A Tutorial on CLARION 5.0

Ron Sun

July 22, 2003

Note that those sections marked with "*", as well as appendices, may be skipped on first reading.

Contents

Chapter 1

Introduction and Motivation

CLARION, which stands for *Connectionist Learning with Adaptive Rule Induction ON-line*, was proposed in its earlier forms in Sun (1997, 1999, 2000) and Sun et al (2001). It is a cognitive architecture that incorporates the distinction between implicit and explicit processes and focuses on capturing the interaction between these two types of processes.

In this chapter, we outline the model. We first discuss the representations involved in the model, and then the learning processes involved. Finally, we summarize the basic postulates of the model. (In later chapters, we will develop technical details of the model.)

1.1 Representation

Let us first consider the representational forms that need to be present in the cognitive architecture. We take note of the fact that the inaccessible nature of implicit knowledge is best captured by subsymbolic, distributed representation provided, for example, by a backpropagation network (Rumelhart et al 1986). This is so because distributed representational units in the hidden layer(s) of a backpropagation network are capable of accomplishing computations but are subsymbolic and generally not individually meaningful (Rumelhart et al 1986, Sun 1994); that is, they generally do not have associated semantic labels. This characteristic of distributed representation, which renders the representational form less accessible, accords well with the relative inaccessibility of implicit knowledge (Reber 1989, Seger 1994, Cleeremans et al 1998).

However, it is generally not the case that distributed representation is not accessible at all, but it is clear that distributed representation is less accessible, not as direct and immediate as localist representation. Distributed representation may be accessed through indirect, transformational processes, as opposed to the direct access of localist/symbolic representation. As Kirsh (1990) put it, "explicitness [of representation] really concerns how quickly information can be accessed..... It has more to do with what is present in a process sense, than with what is present in a structural sense". Likewise, in Hadley (1995), the notion of "immediate usability" was used in defining explicit representation. Sun (2001) provided a further explication of this idea utilizing the notion of Kolmogorov complexity.

In contrast, explicit knowledge may be best captured in computational modeling by symbolic or localist representation (Clark and Karmiloff-Smith 1993), in which each unit is more easily interpretable and has a clearer conceptual meaning (namely, a semantic label). This characteristic of symbolic or localist representation captures the characteristic of explicit knowledge being more accessible and more manipulable (Smolensky 1988, Sun 1994). Again, accessibility here refers to the direct and immediate availability of mental content for the major operations that are responsible for, or concomitant with, consciousness, such as introspection, forming higher-order thoughts, and verbal reporting, as well as meta-level control and manipulation.

The dichotomous difference in the representations of the two different types of knowledge leads to a two-level architecture, whereby each level uses one kind of representation and captures one corresponding type of process, implicit or explicit. Sun (1994, 1995, 1999, 2002), Dreyfus and Dreyfus (1987), and Smolensky (1988) provided a great deal of theoretical arguments for such two-level models, which we will not repeat here (the reader is referred to these publications).

At each level, there may be multiple modules, including both *action-centered* modules and *non-action-centered* modules (Schacter 1990, Moscovitch and Umilta 1991). Each of them resides in a separate subsystem (the action-centered subsystem and the non-action-centered subsystem, respectively). 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 inaccessible (directly), and non-action-centered knowledge (roughly, declarative knowledge) is not necessarily accessible (directly). Although it has been argued by some that all procedural knowledge is inaccessible and all declarative knowledge is accessible, such a clean mapping of the two dichotomies is untenable in our view.

Thus, beside the distinction and separation of implicit and explicit processes, we also need the distinction and separation of action-centered and non-action-centered processes. Non-action-centered knowledge does not involve action recommendations, and therefore does not necessarily go in only one direction—retrieval can often be accomplished in more than one ways. For example, if a chunk of non-action-centered knowledge consists of a set of attributes, then we should be able to retrieve it based on any subset of these attributes. On the other hand, action-centered knowledge is usually one-way only — from current state assessment to action recommendations, and usually does not go in the other direction. [1]

Let us look into action-centered knowledge first. At the bottom level, action-centered knowledge is highly modular: A number of backpropagation networks co-exist, each adapted to a specific modality, task, or stimulus type. This assumption is consistent with the well known modularity claim made by many (e.g., Fodor 1983; Karmiloff-Smith 1986; Cosmides and Tooby 1994).

At the top level, action-centered knowledge can reside in different modules, in correspondence with the bottom-level structure. But in general, as we know, explicit knowledge at the top level is less modularized compared with implicit knowledge at the bottom level (Fodor 1983). That is, explicit knowledge may be viewed as residing in a more centralized and less compartmentalized structure.

There is also a working memory for keeping information on a temporary basis. Jackendoff (2002) argued that working memory cannot just consist of the nodes in long-term memory that are activated in current processing. [2] The view of working memory as a separate component seems more appropriate (Baddeley 1986). In our view, working memory is a temporary

[1] Therefore, the difference between the two kinds may be discerned through, for example, contrasting symmetric vs. asymmetric retrieval (Anderson 1983, 1993, Johnson 1998).

[2] For example, in the phrase "the big star's beside the little star", the word "star" occurs twice. These two instances cannot be kept distinct if each consists simply of an activation of the entry for "star" in the long-term memory.

and transient storage used for facilitating action decision making. It may point to long-term memory, and serve as its substitute on a temporary basis.

In addition, there is also a goal structure (for example, a goal stack as in Anderson 1993 and Anderson and Lebiere 1998). In a way, we may view the goal structure as part of the working memory. The goal structure is used to organize the activities of an agent: It organizes various drives, desires, needs, and motivations, and their interactions in an agent (Hull 1943, Tyrell 1993). It then directs the actions of the agent accordingly, so that the activities of the agent are consistent with its own drives, desires, and needs. Thus, the goal structure is a necessary part of a cognitive architecture.

On the other hand, non-action-centered modules (at both levels) represent more static, declarative, generic types of knowledge. The knowledge there includes what is commonly referred to as "semantic" memory, i.e., general knowledge about the world in a conceptual, symbolic form (Quillian 1968, Tulving 1972), although implicit forms of "semantic memory" modules are also included (in the bottom level).

Non-action-centered modules serve as a long-term memory system that keeps track of all non-action-centered knowledge, as well as episodic information. Therefore, it can be involved in various kinds of reasoning. Although it was claimed by some that such involvement was the function of short-term memory, it is justified to involve long-term memory: Given the huge quantity of information that is likely needed to perform complex tasks, there is no other way but to involve long-term memory in on-going processing. See, for example, Ericssoon and Kintsch (1995) for similar arguments.

The co-existence of episodic memory and semantic memory in this architecture deserves some comments. Episodic memory is not concerned with current action decision making; therefore, it is not part of the action-centered modules. Although considered part of the non-action-centered subsystem, it is distinct from other non-action-centered modules (that is, the semantic memory modules). [3] We feel that the distinction between episodic memory and semantic memory in CLARION is justified, because of

[3] This distinction has been argued for by some (e.g., Tulving 1972, Bower 1996), while objected to by others (Anderson 1983, Humphreys et al 1989).

the need to separate knowledge regarding specific experience and knowledge that is more generic and not experience-specific. [4] See Tulving (1972) for a similar view. [5]

In both action-centered and non-action-centered modules, explicit knowledge is represented as rules and chunks. Rules include action rules (in action-centered modules) and associative rules (non-action-centered rules, in non-action-centered modules). For representing these rules, there are chunks involved, which describe conditions and conclusions of rules.

Reasoning is evidential: Humans often evaluate available evidence and sum up evidential support when reaching conclusions (Zadeh 1988, Smithson and Oden 1999, Sun 1995). Since in CLARION reasoning is performed based on rules, we term the evidential support for a conclusion reached based on a rule *rule support*. There are various forms of "rule support" available in the literature, ranging from Dempster-Shafer evidential calculus, through fuzzy logic (Zadeh 1988, Sun 1995), to probabilistic reasoning (Pearl 1988).

Moreover, human memory is recency-based. Anderson (1993) showed that memory availability should be linked to the odds of an item would be needed, which in turn could be determined by a power function of the lapsed time since the previous access of that item. We believe that the retrieval of both rules and chunks (i.e., all explicit knowledge) should be subject to this calculation, which is termed *base-level activation* (or simply BLA, following Anderson 1993).

In addition, we need measures of effectiveness, or usefulness, of rules. That is, we need some kind of utility measure (Anderson 1993, Luce 2000), which captures rationality in decision making. Such a measure decides whether we should keep a particular rule or not, or use a particular rule or not.

Summarizing the above discussion, there are plenty of choices in terms of numerical measures associated with explicit knowledge, and possibilities for their use. In CLARION, we adopted some simple forms of the above

[4] Thus, there are in fact three separate memory systems in CLARION: procedural (action-centered), episodic, and semantic (non-action-centered).

[5] Note also that we include both implicit and explicit forms of semantic memory and both implicit and explicit forms of episodic memory (at the bottom and the top level of the non-action-centered subsystem, respectively).

schemes. We made slightly different choices for action-centered knowledge and non-action-centered knowledge. See details later.

In addition to action-centered and non-action-centered modules, there is yet another major component in CLARION (the motivational subsystem), which is responsible for motivational states, or "drives" and "goals". In CLARION, drives are the motivational force behind action decision making (as well as behind non-action-centered processes). Goals are abstracted from drives, to provide specific and tangible motivation and context for actions. Actions are selected based on their "values" with regard to the current goal. Such "values" (or evaluations) of actions are the motivation behind decisions to take specific actions. Goals may be set based on drive states (by a meta-cognitive subsystem). Parameters for action decision making and parameters for reasoning may also be determined based on motivational states (by the meta-cognitive subsystem).

In sum, on the one hand, action-centered modules include implicit bottom-level action decision networks, explicit top-level action rule groups, working memory, and goal structure. These action-centered modules are the primary "anatomic" components of the mind, as they are the ones that decide what an agent does every step of the way. Non-action-centered modules, on the other hand, include explicit general knowledge store (i.e., explicit semantic memory), and implicit associative memory networks (i.e., implicit semantic memory). The non-action-centered modules supplement action-centered modules in a variety of ways. These two sets of modules constitute two subsystems. They operate under the influence of motivational and meta-cognitive processes.

1.2 Learning

From the preceding analysis, some further questions may arise. First of all, in relation to the bottom level,

(i) How should an agent decide an action based on information about the current state (the input)? In other words, what are the parameters? And what is the mapping function from the current state and the current parameters to the the action?

(ii) Similarly, how should an agent produce an associated output for a given input when reasoning? What is the mapping function from the

input and the current parameters to the the output?

(iii) How should an agent adjust parameters to increase or decrease the likelihood of certain output in accordance with feedback (e.g., reinforcement)?

There are a few possible ways of answering these two questions computationally. One way of implementing a mapping function is to use a multi-layer neural network (e.g., a three-layer backpropagation network). Such networks may map states to evaluations of actions, or input to their associated output. [6] They involve distributed representation, which may aptly capture the nature of implicit knowledge (as discussed earlier).

Adjusting parameters of this mapping function to change output (that is, learning implicit knowledge) may be carried out in ways consistent with the nature of distributed representation (e.g., as involved in backpropagation networks). Often, reinforcement learning can be used (Sutton and Barto 1995), especially Q-learning (Watkins 1989) (implemented using backpropagation networks). In this learning setting, there is no need for external teachers providing desired input/output mappings. On the other hand, in the learning settings where desired input/output mappings are available, straight backpropagation—a supervised learning algorithm—can be used for a network (Rumelhart et al 1986). Supervised learning requires the a priori determination of a uniquely correct output for each input. [7]

These (implicit) learning methods may be cognitively justified. For instance, Shanks (1993) showed that human instrumental conditioning was best captured by associative memory models (i.e., neural networks). Cleeremans (1997) argued at length that implicit learning could not be captured by symbolic models but neural networks. Sun (1997, 1999) made similar arguments.

