An Introduction to CLARION

Ryan Allison

Rensselaer Polytechnic Institute

An Introduction to CLARION

Psychologists have been puzzling over the workings of the mind for centuries. There has been a multitude of methods for analyzing and modeling the cognitive functioning of humans and animals. One method, known as cognitive modeling, uses computers to simulate human cognition and behavior. Researchers go about this by writing software code. Using a programming language (like LISP), they can develop functions and procedures which perform much like humans would. Efforts have even been undertaken to develop whole architectures which try to duplicate the functioning of the human mind. By comparing the results of simulations using these programs with experiments involving humans, cognitive scientists can justify the procedures they develop. These cognitive architectures are symbolically based, however, and do not attempt to mimic the human mind on the organic level. But by understanding the processes and functions of the mind, we will be closer to understanding how the brain functions biologically.

One such cognitive architecture, the topic of this paper, is called CLARION. This stands for *Connectionist Learning with Adoptive Rule Induction ON-Line*. Development of this project has been led by Dr. Ron Sun, who has written a guide of CLARION entitled *A Tutorial on CLARION 5.0* (2003). The purpose of this paper is to provide a simpler, less technical introduction to

CLARION and is based on the tutorial supplied by Dr. Sun. A copy of this tutorial can be currently accessed at http://www.cogsci.rpi.edu/~rsun/sun.tutorial.pdf.

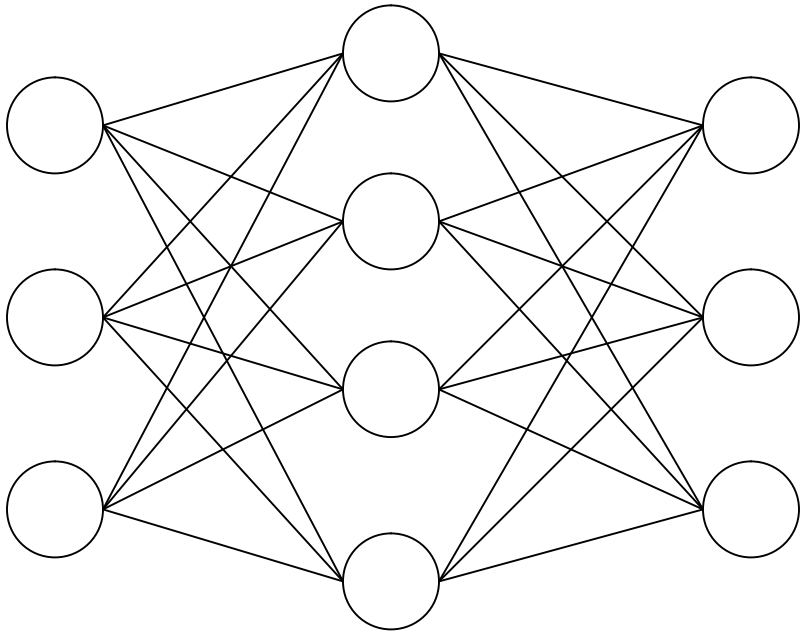
General Architecture

Before discussing the workings of each part of CLARION, we must first describe the general structure involved. A major question in the development of a cognitive architecture is how the basic units of thought are going to be represented. Science has yet to find, at least completely, how nature stores and manipulates information within the brain, so cognitive scientists must develop systems which represent, on a working level, these units. By this, it is meant that the knowledge systems may not be directly identical to human knowledge representation, but are functionally very similar. It is the hope that science can synthesize the workings of the human mind without having to understand the exact biology involved.

Simulating knowledge representation runs in to difficulty since there is not necessarily a single type of knowledge. In CLARION, it is split into implicit knowledge and explicit knowledge (Sun, 2003). Both represent different types of information and therefore must be handled differently. For instance, implicit knowledge is not directly accessible, but can be used in computation. Rumelhart et al, Sun, Reber, Seger, and Cleeremans et al have provided support in regards to the argument that a backpropagation network can represent the subsymbolic, distributed representation of implicit knowledge (as cited in Sun, 2003). Sun (2003) draws from Rumelhart et al and Sun when he explains that the units in a

backpropagation network “are capable of accomplishing computations but are subsymbolic and not individually meaningful; that is, they generally do not have associated semantic labels,” (2003, p. 5).

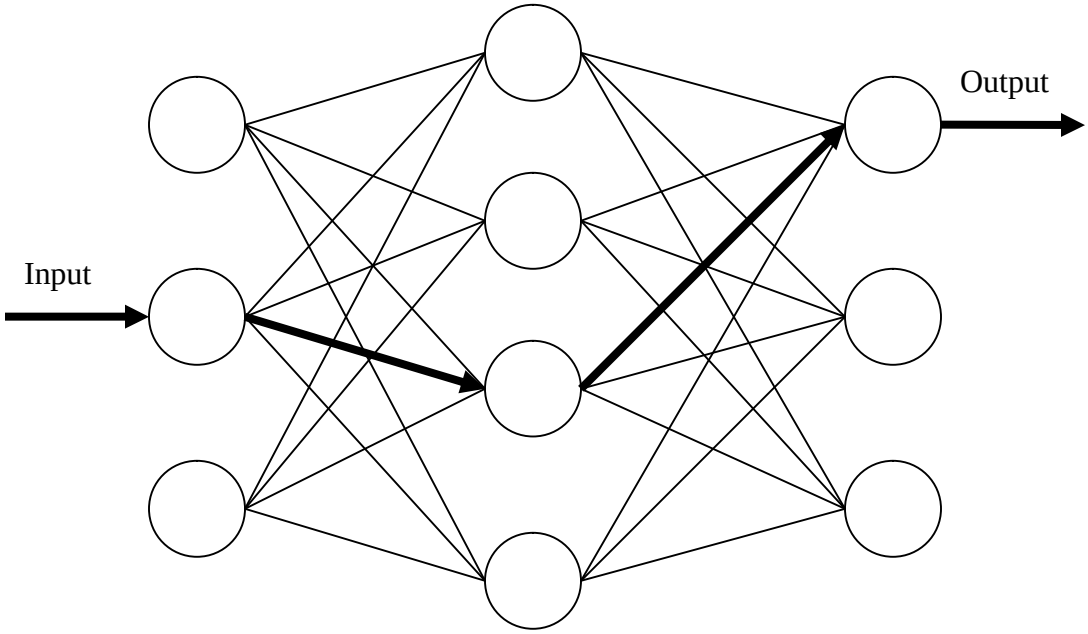
A backpropagation network is a type of neural network in which the connections between nodes are adjusted to produce the optimal output. A neural network is a collection of nodes which are connected to each other by pathways that allow a signal. They are based on the biology of the brain where special cells, neurons, are connected to each other and pass an electrical signal. The following figure is a simplified example of a neural network:



In this diagram, each circle is a “node” and the lines between them represent the connections. This is a small representation of a network, in actuality there would be thousands, if not millions of nodes and connections.

