDITION (REMOVAL) or INCREASE (DECREASE) and the discounted size of the actual change was at least the amount expected for every expected result and was significantly above for at least one result
- BELOW: every expected result was INCREASE (DECREASE) and the actual result was ADDITION (REMOVAL) or INCREASE (DECREASE) and the discounted size of the actual change was no more than the amount expected for every expected result and was significantly below for at least one result
- EXPECTED: every expected result was INCREASE (DECREASE) and the actual result was ADDITION (REMOVAL) or INCREASE (DECREASE) and the discounted size of the actual change was about the amount expected for every expected result or some results were significantly above and some significantly below the amount expected. A special case arises if the actual change was below expectation but the expected was an increase and the change resulted in full activation or the expected was a decrease and the change resulted in a removal. In this case the change has done as much as possible, so is still categorized as EXPECTED. This is the reason for recording if a change results in full activation
- MIXED: some results would, in themselves, be categorized as ABOVE, EXPECTED or BELOW but for others either no changes at all occurred or the result was opposite to what was expected (i.e. had expected of INCREASE and an actual of DECREASE or REMOVAL or an expected of DECREASE and an actual of INCREASE or ADDITION)

- NONE: for every expected result, either no changes at all occurred or the result was opposite to what was expected (i.e. had expected of INCREASE and an actual of DECREASE or REMOVAL or an expected of DECREASE and an actual of INCREASE or ADDITION)

Note that “no changes at all occurred” can occur because the instantiated Behavior or Behavior Stream had unbound arguments.

7.8 Using Results to Update the Selected Behavior Scheme or Behavior Stream Scheme

Once the overall effect and the feeling effect of the execution have been determined the selected Behavior Scheme or Behavior Stream Scheme is updated. Updates are made to the expected results, curiosity, base activation and reliability.

7.8.1 Updating Expected Results

Section 4.1.3 explains that Result Conditions have a size of effect which holds the expected change for each cognitive cycle. It is this size of effect that is updated after execution.

More than one implementation has been tried for this update including giving extra weight to the first few executions of a Behavior Scheme or Behavior Stream Scheme and giving extra weight to executions that affected feelings. Also some implementations have only updated the size of effect if the execution was successful (more specifically, had an overall effect of ABOVE, EXPECTED or BELOW).

However, the current implementation simply keeps the size of effect to be the average effect over the most recent executions and is updated regardless of the overall effect. If the Result Condition is part of an unbound argument it is updated using zero as the current effect.

7.8.2 Updating Curiosity

One of the main reasons for implementing a more sophisticated overall effect was to update curiosity differently for each effect. Basically the idea was that the agent would very quickly loose curiosity about a Behavior Scheme or Behavior Stream Scheme that produced EXPECTED results but would become more curious about any other result. Under this scheme curiosity would be increased by varying factors for overall effects of ABOVE, BELOW, MIXED or NONE and reduced for EXPECTED. This affect would be moderated by the number of executions, so that, after several executions, curiosity would be reduced regardless of the overall effect.

The expectation was that, because of the updating of expected results, after a few executions, useful Behavior Schemes or Behavior Stream Schemes would start to produce EXPECTED overall effect.

This worked well in some simple environments but did not work well in Tyrrell’s world simply because stable EXPECTED results are never achieved. Instead the overall effect tends to continuously swing from ABOVE to BELOW, with very frequent MIXED and NONE caused by the fact that executions often fail even when performed under the “right” conditions.

Consequently, the current implementation simply decays the curiosity whenever a Behavior Scheme or Behavior Stream Scheme is executed regardless of the overall effect.

This calls into question the need to have such a sophisticated overall effect. Similarly, holding a size of effect for each cognitive cycle and categorizing actual changes as ADDITION, REMOVAL, INCREASE and DECREASE does not work well in Tyrrell’s world, again because the results vary so much between executions and many results do not fit into these tidy categories (e.g., in Tyrrell’s world there will often be a increase followed by a decrease of a Node’s current activation during the “wait and see” period).

Consequently, in an implementation really aimed at Tyrrell’s world, it would be better to replace both the recording of actual results and the categorization of overall effects with a more crude but less brittle approach. For example, the size of effect could be replaced by a simple discounted sum over all cognitive cycles and the categorization of overall effects could be replaced by one that simply had SUCCESS or FAILURE.