Next, explicit knowledge at the top level can also be learned in a variety of ways (in accordance with localist representation used there).

[6] For example, for representing action-centered implicit knowledge, a network maps states to "values" (evaluations) of actions, and various weights in different layers of the network can be adjusted to change action evaluations. Actions are selected based on "values" of different actions in a given state.

[7] For example, when explicit knowledge is available at the top level, corresponding implicit knowledge may be learned through assimilating the explicit knowledge by using supervised learning.

Because of the representational characteristics, one-shot learning based on hypothesis testing is preferred (Bruner et al 1956, Busemeyer and Myung 1992, Nosofsky et al 1994, Sun et al 2001). With such learning, an agent explores the world, and dynamically acquires representations and modify them as needed, reflecting the dynamic (on-going) nature of everyday activities (Heidegger 1927, Vygotsky 1962, Sun 2002).

Some questions arise in this regard:

- How should an agent decide when to acquire an explicit representation?
- How should an agent decide when to refine an explicit representation?
- What are possible ways by which an agent acquires or refines an explicit representation?

We notice that the implicit knowledge already acquired in the bottom level may be utilized in learning explicit knowledge, through *bottom-up learning* (Sun et al 2001). The basic idea of bottom-up learning of action-centered knowledge is as follows: If an action chosen (by the bottom level) is successful (i.e., it satisfies a certain criterion), then a rule is extracted. Then, in subsequent interactions with the world, the rule is refined by considering the outcome of applying the rule: If the outcome is successful, the condition of the rule may be generalized to make it more universal; if the outcome is not successful, then the condition of the rule should be made more specific and exclusive of the current case. [8] This is an on-line version of hypothesis testing processes studied (in different contexts) by, for example, Bruner et al (1956) and Nosofsky et al (1994).

Conceivably, other types of learning of explicit knowledge are also possible, such as hypothesis testing without the help of the bottom level.

Moreover, once explicit knowledge is established at the top level, it may be assimilated into the bottom level. The assimilation process, known as *top-down learning*, may be carried out in a variety of ways. See details in later chapters.

1.3 Basic Postulates

Here is a summary of the basic theoretical hypotheses of the CLARION architecture as discussed in this chapter:

[8] We need a rational basis for making these above decisions. We will address details in subsequent chapters.

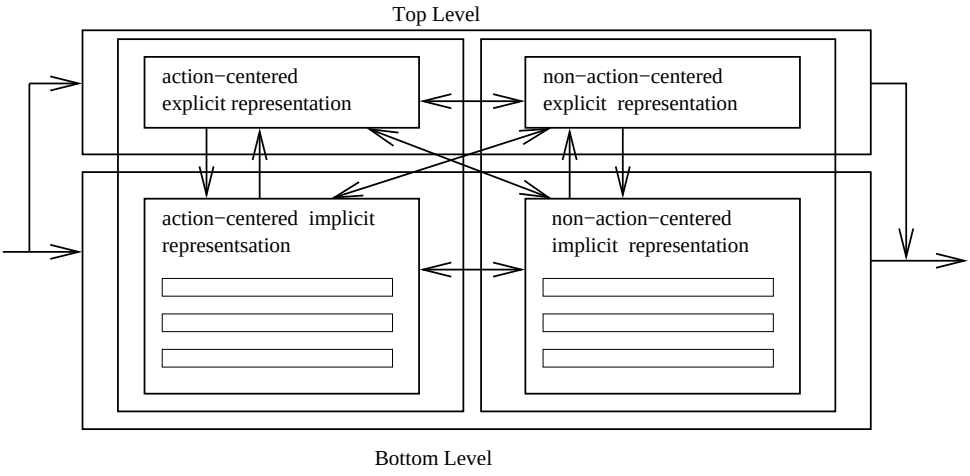

Figure 1.1. The CLARION architecture.

- Representational difference: The two levels of the architecture employ two different types of representations and thus have different degrees of accessibility.
- Learning difference: Different learning methods are used for the two levels.
- Bottom-up and top-down learning: There are two possible directions of learning that go from one type of kowledge to the other (Sun 2002).
- Action-centered vs. non-action-centered representation: At each level, there are both action-centered and non-action-centered representations present.

These hypotheses together form the basis of the CLARION architecture (Sun et al 1998, Sun 1997, 1999, 2002). See Figure 1.1 for a sketch of the architecture.

Chapter 2

The Action-Centered Subsystem

2.1 Overall Action Decision Making

The overall algorithm of CLARION's action decision making consists of a structure that goes from perception to actions and ties them together through two levels of cognitive processes in the ACS. It can be stated as follows:

1. Observe the current state x.
2. Compute in the bottom level of the ACS the "value" of each of the possible actions (a_i's) in the current state x: $Q(x, a_1)$, $Q(x, a_2)$, ..., $Q(x, a_n)$.
3. Find out all the possible actions (b_1, b_2, ..., b_m) at the top level of the ACS, based on the the current state information x (including internal information such as goal structure and working memory) and the existing rules in place at the top level.
4. Choose an appropriate action a, by integrating (in some way) the outcomes of the two levels.
5. Perform the chosen action a, and observe the next state y and (possibly) the reinforcement r.
6. Update the bottom level of the ACS in accordance with an appropriate algorithm (to be detailed later), based on the feedback information.
7. Update the top level of the ACS using an appropriate algorithm (for constructing, refining, and deleting rules, to be detailed

later).

8. Go back to Step 1.

State x is represented as a set of dimension-value pairs: $(dim_1, val_1)(dim_2, val_2)....(dim_n, val_n)$. Each dimension is discrete, with a fixed set of possible values. Each dimension is either ordinal or nominal. A binary dimension is, for example, a special case of a discrete ordinal dimension.

Note that state x may be filtered before it is delivered to the bottom level, and may be further filtered before it is delivered to the top level. Each level of the model thus may see a (different) subset of these dimension-value pairs, due to differential filtering (see the chapter on the meta-cognitive subsystem). Each module within each level (for example, each backpropagation network at the bottom level, or each rule type within each rule group at the top level) may see a (different) subset of the dimension-value pairs due to differential filtering. At the bottom level, each dimension-value pair that is observable to a bottom-level module is sent to the corresponding input node of that module. At the top level, these dimension-value pairs that are observable to different modules are matched against rule conditions in these modules respectively.

Note that action output from the ACS need not be primitive actions, although primitive actions should always be included as possible action output from the ACS. Some action output may involve complex structures. For example, an action output may consist of multiple primitive actions organized in complex ways. In particular, temporally extended actions (or action sequences) can be used, which may or may not have simple termination conditions (Lorenz 1950, Tinbergen 1951, Mataric 2001). The existence of such complex actions (or action routines), or their status, may be indicated through (internal) sensory input to an agent (as a form of proprioceptive feedback).

2.2 The Bottom Level

The bottom level captures implicit processes of action decision making. It is a simple and direct mapping from perceptual information to actions, through learned or pre-wired "reflex arcs". We will term this part IDNs, or implicit decision networks.

It is implemented in backpropagation neural networks, in which distributed representation is involved. That is, representational units (in the hidden layer of a backpropagation network) are capable of accomplishing tasks but are generally subsymbolic and not individually meaningful (see Rumelhart et al 1986, Sun 1994); that is, they generally do not have an associated semantic label. This characteristic of distributed representation accords well with the inaccessibility of implicit processes.

2.2.1 Representation

The input to the bottom level of the ACS consists of three groups of information:

- Sensory input. The sensory input is divided into a certain number of dimensions, each of which has a certain number of possible values. [1] At each step, the sensory input, as part of the current state, is in the form of $(dim_1, val_1)(dim_2, val_2)....(dim_n, val_n)$. Each possible value of each dimension is represented by a separate node.
- Working memory items. Working memory items are presented as part of the current state at each step of action decision making. The working memory consists of a certain number of items, up to a maximum limit. In each item, there is a set of dimension-value pairs. Each possible value of each dimension is represented by a separate node. If the base-level activation (BLA) of WM item i is above a threshold ($B_i^w > threshold_{WM}$), the item will be used as part of the current state (and input to the bottom level), with a strength of 1 (otherwise, the strength is considered zero, as if the item does not exist).
- Optionally, there can be a few additional, special working memory items, namely *flags*, that do not correspond to any sensory input dimensions. Each flag can be either on or off. Each flag is represented by an individual node. These flags are treated as part of the current state.
- One item from the goal structure. The goal structure may store a certain number of goals, up to a maximum limit. Each goal is represented by a set of dimension-value pairs. In each goal, there is

[1] It may in fact include "internal" sensory input as well. For example, it may include sensory input from the agent's own body (i.e., proprioceptive input).

a goal dimension, in which each possible goal code (from a finite pre-specified set of goal codes) is represented by an individual node. Along with this goal dimension, there are a number of parameter dimensions in each goal. Each value of each parameter dimension is represented by an individual node. At most one goal can be *active* at a time. [2] The active goal is presented as part of the current state (in the form of a set of dimension-value pairs, include the goal dimension) at each step.

These three sets of information are stored in the *current state buffer*, which is a structure that holds state information for use by the bottom level as well as the top level. (The current state buffer may be subject to information selection and filtering by the meta-cognitive subsystem. So it it not the same as the raw state information. Information received by the top level may be subject to further filtering. More later.)

The output of the bottom level are action choices. It consists of three groups of actions: working memory actions, goal actions, and external actions:

- For any particular domain, there are a number of possible (domain-specific) external actions, each of which is represented by a set of dimension-value pairs. That is, each external action has a number of action dimensions, each of which has a number of possible values. Each value of each dimension is represented as an individual node. These external actions may be used to change the state of the world (the internal or external state). *Do-nothing* is included as a special case. Only one external action can be selected at each step.
- Goal actions are either for setting up a goal, or for removing a goal. See the section on the goal structure for details of goal actions. See the section on coordination regarding the coordination of goal actions and other action types. Each goal action is constituted by a number of dimensions, each of which takes on one value out of a set of possible values. Each value of each goal action dimension is represented by a separate node.

[2] In case a goal stack is used, the active goal is the one at the highest non-empty slot of the stack. Other slots contain similar sets of goal codes and parameter values but are not considered active and thus have no effect, unless they become the highest non-empty slot of the goal stack (as a result of popping off higher ones).

- Working memory actions are either for storing an item in the working memory, or for removing an item from the working memory. See the section on working memory for details regarding WM actions. Each working memory action is constituted by a number of dimensions, each of which takes on one value out of a set of possible values. Each value of each working memory action dimension is represented by a separate node.
- Optionally, flags may be used and thus there may be two additional working memory actions for each flag used: *set* $flag_i$ and *reset* $flag_i$, where i is a flag number and setting a flag turns it on and resetting a flag turns it off. In addition, there is *reset-all-flag*. Each action is represented by an individual node; together they constitute a flag action dimension.

These three groups of actions may be computed by one network, or by three separate networks. The goal action network and the working memory action network are optional: Goal actions and working memory actions may be computed by the external action network, in case the goal action network or the working memory action network do not exist separately. The external action network, however, may be further divided up into several networks. Each of these external action networks may be responsible for a different (sub)task (e.g., a different set of states or a different set of state sequences; see Sun and Peterson 1999, Sun and Sessions 2000), and may receive differently filtered state information and recommend different actions (different action dimensions/values).

An eligibility condition may be specified for each external action network, so that not all of these networks may be applicable at a particular step. Ideally, only one external action network is applicable at each step. Inapplicable networks are simply ignored. [3] The eligibility condition of an external action network may be specified based on goals: Each subtask may have a different goal, and the current goal indicates the eligibility of different networks. The eligibility condition may also be specified based on more elaborate information (such as the current state along with the current goal, or certain patterns in the past state sequences).