In humans, the nodes are neurons, the pathways are synapses, and the signal is an electrical impulse. In computers, the nodes are small pieces of

memory, the pathways are references to the location of the nodes in memory, and the signal is a small amount of data. A signal traveling down different pathways and different nodes will produce different outputs. Adjusting the network consists of training, namely, repeatedly matching a specific input to a specific output and adjusting the weights of the connections between them. Once a network has been trained to a given input, the signal will travel the weighted connections to the trained output. The following figure repeated below, but modified to demonstrate:



Here, a specific path has been trained so that when the input node is triggered, the signal travels through the network to the trained output node. Future training might strengthen this particular path even more, or, if a new response is desired, it might lead to a new connection to a different output node.

This is just like in the brain, where a stimulated neuron will pass its signal to the next neuron, which will then pass it to the next, and so on until it reaches

the destination cells. For instance, a decision to move your arm up would involve sending a signal through the neurons that have been trained to connect the brain to the muscle in your arm. The more this route is used, the stronger the connections between the neurons become and the easier for this path to be used.

Just like neurons, nodes are not individually useful. Only when the node is a part of a network can it be constructive. Thus, a network is a system of many, small parts which are not functional on their own, only as a group. Basically, it allows for the computation of information (via controlling the pathway from input to output), but not direct access to the individual components involved in the computation. Since implicit knowledge represents the lower-level, almost automatic processing of information, it is easy to see why backpropagation networks are used.

Explicit knowledge, on the other hand, is more appropriately represented in a symbolic or localist way (Clark and Karmiloff-Smith, as cited in Sun, 2003). According to Sun, symbolic representation means that “each unit is more easily interpretable and has a clearer conceptual meaning (namely, a semantic label),” (2003, p. 6). So, unlike implicit knowledge, explicit knowledge consists of more substantial pieces of information which can be directly accessed and manipulated (Smolensky, Sun, as cited in Sun, 2003). This knowledge needs to be easily accessible since it is the basis of complex thought. While implicit knowledge represents sub-conscious information, explicit knowledge represents more conscious information. Explicit knowledge within CLARION is thus divided

into rules and chunks, which are easier to explain after introducing subsystems and modules (Sun, 2003).

Since both types of knowledge are necessary to develop a model of cognition, it is appropriate to include both distributed and localist representation (Sun, 2003). The developers of CLARION chose to use both types of representation, in the form of a two-level architecture, so that implicit and explicit knowledge are both handled properly. Sun, Dreyfus and Dreyfus, and Smolensky have given support in favor of this two-level architecture (as cited in Sun, 2003).

Within each level, there exist numerous modules that handle the functions of cognition. In addition, these modules are parts of separate subsystems. Each subsystem thus has a module (or modules) for implicit and explicit processes (Sun, 2003). A subsystem refers to a set of functions which handle a particular aspect of cognition. For example, there is the action-centered subsystem. This subsystem contains all the processes which govern the actions of the cognitive model. It consists of action-centered modules which exist in either the implicit level or the explicit level. Another subsystem, the non-action-centered subsystem, also contains modules in both levels of representation. Sun explains that “the reason for having both action-centered and non-action-centered modules (at each level) is because, as it should be obvious, action-centered knowledge (roughly, procedural knowledge) is not necessarily accessible (directly),” (2003, p. 6). So, while action-centered knowledge is usually implicit, it

is not necessarily so, as is non-action-centered knowledge not necessarily explicit. The following figure may help to clarify the organization.

		Sub-system	
		Action-centered	Non-action-centered
Knowledge	Top-level (Explicit)	Module(s)	Module(s)
	Bottom-level (Implicit)	Module(s)	Module(s)

This figure shows how each sub-system has modules in both levels of the knowledge architecture. In addition to the action-centered sub-system and the non-action-centered subsystem, there are two other subsystems. These are the motivational subsystem and the meta-cognitive subsystem. The purpose of the motivational subsystem is to guide the agent's overall behavior. It achieves this by forming goals and sending them to the goal structure. The meta-cognitive subsystem is the main controller of the cognitive architecture. It manages the processes of the other subsystems so as to provide structured behavior.

Within the modules of the subsystems are the elemental parts of knowledge within CLARION, dimension/value pairs, rules, and chunks (Sun, 2003). The modules in the bottom-level of a subsystem are made of dimension/value pairs. They are of the format "(dimension, value)." A dimension would be an aspect of an object and the value(s) would be the adjective or

intensity. For example, “(height, tall)” would be describing something or someone that is tall. The modules which contain these pairs represent implicit knowledge.

The input into CLARION, known as the state of the world, is a series of dimension/value pairs, by which they are describing how and what things are in the environment. The “state of the world” refers to the environment and condition the agent is in at a given time. More specifically, it consists of the objects in the environment which the agent senses, as well as information about items in working memory and the current goal. It can be thought of as a snapshot of everything around and within the agent. The senses convert their raw input (light for eyes, sound waves for ears, etc.) into dimension/value pairs. These pairs, in addition to the dimension/value pairs of the working memory and goals, constitute the current world state, which is then used as input for the action-centered subsystem.

For explicit knowledge, chunks and rules are used. They exist in the top-level modules of each subsystem. Chunks serve as a way of tying dimension/value pairs into a group so that they describe a particular thing (Sun, 2003). Each chunk has an identifier so that it is unique. Sun provides an excellent example of a chunk, “*table-1: (size, large) (color, white) (number-of-legs, four)*” (2003, p. 63). In his example, the chunk is named “*table-1*” and has three dimension/value pairs, each one describing an aspect (dimension) of the table. Another chunk, maybe “*table-2,*” could have entirely different values for those dimensions, and thus be a different object in the world state.

Rules are the other base elements used in CLARION's explicit knowledge representation. Rules differ slightly based on which subsystem they are in (Sun, 2003). For action-centered modules, they are action rules. If they are in non-action-centered modules, they are associative rules. To generalize them, rules are basically conditional statements which serve to output information that is specific to the module it is in. What is meant by a conditional statement is a simple test to see if the requirements (stated in the "condition" of the rule) are met. A simple, real-world example is this: IF [The door is open], THEN [Shut it]. The first part of this rule ([The door is open]) is the condition. If the current state of the world that the agent is in has a door that is open, then this condition would be true. Whenever the condition of a rule is true, the output at the end of it is triggered (or in this case performed). If the condition is false (i.e. there is no door or all doors are closed), then the rule is simply ignored and the action is not triggered. An action rule in CLARION is much the same as the example above, albeit more detailed as to the condition and result.