In learning mode, curiosity drives Behavior Scheme or Behavior Stream Scheme selection for execution. Also, as described below, the creation of Behavior Stream Schemes requires that the Behavior Schemes that are to be included in the Behavior Stream Scheme need to be executed within some short time span. So, when a Behavior Scheme becomes a candidate for inclusion in a Behavior Stream Scheme, it is important that potential successor or predecessors in a new Behavior Stream Scheme are likely to be executed. But if their curiosity is low it is unlikely that they will. Consequently, when a Behavior Scheme becomes a candidate for inclusion in a Behavior Stream Scheme the curiosity is increased in all potential successor or predecessors. Note that this is equivalent to Drescher’s Inverse Actions in section 3.4.2 of “Made-Up Minds”.
This measure has ensured that potential successor or predecessors have a good chance of running but doesn’t in any way ensure that they are run (they are competing with many others). This is discussed more in section 7.11.
A Behavior Scheme is a candidate for inclusion in a Behavior Stream Scheme as a predecessor when all of the following are true:

- It is reliable
- It has some significant results
- It has no insignificant results
- It has a Result Condition that matches a Context Condition of the Behavior Scheme or Behavior Stream Scheme just executed (the Result Condition produces a positive result on the same node id in the non negated Context Condition or the Result Condition produces a negative result on the same node id in the negated Context Condition)

A Behavior Scheme is a candidate for inclusion in a Behavior Stream Scheme as a successor when all of the following are true:

- Its base activation exceeds a threshold
- It is reliable
- It has some significant results
- It has no insignificant results
- It has a Context Condition that matches a Result Condition of the Behavior Scheme or Behavior Stream Scheme just executed (the Result Condition produces a positive result on the same node id in the non negated Context Condition or the Result Condition produces a negative result on the same node id in the negated Context Condition)

These restrictions increase the chances of producing reliable Behavior Stream Schemes and the restriction on a successor having base activation exceeding a threshold means that the agent tries to find Behavior Stream Schemes that affect feelings. This restriction is only on successor as it's the last Behavior Scheme in the Behavior Stream Scheme that is likely to produce the feeling affect.

7.8.3 Updating Base Activation

The update to base activation is relatively simple – it is updated based on the feeling effect as described in section 6.3 of "A Cognitive Architecture Capable of Human Like Learning".

7.8.4 Updating Reliability

Since reliability is implemented as the ratio of the number of successful executions to the total number of executions and the total number of executions is always updated when a Behavior Scheme or Behavior Stream Scheme is executed, updating reliability is simply a matter of incrementing number of successful executions when appropriate.

In the current implementation, the number of successful executions is incremented when the overall effect is ABOVE, EXPECTED or BELOW. It is not incremented when the overall effect is NONE or MIXED.

7.9 Creating New Behavior Schemes by Specialization

New Behavior Schemes are created from the Behavior Scheme that has been executed if the existing Behavior Scheme is reliable and has no insignificant results. These conditions stop the agent building new Behavior Schemes that are very unlikely to prove useful.

The new Behavior Schemes constructed depend on the overall effect as follows:

If the overall effect is EXPECTED or ABOVE then:

- The high overall effect may be an indication that some condition was true in the environment that helped the Behavior Scheme to succeed and is not currently part of the Context Conditions. So new Behavior Schemes are created by adding one new Context Condition created from an object that was in the Conscious Broadcast Merged History at the time the Behavior Scheme was selected and had current activation above a threshold, as long as the new Behavior Scheme is not a duplicate

- Since this Behavior Scheme is currently good at predicting its existing results, it is a good candidate to see if it also produces other results not currently part of the Result Conditions. So new Behavior Schemes are created by adding one new Result Condition created from actual results as determined above, that had changes above a threshold, as long as the new Behavior Scheme is not a duplicate. A lower threshold is used for results that affect feeling nodes as these are often small but still important
- It may be possible that high overall effect can be achieved without one of the existing Context Conditions. So new Behavior Schemes are created by dropping one of the existing Context Conditions

If the overall effect is NONE or BELOW then:

- The low overall effect may be an indication that some condition was true in the environment that hindered the Behavior Scheme from succeeding and is not currently part of the Context Conditions. So new Behavior Schemes are created by adding one new negated Context Condition created from an object that was in the Conscious Broadcast Merged History at the time the Behavior Scheme was selected and had current activation above a threshold, as long as the new Behavior Scheme is not a duplicate
- If the Behavior Scheme has more than one Result Condition, the low overall effect may be because one of the Result Conditions is not relevant to the Context Conditions. So new Behavior Schemes are created by dropping one of the existing Result Conditions that had actual results that were significantly below expected results

If the overall effect is MIXED then:

- Some Result Conditions performed well and this may be an indication that some condition was true in the environment that helped these Result Conditions to succeed and is not currently part of the Context Conditions. So new Behavior Schemes are created by adding one new Context Condition created from an object that was in the Conscious Broadcast Merged History at the time the Behavior Scheme was selected and had current activation above a threshold, as long as the new Behavior Scheme is not a duplicate
- Some Result Conditions performed poorly and this is an indication that some condition was true in the environment that hindered the Result Conditions from succeeding and is not currently part of the Context Conditions. So new Behavior Schemes are created by adding one new negated Context Condition created from an object that was in the Conscious Broadcast Merged History at the time the Behavior Scheme was selected and had current activation above a threshold, as long as the new Behavior Scheme is not a duplicate

7.10 Creating New Behavior Schemes by Generalization

If a Behavior Scheme or Behavior Stream Scheme is chosen for execution part of the analysis after execution is to look for generalization opportunities.