[3] When there are multiple applicable networks, we need consider the coordination of them, which will be addressed later.

Within each such network, each action (of a particular action type: the WM actions, the goal actions, and the external actions; or of a mixed type) consists of one or more action dimensions. Each dimension may have one of a number of possible values. Thus, at each step, each output may be in the form of $(dim_1, val_1)(dim_2, val_2)....(dim_n, val_n)$. For example, *(type: set-goal) (goal-id: chase-prey) (speed: fast)*. Each possible value of each output dimension is represented by an individual node at the bottom level. [4]

See Figure 2.1 for a sketch of the overall structure of the bottom level of the ACS (i.e., the IDNs).

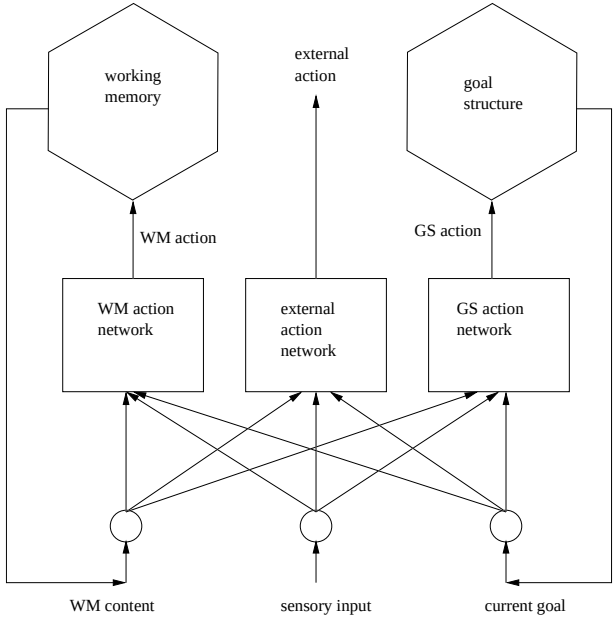

Figure 2.1. The three networks at the bottom level.

The majority of these networks are amenable to learning. The detail of learning is described in the next subsection. However, some of these networks may be (relatively) fixed: that is, they may reflect pre-wired instincts or reflexes, acquired through long evolutionary processes, that are

[4] Each action as a whole constitutes a chunk, which is represented by an individual node at the top level, i.e., by an action chunk. See the discussions on the top level later.

not (easily) changeable. Specifically, some external action networks may be designated as being fixed, representing pre-wired innate reflexes. Likewise, the goal action network may be designated as being fixed, reflecting innate priority setting. In the current implementation of CLARION, if one designates any particular network to be *fixed*, it will have a learning rate of 0.

As evident from the above discussion, this part of the system is clearly teleoreactive (as termed by Benson and Nilsson 1995): It reacts to the current state of the world (thus, it is reactive in the sense as discussed in earlier chapters), while it takes into consideration the current goal to be achieved (thus, it is also teleological).

2.2.2 Learning

In terms of learning Q values, there are the following options. These options need to be specified with respect to each backpropagation network (each IDN) used.

2.2.2.1 Q-learning

Q-learning (Watkins 1989) is a reinforcement learning algorithm. In the algorithm, $Q(x, a)$ estimates the maximum (discounted) total reinforcement that can be received from the current state x on after action a is performed:

$$\forall x \forall a : Q(x, a) = \max_{a_i : i=1,2,3,....} (\sum_{i=0}^{\infty} \gamma^i r_i) \tag{2.1}$$

where γ is a discount factor that favors reinforcement received sooner relative to that received later, a_i is an action that can be performed at step i (with $a_0 = a$), and r_i is the reinforcement received at step i (which may be a positive/negative value or zero). Sequential behavior is made possible by such Q values, which provide estimation of total reinforcement on a step-by-step basis and thus guidance to an agent on a step-by-step basis.

The updating of $Q(x, a)$ is based on:

$$\forall x \forall a : \Delta Q(x, a) = \alpha(r + \gamma e(y) - Q(x, a)) \tag{2.2}$$

where γ is a discount factor, y is the new state resulting from action a in state x, and $e(y) = \max_b Q(y, b)$. Note that x and y include sensory input (internal and external), working memory items (if there is any activated with a BLA above a certain threshold), and the current goal (if it exists).

Thus, the updating is based on the *temporal difference* in evaluating the current state and the action chosen. In the above formula, $Q(x, a)$ estimates, before action a is performed, the (discounted) total reinforcement to be received from the current point on if action a is performed, and $r + \gamma e(y)$ estimates the (discounted) total reinforcement to be received from the current point on, after action a is performed; so their difference (the temporal difference in evaluating an action) enables the learning of Q-values that approximate the (discounted) total reinforcement. Q-learning allows sequential behavior to emerge. due to its use of estimates of future reinforcement.

Q-learning can be implemented in backpropagation networks (Lin 1992). Applying Q-learning, training of the backpropagation network is based on minimizing the following error at each step:

$$\forall x \forall i : err_i = \begin{cases} r + \gamma e(y) - Q(x, a_i) & \text{if } a_i = a \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

where i is the index for an output node representing the action a_i, a is the action actually performed, and x is the current state. Based on this error measure, the backpropagation algorithm is applied to adjust internal weights. Note that weights are randomly initialized before training starts.

In case there are multiple action dimensions, this process is carried out for each of these dimensions, separately.

2.2.2.2 Simplified Q-learning

In simplified Q-learning, temporal credit assignment is not involved. This is useful when an agent can rely on immediate feedback and thus there is no need for backwards temporal credit apportionment. That is,

$$\forall x \forall a : \Delta Q(x, a) = \alpha(r + \gamma \max_b(y, b) - Q(x, a)) = \alpha(r - Q(x, a)) \tag{2.4}$$

where x is the current state, a is one of the output, r is the immediate reinforcement at the current step, and $\gamma \max_b(y, b)$ is set to zero.

The error used in the simplified Q-learning is as follows:

$$\forall x \forall i : err_i = \begin{cases} r - Q(x, a_i) & \text{if } a_i = a \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

where i is the index for the output node representing action a_i, a is the action performed, and x is the current state. Based on this error measure, the backpropagation algorithm is applied to adjust internal weights. Weights are randomly initialized before training starts.

A special case of this is step-wise prediction. In the case of step-wise prediction, $r = 1$ if a is the correct prediction; $r = 0$ if a is not the correct prediction.

This version of the Q-learning algorithm is in fact similar to straight backpropagation, except that we only update one of the output at each step—the output that indicates the current prediction.

2.2.2.3 Backpropagation learning

When a correct input/output mapping is available for a step, backpropagation can be directly applied using the error measure of the absolute difference between the desired output and the actual output:

$$\forall x \forall i : err_i = target(x, a_i) - Q(x, a_i) \tag{2.6}$$

where $target(x, a_i)$ is the target output value of output node i (representing action a_i) given current state x, and $Q(x, a_i)$ is the actual output value of output node i (representing action a_i) given current state x.

2.2.2.4 External instructions and top-down assimilation

Yet another possibility is top-down assimilation—the process by which externally given explicit knowledge (i.e., rules given externally and wired up at the top level) are assimilated into the bottom level

First, externally given explicit knowledge may be expressed as rules at the top level (in the forms of FR or IRL rules; to be explained later). Assimilation can then be done either using supervised learning in which the top level serves as the teacher, or through gradual practice (with Q-learning) guided by the top-level knowledge. If supervised learning for assimilation is to be performed, a frequency parameter needs to be specified,

which indicates the frequency of supervised learning relative to regular reinforcement learning (m steps of supervised learning every n steps of reinforcement learning). Assimilation through gradual practice guided by the top-level knowledge is always (automatically) performed.

Through assimilation, explicit knowledge becomes procedural, embodied skills and thus becomes more effective (see, for example, Anderson 1982, Dreyfus and Dreyfus 1987).

2.2.2.5 Imitative learning

Imitation is an important learning mechanism for humans (as well as other social animals). Imitation learning can be accomplished in the following way: The bottom level learns through imitating another agent, and then on that basis, explicit knowledge may be extracted at the top level (to be discussed later).

For the bottom level to learn through imitation, a special reinforcement function needs to be used. This special reinforcement function rewards state-action pairings (a certain action in a certain state) identical to those performed by the agent to be imitated. With the help from this special reinforcement function, imitative actions (actions that are the same as those performed by the agent to be imitated in their respective states) are encouraged.

This special reinforcement function is constructed on line, during the imitation learning process. The construction is based on observing the state-action pairings of the agent to be imitated, and then providing a reward to the learning agent for the same pairings when they occur (at the same time or at a later time).

2.2.2.6 An End Note to Learning

Note that these above learning methods (Q-learning, Backpropagation, and so on) may be under the control of a meta-cognitive subsystem, which determines, at each step, whether a particular learning method should be operating or not (see the chapter on the meta-cognitive subsystem).

With Q-values in place, the agent can select an action in a given state, by using a *max* function (that is, selecting the action that maximizes the Q-value in the current state, or in other words, selecting the best action

for the current state), or by using a *softmax* function (e.g., based on a Boltzmann distribution). See details later.

2.2.3 Reinforcement Functions

In case that Q-learning or simplified Q-learning is used, we need to specify reinforcement functions that determine r. We may have the following functions separately specified:

- The reinforcement function for external actions: $r_1(x)$, which must be specified if Q-learning or simplified Q-learning is chosen, where x includes sensory input, working memory, and the active goal. [5] If there are multiple external action networks, as an option, multiple such functions may be specified, one for each external action network.
- The reinforcement function for goal actions: $r_2(x)$. This function is optional.
- The reinforcement function for working memory actions: $r_3(x)$. This function is optional.

We may choose to specify one reinforcement function (i.e., r_1) that encompasses all three functions. If only one reinforcement function is provided (that is, if one external action reinforcement function is provided, which is mandatory), then the output of this function is given to all three types of networks (goal action, working memory action, and external action).

If separate reinforcement functions are given for external actions, goal actions, and working memory actions, then the corresponding networks use these reinforcement functions respectively. Furthermore, separate reinforcement functions may be provided to different networks for external actions, if multiple external action networks are used.

Note that reinforcement is to be determined based on agent's needs, desires, and goals. Therefore, they may be determined by the motivational and the meta-cognitive subsystem of an agent. See the chapter on the meta-cognitive subsystem.

[5] There may be other information as well, for example, past state(s), past action(s), and so on. If a meta-cognitive subsystem is involved in deciding this function, it may also involve drive states and so on (see the later chapter on the meta-cognitive subsystem).

2.3 The Top Level

At the top level, in contrast to the bottom level, explicit knowledge may be best captured by a symbolic or localist representation (Clark and Karmiloff-Smith 1993), in which each unit is easily interpretable and has a clear conceptual meaning, i.e., a semantic label. This characteristic captures the property of explicit knowledge being accessible and manipulable (Smolensky 1988, Sun 1994). In terms of acquiring/learning explicit action-centered knowledge at the top level, a number of explicit rule learning algorithms may be used. This part of CLARION is termed ARS, that is, the *action rule store.*

2.3.1 Representation

Input state x to the top level, similar to the input to the bottom level, is made up of a number of dimensions (e.g., $x_1, x_2, ..., x_n$). Each dimension can have a number of possible values (e.g., $v_1, v_2, ..., v_m$). Each dimension is either ordinal (discrete or continuous) or nominal. As in the case of the input to the bottom level, the input consists of three groups of information: sensory input, [6] working memory items, and the currently active goal. The sensory input is divided into a number of dimensions, each of which has a number of possible values. The goal input similarly consists of a number of dimensions, which include a goal dimension and its parameter dimensions. The working memory items are also divided into dimensions as discussed earlier. All the information comes from the *current state buffer* (but subject to filtering; see the chapter on the meta-cognitive subsystem). [7]

Rules are usually in the following form: *current-state-condition* $\longrightarrow$ *action.* The left-hand side of a rule is a conjunction of individual elements each of which refers to a dimension x_i of state x. Each element specifies a value or a value range, i.e., $x_i = v_i$ or $x_i \in (v_{i1}, v_{i2}, ..., v_{in})$. The right-hand

[6] It may in fact include sensory input from the agent's own body. So, in this sense, it includes "internal" sensory input as well.