Rules have dimension/value pairs as their condition (represented by a chunk), so if the dimension/value pairs from the current world state match the dimension/value pairs of the rule's condition, then the output of the rule will be initiated (Sun, 2003). For example, if the decision/value pairs in an action rule's condition were to match the decision/value pairs of the world state, then an action recommendation would be triggered. Associative rules also have dimension/value pairs as conditions, but their output is in the form of a memory chunk. In both cases of rules, the dimension/value pairs of the condition are

associated into one or more chunks. The output of rules are also chunks, but are handled differently depending on the type of rule. The reasoning behind this will be addressed in more detail later, as will the reason why the conditions are chunks. As a note, the world state can (and usually will) have many more dimension/value pairs than a rule and still activate it, as long as the rule's condition pairs are all matched. In other words, for every dimension/value pair in the rule's condition, there has to be a dimension/value pair in the world state that is the same. In truth, CLARION does support the option of allowing partial matching (where not all values have to be matched exactly) (Sun, 2003). However, in an effort to keep this document as simple as possible, the logic will not be covered here. The reader is encouraged to refer to Sun's tutorial (2003) if understanding of this process is desired.

As stated before, CLARION has a two-level knowledge architecture, top-level explicit knowledge and bottom-level implicit knowledge. These two types of knowledge are linked together by the relationship between dimension/value pairs, chunks, and rules. The dimension/value pairs exist in the bottom-level backpropagation networks that were described earlier. Each separate dimension and each separate value is a node in a backpropagation network and represents a piece of implicit knowledge (Sun, 2003). In order to form a dimension/value pair, the dimension node is linked to the value node. Chunks exist in the top-level and therefore represent explicit knowledge, but are linked to their associated dimension/value pairs in the bottom-level. More specifically, the

chunk nodes are linked to the dimension and value nodes. The reasoning behind this will be addressed in later sections.

An individual dimension or value node by itself is meaningless, but when associated with others via a chunk, it becomes a part of an item of explicit knowledge. The name of the chunk serves as the semantic label for the concept represented by the associated dimension/value pairs in the bottom-level (Sun, 2003). Rules consist of chunks, so are therefore also present in the top-level. However, the chunks are associated with dimension/value pairs, so rules are also connected to the bottom level. In this manner, the designers of CLARION have been able to represent implicit and explicit knowledge accurately, as well as allowing them to interact.

Action-centered subsystem

Structure

The action-centered subsystem serves as the part of the mind which controls the physical and mental actions of the agent. This would include such things as interaction with the environment (external actions), manipulation of items in memory (working memory actions), and adjustments to goals (goal actions) (Sun, 2003). More specifically, the action-centered subsystem takes in data and produces recommendations, from which the best action is chosen. The resultant chosen recommendation is then performed by the appropriate functions which govern the respective actions. For example, a recommendation to pick up an object, if chosen to be executed, would trigger the activation of base-level

motor control. Or, if the recommendation to access a memory was chosen, the retrieval of the desired information would be triggered.

Before any actions can be taken by the agent, the current state of its environment must be taken in. As stated earlier, the environment is perceived by the agent's senses and then converted into dimension/value pairs. These pairs, along with working memory items and the current goal, are then sent to the action-centered subsystem so that actions can be chosen and performed (Sun, 2003). Within the action-centered subsystem, the state, memory items, and current goal are sent to both top-level and bottom-level modules. The modules in each level produce action recommendations based on the information given. From all of the action recommendations which were triggered, the best one is selected and executed. This does not signal the end of the process, however. The agent now perceives the new world state, as well as any possible reinforcement. Reinforcement is a factor of learning in the action-centered subsystem and used to modify the agent's future responses. This is achieved through adjusting the connections in the backpropagation networks (for the bottom-level) and modifying the action rules (for the top-level) (Sun, 2003). Once this has been completed, the agent can then start a new round of action decision making.

Bottom-level

The action-centered subsystem at the bottom-level (implicit) has been designed to be highly modular. This means that there can be numerous

backpropagation networks, each being specific to a task or a stimulus (Sun, 2003). These networks can be classified as external action networks, working memory action networks, or the goal action network. External action networks are those actions whose results will affect the world state (i.e. the agent's environment). The working memory network contains those actions which manipulate the items in working memory. Likewise, the goal action network has the actions which will alter the goals of the agent. It is actually not required to have separate working memory networks and goal action networks, both can be a part of the external action networks (Sun, 2003). There can be, however, multiple external action networks. This allows external action networks to become specific to particular situations or responses. Much like human reflexes, these networks produce a response (output) given a stimulus (input) without any conscious thought by the agent (like recoiling if a hand touches fire).

By applying the process of learning, these networks can be modified and the responses of the agent changed. Some of these networks, however, could represent the reflexive, innate actions of instinct (Sun, 2003). These fixed networks would be the results of the evolutionary process shaping the behavior of the agent's ancestors. Those agents which were "stronger" than others, because of some ability or reflex, would pass this on to the descendants. This ability, if it was extremely beneficial to the agent, would eventually become a fixed part of the agents' action networks. Of course a simulation involving CLARION would not have ancestors and therefore not be a product of evolution.