The following generalizations of a Behavior Scheme are performed:

- Drop zero effect results
- Drop a possibly redundant Context or Result Condition

- Replace the Slipnet (or PAM) node in a Context or Result Condition by a more general one

Dropping zero effect is also applied to Behavior Stream Schemes; the others are not due to the difficulty with keeping the Result and Context Conditions of the Behavior Stream Scheme consistent with the underlying Behavior Schemes that it is based on. It was later realized that this wouldn't be a problem for dropping redundant conditions, but this has not currently been implemented.

7.10.1 Drop Zero Effect Results

If a Behavior Scheme or Behavior Stream Scheme has a Result Condition with size of effect close to zero, and it has other non zero Result Conditions, then a new Behavior Scheme or Behavior Stream Scheme is created which is the same as the one being analyzed but without the Result Condition as long as the new Behavior Scheme or Behavior Stream Scheme is not a duplicate.

7.10.2 Drop Redundant Context or Result Conditions

If another Behavior Scheme can be found that is the same as the Behavior Scheme being analyzed except that one Result or Context Condition is different, then it may be that this Result or Context Condition is redundant. So a new Behavior Scheme is created which is the same as the one being analyzed but without the Result or Context Condition as long as the new Behavior Scheme is not a duplicate.

This process is conceptually simple but is not so simple to implement due to:

- Difficulties in making the matching efficient enough
- The fact that two Behavior Schemes may have their Result and Context Conditions in different arguments means that comparisons need to be made to reorganized variants of the Behavior Schemes

The Behavior Scheme being analyzed may participate in conflictor/conflictee links. If the newly created Behavior Scheme has dropped a Result Condition then this may require removal of a conflictor link. This would be relatively easy since the Result Conditions/Context Conditions that constitute the conflict are recorded.

However, if the newly created Behavior Scheme has dropped a Context Condition then this may require removal of a conflictee link. This is not so easy since currently conflictee links only contain the Behavior Schemes that constitute the link, not the Result Condition/Context Condition.

So currently, the newly created Behavior Scheme has no conflictor/conflictee links – these have to be relearnt.

The Behavior Scheme being analyzed may also be part a one or more Behavior Stream Schemes. An attempt was made to find all Behavior Stream Schemes that contain the Behavior Scheme being analyzed and make new variants with the Behavior Scheme being analyzed replaced by the newly created Behavior Scheme. Conceptually this is achievable because any successor/success links that refers to the dropped Result or Context Condition can be dropped from the Behavior Stream Scheme variant. This was implemented but, during testing, did not seem to be working correctly, and is currently disabled. Consequently the participation of the newly created Behavior Scheme in Behavior Stream Schemes will have to be relearnt.

This discussion on conflictor/conflictee links and Behavior Stream Schemes also applies to the previous section.

7.10.3 Replace a Node by a more general one

If another Behavior Scheme can be found that is the same as the Behavior Scheme being analyzed except that a Result or Context Condition in the Behavior Scheme being analyzed refers to node_1 and the Result or Context Condition in the other Behavior Scheme refers to node_2 and node_1 and node_2 have a generalization in the Slipnet, then it may be that they can be replaced by a more general Behavior Scheme. So a new Behavior Scheme is created which is the same as the one being analyzed but with the Result or Context Condition changed so that it refers to the more general one as long as the new Behavior Scheme is not a duplicate.

This generalization relies on parts of the LIDA development that are outside of Procedural Memory and specifically on:

- Nodes in the Slipnet having the concept a how general they are (equivalently conceptual depth)
- Procedural Memory having enough access to the Slipnet to be able to access the generalization hierarchy

Since it is not currently know how other parts of LIDA will be implemented a temporary Slipnet Procedure View has been constructed that is sufficient for this purpose. This will be replaced when a full LIDA implementation is available.

Updating conflictor/conflictee links and making new variants of any Behavior Stream Schemes that the Behavior Scheme being analyzed is part of is not even conceptually easy, and no attempt has been made to implement this. This information will have to be relearnt for the newly created Behavior Scheme, if it is still relevant.

7.11 Create Behavior Stream Schemes

If a Behavior Scheme or Behavior Stream Scheme is chosen for execution, part of the analysis after execution is to look for opportunities to create new Behavior Stream Schemes.

In order to create a new Behavior Stream Scheme, a successor relationship must be found between a Behavior Scheme or Behavior Stream Scheme that was recently chosen for execution and the Behavior Scheme or Behavior Stream Scheme being analyzed (see Section 6.3.1 of “A Cognitive Architecture Capable of Human Like Learning” which mentions “executing within some small time span”).