[7] Note that the order in which working memory items appear to the top level may or may not be relevant. The condition of an action rule may require a working memory item satisfying a certain criterion (e.g., dimension i has value j), and this requirement will be matched against each of all the working memory items. The first working memory item meeting the criterion will be selected. Alternatively, the condition of an action rule may specify a particular working memory item (e.g., the kth WM item).

side of a rule is an action recommendation, which also consists of a set of dimension-value pairs concerning actions. [8]

In the condition of a rule, all of the specified dimensional values together, constitute a chunk. A chunk is represented as a (chunk) node at the top level, whereas its dimensional values are represented as separate nodes at the bottom level.

The output of a rule, similar to the output from the bottom level, is an action recommendation. It may be one of the three types: working memory actions, goal actions, and external actions, or their combinations thereof. Hence, rules may belong to one of the three action types: external action rules, working memory action rules, and goal action rules, but rules may also be the combinations of these types thereof.

An action is divided into a number of action dimensions, each of which can take on one of a number of possible values. An action, in the right-hand side of a rule, constitutes a chunk, which is represented as a chunk node at the top level, the same way as the condition of a rule. [9]

Each chunk node at the top level is connected to all the specified dimensional values (represented by separate nodes) at the bottom level. Dimensional values in the bottom level serve as features of a chunk, and the chunk node at the top level in turn serves to identify and label this set of dimensional values, or features, as a whole. Each chunk is a concept, given or learned, based on which complex forms of reasoning may be performed (see the chapter on the non-action-centered subsystem regarding reasoning).

The connection from the chunk node at the top level to the nodes representing the dimensional values at the bottom level may involve two types of links: AND links and OR links. When one dimension is allowed to take on one of a number of allowable values, an OR link is used. When values of multiple dimensions are combined, an AND link is used. These two types of links may be straightforwardly implemented as simple logical operations. (Alternatively, see Appendix at the end of this chapter, regarding more complex, connectionist implementations of AND and OR links.)

[8] Alternatively, the rules can be in the forms of *current-state-condition* $\longrightarrow$ *action new-state* or *current-state-condition action* $\longrightarrow$ *new-state*. The encoding of the alternative forms of rules is similar.

[9] Note that each of the dimensional values is represented as a separate (feature) node at the bottom level, as described before.

In addition, for an AND link, if partial match is allowed, there will be a weighted-sum activation function that calculate the partial strength of the chunk node due to partial match—based on how many of its feature dimensions are satisfied (i.e., how many dimensions have a required value). When features (dimensional values) at the bottom level are activated, corresponding chunk nodes at the top level are also activated to a corresponding strength level (based on a weighted-sum calculation when partial match is allowed; see details later).

We may easily translate the structure of a set of rules into that of a network. When given a set of action rules, a rule network can be wired up, in which conditions and conclusions of rules are represented by corresponding chunk nodes. For each rule, a link is established, which connects the chunk node representing the condition of the rule to the chunk node representing the conclusion. Notice that, when partial match is allowed, current state x "activates" all rule condition chunks that are contained in it: That is, it "activates" all the rule condition chunks that specify a subset of dimension-value pairs as indicated by state x.

With rules in place, an agent can select an action in a given state by choosing an applicable rule. Rules compete with each other to be used in action decision making, The competition can be done with a *max* function [10] or with a *softmax* function (details later). The competition may be based on a utility measure. In case that no utility measure is available, a random selection can be made. (See section on integration later for more details.)

Finally, we may divide rules into *groups*, with each group corresponding to one of the bottom-level networks (the WM action network, the goal action network, or various external action networks). There may be multiple external action rule groups. Each external action rule group obeys the same eligibility condition as specified for the corresponding bottom-level external action network. Within each rule group, we may further divide rules into *sets*, based on their knowledge types (e.g., RER rules, IRL rules, or fixed rules, to be discussed later). See Figure 2.2 for a sketch of the division.

[10] In this case, select the action that maximizes a measure in the current state, or in other words, select the best action for the current state according to that measure.

	EXT_1	....	EXT_n	GS	WM
BL	net_{EXT_1}	....	net_{EXT_n}	net_{GS}	net_{WM}
RER	$ruleset_{EXT_1,RER}$	....	$ruleset_{EXT_n,RER}$	$ruleset_{GS,RER}$	$ruleset_{WM,RER}$
IRL	$ruleset_{EXT_1,IRL}$	....	$ruleset_{EXT_n,IRL}$	$ruleset_{GS,IRL}$	$ruleset_{WM,IRL}$
FR	$ruleset_{EXT_1,FR}$	....	$ruleset_{EXT_n,FR}$	$ruleset_{GS,FR}$	$ruleset_{WM,FR}$

Figure 2.2. Divisions of Knowledge: knowledge types (in rows) versus network/group type (in columns).

2.3.2 Bottom-Up Rule Learning (RER)

Using the *Rule-Extraction-Refinement* algorithm (RER), an agent learns rules using information from the bottom level, which captures a bottom-up learning process.

The basic idea of this algorithm is as follows (for each IDN and its corresponding rules): If an action decided by the bottom level is successful (i.e., if it satisfies a certain criterion), then the agent constructs a rule (with its action corresponding to that selected by the bottom level and with its condition specifying the current state as observable to this particular module of the agent), and adds the rule to the top level. Then, in subsequent interactions with the world, the agent refines the constructed rule by considering the outcome of applying the rule: If the outcome is successful, the agent may try to generalize the condition of the rule to make it more universal; if the outcome is not successful, then the condition of the rule should be made more specific (and exclusive of the current state). Specifically,

1. Update the rule statistics (to be explained later).
2. Check the current criterion for rule extraction, generalization, and specialization:
2.1. If the result is successful according to the current rule extraction criterion, and there is no rule matching the current state and action, then perform *extraction* of a new rule: condition $\longrightarrow$ action. Add the extracted rule to the ARS.
2.2. If the result is unsuccessful according to the current specialization criterion, revise all the rules matching the current state and action through *specialization*:
2.2.1. Remove the rules from the rule store.

2.2.2. Add the revised (specialized) versions of these rules into the rule store.
2.3. If the result is successful according to the current generalization criterion, then generalize the rules matching the current state and action through *generalization*:
2.3.1. Remove these rules from the rule store.
2.3.2. Add the generalized versions of these rules to the rule store.

One may find empirical data and arguments in favor of this kind of algorithm in, for example, Bruner et al (1956) and Haygood et al (1970).

Let us discuss the details of the operations used in the above algorithm and criteria measuring whether a result is successful or not.

At each step, we examine the following information: (x, y, r, a), where x is the state before action a is performed, y is the new state after an action a is performed, and r is the reinforcement received after action a. Based on that, we update (in Step 1 of the above algorithm) the positive and negative match counts (i.e, $PM_a(C)$ and $NM_a(C)$), for each rule condition and each of its variations (e.g., resulting from the rule condition plus/minus one possible value in one of the input dimensions), denoted as C, with regard to the action a performed. That is, $PM_a(C)$ (i.e., Positive Match) equals the number of times that a state matches condition C, action a is performed, and the result is positive; $NM_a(C)$ (i.e., Negative Match) equals the number of times that a state matches condition C, action a is performed, and the result is negative.

Positivity or negativity (for updating PM and NM) may be determined based on a certain criterion. The default criterion is as follows:

$$\gamma \max_b Q(y, b) + r - Q(x, a) > threshold_{RER} \tag{2.7}$$

which indicates whether or not action a (chosen according to a rule) is reasonably good (Sun and Peterson 1997, 1998b). [11] Alternative criteria are also possible. For example, when immediate feedback is given, positivity

[11] Due to function approximation in the bottom level, the Q value landscape tends to be smoother, and the output tend to be more uniform, than what is optimal. This suboptimal and uniform output are the result of function approximation in the backpropagation network used for computing the Q values, and the inevitable generalization and over-generalization by function approximators. This criterion identifies these lost peaks of the Q value landscape, and extract rules accordingly (Sun and Peterson 1998).

may be determined by the immediate feedback: If $r > threshold_{RER}$, then it is positive; otherwise, it is negative.

Each statistic is updated with the following formulas:

- $PM := PM + 1$ when the positivity criterion is met;
- $NM := NM + 1$ when the positivity criterion is not met.
- At the end of each episode, they are discounted: $PM := PM * 0.90$ and $NM := NM * 0.90$. The results are time-weighted statistics, which are useful in nonstationary situations.

Based on these statistics, the (default) information gain measure (IG) may be calculated:

$$IG(A,B) = log_2 \frac{PM_a(A) + c_1}{PM_a(A) + NM_a(A) + c_2} - log_2 \frac{PM_a(B) + c_1}{PM_a(B) + NM_a(B) + c_2} \tag{2.8}$$

where A and B are two different rule conditions that lead to the same action a, and c_1 and c_2 are two constants representing the prior (the default values are $c_1 = 1, c_2 = 2$). Essentially, the measure compares the percentages of positive matches under different conditions A and B (with the Laplace estimator). [12] If A can improve the percentage to a certain degree over B, then A is considered better than B. In the following algorithm, if a rule is better compared with the corresponding match-all rule (i.e, the rule with the same action but with the condition that matches all possible states), [13] then the rule is considered successful.

We may decide on whether or not to extract a rule based on the positivity criterion, which measures whether the current step is successful or not, fully determined by the current step (x, y, r, a):

- *Extraction*: If the current step is positive (according to the current positivity criterion) and if there is no rule that covers this step in the top level (matching both the state and the action), set up a rule $C \longrightarrow a$, where C specifies the values of all the dimensions exactly

[12] This is a commonly used method, especially in Inductive Logic Programming, and well justified on the empirical ground. See, for example, Lavrac and Dzeroski (1994).

[13] Clearly, there are as many match-all rules as the number of possible actions. That is, the number of match-all rules is equal to $|dim_1| \times |dim_2| \times ..... \times |dim_n|$, where $1, 2, ..., n$ are all the action dimensions, and $|dim_i|$ measures the number of possible values in dimension i.

as in the current state x and a is the action performed at the current step. [14]

When extracting an initial rule, if there are working memory items with base-level activations greater than $threshold_{WM}$, then these items may be included in the condition of the rule extracted (provided that these working memory dimensions are selected). Similarly, if there is an active goal, then the extracted rule may include the currently active goal in its condition (provided that the goal dimension is selected).

We may specify the dimensions to be included in an extracted rule (see the appendix chapter on options and parameters). For example, we may choose to include or exclude goals from the conditions of extracted rules. Likewise, we may choose to include or exclude working memory items from the conditions of extracted rules. We may also selectively include (or exclude) sensory input dimensions. Similarly, action dimensions may also be selectively included (or excluded).

The issue of the proliferation of rules needs to be addressed. As described before, the extraction of rules is subject to a positivity criterion. In addition, a probability parameter (p_{re}) determines how likely a rule will be extracted, given that the criterion for rule extraction (e.g., the default criterion as described above) is met. On the other hand, a density parameter (d_r) determines the minimum frequency of matching of input necessary to keep an action rule. For example, if this parameter is set at $1/n$, then at least one encounter of an input that matches a rule is necessary every n steps in order to keep the rule. Otherwise, the rule will not be maintained and will thus be deleted.