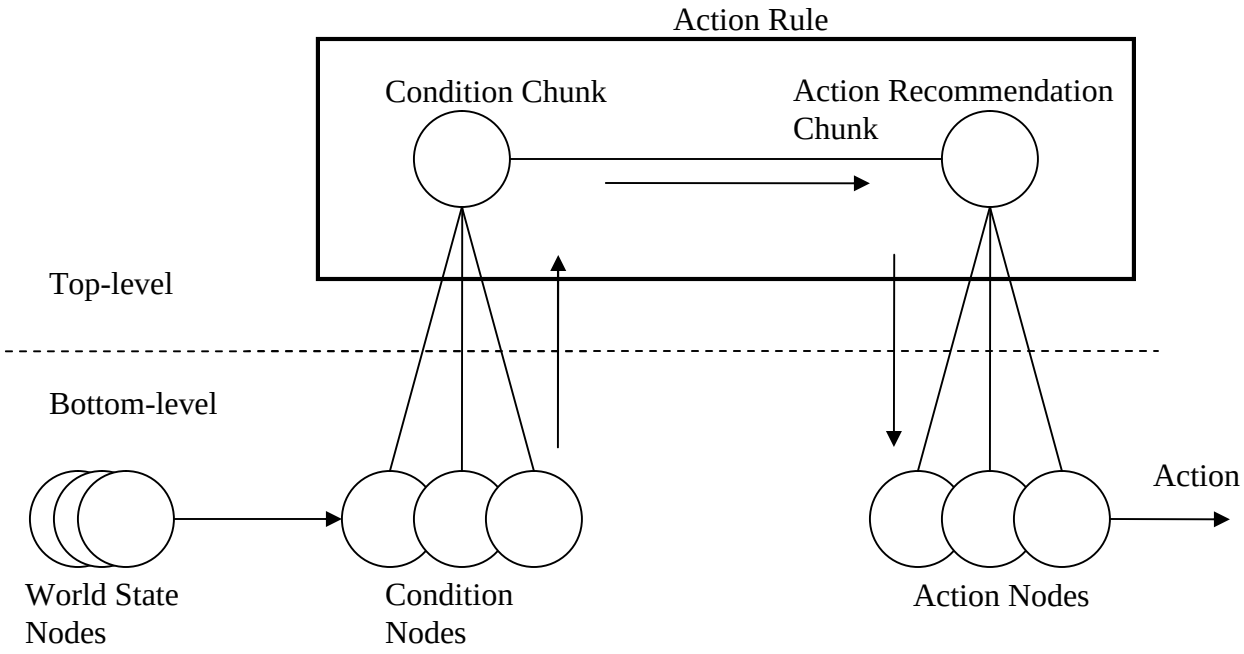
However, since the goal of CLARION is to represent the human mind, it must therefore take into account these instinctual products of evolution.

Top-level

The top-level (explicit) of the action-centered subsystem is also made of multiple modules. These, however, are not as numerous as the bottom-level modules, since explicit knowledge is not as modularized (Fodor, as cited in Sun, 1983). "That is, explicit knowledge may be viewed as residing in a more centralized and less compartmentalized structure," (Sun, 2003, p. 7). So, unlike the bottom-level, there are not individual modules to handle specific instances of explicit knowledge, but more generalized ones. For instance, while performing a new task, an agent might use top-level modules which were used in previous, similar tasks. Let's say that an agent has experience taking car engines apart. On encountering a new, unfamiliar engine, the agent would not automatically know how to disassemble the engine, but might use knowledge from existing top-level modules to formulate a plan of action.

The input to the top-level of the action-centered modules is the same as that given to the bottom-level. The world state, along with the items in working memory and the current goal, is compared with the rules in the top-level. Each rule, also like the bottom-level, gives an action recommendation if triggered. Since action recommendations are divided into external, working memory, and goal types, the rules can also be grouped accordingly. Thus, there are external action rules, working memory action rules, and goal action rules (Sun, 2003).

As mentioned earlier, the condition of each rule is compared to the world state. If the dimension/value pairs of the rule are matched, then the action recommendation of that rule is activated. Remember that the action recommendation of a rule is also a chunk, linked to the action nodes in the bottom-level networks. After each rule has been compared, all of the activated action recommendations are then considered and the best one selected. The method of selecting the best rule can be governed by a utility measure (Sun, 2003). The utility measure would be a mathematical function which evaluates how constructive each action recommendation would be and assigns them each a "score." The recommendation with the highest "score" would be the action that is the most beneficial to the agent. If a utility measure is not used in choosing a recommendation, then random selection can be used (Sun, 2003). After choosing the best recommendation, that chunk activates the action nodes in the bottom-level that it is linked to. The following figure is offered to help in understanding this process and is a rough approximation of how the respective elements are associated.



This figure demonstrates the relationships between the pieces involved in action-decision making. The world state nodes (dimension/value pairs) are compared to the condition nodes (dimension/value pairs) of a rule in the bottom-level. If they are matched, then the condition nodes activate the condition chunk in the top-level that they are linked to. This condition chunk then activates the conclusion chunk (i.e. an action recommendation) it is linked to. As the box shows, the condition chunk and the conclusion chunk it is linked to form a rule. If the conclusion chunk of this rule is selected, then the conclusion chunk would activate the action nodes it is linked to in the bottom-level. Lastly, these action nodes would activate the base level functions which they are associated with (i.e. muscle movement, memory management, etc.).