In order to find such a relationship a history of recent execution selections is required. It is not clear how this history will be maintained in a full implementation of LIDA, but currently it is simply a limited length list of previous selections. When a new selection is added to the start (most recent position) in the list the oldest entry in the list is discarded.

Let the Behavior Scheme or Behavior Stream Scheme being analyzed be referred to as ps (for potential successor) and a recently executed Behavior Scheme or Behavior Stream Scheme be referred to as pp (for potential predecessor). Also let the Behavior or Behavior Streams instantiated from ps and pp that were actually chosen for execution be ips and ipp.

Then, all of the following need to be true for a new Behavior Stream Scheme to be created:

- ips and ipp must have been fully bound. This is required for the next bullet
- there must be a Post Condition in ipp and a Pre Condition in ips that were bound to the same property and actual object
- the Post Condition in ipp must increase the property if the Pre Condition in ips is not negated and decrease it otherwise
- pp and ps must both be reliable
- pp and ps must not have any insignificant results
- ps must have activation above a threshold. This ensures that we find Behavior Stream Schemes that affect the feeling of the agent
- the new Behavior Stream Scheme must not be a duplicate

The actual creation of a new Behavior Stream Scheme requires creation of enhanced Context Conditions and Result Conditions as described in section 4.2.2 and successor links as described in section 4.2.1.

This is mostly straightforward but there are a few points worth mentioning about the creation of Context Conditions and Result Conditions in the Behavior Stream Scheme:

- the Context Conditions in the Behavior Stream Scheme are a merge of the Context Conditions in pp and ps except that if the pp and ps have the same Context Condition except that it is negated in one but not the other then it is not included
- if a Context Condition in ps matches a Result Condition in pp then the Context Condition is not included (this is the whole point of creating the Behavior Stream Scheme – pp has results that make the context of ps true, and these should not be included in the Behavior Stream Scheme)
- the Result Conditions in the Behavior Stream Scheme are a merge of the Result Conditions in pp and ps except that if the pp and ps have the same Result Condition except that one produces an increase and one produces a decrease then it is not included
- the size of effect in the merged Result Conditions should hold the expected effect for each cognitive cycle after the Behavior Stream Scheme has run. But effects that come from ps will only start after ips runs and it is not known when this will be in relation to the running of ipp (for example, in the history of recent executions ips may have run 10 cognitive cycles after ipp, but perhaps it could have run after 2 cognitive cycles – it depends when ipp produces its effects and how much of those effects are required before ips can run). The size of effect in the merged Result Condition does not have to be accurate as it will be updated when the Behavior Stream Scheme is run, so as a

first approximation, it is assumed that ips will run after ipp has produced all its effects. So, for example, if pp has two Result Conditions with size of effects of [0.1, 0.3, 0.4] and [0.4, 0.5] which have to be merged with Result Conditions in ps with size of effects of [0.8, 0.9] and [0.6] respectively, the merged size of effects will be [0.1, 0.3, 0.4, 0.8, 0.9] and [0.4, 0.5, 0.0, 0.6]. This basically assumes that ips will run after the whole [0.1, 0.3, 0.4] effect is produced.

The creation of a Behavior Stream Scheme is basically the recognition of a successor link between two Behavior Schemes or Behavior Stream Schemes. These Behavior Schemes or Behavior Stream Schemes may already be included in other Behavior Stream Schemes but without this successor link. New versions of all such Behavior Stream Schemes are created which do include this successor link.

The restriction to only creating Behavior Stream Scheme between the Behavior Scheme or Behavior Stream Scheme being analyzed and one that was recently chosen for execution has meant that Behavior Stream Schemes that could be very useful are often not created. Behavior Stream Schemes would be created more reliably if all Behavior Schemes and Behavior Stream Schemes were considered, though this would require changes because the current mechanism relies on information only available in the instantiated Behaviors. This is discussed further in section 9.4.

7.12 Create Conflictor Links

If a Behavior Scheme or Behavior Stream Scheme is chosen for execution, part of the analysis after execution is to look for opportunities to create new conflictor links between Behavior Schemes.

In order to create a new conflictor link, a conflictor relationship must be found between a Behavior Scheme that was recently chosen for execution and the Behavior Scheme being analyzed.

This process uses the same history of recent execution selections used for the creation of Behavior Stream Schemes.

Let the Behavior Scheme being analyzed be referred to as ps (for potential successor) and a recently executed Behavior Scheme or Behavior Stream Scheme be referred to as pp (for potential predecessor). Also let the Behaviors instantiated from ps and pp that were actually chosen for execution be ips and ipp.