Generalization and *specialization* operators is based on the information gain measure. Generalization amounts to adding an additional value to one input dimension in the condition of a rule, so that the rule will have more opportunities of matching input, and specialization amounts to removing one value from one input dimension in the condition of a rule, so that it will have less opportunities of matching input.

- *Generalization*: If $IG(C, all) > threshold1$ and $\max_{C'} IG(C', C) \geq 0$, where C is the current condition of a rule (matching the current state and action), *all* refers to the corresponding match-all rule (with the

[14] Potentially, we might use attention to focus on fewer input dimensions. See the chapter on the meta-cognitive subsystem for further details.

same action as specified by the original rule but with a condition that matches any state), and C' is a modified condition such that $C' = C$ plus one value (i.e., C' has one more value in one of the input dimensions) [**that is, if the current rule is successful and a generalized condition is potentially better**], then set $C'' = argmax_{C'} IG(C', C)$ as the new (generalized) condition of the rule. [15] Reset all the rule statistics. Any rule covered by the generalized rule will be placed in its children list. [16]

- *Specialization*: If $IG(C, all) < threshold2$ and $\max_{C'} IG(C', C) > 0$, where C is the current condition of a rule (matching the current state and the current action), *all* refers to the corresponding match-all rule (with the same action as specified by the original rule but with a condition that matches any states), and C' is a modified condition such that $C' = C$ minus one value (i.e., C' has one less value in one of the input dimensions) [**that is, if the current rule is unsuccessful, but a specialized condition is better**], then set $C'' = argmax_{C'} IG(C', C)$ as the new (specialized) condition of the rule. [17] Reset all the rule statistics. Restore those rules in the children list of the original rule that are not covered by the specialized rule and the other existing rules. Note that, unless no dimension has more than one value, removing the last value from a dimension is not allowed. Removing the last value from a certain dimension of a rule makes it impossible for a rule to match any state. In that case, the rule is deleted.

In this type of learning, although the accumulation of statistics is gradual, the acquisition and refinement of rules is one-shot and all-or-nothing. Therefore, this type of learning is quite different from the gradual weight tuning at the bottom level (the IDNs).

[15] Dominowski (1972) showed that even when their hypotheses were correct for the current step, subjects tended to shift their hypotheses in some ways any way.

[16] The children list of a rule is created to keep aside and make inactive those rules that are more specific (thus fully covered) by the current rule. It is useful because if later on the rule is deleted or specialized, some or all of those rules on its children list may become active again if they are no longer covered.

[17] Clearly, we should have $threshold2 \leq threshold1$ to avoid oscillation.

Note that there are some other options (and other parameters) concerning RER. They are explained in Appendix. [18]

Explicit knowledge acquired in this way is clearly concerned with existentially and ecologically significant aspects of the world. In other words, concepts and rules learned are concerned with those and only those aspects of the world that have significant bearings on an agent in its interaction with the world. They are not just "objective" classifications of objects in the world (Lave 1988), but the result of the interaction of an agent with its world, manifesting the regularities encountered in such interactions. Thus, they are not objective, but task-oriented and grounded in the interaction between an agent and its world (Heidegger 1927).

2.3.3 Independent Rule Learning (IRL)

A variation of the above algorithm is independent rule learning, without using the bottom level for the initial extraction. In independent rule learning (IRL), either completely randomly or in a particular domain-specific order, rules of various forms are independently generated (hypothesized and wired up) at the top level. Then, these rules are tested through experience using an IG measure. If the IG measure of a rule is below a threshold, then the rule is deleted. (One may find in Trabasso and Bower 1968, Levine 1970, and Dienes and Fahey 1995 data and arguments in favor of such an algorithm.)

In the current implementation of CLARION, one may specify several subsets of rules. Each subset is specified through a rule template, and ranges of its parameter values. These rule subsets will be tested in the order specified. That is, if one subset of rules (generated from one template) fails the test and consequently all the rules in the subset are deleted (one by one), then the next subset of the rule (i.e., the next template) will be tested, and so on.

The positivity criterion and the IG measure for IRL need to be specified. The positivity criterion in calculating IG may be either based on

[18] Note also that, like other learning methods discussed here, this type of learning may be under the control of a meta-cognitive subsystem, which determines, at each step, whether a learning method should be operating or not (see the chapter on the meta-cognitive subsystem).

information from the bottom level (as specified before for RER) or based on other pieces of information (e.g., external information such as immediate reinforcement). For example, one possible positivity measure is:

$$\gamma \max_b Q(y,b) + r - Q(x,a) > threshold_{IRL} \tag{2.9}$$

which is similar to the one used in RER. One possible IG measure for IRL rule testing is:

"If $IG(C, random) < threshold3$, we delete the rule C."

where *random* refers to completely random actions. This measure is equivalent to the following measure (with respect to any particular domain): [19]

"If $IG(C) = log_2 \frac{PM(C)+c_5}{PM(C)+NM(C)+c_6} < threshold4$, we delete the rule C."

When appropriate, generalization and specialization may also be performed based on the IG measure as before. Specialization consists of removing allowable values from the condition of a rule (the same way as with RER rules). Generalization then consists of adding allowable values to the condition of a rule (again, the same way as with RER rules).

Often, an initial IRL rule (specified a priori to be tested through experience) does not involve values of input dimensions in its condition (see, e.g., the rules used in simulating process control tasks). In that case, we consider the rule condition consists of all the values in all the input dimensions, so the rule is at its most general form (thus, no generalization may be performed on such a rule initially, but specialization can be performed). Deletion can also occur when specialization leads to an overly specialized condition that has no possibility of matching any input.

A density parameter (d_r) determines the minimum frequency of matching of input necessary to keep an IRL rule, in ways similar to RER rules (as discussed earlier). There are also other options and parameters related to IRL, similar to those concerning RER (see Appendix).

[19] The match-all rule is not used here for comparison purposes, because IRL rules are supposed to be general rules; that is, in different states, different action may be recommended. Without the restriction of the same action always being recommended by the rule, the match-all rule becomes a random-action rule, as in the above formula. For this reason, no subscript referring to action "a" is used here (as opposed to the default IG measure for RER).

2.3.4 Fixed Rules

Some of the rules at the top level may be pre-wired and fixed throughout simulation. (In contrast, IRL rules may be pre-determined to some extent but not completely fixed.) This type of rule (fixed rule, or FR for short) may represent either genetic pre-endowment of an agent (presumably acquired through long evolutionary processes), or prior knowledge acquired through prior experience in the world. Alternatively, FRs may be given by external sources (for example, via instructions or textbooks).

Externally given FRs enable top-down learning (assimilation). [20] With the FRs in place, the bottom level learns under their guidance. Initially, the agent relies mostly on the FRs in its action decision making. But gradually, through observing actions directed by the FRs, more and more knowledge is acquired by the bottom level. And therefore, the agent gradually becomes more and more reliant on the bottom level (given that parameters for cross-level integration are adaptable). Hence, top-down learning takes place. This kind of top-down assimilation through gradual practice guided by the top-level knowledge is always carried out "automatically"—It comes with the basic structure of CLARION.

Top-down assimilation can also be accomplished through supervised learning (in which the top level serves as the teacher). In that case, a frequency parameter needs to be specified, which indicates the frequency of supervised learning relative to that of (regular) reinforcement learning—x steps of supervised learning for every y steps of reinforcement learning.

An advantage of fixed rules is that they can represent more than simple propositional structures. FRs can involve more complex structures, which establish more complex mappings between input (conditions) and output (actions), as well as more complex action sequences. The idea is essentially similar to the notion of "schemas" (Arbib 1980, Dretcher 1989), as well as the notion of "abstract behaviors" (Mataric 2001).

Note that one issue that is not fully dealt with here is the explanation of the acquisition of FRs. Aside from the simpler parts of this issue, such as FRs being given externally by an instructors or via a manual or a textbook, in general, it appears that this issue is a difficult one. We should note that Rosenbloom (1991), Anderson (1993), and Anderson and Lebiere

[20] Other rule types, such as IRL, may also enable top-down assimilation.

(1998) used such rules in SOAR, ACT-R, and other existing cognitive architectures, and they did not provide an explanation of this issue either. [21] FRs are used here merely as a convenient way of representing certain (relatively fixed) knowledge whose origin, at present time, is hard to account for.

2.3.5 Imitative Learning *

In imitative learning using the top level, an agent observes the state and the action of another agent. The imitating agent sets up an action rule in strict correspondence with the observed state and the observed action of the imitated agent at each step. [22]

Note that if imitative learning is going on in the bottom level (see the section earlier on learning in the bottom level), then through RER learning, the top level may extract corresponding rules. But this process (indirect imitative learning via the bottom level) is separate from the afore-specified (direct) imitative learning.

2.3.6 Plan Extraction *

See Sun and Sessions (2000 b) for a detailed explanation of this aspect.

2.3.7 A Note on Knowledge Types

A final note concerning these different types of rules is in order. We hypothesize that these different types of rules represent different "layers" of knowledge, from the most fluid to the most entrenched: One type is innate or socioculturally indoctrinated, and thus relatively fixed. Another type is based on known knowledge templates (innate, sociocultural, or empirical), but tunable with experience and thus exhibits some flexibility. Yet another type is completely empirical—It is constructed anew from experiences of interacting with the world. These types of knowledge are fundamentally

[21] To account for this issue, we may need much more elaborate, and task-specific, mechanisms that incorporate a priori knowledge (Anderson 1993), domain-specific constraints, complex human intuition (Bowers et al 1990), and logical/mathematical reasoning (Berry 1983), to come up with a set of rules. The detailed accounting of such learning is very much lacking in the existing literatures.

[22] The MCS may dictate when this learning is to be performed, the same as other learning methods.

different from each other in some sense. Therefore, we keep them separate in CLARION, represented by different rule sets (or "knowledge types"). [23]

2.3.8 Rule Support, Utility, and Base-Level Activation

Below, we will define a few useful numerical measures that are associated with rules (and also with action chunks in some cases), and then we will address their uses in rule-based reasoning.

2.3.8.1 Definitions of Some Numerical Measures

First of all, rule support measures the degree to which the conclusion of a rule is supported by the available information in relation to the condition of the rule. When partial match is not allowed, it is simply either 0 (not supported) or 1 (supported).

When partial match is enabled, a weighted sum is involved in computing rule support. As mentioned before, the condition of a rule is represented by a chunk, which is linked to the dimension-value representation at the bottom level. Generally speaking, dimensional values at the bottom level, once activated by input, go bottom up to activated all relevant chunk nodes. The strength of a chunk node (representing the condition of a rule) is set as follows:

$$S^c_{c_k} = \sum_i A_i * W_i^{c_k} \tag{2.10}$$

where $S^c_{c_k}$ is the strength of chunk k, A_i is the activation of the ith dimension of chunk c_k (concerning dimension i, specifying the extent to which the allowable value with the strongest activation in that dimension is activated), and $W_i^{c_k}$ is the weight of the ith dimension of chunk c_k. (where the default is $W_i^{c_k} = 1/n$, where n is the number of dimensions specified for chunk c_k). Then, based on the strength of the condition chunk of a rule, the support for the rule k is computed as follows:

$$S^r_k = S^c_{c_k} * W^r \tag{2.11}$$

[23] In reality, though, the boundaries between them are not so clear-cut. For example, one piece of knowledge may migrate from one rule set (one knowledge type) to another.

where k indicates the kth rule at the top level, S_k^r is the support for rule k, $S_{c_k}^c$ is the strength of chunk c_k (which represents the condition of rule k), and W^r is the weight of the rule (where the default is $W^r = 1$).