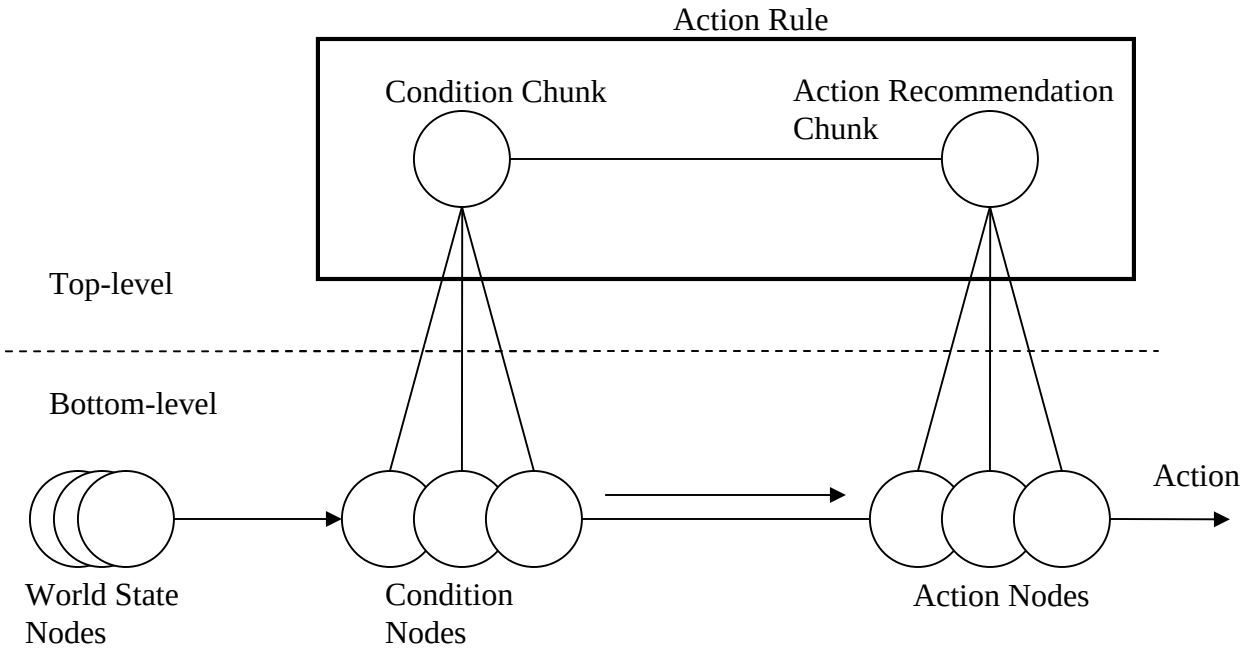
Learning and Reinforcement

In order for an agent to continuously exist in a changing environment, it has to be able to learn. The process of learning involves altering the top-level rules and bottom-level network connections so as to produce new (and hopefully beneficial) behavior. Learning in the bottom-level can be achieved through several different methods. The first of such methods is known as Q-learning (Watkins, as cited in Sun, 2003). This is an algorithm which uses estimation of reinforcement to influence behavior. The current state and the state after a particular action is executed are compared. A value is then calculated which represents how beneficial this action would be to the agent. The agent determines how beneficial an action is by calculating reinforcement (Sun, 2003). Reinforcement is the result of a function which factors in the goals and desires of the agent, as well as the state of the world (including the agent). Sun explains, "reinforcement signals are derived from measuring the degree of satisfaction of the drives and the goals of the agent," (2003, p. 111). By comparing how close a state is to the goal of the agent, the reinforcement function produces a value that represents how desirable a state is. A state that is closer to achieving a goal than another state would be more desirable and so have a higher reinforcement. The action recommendation that leads to the state with the highest reinforcement would be the "best" and therefore be chosen. This is much like the psychological concept of operant conditioning, where reinforcement in the environment increases or decreases behavior. In Q-learning, the total estimated reinforcement before and after an action is performed is used to develop Q

values. The Q values represent the predicted total reinforcement and thus provide a method for determining sequential actions (Sun, 2003).

In addition to Q-learning, there are other methods of learning within the action-centered subsystem. One such method is that of top-down assimilation (Sun, 2003). In this, externally given explicit knowledge that has been encoded in the top-level is used to generate bottom-level networks. The purpose of this is to take top-level rules which have been used repeatedly and represent them in the backpropagation networks of the bottom-level. By taking this knowledge and representing it in the bottom-level, the agent makes the rule almost automatic, and thereby more efficient (Anderson, Dreyfus and Dreyfus, as cited in Sun, 2003). An example of this would be learning to use a screwdriver. When first used, a person would have to use their top-level knowledge to formulate a procedure (i.e. hold screwdriver, put to screw, twist handle, etc.). Initially, this might be difficult for an individual to perform. Mistakes might be made (i.e. turning handle in wrong direction), but reinforcement from correct actions would strengthen the proper sequence. With continued practice, the procedure could be assimilated into the bottom-level, so that using the screwdriver would be almost thoughtless. This can be seen by observing a novice who must concentrate fully on the task and comparing them to an expert (like a carpenter) who is able to perform the procedure while talking to someone else.

The following figure may help in understanding top-down assimilation. It is based on the figure used before to demonstrate an action rule, but has been modified to show the result of top-down assimilation.



In this figure, you can see that, after top-down assimilation, the condition nodes and action nodes of a rule have become linked directly on the implicit level. Now when the condition nodes are activated, they immediately activate the linked action nodes. This process is more efficient by saving time and mental effort. With the condition and recommendation linked like this, the top-level action rule can eventually be removed.

Yet another learning method in the action-centered subsystem is that of imitative learning (Sun, 2003). This involves an agent observing another agent performing a task and then trying to mimic it. As opposed to top-down assimilation, this process is bottom-up, meaning bottom-level knowledge is used to formulate top-level knowledge. Sun explains that for the bottom-level to initially acquire the knowledge, there needs to be a unique reinforcement function (2003). This reinforcement function assigns value to those actions, in a given

state, which are the same as another agent's actions, in the same given state. This special reinforcement means that actions that imitate another agent are more likely to be selected. Notice that they are more likely to be chosen, not necessarily chosen. This is so that the agent will not automatically perform an imitative action rather than a different, already known action which provides a better result. So imitation is encouraged, but not at the sacrifice of performance.

Once the action is imitated by the agent, reinforcement is sensed and the backpropagation network is updated. Specifically, the connections between the nodes which were activated by the imitative action are strengthened. The more the given state is paired with the imitated action, the stronger the connections will become and the more likely the action will be performed the next time the given state occurs. Now that the action is stored in the bottom-level, the top-level can extract rules based on it. Imitative learning can also be applied directly to the top level, meaning that rules are created which are based on the direct observation of an agent by another agent (Sun, 2003). This can be thought of as a more conscious form of imitation, where the agent actively decides to mimic the actions of another agent.

The process of rule extraction from the bottom-level to the top-level of the action-centered subsystem is handled by the Rule-Extraction-Refinement algorithm (Sun, 2003). This algorithm takes a successful action that was recommended by the bottom-level, and develops a top-level rule that represents it. The new rule is then revised based on future states of the world. Depending on if the rule is successful or not, it will be expanded or narrowed. The agent will

attempt to apply a successful rule to broader situations, meaning more opportunities for it to be applied. A rule that is not successful, however, may be too general and so must be focused more (Sun, 2003). Focusing a rule involves removing a value from its condition so that it will have a decreased chance of being activated. A rule is discarded when all the values in the condition have been removed, since there is no state that will ever be able to activate it (Sun, 2003). Mathematical functions are used to evaluate the extracted rules and determine if they should be revised.

Another type of learning that exists in the top-level of the action-centered subsystem is Independent Rule Learning (Sun, 2003). This is a slightly different method of generating rules for the top-level. Instead of using the bottom-level to form new rules, Independent Rule Learning formulates them through a sort of random generation. The algorithm, which is similar to the one for extracting rules, uses pre-specified templates to create subsets of rules. These subsets are then tested to see if they prove to be useful. Subsets which are found to be ineffective are deleted (Sun, 2003). As with the rules generated by rule extraction, these rules can be adjusted so that they apply to more universal or more specific situations.

Fixed rules exist in the top-level of the action-centered subsystem and represent knowledge that has been given from a source outside of the agent or acquired from past situations (Sun, 2003). Rules of this type can also represent instinctual knowledge that results from the evolutionary process. In this regard, fixed rules are similar to the fixed networks of the bottom-levels. With fixed rules,

top-down assimilation can apply the top-level knowledge to the bottom-level. As with Q-learning, repeated use of the fixed rules strengthens the connections of the nodes in the bottom-level. Eventually, the appropriate backpropagation network can automatically generate the same results, but more efficiently (meaning less mental effort). Sun notes that fixed rules are more versatile and can represent more complex concepts than other rules (2003). He also explains that this is akin to the psychological concepts of "schemas" (Arbib, Dretcher) and "abstract behaviors" (Mataric, as cited in Sun, 2003).