Then, all of the following need to be true for a new conflictor link to be created:

- ips and ipp must have been fully bound. This is required for the next bullet
- there must be a Post Condition in ipp and a Pre Condition in ips that were bound to the same property and actual object
- the Post Condition in ipp must increase the property if the Pre Condition in ips is negated and decrease it otherwise
- pp and ps must both be reliable
- pp and ps must not have any insignificant results
- the new conflictor link must not be a duplicate

8 Results

Some testing has been done in some very simple, very predictable environments, but the only significant testing has been done in Tyrrell's world, and that is what is reported here.

8.1 *Introduction*

The version of Tyrrell's World used is the Java implementation available from http://w2m.comp.dit.ie/services/tyrrellse/index.html.

Tyrrell's world is a challenging one for a learning agent for several reasons which are listed here and described in more detail below:

- The coarseness of directions and movements
- The low reliability of actions and no feedback on failure
- The invisible world boundary
- The missing "mated" signal
- Inconsistencies between specific and general food perception
- The small size of "successful" actions
- The size of the search space
- The dynamic (and dangerous) nature of the world

8.1.1 The coarseness of directions and movements

As in many simulated world, Tyrrell's world is divided into squares. Directions are specified using the 8 directions north, northeast, east, southeast, south, southwest, west, northwest but the animal's perception extends beyond the immediate adjacent squares. This makes learning harder because the exact direction of perceived objects cannot be communicated to the animal.
For example, suppose there is food two squares to the north and 1 square to the east. The animal will likely receive a perception of "food to the north east". But, if the animal does the sensible action of "move to the north east", then the perception will not change to "food in animal square", but rather "food to the north".

This is not the case in the real world where an animal is able to move directly towards perceived food (unless there are obstacles in the way).

The effects of this have been somewhat reduced by changing the parameter "number of squares distance that can be perceived" from 3 to 1.5. This was the only change made to the default parameters.

8.1.2 Low reliability of actions

Actions are unreliable and the world situation may change between the time an action is chosen and when it is completed. If the action fails then there is no feedback. For example, if the agent decides to "move north" the move may fail, but the animal is not told. So, if the animal was moving north in order to reach food, then the perception is that the move occurred but the food is not available. This is not the case in the real world where other signals would make it clear that the move failed.

If the action does succeed then the intended result may fail because the world has changed. For example a move towards a mate may not result in a mate in the animal square, as the mate may have moved as well. This is obviously a feature of the real world also.

8.1.3 Invisible World Boundary

Unlike the world described by Tyrrell in “Computational Mechanisms for Action Selection – 1993” the Java implementation has no perception of the world boundary. This makes learning more difficult as any actions that try to cross the boundary will fail inexplicably. This is not normally a feature of the real world, though may be similar to a fly trying to get through a closed glass window.

8.1.4 Missing Mated Signal

When mating occurs there is no perception of this occurring. This is obviously not the case in the real world. This makes it impossible for the animal to realise the importance of this event (it’s the only event that can affect the final score). In order to overcome this, the interface between Tyrrell’s world and LIDA has been changed to add such a perception when the mating action is chosen when a courted mated is in the animal’s square.

8.1.5 Inconsistent Food Perception

When food is not in the animal square the animal can only perceive the presence of food, not the type of food. Food in the animal square however can be perceived both as food and as a specific type of food. This is important as the agent has 3 eating actions and the right one must be chosen for eating to occur. This makes it harder to learn Behavior Stream Schemes as the two Behavior Schemes

> Move north: Food to the north → food in animal square
> and
> Eat cereal food: cereal food in animal square -> reduced protein shortage

do not make a sequence. Instead a sequence must be made from either

> Move north: Food to the north → cereal food in animal square
> and
> Eat cereal food: cereal food in animal square → reduced protein shortage

or

> Move north: Food to the north → food in animal square
> and
> Eat cereal food: food in animal square → reduced protein shortage

both of which involve very unreliable Behavior Schemes.

This is not the case in the real world where an animal could perceive a specific food at a distance.

8.1.6 The small size of “successful” actions

Often the effect of “doing the right thing” is hard to distinguish from noise.
For example, the Behavior Stream Scheme

Move southwest and eat fruit

sometimes increased carbohydrate shortage by .0023 (because of the move) and only decreased in by .0024 (because of the eat). This makes it hard to establish the usefulness of this Behavior Stream Scheme.

8.1.7 The size of the search space

Tyrrell’s world has 35 actions, 21 non directional perceptions and 15 directional perceptions (each with 9 values) giving 21 + (15 x 9) = 156 perceptions in total.

So even if Behavior Schemes of the form

Action: perception 1, perception 2 -> result 1, result 2,

(where each perception can be negated or not), were the only ones considered, there would be

35 x 2 x156 x 2 x156 x 156 x156 = 82,913,725,440

possibilities.

This is obviously still much simpler that the real world.

8.1.8 The dynamic and dangerous nature of the world

In Tyrrell’s world there are predators, mates and irrelevant animals that all move around. Food supplies are sometimes poisonous and can also be exhausted. Food supplies and other objects also sometimes appear unexpectedly. The animal’s memory is short term and error prone and the animals perception of where it is also error prone. The animal’s abilities are also error prone and this increases at night and when the animal is damaged.