When there are multiple rules reaching the same conclusion, they are combined in the strength of the conclusion (action) chunk as follows:

$$S_{c_j}^c = \max_{\text{all rules } k \text{ leading to } j} S_k^r \tag{2.12}$$

where j indicates the jth action at the top level.

The formula for rule support allows "partial match" (as an option): When one or more of these dimensional values are not activated to the full extent of 1 or not present at all, the rule can still be applied, although it generates a conclusion with a relatively lower strength level (< 1). A threshold ($threshold_{PM}$) may be specified regarding the minimum strength level that is acceptable. However, we generally require that the goal component of a rule condition (if exists) be matched exactly (as in, for example, Anderson and Lebiere 1998). [24]

On the other hand, rule utility measures the effectiveness, or usefulness, of a rule, in terms of cost and benefit. Utility may be determined by functions that calculate utility values on the fly. The default formula is

$$benefit_j = \frac{c_7 + PM(j)}{c_8 + PM(j) + NM(j)} \tag{2.13}$$

where the default parameter values are $c_7 = 1$ and $c_8 = 2$, and

$$cost_j = \frac{\text{execution-time-of-rule-j}}{\text{average-execution-time-of-rules}} \tag{2.14}$$

where *execution-time-of-rule-j* and *average-execution-time-of-rules* need to be estimated and set (either as constants or as functions that compute these two quantities). Overall, we have

$$U_j^r = benefit_j - \nu \times cost_j \tag{2.15}$$

The interpretation of the measure is as follows: The benefit of a rule is calculated based on the positive match ratio—how many positive matches a rule may produce within the context of all the possible matches by the rule.

[24] Note that the above description of partial match is in fact an approximation of similarity-based reasoning (Sun 1995). See the chapter on the non-action-centered subsystem for a discussion of similarity-based reasoning.

The cost is calculated based on the execution time considerations: It can be calculated based on types of operations involved and their pre-determined costs. However, it need not be a precise measurement of execution times; it can be set as a constant, which varies from rule to rule (based on types of operations involved), roughly corresponding to the execution time of a rule. The utility of a rule is the comparison of its benefit and its cost (i.e., benefit minus cost with a scaling factor), as commonly used. [25]

Furthermore, we need a measure of past uses of knowledge, which is needed to capture recency effects (that is, priming effects). One such measure is base-level activation (or BLA) of Anderson (1993). The base-level activation of an action rule may be a recency-based value:

$$B_j^r = iB_j^r + c * \sum_{l=1}^{n} t_l^{-d} \tag{2.16}$$

where t_l is the lth encoding/use of the rule, the default values of the parameters are $c = 2, d = 0.5$, and iB_j^r is the initial value of B_j^r. This quantity specifies the odds of needing a particular rule based on the history of use of a rule (Anderson 1993), which decays gradually following a power law. It represents a priming effect on a rule resulting from prior uses of that rule. This priming effect is posited to be proportional to the odds of needing it (Anderson 1993). [26]

Similarly, the base-level activation of the resulting chunk from a rule (which is an action specification) may also be a recency-based value:

$$B_j^c = iB_j^c + c * \sum_{l=1}^{n} t_l^{-d} \tag{2.17}$$

where t_l is the lth encoding/use of the chunk (the action), the default values of the parameters are $c = 2, d = 0.5$, and iB_j^c is the initial value of B_j^c. This quantity specifies the odds of needing a particular chunk based on the history of its use (Anderson 1993). It represents a priming effect on a chunk resulting from prior uses of that chunk. [27]

[25] Alternatively, utility may be determined based two pre-set constants: *benefit* and *cost*.
[26] Note that one need to specify an initial base-level activation that represents prior experience before a simulation begins. The initial base-level activation is normally zero, unless a rule with which the BLA is associated exists before simulation starts.
[27] Again, one must specify an initial base-level activation that represents prior experience before a simulation starts. The initial base-level activation is normally zero, unless the chunk with which the BLA is associated exists before simulation starts.

2.3.8.2 Applications of the Numerical Measures

These measures can all be applied to the ACS.

An action recommendation takes into account the combination of the following factors: (1) how often and how recent an action or a rule has been selected, which determines to some extent how likely it will be useful at the current step, (2) how strong the condition of a rule is activated, which measure the degree of support that the conclusion has (including how many elements of the condition are matched and how strong they are activated), and (3) the utility of a rule, which measures its general usefulness. The first factor is represented by the base-level activation, and the second by the weighted sum in rule support, and the third by the utility measure. Thus, the impact of an action recommendation is determined by (1) the base-level activations of the rule applied and its resulting action chunk, (2) the support provided to the rule applied, and (3) the utility of the rule applied.

These measures may be used in selecting output. They may also be used in calculating the response time of applying a rule. There are many possible options for using these three measures. The specific choices made for the ACS of CLARION are explained below.

The output from the top level is determined based on a competition of all the rules matching the current input, based on utility. As an approximation (Luce 1959), we use the Boltzmann distribution to select an action recommendation out of all the rules whose conditions match the current input. This Boltzmann distribution is constructed using U_j^r of all the rules whose conditions match the current input, If rule utility is not used (which is an option), the Boltzmann distribution is constructed using the rule support. The probability of selecting a particular rule i is:

$$p(i|x) = \frac{e^{U_i^r/\tau}}{\sum_j e^{U_j^r/\tau}} \tag{2.18}$$

where x is the current state, i and j indicate rules matching the current state x (j ranges over all such rules), and τ controls the degree of randomness (temperature) of the decision-making process.

The interpretation of the Boltzmann selection process is as follows: It implements a "softmax" function. The stronger the utility of a rule is, the faster it reaches that asymptotic level. Consequently, the stronger the

utility of a rule is, the more likely that it reaches a utility threshold first (which is not explicitly specified in the model). Assume that the first rule reaches the threshold will be selected (to win the competition among all applicable rules). Thus, the stronger the utility of a rule is, the more likely it will be selected. (See Sun et al 1998, 2001 for further justifications.)

On the other hand, the decision time of a rule may be determined by the base-level activation of the rule as well as the base-level activation of the resulting action chunk (the action recommendation). The use of these two kinds of BLAs captures two kinds of priming effects: the priming of rules and the priming of action recommendations (Anderson 1993). See the chapter on RTs for further details regarding response times.

2.4 Integrating the Outcomes of the Two Levels

A number of methods for integrating the outcomes from the two levels of the ACS may be used (e.g., stochastic selection of levels, bottom-up rectification, and top-down guidance). In the following exposition of these methods, we assume that there is only one bottom-level network, and correspondingly, there is only one rule group. The extension of this, the coordination of multiple rule groups, will be addressed in a later section.

2.4.1 Stochastic Selection

Using *stochastic selection*, at each step, with probability P_{RER}, if there is at least one RER rule indicating a proper action in the current state, we use the outcome from that rule set (through competition based on rule utility); otherwise, we use the outcome of the bottom level (which is always available). [28] With probability P_{IRL}, if there is at least one IRL rule indicating a proper action in the current state, we use the outcome from that rule set (selected based on utility); otherwise, we use the outcome of the bottom level (which is always available). With probability P_{FR}, if there is at least one fixed rule indicating a proper action in the current state, we use the outcome from that rule set (selected based on utility); otherwise, we use the outcome of the bottom level (which is always available). With probability

[28] Note that, in that case, we update PM_{BL} or NM_{BL} (the positive or negative match counts of the bottom level) accordingly. But we do not update NM_{RER} or PM_{RER} because no RER rule was applied. The same below with regard to IRL and FR rules.

$P_{BL} = 1 - P_{RER} - P_{IRL} - P_{FR}$, we use the outcome of the bottom level. The selection probabilities may be determined based on the "utility" of each rule set, as discussed later. There exists some psychological evidence for such intermittent use of rules; see, for example, VanLehn (1991), Anderson (1993), and Sun et al (2001).

With this method, when we use the outcome from a rule set at the top level, we use the action recommendation of a stochastically selected rule. We stochastically select a rule, based on rule utility, out of all the rules matching the current state in a rule set. The probability of selecting a particular rule i is:

$$p(i|x) = \frac{e^{U_i^r/\tau}}{\sum_j e^{U_j^r/\tau}} \tag{2.19}$$

where x is the current state, a is an action recommendation, and τ controls the degree of randomness (temperature) of the process. This method is known as Luce's choice axiom; see Luce (1959).

When we use the outcome from the bottom level, we use a stochastic process based on the Boltzmann distribution of Q values for selecting an action:

$$p(a|x) = \frac{e^{Q(x,a)/\tau}}{\sum_i e^{Q(x,a_i)/\tau}} \tag{2.20}$$

where x is the current state, a is an action, and τ controls the degree of randomness (temperature) of the process. [29]

Note that deterministic selection of one level or the other is a special case of this method. Deterministic selection is useful when a task is completely implicit (using only the bottom level) or completely explicit (using only the top level).

With regard to the selection probabilities, they may be either (1) fixed (pre-set), or (2) variable. If they are fixed, one must specify those probabilities (as constants): P_{BL}, P_{RER}, P_{IRL}, and P_{FR}.

If these probabilities are variable, they are determined through a process known as "probability matching", as follows:

$$P_{BL} = \frac{\beta_{BL} * sr_{BL}}{\beta_{BL} * sr_{BL} + \beta_{RER} * sr_{RER} + \beta_{IRL} * sr_{IRL} + \beta_{FR} * sr_{FR}} \tag{2.21}$$

[29] If there are multiple action dimensions, the Boltzmann decision process takes place in each dimension individually, to select one value for each action dimension.

$$P_{RER} = \frac{\beta_{RER} * sr_{RER}}{\beta_{BL} * sr_{BL} + \beta_{RER} * sr_{RER} + \beta_{IRL} * sr_{IRL} + \beta_{FR} * sr_{FR}} \quad (2.22)$$

$$P_{IRL} = \frac{\beta_{IRL} * sr_{IRL}}{\beta_{BL} * sr_{BL} + \beta_{RER} * sr_{RER} + \beta_{IRL} * sr_{IRL} + \beta_{FR} * sr_{FR}} \quad (2.23)$$

$$P_{FR} = \frac{\beta_{FR} * sr_{FR}}{\beta_{BL} * sr_{BL} + \beta_{RER} * sr_{RER} + \beta_{IRL} * sr_{IRL} + \beta_{FR} * sr_{FR}} \quad (2.24)$$

where sr stands for success rate and β is a weighting parameter. [30]

Note that, although the bottom versus the top level is the primary distinction made, RER versus IRL versus FR is also a significant distinction. The elevated importance of the distinction among different rule types may be justified by positing that these different types of rules represent different "layers" of knowledge: Some are (almost) innate, some others are based on known knowledge templates, and yet some others are completely empirical—constructed from experiences of interacting with the world. Therefore, they are substantially different in many ways, and the selection among them has to be made quite early on in combining decisions from different components.

2.4.2 Combination

Another possibility is combining rule utility (or rule support if rule utility is not used) with Q values of the bottom level first and then selecting an action based on the combined values, using the Boltzmann distribution. A positive conclusion reached by the ARS can compete with (add to) the action recommendation by the IDNs. A negative conclusion reached by the ARS can be used to "veto" (in some sense) the corresponding action

[30] In the above equations, $sr_{BL} = \frac{c_3 + \Sigma PM_{BL}}{c_4 + \Sigma PM_{BL} + \Sigma NM_{BL}}$ where the summation is over all state/action pairs matching the input/output of the bottom level; c_3 and c_4 are constants representing the prior (the default is $c_3 = 1$ and $c_4 = 2$). Similarly, $sr_{RER} = \frac{c_3 + \Sigma PM_{RER}}{c_4 + \Sigma PM_{RER} + \Sigma NM_{RER}}$, $sr_{IRL} = \frac{c_3 + \Sigma PM_{IRL}}{c_4 + \Sigma PM_{IRL} + \Sigma NM_{IRL}}$, and $sr_{FR} = \frac{c_3 + \Sigma PM_{FR}}{c_4 + \Sigma PM_{FR} + \Sigma NM_{FR}}$. The parameters, $\beta_{BL}, \beta_{RER}, \beta_{IRL}$, and β_{FR}, provide differential emphases to different knowledge types, which reflect innate biases, and they must be pre-specified.

recommendation by the IDNs through competing with (adding to) the outcomes of the IDNs.