Action Selection

As explained before, both the top-level and bottom-level of the action-centered subsystem control the actions of the agent. Both levels are fed the current state information and each produces recommended actions which are appropriate. Not all of the recommended actions can be performed, however, so the agent must pick among them. There exist several possibilities for integrating and then selecting from these action recommendations. In one method, top-level rules are analyzed and if there are no appropriate ones, a recommendation is selected from the bottom-level (Sun, 2003). In another method, the recommendations from both levels are combined and then a selection is made. While not covered here, Sun's tutorial (2003) explains these methods as well as the mathematical calculations involved. For the purpose of this paper, it is sufficient to know that the recommendation chosen, regardless of being from the top-level or bottom-level, is deemed the choice which will produce the most

favorable situation for the agent, given the current internal and external state of the world.

Working Memory

Another important part of the CLARION architecture is the working memory. This serves as a medium between the cognitive processes and the long-term memory store. In humans, this is analogous to short-term memory. It is a temporary storage for a limited amount of data which is currently being used by the action-centered modules. The short-term memory of humans is also of a limited size, commonly believed to be around seven “items”. Storing information in the working memory is “for the express purpose of facilitating subsequent action decision making and/or reasoning,” (Sun, 2003, p. 50). Pieces of information are pulled from the long-term memory store, placed in the working memory, used, and then removed. This interaction with the working memory is achieved by working memory actions activated by the action-centered subsystem (Sun, 2003). Since it only holds a limited amount of data, information must be removed in order to allow new information to be inserted. Also, the items in working memory are susceptible to decay. This is similar to the concept of memory decay in humans. As time passes, unused items in working memory lose strength and eventually are removed. In order to keep an item in working memory, it has to be refreshed by a working memory action. Also, it is important to note that the designers of CLARION have made working memory “used solely for action-decision making, not for general memory access,” (Sun, 2003, p. 53).

In CLARION, working memory is a separate system, but is considered a part of the action-centered subsystem.

Goal Structure

In order for an agent (human or cognitive model) to complete complex tasks, there needs to be a process dedicated to directing the agent's actions in completing the task. This process would need to keep track of the current goal the agent is trying to achieve, as well as if the goal has been reached or not. In addition, it would also need to be able to manage multiple goals as well as sub-goals. Returning to the previous example of a human disassembling a car engine, the agent must have an overall goal of taking it apart, as well as many sub-goals that must be completed first (like removing the air filter, removing the intake manifold, etc.). Thus, a goal structure is required so as to guide the agent's actions in a constructive manner (Sun, 2003). Sun gives an example of a goal structure, the goal stack (Anderson, Anderson and Liebre, as cited in 2003). Goal actions, from the action-centered subsystem, are the ways in which the agent can manipulate the goal structure. As with working memory, the goal structure is a separate system, but considered a part of the action-centered subsystem.

The goal stack is a way of organizing goals and subgoals into an ordered list. When a main goal is given to the goal stack, it is placed in the list. If the main goal cannot be accomplished, then a subgoal is created. This subgoal's purpose, in the end, is to make the main goal achievable. It is put on the goal

stack, by a goal action, on top of the main goal and becomes the currently active goal. The agent then attempts to satisfy this goal. If it cannot, another subgoal is created which serves to aide in achieving the previous subgoal. It is put on the top of the stack (above the previous goal) and becomes the currently active goal. This continues until there is a subgoal which can be achieved or cannot be broken into smaller goals. If the latter is the case, then the current goal (or subgoal) fails. In either case (success or failure), the current goal is removed from the top of the stack and the next goal becomes the active one. The process of checking the attainability of a goal is repeated with respect to the now current goal. This continues until the main goal is a success or a failure.

As a simple example, let's say the main goal is disassembling the car engine. It is placed on the goal stack and checked to see if it can be satisfied. It is not, so a subgoal (say, remove the air filter) is created and put on the stack as the current goal. This goal is checked and fails since the air filter cannot be removed. A new sub goal is created, namely, remove the wing-nuts securing the air filter. It is placed on the stack and becomes the current goal. This goal is attainable since there is an action which can satisfy it. The action is performed and the current goal (remove wing-nuts) is deleted from the top of the stack, making the previous goal (remove air filter) the current one. The process is repeated as before. Assuming nothing is rusted together (thereby causing a goal to be unattainable), the car engine will be taken apart piece by piece until it is disassembled. This example shows how a goal structure is necessary in order for the actions of an agent to be productive.

Another goal structure that CLARION can use is the goal list. This structure is a linear list of goals, like the goal stack. Also like the stack, goal actions are required in order to alter it. It is different from the goal stack, however, in that each goal item can be accessed randomly (Sun, 2003). The major difference between the two structures is in how a goal becomes the current goal. When there is more than one goal in the list, each item competes against each other to become the current goal. In addition, the “strength” of the each goal item fades over time, so older goals are weaker than newer goals. In this way, the goal list generates a procedural order much like the goal stack (Sun, 2003). If a new goal has to be generated, because the existing ones are not attainable, it is more recent and therefore stronger than the previous goals. It remains the current goal until it is completed or a newer goal is created and therefore stronger than it. As active goals are completed, they are deleted. This allows the most previous goal to be the strongest and, thus, become the current goal.

The accessibility of older goals means the goal list is more versatile than the goal stack. It allows switching between goals as necessary. If the environment changes so that an older goal is now achievable, the strength of said older goal will increase and (possibly) become the current goal. This resumption of older tasks is not possible in the goal stack without first completing the goals on top of it. Like the goal list, humans are able to switch between goals as desired or needed. This aspect of the goal list makes it more similar to human

behavior than the goal stack and therefore a better choice for a goal structure (Sun, 2003).

Non-action-centered subsystem

Structure

Having described how the action-centered subsystem is constructed, the structure of the non-action-centered subsystem will be addressed. While the action-centered subsystem encompasses the processes which govern the actions of an agent, the non-action-centered subsystem contains what can be considered general knowledge (Sun, 2003). Sun explains that general knowledge includes what is known as "semantic" memory, "i.e., general knowledge about the world in a conceptual, symbolic form," (2003, p. 8). He also points out that semantic memory is not solely in the top-level, but included in the bottom-level as well. This is the type of knowledge which includes ideas, objects, and facts. For example, if you thought about an apple, you would probably have a mental image of an apple form in your mind. This is because you have stored knowledge in your head which represents your concept of "an apple." The word "apple," a mental picture of an apple, and the taste of an apple, are all pieces of information that you have stored in your mind that symbolize an aspect of the real-world object. These specific attributes would be located in the bottom-level. By being linked to the top-level knowledge, these attributes are associated with each other and form the mental concept of "an apple."