All these factors mean that an action that works once in a given perceived situation may well not work the next time.

Conversely an action can seem to produce an absurd result. For example, since food can appear unexpectedly, a completely unrelated action can seem to produce food.

These factors make learning significantly harder.

These factors also occur in the real world also, but higher animals are often trained by parents in a safer, more predictable environment first.

8.2 Successes In Tyrrell's World

When run in Tyrrell's world the procedural learning mechanism finds Behavior Schemes and Behavior Stream Schemes that can keep the animal clean, provide water and food and achieve mating.
These are discussed separately below.

8.2.1 Keeping Clean

The simple Behavior Scheme

> Clean: Dirty Animal → reduced Dirtiness

Is found, run several times and has high base activation and reliability.

8.2.2 Providing Water

The Behavior Scheme:

> Drinking: water in animal square → reduced thirstiness, reduced water in animal square

and

Behavior Schemes of the form:

> Move north east: water to north east → water in animal square

are found and combined into Behavior Stream Schemes of the form:

> Move north east followed by Drinking: water to north east → reduced thirstiness

These Behavior Stream Schemes have high base activation and reliability.

8.2.3 Providing food

Despite the problem with inconsistent food perception mentioned above, Behavior Schemes of the form:

> Move north east: Food to the north east → cereal food in animal square
> and
> Eat cereal food: Cereal food in animal square → reduced protein shortage, reduced cereal food in animal square

are found and usually (see section 7.11 for the unreliability of the current mechanism) combined into Behavior Stream Schemes of the form:

> Move north east followed by Eat cereal food: Food to north east → reduced protein shortage

To achieve this, the reliability threshold discussed in section 4.1.9 needs to be set around 0.2.
These Behavior Stream Schemes have high base activation and low reliability.

8.2.4 Mating

Despite the fact that Court displaying is one of the low reliability actions discussed in section 8.1.2, Behavior Schemes of the form:

> Move north east: Mate to the north east → mate in animal square
> and
> Court displaying: Mate in animal square → mate courted
> and
> Mate: Mate courted → mated

are usually (court displaying is marginally reliable) found and sometimes (see section 7.11 for the unreliability of the current mechanism) combined into Behavior Stream Schemes of the form:

> Move north east followed by Court displaying followed by Mate: Mate to north east → mated

Again, to achieve this, the reliability threshold discussed in section 4.1.9 needs to be set around 0.2.
These Behavior Stream Schemes have high base activation and low reliability.

8.3 Failures In Tyrrell's World

In Tyrrell's world the agent fails in the following ways that are discussed in more detail below:

- Behavior Schemes with redundant Context Conditions end up with higher base activation than Behavior Schemes with no redundant Context Conditions
- Behavior Schemes with missing Context Conditions can end up with higher base activation than Behavior Schemes with no missing Context Conditions
- Behavior Schemes with missing Result Conditions can end up with higher base activation than Behavior Schemes with no missing Result Conditions
- Some Behavior Schemes are unexpectedly successful
- Effective strategies to avoid predators have not been identified
- The significance of night has not been identified
- The significance of sleeping has not been identified
- The significance of the den has not been identified

8.3.1 Redundant Context Conditions

As well as creating

> Clean: Dirty Animal → reduced Dirtiness

the learning mechanisms also create other Behavior Schemes such as:

> Clean: Dirty Animal, animal short of water → reduced Dirtiness

and

> Clean: Dirty Animal, no Food to the South → reduced Dirtiness

Since “animal short of water” and “no Food to the South” are almost always true, the Context Conditions for these Behavior Schemes usually result in the same executability as the simpler Clean: Dirty Animal → reduced Dirtiness.

But, because they have the extra Context Condition, the more complex Behavior Schemes more often become potential successors and have their curiosity increased as described in section 7.8.2. This results in their being run more often and ending up with higher base activation.

It should be noted that the existence of the mechanism described in section 7.8.2 exacerbates the problem rather than causing it, since, such Behavior Schemes will (usually) have the same executability and base activation as the more obvious one so, even in “normal” mode (see section 7.3) may get chosen for execution more often (since the choice between them is arbitrary) which will increase their base activation. Once their base activation is higher than the more obvious Behavior Scheme, then they will continue to be preferred by the normal selection algorithm.

There doesn’t seem to be a mechanism in “A Cognitive Architecture Capable of Human Like Learning” to address this problem. No additional mechanism has been built so currently there is no mechanism to detect and drop Behavior Schemes with redundant Context Conditions.

8.3.2 Missing Context Conditions

This is the converse of that described in the previous section.