We combine the corresponding values for an action from the two levels by a weighted sum. That is, at the top level, the RER rule set indicates that action a has a value v_a^{RER}, which should be the maximum rule utility for that action (i.e., the highest utility from the rules matching the current state and recommending action a), [31] or rule support if rule utility is not used. Similarly, the IRL rule set indicates that action a has a value v_a^{IRL}. The fixed rule set indicates that action a has a value v_a^{FR}. The bottom level (the IDNs) indicates that a has a value q_a (i.e., the Q-value of a). Thus, the combined value for action a is $v_a = w_{RER} * v_a^{RER} + w_{IRL} * v_a^{IRL} + w_{FR} * v_a^{FR} + w_{BL} * q_a$. Stochastic decision making with Boltzmann distribution, based on combined values for different actions generated by the weighted sum, is then performed to select an action out of all the possible actions. [32]

There are some indications of the plausibility of this method. There are reasons to believe that, in skill learning and performance, the top-level outcome is sent down to the bottom level, and there it is combined with the bottom-level outcome (Sun et al 2002). It has been shown that explicit knowledge influences skilled performance in humans, but explicit knowledge does not directly control skilled performance (Stanley et al 1989). Alternatively, in reasoning, there are indications that the bottom-level outcome is sent up to the top level, and there it is combined with the top-level outcome (Sun 1995). Research has shown that implicit knowledge influences explicit reasoning, although the influence is not direct either (Nisbett and Wilson 1977, Sun and Zhang 2003).

With regard to the weights, they may be either (1) fixed (pre-set), or (2) variable. If they are fixed, one must specify these weights as constants: w_{BL}, w_{RER}, w_{IRL}, and w_{FR}. If they are variable, "probability matching" equations similar to those used in the variable stochastic selection case discussed earlier are applicable for determining w_{BL}, w_{RER}, w_{IRL}, and

[31] That is,

$$v_a^{RER} = \max_{j:\text{leading to } a} U_j^r \tag{2.25}$$

[32] If there are multiple action dimensions, the Boltzmann decision process takes place in each dimension individually, to select one value for each action dimension.

w_{FR}.

Furthermore, the weighted-sum method may be viewed as one example of more general (and more complex) ways of integrating the outcomes of the two levels. Such more general ways of integrating the two levels include bottom-up rectification and top-down guidance. In *bottom-up rectification*, the bottom-level outcome is sent up to the top level, and there it is integrated with the top-level outcome. This is likely to happen in reasoning situations (Nisbett and Wilson 1977). The integration at the top level may take the form of the top level utilizing and rectifying the outcome from the bottom level with the knowledge at the top level. Using weighted-sums to put together both types of knowledge is but one possible way of accomplishing this; other, more complex ways are also possible, and they have been used in some prior simulations (see Mathews et al 2004). In *top-down guidance*, the top-level outcome is sent down to the bottom level, and there it is integrated with the bottom-level outcome. This happens mostly in skill learning and skilled performance. The integration at the bottom level may take the form of the bottom level utilizing the outcome of the top level, along with its own knowledge, in making its action decisions. Again, using weighted-sums is but one possible way of accomplishing this.

Given all of the afore-discussed possibilities, the way by which the outcomes of the two levels are integrated in a particular task domain may need to be determined based on task domain characteristics. Some of the parameters involved may be set by a meta-cognitive subsystem (see the chapter on that subsystem).

2.5 Goal Structure and Working Memory *

2.5.1 Goal Structure *

The goal structure (or GS for short) is used to bring focus to the activities of an agent: organizing its actions in a prioritized, coherent, and orderly way, in relation to the desires and needs of the agent.

A goal structure provides a way of representing various motivations and their interactions in an agent (e.g., Anderson 1983, Rosenbloom et al 1993). A goal that is currently active represents a certain drive state that is the result of combining various desires, needs, and priorities of the

present time. Besides making explicit pre-wired, (largely) implicit, internal drives, a goal structure may also represent explicitly set goals in explicit reasoning and problem solving (e.g., Anderson 1993). Furthermore, the goal structure may also be used to handle externally given goals (which may be received through instructions or other means of explicit communication). A practical reason why a goal structure is used is that it provides specific and tangible motivations for the actions chosen and performed by the ACS. In the ACS, actions are selected based on values of actions with regard to the current state and the current goal. Such values are the motivating force for performing specific actions. The goal structure provides a framework for motivation (see the chapter on the motivational subsystem for more details).

Using goals is often preferable to direct mappings from sensory input to actions. For example, it avoids the problem of "dithering". See further treatment of this topic in the chapter on the motivational subsystem. [33]

In CLARION, the goal structure can be either in the form of a list, from which an active goal can be selected, or in the form of a stack, in which case the top item on the stack is the active goal.

*2.5.1.1 Goal Stack **

Let us look into the goal stack first. A goal stack is a linear structure of multiple items in which only the top item may be accessed. Items may be removed from or added to the top of the stack. Therefore, only one goal item, at the top of the stack, may be active at a time. A currently active goal item becomes inactive when another goal item is added to the top of the goal stack. A previously active goal item (now in the middle of

[33] Suppose an animal has equal food and water needs and situates between food and water sources. The animal may visit the food source first and reduce its food deficit only slightly. Then, because its water need becomes slightly more urgent, it may rush to the water source, similarly reduce its water deficit slightly, and then rush back to the food source, and so on. Through setting up goals (such as "getting a sufficient amount of water" and "getting a sufficient amount of food") and using the action routines for these goals respectively, the animal my be more persistent on the activity related to one goal before switching to another (Tyrell 1993). For another example, in order to capture a prey, it may be necessary to move away from the prey temporarily. Such a behavior may be best captured by a subgoal of "getting behind the prey to avoid detection" in the service of the goal "capturing the prey".

the stack) may become active again when the goal items on top of it are removed.

Representationally, at each slot of the stack, a goal symbol and its parameters are represented at the top level in the form of a chunk (that is, a set of dimension-value pairs represented together by one chunk node). Each goal chunk contains a goal dimension, which specifies one value out of the (finite, pre-specified) set of all possible values of the goal dimension. That is, it specifies one goal symbol out of the pre-determined set of all possible goal symbols. Each goal chunk also contains a number of parameter dimensions. Each parameter dimension specifies one value out of the set of all the possible values. The size of a goal chunk (i.e., the number of optional parameter dimensions) may be determined based on characteristics of specific task domains. [34] Goal chunks at the top level of the stack are created whenever they are needed (e.g., due to a goal action that creates a goal item on the stack). The top goal item on the goal stack (i.e., the active goal) is represented by a goal chunk at the highest non-empty slot of the stack.

The goal stack is also represented at the bottom level. A goal chunk node is linked to its *feature* nodes at the bottom level that represent the dimension-value pairs specified by the chunk. These feature nodes are connected to the parts of the input layers of the IDNs specifically designated for goal stack information (as was discussed previously).

Goal actions are used for pushing a goal item onto the goal stack, or for popping a goal item off the goal stack. The actions are in the form of "*push i* {dim, value}" or "*pop*", where i is one of the possible goal symbols (which are specific to a particular domain), and {*dim, value*} denotes a set of (optional) parameter dimension-value pairs that are set along with the goal i. There is also an additional action *do-nothing* (no goal action to be performed).

Goal actions can be performed either by the top level alone (using rules, especially FRs), or by a combination of the top level and the bottom level.

A goal stack allows the emergence and application of "routines" (or "subroutines"), that is, relatively fixed patterns of behaviors (in other

[34] The size of the goal stack (the maximum number of goal items on the stack) is also determined based on characteristics of task domains.

words, a frequently repeated sequence of actions; Lorenz 1950, Tinbergen 1951, Tyrell 1993, Sun 2001). Once a goal item is pushed onto the stack, a routine (or subroutine) for accomplishing the goal, if exists, is automatically initiated, through action selections that are suitable for accomplishing the goal at every step of the way. An initiated routine can keep running, until interrupted or terminated. They may be terminated due to the popping of the current goal item off the stack. They may also be interrupted due to a new goal item being pushed onto the stack.

The problem with the goal stack is that it is too idealistic for modeling cognitive processes realistically: An agent invariably pushes a sequence of subgoals of the current goal onto the stack; when a subgoal is completed, it reliably returns to the previous goal. Much subtlety and complexity in human activities involving goal coordination are lost. Thus, a goal stack is merely a rough approximation and abstraction of a complex motivational and meta-cognitive process. We next turn to the goal list, which is a more realistic approach toward cognitive modeling of motivational processes. [35]

*2.5.1.2 Goal List **

A goal list is a randomly accessible, linear structure that contains a set of goal items, each with a base-level activation and constituted by a set of dimension-value pairs. Each goal item is represented as a chunk node at the top level of the goal list, which describes a set of dimension-value pairs (including the goal dimension and its value, as well as a number of parameter dimensions and their values). The goal chunk is connected to the nodes at the bottom level that represent its features (dimension-value pairs, which are connected to the input layers of the IDNs). The size of a goal chunk (the number of parameter dimensions) is determined by domain characteristics. [36]

Actions on a goal list are similar to the actions on a goal stack. The actions are in the form of "*set i* {dim, value}" or "*reset i* {dim, value}", where i is one of a set of possible goal symbols (which are specific to a particular domain), and {*dim, value*} represents a set of (optional)

[35] The goal list requires more effort to implement in simulations. This is the reason why the goal stack is included in CLARION.

[36] The size of the goal list (the maximum number of goal items on the list) is also determined based on characteristics of task domains.

parameter dimension-value pairs that are set along with the goal i. The goal setting action can set a goal on any of the available (empty) slots on the goal list. The exact location on the list does not matter. The resetting action removes the goal chunk(s) that matches the goal symbol and the (optional) parameter dimension-value pairs that come with the action. There is also an additional action *do-nothing*.

Each goal chunk has a base-level activation (BLA) associated with it. The BLA is set when the goal is initially set. The BLA decays slowly over time. That is,

$$B_i^g = iB_i^g + c * \sum_{l=1}^{n} t_l^{-d} \tag{2.26}$$

where i indicates an item (a goal chunk) on the goal list, l indicates the lth setting of that item, [37] t_l indicates the time since the lth setting of that item, and iB_i^g is the initial value of B_i^g (normally uniformly set at, e.g., $iB_i^g = 0$). c and d are constants.

The same as goal stack actions, goal list actions can be performed either by the top level alone (using rules, especially FRs), or by a combination of the top level and the bottom level.

Goal items (with BLAs) on a goal list compete with each other to become *active*. Some restrictions is necessary: Only goal items with $B_i^g > threshold_{GS}$ participate in the competition of the goal list. The competition is through the Boltzmann Distribution. Only one goal item (the one that wins the competition) becomes active (similar to the top item of the goal stack). The active goal item is accessible to the bottom level and the top level of the ACS; all the other goal items are not. The strength of the active goal is 1.