Top-level

The top-level of the non-action-centered subsystem within CLARION is known as the general knowledge store (Sun, 2003). This is where the explicit, declarative knowledge is located. It is similar to the top-level of the action-centered subsystem in several ways. One way is that the top-level consists of rules and chunks. The chunks of the general knowledge store are linked to dimension/value pairs, just like in the action-centered subsystem. Whereas chunks were connected to action recommendation chunks in the action-centered subsystem, declarative knowledge chunks in the non-action-centered subsystem are connected to other declarative knowledge chunks (Sun, 2003). These connections between chunks are called associative rules. Activating a chunk will in turn activate the conclusion chunks it is linked with via associative rules, as well as the dimension/value pairs in the bottom-level. Likewise, activating the dimension/value pairs in the bottom-level network will activate the linked chunks in the top-level. In this way, explicit ideas are linked to other ideas.

Bottom-level

The bottom-level of the non-action-centered subsystem consists of networks that represent implicit declarative knowledge (Sun, 2003). As in the action-centered subsystem, it consists of networks of dimension/value pairs. These pairs can become mapped to each other so that an input of certain dimension/value pairs produces an output of associated dimension/value pairs. One possible use of these associative memory networks is to provide a method

of predicting future states (Sun, 2003). Training would involve linking the dimension/value pairs of the input to a specific state. Then, if only a portion of the dimension/value pairs are supplied to the input, the links would indicate the most likely state.

These networks could also serve as the location of knowledge which has been used repeatedly and often. By matching a given input to a given output, the knowledge would be almost automatic. For instance, this might apply to word recognition. An experienced reader does not have to consciously think about each word in the sentence and remember what it is. The sensory input is sent by the action-centered subsystem to the associative memory networks. The dimension/value pairs of the input would be linked to the dimension/value pairs of the chunks which represent that word. This would explain why those learning a new language must perform much more mental effort in recalling data on a given word when compared to those who are fluent.

Reasoning

Reasoning is the process by which new ideas or relationships are formulated from already known ones. In order for reasoning to work, ideas must exist as well as relations between these ideas. Thought can then move from idea to idea along these associations, until a desired conclusion is reached. Reasoning in CLARION involves using associative rules and chunks to draw conclusions. At the beginning of the reasoning process, input is sent to the non-action-centered subsystem (Sun, 2003). Depending on the type of input, it is

either sent to the top-level or the bottom-level. Chunks that are sent as input activate chunks in the general knowledge store, whereas dimension/value pairs sent to the associative memory networks activate other dimension/value pairs. Once the respective pieces of knowledge (chunks or pairs) are activated, they proceed to activate their associated chunks and pairs. In the end, an input to either the general knowledge store or the associative memory networks will activate items in both the top-level and the bottom-level.

In the associative memory networks, the triggered dimension/value pairs take part in associative mapping, where the product is activation of output dimension/value pairs (Sun, 2003). Meanwhile, the associative rules for all the activated chunks in the top-level are executed and conclusion chunks are the result. This is referred to as forward chaining reasoning. Once the conclusion chunks and the output dimension/value pairs are determined, another round of activation occurs. This time, the conclusion chunks activate their related dimension/value pairs in the bottom-level and the output dimension/value pairs activate the associated chunks in the top-level. It is then possible for the process to be repeated with the new chunks and dimension/value pairs.

There is an optional method of reasoning in CLARION called similarity-based reasoning (Sun, 2003). This reasoning uses inference to create associations between chunks. A chunk is compared to another chunk using an algorithm. If they are sufficiently similar, one chunk is said to infer another. When reasoning occurs and chunks are activated, the inferred chunks may also be activated. This allows a broader spectrum of conclusions to be drawn.

Learning

Since the non-action-centered subsystem encompasses both top-level and bottom-level knowledge, there are separate processes for learning in each. At the top-level, explicit knowledge can be encoded based on information attained from outside the agent (Sun, 2003). This information would be read by the senses and sent to the action-centered subsystem. There, a decision would take place to encode the information into memory. In order for the non-action-centered subsystem to encode externally given knowledge, the action-centered subsystem must first convert it into a piece of explicit knowledge. The associative rule or chunk can then be sent to the non-action-centered subsystem and saved to memory. In addition, the action-centered subsystem can encode states and actions performed as chunks, which are also then sent to the non-action-centered subsystem (Sun, 2003). By storing the states and actions experienced by the agent, it can have a reference to draw on in future decision making.

Other explicit knowledge can be derived from the bottom-level networks. This bottom-up learning is much like the rule extraction of the action-centered subsystem (Sun, 2003). Within the associative memory network, associative mapping strengthens the connections between input and output dimension/value pairs. This also creates an association between the chunks in the top-level general knowledge store. An associative rule is created where the chunk of the input pairs is the condition, and the chunk of the output pairs is the conclusion.

As in the action-centered subsystem, a rule was created to associate the chunks involved in a bottom-level network mapping.

The learning of implicit declarative knowledge requires the use of a special memory known as episodic memory. Simply, this memory stores information about experiences (Sun, 2003). As actions are taken and goals met, new information is added to the episodic memory, for reference later. This memory is stored separately, as explained by Sun, “we feel that the distinction between episodic memory and semantic memory in CLARION is justified, because of the need to separate knowledge regarding specific experience and knowledge that is more generic and not experience-specific,” (2003, p. 8). So the long-term memory stores symbolic representations of objects (or concepts) with no regard to context, while the episodic memory stores information regarding how and when the agent interacted. In addition, the episodic memory contains the associations used and learned in the past. These associations are randomly chosen when implicit learning is to be initiated (Sun, 2003). They are then fed to the associative memory networks as the input and output. Those associations which are beneficial are then trained into the network.

Motivational subsystem

The motivational subsystem is the part of cognition which supplies the reasons behind an agent’s actions. While goals are used to direct actions on a task-based level, drives are what determine the actions overall (Sun, 2003). They are the reason behind why certain goals are chosen as the active goal.

Drives are related to needs, namely, drives are the desire to meet an agent's needs. The needs of an agent are usually considered those things which are necessary for the agent to survive. One such need, at least for a human, is the need for water. As the need for water increases, so does the drive to satisfy it. Eventually, the motivational subsystem will deem it necessary to acquire water and so recommend goals to achieve this.