Suppose we have two Behavior Schemes:

> Drinking: water in animal square, animal thirsty → reduced thirstiness

and

> Drinking: water in animal square → reduced thirstiness

The first of these is more reliable, but the second is usually reliable. For the same reasons as described in the previous section, the second, less reliable Behavior Scheme may end up with higher base activation when running in “normal” mode. Again, there is no mechanism to detect and drop Behavior Schemes with missing Context Conditions.

8.3.3 Missing Result Conditions

This is similar to that described in the previous two sections.

Suppose we have two Behavior Schemes:

> Drinking: water in animal square → reduced thirstiness, reduced water in animal square

and

Drinking: water in animal square → reduced thirstiness

The first of these more accurately describes the effects of taking the action. When running in normal mode, the activation that these two Behavior Schemes will receive will depend on what conflictor and successor activations they are passed. If no successor activations have been identified and the first Behavior Scheme gets reduced activation because the reduction in water conflicts with another Behavior Scheme, then the second Behavior Scheme will be selected and receive further base activation if successful. Once it receives higher base activation it will continue to be selected even though it is really "cheating" – it doesn't disclose its full effects so conflictor activation it should have received doesn't happen.
Again, there is no mechanism to detect and drop Behavior Schemes with missing Result Conditions.

8.3.4 Unexpectedly successful Behavior Schemes

As mentioned in section 8.1.8 an action can seem to produce an absurd result and this can happen often enough to be considered reliable. The real problem here is that legitimate Behavior Schemes (such as Court displaying: Mate in animal square → mate courted) are unreliable so that the reliability threshold has to be set very low (currently 0.19). With the threshold set low, obviously absurd Behavior Schemes (such as Court displaying: Protein shortage → Food to the south) are considered reliable.

8.3.5 Too many sub-optimal Behavior Schemes

The end result of the previous four sections is that the learning mechanism identifies many sub-optimal Behavior Schemes which are reliable. These then participate in the creation of Behavior Stream Schemes. The end result is the creation of many Behavior Schemes that are very unlikely to really be helpful to the agent.

"A Cognitive Architecture Capable of Human Like Learning" recognized that many redundant Behavior Schemes would be created but expected them to be quickly dropped. For those cases described in the previous four sections, this is not happening, resulting in many sub-optimal Behavior Schemes, which in turn are used to create sub-optimal Behavior Stream Schemes.

8.3.6 Handling Predators

Consider again a Behavior Scheme of the form

Eat cereal food: Cereal food in animal square → reduced protein shortage, reduced cereal food in animal square

Since Tyrrell's world includes predators, it would be better if this Behavior Scheme included another Context Condition about predators such as:

Eat cereal food: Cereal food in animal square, no predator in sight → reduced protein shortage, reduced cereal food in animal square

The existing mechanisms should be sufficient to learn such a scheme as the presence of the predator is very likely to result in the normal effect not occurring in which case the mechanism described in section 7.9 when a NONE or BELOW result occurs (“some condition was true in the environment that hindered the Behavior Scheme from succeeding”) should be sufficient to create a new Behavior Scheme with the predator Context Condition. However, in Tyrrell’s world the animal usually dies at this point and the Tyrrell world implementation terminates the run, and the analysis phase is not invoked.

Consequently, the currently created Behavior Schemes and Behavior Stream Schemes basically ignore the presence of predators (a very dangerous thing to do!).

8.3.7 Ineffectiveness at night

In Tyrrell’s world the animal is much less effective at night and the animal can perceive “night proximity”. Since, as explained in section 6.1.3 any activation that is significant (above a “noise threshold”) is considered “true” and activation below that threshold is considered “false”, this perception would need to be converted to true and false by choosing a threshold that correctly represents the point at which the animal is significantly impaired.

Once a “night” perception was available the animal should be able to refine Behavior Schemes such as:

> Move north east: Food to the north east → cereal food in animal square

to

> Move north east: Food to the north east, not night → cereal food in animal square

Converting “night proximity” to a “night/not night” perception is not expected to be difficult but has currently not been attempted.

8.3.8 Sleeping to cure fatigue

Tyrrell’s world includes the action of sleeping but it does not include the concept of fatigue.
The best action for the animal to choose at night is to sleep, not because it feels fatigue but because it results in the least reduction of food and water and there is nothing effective the animal can do at night (see section 8.3.7).

If all Behavior Schemes such as

> Move north east: Food to the north east → cereal food in animal square

were dropped as unreliable and replaced by:

> Move north east: Food to the north east, not night → cereal food in animal square

then the action of Sleeping might be chosen as the only Behavior Scheme available at night. But the problems described in sections 8.3.1 through 8.3.5 means that currently the animal will continue to choose ineffective Behavior Schemes at night.