Some drives are more important than others, and so they can be arranged in a hierarchical manner (Sun, 2003). Drives which are higher in the hierarchy are more likely to be satisfied at a given moment than a lesser drive. An example of this would be the need to avoid danger and the need to reproduce. An agent would not reproduce if it would put it in serious danger. The more important need of self-preservation overcomes the need to mate.

The motivational subsystem also takes into account the accessibility of a need when it chooses goals (Sun, 2003). This means that the object that the drive requires must be available to the agent in order for the drive to be satisfied. Setting a goal to satisfy a need will not succeed if the object is not present. Thus, different goals may have to be set so that the agent can find the object in need, and then satisfy the drive. Basically, if the agent requires water, but there is no water, than the agent must first find water. As you can see, depending on if the need is available or not will determine the goals set. Also, if a need is not available but not drastically desired, then the motivational subsystem may try to satisfy other needs which are accessible (like eating food). Sun refers to Tyrell when he explains the requirements of drives:

- “Proportional activation” – How strong a drive to satisfy a need is should be related to how badly the need is required. The more desperate the need, the stronger the desire to satisfy it.
- “Opportunism” – The availability of a need satisfier should influence the decision process. In other words, needs that can be satisfied immediately should have more consideration than other needs.
- “Contiguity of actions” – An agent should finish the process of satisfying a drive before switching to another.
- “Persistence” – Drives should be satisfied to the point that maximizes the time between satisfying. This means that an agent should not minimally meet a need and then proceed to another. If this were the case, the agent would continuously be moving between needs and only satisfying a bit at a time. Thus, the agent should fill itself to a good amount before moving on.
- “Interruption when necessary” – There are cases, however, where the process of meeting a need should be stopped and another one started. A good example is that of an animal drinking at a pond. If it is attacked, the need to avoid danger should become the prominent drive.
- “Combination of preferences” – The needs of all the drives should be considered when setting goals so that multiple drives can be met at once. Sometimes it is best for an agent to satisfy many needs marginally instead of one need excessively. For example, it

> might be best for a hungry animal to stay in its present location where there is food and water, instead of going to a place where there is more food but no water. (2003)

Using these requirements, the designers of CLARION have created a motivational subsystem which properly represents the need-satisfying behavior of humans (as well as animals).

Meta-Cognitive subsystem

The meta-cognitive subsystem is the main controller of cognition. It regulates the activities of all the other subsystems (Sun, 2003). It does this so as to make the interaction of these subsystems more efficient. For example, it is this subsystem which adds goals to the goal structure in response to the drives of the motivational subsystem. The meta-cognitive subsystem takes into account the various drives and their strengths, and determines which one should be the current focus. It can even combine drives so that multiple needs can be met, pending the situation allows it (Sun, 2003). The meta-cognitive subsystem also handles the passage of information from the action-centered subsystem to the non-action-centered subsystem. When a memory action is triggered in the action-centered subsystem, the meta-cognitive subsystem transfers the information to the non-action-centered subsystem. The meta-cognitive subsystem governs which learning methods are to be employed, as well as when (Sun, 2003). Depending on the situation, certain learning methods may be applied as opposed to others. In addition, parameters of the action-entered and

non-action-centered subsystems are set by it (Sun, 2003). By adjusting the parameters of the separate subsystems, the meta-cognitive subsystem can modify the individual processes in each, increasing the overall efficiency of the system. These are just some examples of the responsibilities of the meta-cognitive subsystem. By controlling the activities of the individual subsystems, the meta-cognitive subsystem is able to join them into a functional model of cognition.

One other aspect of the meta-cognitive subsystem is that it filters information when it is passed between subsystems (Sun, 2003). It does this by suppressing certain dimensions and letting others go through. Suppressing a dimension/value pair is done by removing the activation supplied to it, thereby preventing it from activating anything else it is connected to. Other dimension/value pairs can be focused onto by increasing their activation. The decisions as to what to suppress are made by the meta-cognitive subsystem based on information gathered from various sources. The working memory, the current goal, and drives are all examined, as well as other locations (Sun, 2003). Based on these factors, the meta-cognitive subsystem can choose what information is most pertinent and then focus the cognition of the agent to that. This prevents unnecessary cognitive effort to be performed on information that is not currently relevant. Suppressing dimensions is just another way the meta-cognitive subsystem can direct cognition.

Yet another function of the meta-cognitive subsystem is the calculation of reinforcement resulting from drives (Sun, 2003). It observes the current state of

the agent and determines if the current action satisfies one of the needs. If a need is satisfied by the action, reinforcement is generated and applied. Thus, actions which satisfy needs are more likely to be chosen and used by the agent.

As you can see, the meta-cognitive subsystem can control aspects of every other part of cognition. It can make adjustments, focus attention, defocus attention, set goals, and provide reinforcement to behavior. This is done so that the separate subsystems can function together as an efficient whole.

Conclusion

All of the subsystems of CLARION have been described as well as their approximate functioning. The reader can see that each subsystem is specialized and intended for a specific purpose. This is just like in the human brain, where there are separate parts which handle specific tasks. The action-centered subsystem handles the actions of the agent, the non-action-centered subsystem handles the general knowledge of an agent, the motivational subsystem contains the innate drives of the agent, and the meta-cognitive subsystem regulates the flow of them all. Information about the world is taken in by the senses, filtered by the meta-cognitive subsystem, and then sent to the action-centered subsystem. There, possible actions are determined based on the state of the world. From these actions are chosen those that will benefit the agent, based on the goals, drives, and reinforcement. The actions are performed, whether it is an external action, accessing a working memory, or retrieving data from long-term memory. If long-term memory is to be accessed, the meta-cognitive subsystem filters the

information as it goes from the action-centered subsystem to the non-action-centered subsystem. Meanwhile, learning is taking place in both subsystems in various forms. Once the actions are complete, the agent can take in the new state of the world and yet again decide how to behave. These parts interact together to form a cognitive architecture.

Each aspect of the CLARION architecture is designed to represent an aspect of cognition within the human mind. These are integrated in the most accurately known way. However, it is far from perfect. There still exist many unknowns in human cognition, so CLARION will have to continue evolving. Through continued experimentation, both human and simulated, more information can be gathered as to the workings of an intelligent agent. This can be applied to modifying CLARION so that it more accurately represents the human mind.

Works Cited

Sun, R. (2003). *A Tutorial on CLARION 5.0*. Retrieved on October 15, 2003 from Rensselaer Polytechnic Institute, Department of Cognitive Science Web site: http://www.cogsci.rpi.edu/~rsun/sun.tutorial.pdf