8.3.9 Returning to den at night

Not only is the best action for the animal to sleep at night, it is also best to do it in its den, since there it is protected from predators.
To realize this, there would need to be a recognition that predators never appear when the animal is in the den. This would need some history of events and an analysis of this history. What component of the LIDA architecture would hold such a history or perform such an analysis is not clear. It is also not clear how such an analysis would be communicated to Procedural Memory in such a way that it would be able to construct Behavior Schemes of the form:

Move to south: night approaching, den to south → animal in den

Note that such a Behavior Scheme would require the “night proximity” perception to be converted to “night” and “night approaching”.

9 Further Work

As mentioned in section 7, the learning described in this paper has been based on that described in “A Cognitive Architecture Capable of Human Like Learning” by Franklin and D’Mello.

Additional mechanisms, not described in that paper have also been introduced, and these fall into two categories: workarounds to make Procedural Memory capable of running as a standalone process and genuine additional mechanisms.

The main workarounds that have been discussed are:

- Conscious Broadcast Procedure View (section 3.4)
- Conscious Broadcast Merged History (section 7.6.1)
- History of recent execution (section 7.11)
- Slipnet Procedure View (section 7.10.3)

The main additional mechanisms that have been discussed are:

- The use of three statistics instead of one (section 7.2)
- Behavior Selection in Learning Mode (section 7.4)
- Result Conditions with effects for each cognitive cycle (section 4.1.3)
- creating new Behavior Schemes by Specialization and Generalization (sections 7.9 and 7.10)

But, as shown by the failures when testing in Tyrrell’s world, more mechanisms may be required. The following mechanisms (discussed in more detail below) have not been implemented, but could improve the results:

- Variable reliability
- Dropping sub-optimal Behavior Schemes

- More effective exploration
- Behavior Stream Scheme creation that doesn't rely on recent execution

9.1 *Variable Reliability*

As mentioned in section 8.3.4 some Behavior Schemes, such as

Court displaying: Mate in animal square → mate courted

are unreliable but useful. Others, such as

Court displaying: Protein shortage → Food to the south

are unreliable and useless.

The real difference is that

Court displaying: Mate in animal square → mate courted

is the *only* way for the animal to achieve "mate courted", whereas there are far more reliable ways to achieve "food to the south".
The current implementation uses a reliability threshold that establishes whether a Behavior Scheme is reliable enough to participate in further learning.
Rather than using a fixed threshold, the learning would create better Behavior Schemes if it used the most reliable available Behavior Scheme that achieved a particular result.

9.2 *Dropping sub-optimal Behavior Schemes*

Sections 8.3.1 through 8.3.3 described Behavior Schemes that are sub-optimal. "A Cognitive Architecture Capable of Human Like Learning" assumed that such Behavior Schemes would be created but would be quickly dropped and replaced by more optimal ones. The current implementation is not doing this successfully and seems to need explicit mechanisms to recognize that:

- if there are two Behavior Schemes a → c and a, b → c and a, b → c is more reliable, then a → c should be dropped.
- if there are two Behavior Schemes a → c and a, b → c and a → c is just as reliable, then a, b → c should be dropped
- if there are two Behavior Schemes a → b and a → b, c and a → b, c is just as reliable, then a → b should be dropped

9.3 *More effective exploration*

Procedural memory starts off with intrinsic Behavior Schemes but no knowledge of what they achieve or when they are appropriate. The current learning mechanism chooses Behavior Schemes to explore on the basis of a curiosity factor. This ignores the fact that the movement Behavior Schemes play an important role in exploring the agent's world. In other words, the agent should sometimes be choosing movement, not because it is curious about the movement Behavior Scheme itself, but because this Behavior Scheme can position the agent in a new unexplored part of its world. To rephrase again, the agent should be exploring its world as well as exploring its own abilities.

The current mechanism has been surprisingly successful at discovering the type of objects in its world without any explicit attempt to explore the world. But, this may be because Tyrrell’s world conveniently creates objects in many places. If “fruit food” only existed in a single place in the agent’s environment it could easily be missed. It seems that the agent needs a built in mechanism to explore parts of its world that it hasn’t been to. This requires the ability to recognize known places and try to find unknown places and presumably requires involvement of more than just Procedural Memory.

9.4 More reliable Behavior Stream Scheme creation

As mentioned in sections 7.8.2 and 7.11, the current implementation increases the curiosity of Behavior Schemes that appear to be candidates for inclusion in a Behavior Stream Scheme. This is intended to increase the chances that two Behavior Schemes will run “within some short time span”, which is a requirement for Behavior Stream Scheme creation.

This mechanism has been fairly successful, but often Behavior Stream Schemes are missed because of the requirement to run “within some short time span”. Behavior Stream Schemes would be created more reliably if the current mechanism to recognize potential predecessors and successors (section 7.8.2) was used to actually create Behavior Stream Schemes (replacing the implementation in section 7.11). This would result in the creation of less reliable Behavior Stream Schemes (since less matching can be performed when no instantiated Behavior is available), but it would ensure that useful Behavior Stream Schemes are not missed.