Such a goal list can also generate "routines" ("subroutines" or "fixed response patterns") mentioned earlier. One way is through approximating, in some way, stack-like behaviors. Decaying traces of goal setting activities (i.e., base-level activations), coupled with the moment-to-moment state information, can lead to a last-in-first-out way of handling goals. See, for example, Altmann and Trafton (2002) for discussions of this approach.

In the simplest case, an agent can use a rule like the following one to handle the creation of subgoals:

[37] Here, storage/encoding is what matters, not retrieval/use.

> If the current goal is not complete and there is an obstacle to be removed, then create a subgoal to deal with the obstacle and initiate a goal competition.

The newly activated subgoal wins the competition, due to its higher base-level activation (because it is more recent). Then the agent can focus on the subgoal, which is now active. This goal switching is accomplished because the new goal has a higher BLA (not because of it being pushed onto the top of the stack). If there is no other interruption, this subgoal will be accomplished. Once the subgoal is accomplished, it is removed (by a separate goal action rule). At that point, a new round of competition is initiated, and the original goal (that spawned the subgoal) may become active again.

The goal list can handle more complex or more subtle situations that the goal stack cannot easily handle. For example, in order to schedule several activities at the same time, such as carrying on a conversation while searching for a file in the desk drawer, the priorities of different goals, such as "continuing conversation" and "continuing file search", can change dynamically. As the delay time since a question was asked grows, the goal of "continuing conversation" becomes increasingly more important. On the other hand, when the conversation drags on, the goal of "continuing file search" may become dominant again. This kind of alternation of goals may be difficult for the goal stack to handle. [38]

2.5.2 Working Memory *

The Working memory (WM for short) is for storing information on a temporary basis for the express purpose of facilitating subsequent action decision making and/or reasoning.

The working memory involves: (1) action-directed ("deliberate") encoding of information (as opposed to "automatic" encoding), (2) gradual fading of information, (3) action-directed re-encoding (refreshing) of information, and (4) a limited storage capacity. The working memory is in

[38] As will be explained later in the chapter on the motivational subsystem, the setting of goals is carried out by a meta-cognitive subsystem. When the drive for a certain type of action grows, a goal setting action is carried out by the meta-cognitive subsystem. Then, a round of goal competition ensues, which decides on a new active goal (very likely the newly set goal).

the service of the activities of an agent—It is directed by the agent' actions and in turn serves its action decision making.

We stipulate that the information to be stored into working memory be copied from the chunks in the non-action-centered subsystem (which will be described in detail later). This is due to the following two considerations: (1) we must be able to retrieve and hold in working memory items from long-term memory, and (2) we must be able to extract information in the current state, which is, incidentally, also recorded in the non-action-centered subsystem. Therefore, copying from the non-action-centered subsystem is minimally necessary, as well as sufficient, for storing information into working memory.

Working memory may be naturally divided into multiple sections, each for a different kind of sensory information: visuospatial information, auditory/verbal information, and other types of information, respectively. Each section of working memory consists of a certain number of slots. Each slot can hold the content of a chunk (termed a WM item). As usual, a chunk describes a set of dimensions and their respective values. The chunk node at the top level of the working memory is linked to its feature nodes at the bottom level (which are connected to the input layers of the IDNs), where each value of each dimension is represented by an individual node. The number of dimensions of a working memory item is limited (as determined by domain characteristics). The total capacity of the working memory (the maximum number of items in the working memory) is also limited.

Note that we may view the goal structure as part of the working memory, although in CLARION we separate the two structures. A goal structure is somewhat different from other parts of the working memory, primarily due to the use of a set of special goal symbols in directing and coordinating an agent's activities.

Working memory actions may be used for storing an item into working memory, or for removing an item from it. The action may be in the form of "*set* i", whereby the content of a chunk in the retrieval buffer of the non-action-centered subsystem (to be described later) is stored into a working memory slot i and the item is endowed with an initial base-level activation (in accordance with the BLA specification below). The action may also be in the form of "*set* $\{i\}$", whereby the content of a number of chunks from the retrieval buffer is stored into a number of working memory

slots as specified by the action (where $\{i\}$ denotes a sequence of numbers representing working memory slots). [39] They may also be in the form of "*reset i*", whereby the stored values of the ith working memory item (the ith slot) is removed (that is, the corresponding base-level activation is set to 0). Two additional working memory actions are *reset-all-WM* and *do-nothing*.

As mentioned before, there may be special working memory items called flags. Flag-related actions include *set* $flag_i$, *reset* $flag_i$, and *reset-all-flags*.

Working memory actions can be performed either by the top level alone (using rules, especially FRs), or by a combination of the top level and the bottom level.

Working memory base-level activation helps to determine how long past information should be kept around, when there is no ("deliberate") reset action. We may adopt recency-based working memory base-level activation:

$$B_i^w = iB_i^w + c * \sum_{l=1}^{n} t_l^{-d} \tag{2.27}$$

where i indicates an item in the working memory, l indicates the lth setting of that item, t_l is the time since the lth setting of that item, and iB_i^w is the initial value of B_i^w (normally set uniformly, e.g., $iB_i^w = 0$). c and d are constants. [40]

This base-level activation represents the odds of needing a particular item based on past history (Anderson 1993). The fading of information is inversely proportional to the odds of needing it.

In using the working memory items, if the base-level activation of working memory item i is above a threshold ($B_i^w > threshold_{WM}$), the item is used as input to the IDNs and the ARS (with strength 1); otherwise, the strength is set to zero (as if the item does not exist).

[39] In both of the above two cases, the selection of a chunk to be stored into a working memory slot is done with a Boltzmann distribution.

[40] Alternatively, we may use a constant working memory activation for all working memory items at all times. For example, the default may be $B_i^w = 1$, where i indicates the ith item in the working memory. In this case, there is no time limit on working memory items, unless they are reset by a reset action.

This picture of working memory is, in a way, similar to what Baddeley (1986) had in mind, in terms of (1) the existence of the multiple working memory "loops", and (2) the role of covert rehearsal, and so on. It also coincides with Jackendoff's (2000) view concerning a separate representation of working memory, which was argued by Jackendoff (2000) based on recurrences of words/concepts. However, it differs with Anderson's (1993) view, in that working memory items need to be set (and reset) by a specific action, not merely through activation propagation in a general long-term memory system whereby working memory is simply activated items in that general memory (as in ACT-R). The reason is that we believe storing an item in the working memory is a decision that an agent has to make, taking into consideration the utility of doing so (e.g., maximizing total payoffs in some way). Another significant difference to keep in mind is that in CLARION, working memory is used solely for action decision making, not for general memory access. Any other access of working memory information has to be through actions, that is, in an indirect way.

2.6 Coordinating Multiple Action Types *

First, let us look into the coordination of multiple external actions. When there are multiple external action networks and thus multiple external actions rule groups, we need to coordinate the different external actions selected from these different modules (networks along with their respective rule groups). Although an eligibility condition may be specified so that not all bottom-level networks, and correspondingly not all rule groups, may be applicable at a step, there is a possibility that at a particular step, there are multiple applicable/eligible modules (networks and their respective rule groups). Let us consider the coordination of these networks and rule groups below.

Assume that stochastic selection is used for integrating the outcomes of the two groups. In that case, when one knowledge type (RER, IRL, or FR) from the top level is chosen, the corresponding rule set (RER, IRL, or FR) from each external action rule group selects one action (in the way explained in the previous section, based on a Boltzmann distribution), and then these actions (from different rule groups) need to be "coordinated". When the bottom level is chosen, similarly, each external action network selects one action (as explained in the previous section), and then these

actions need to be "coordinated" too.

Assume that a more complex method is used for integrating the outcomes of the two levels (including using the weighted-sum method). In that case, "coordination" is based on the combined outcomes of the corresponding pairs of external action networks and external action rule groups. Each combined pair of external action network/group selects one external action. Then these different external actions from these different pairs need to be "coordinated".

In general, regardless of which method is used, to coordinate these different external actions, we can (1) either choose one external action to perform, randomly, from these selected external actions (from all the eligible/applicable external action modules), or (2) perform all the external actions selected (from all the eligible/applicable external action modules), The choice is domain specific and can be varied.

Second, let us look into the coordination of goal action, WM action, and external action networks. When there are a goal action network, a WM action network, and external action network(s), there are essentially two possibilities for the selection of action types:

- Random selection: The selection between external, WM, and goal actions is based on a probability distribution. The parameters are: P_{EXT}, P_{WM}, and P_{GS}.
- External actions first: Use the chosen external action, provided that the chosen external action is not *do-nothing*. (If the chosen external action is *do-nothing* or if there is no external action available, use a goal action or a WM action.) This is a special case of the previous approach, in which $P_{EXT} = 1$, $P_{WM} = 0$, and $P_{GS} = 0$.
- Perform the chosen action(s) of each of all the action types simultaneously. (If one action type is not available or is *do-nothing*, then we may simply ignore it.)

See Appendix for further details of coordination algorithms.

Note that it is possible that there is no distinction of the goal action module, the WM action module, and the external action module(s)—the same network (and its corresponding rule group) is used to recommend all three types of actions. In that case, there is in fact no issue of coordination.

We simply treat goal actions, WM actions, and external actions as different dimensions of one action specification, and perform them together. [41] [42]

Note also that goal actions are secondary, mostly helping external actions by providing goal information through setting up goal representations in the goal structure. If goal actions are to be learned, hopefully, for the sake of coordinating the two types of actions, while the goal action network or rules learn to set up goals, the external action network or rules will learn to output *do-nothing*. And vice verse. [43] But this is not necessarily the case. Therefore, alternatively, the coordination may be pre-programmed into a model (e.g., through a set of fixed rules). Note that WM actions are similar to goal actions in this regard, and can be handled in a like manner.

entity	evidential support	selection	response time determination
rule	rule support (w/ or w/o PM)	utility	bla
chunk	strength	n/a	bla
WM item	n/a	bla	n/a
GS item	n/a	bla/LIFO	n/a

Figure 2.3. Summary of mechanisms in the ACS.

[41] For example, if both goal actions and external actions are recommended by the same network or by the same selected rule, then there is no issue of coordination—both types of actions are performed at the same time. If goal actions and external actions are recommended by different rules in the same rule group, then there is no issue of coordination either—the same rule selection method (as described before) is used to select only one action. Other cases are dealt with similarly.

[42] It is also possible that two of the three action types are combined while the third type remains separate. In that case, the coordination would be between the two modules (one for the two combined action types and the other for the remaining one).

[43] Learning may be done with a special reinforcement function for goal actions.

2.7 Appendix: Details of Backpropagation Learning

In case that backpropagation, simplified Q-learning, or Q-learning is used to train a network, based on *err* measures, we have the following learning rules, as usual with backpropagation learning. For adjusting output weights (from hidden units to output units):

$$\Delta w_{ji} = \alpha x_{ji} \delta_j$$

$$\delta_j = err_j o_j (1 - o_j)$$

where w_{ji} is the weight associated with the ith input to output unit j (from the ith hidden unit), x_{ji} is the ith input to output unit j (from the ith hidden unit), o_j is the output from output unit j (i.e., $Q(x, a_j)$), and α is the learning rate.

For adjusting hidden weights (from input units to hidden units):

$$\Delta w_{ji} = \alpha \delta_j x_{ji}$$

where w_{ji} is the weight associated with the ith input to hidden unit j (from the ith input unit), x_{ji} is the ith input to hidden unit j (from the ith input unit), α is the learning rate, and

$$\delta_j = o_j (1 - o_j) \sum_k \delta_k w_{kj}$$

where o_j is the output from hidden unit j, k denotes all the units downstream (in the output layer), w_{kj} is the weight associated with the j input to the kth output unit (from the jth hidden unit), and $delta_k$ is equal to $err_k o_k (1 - o_k)$.

2.8 Appendix: Some Options of RER

We may *merge* rules: If either rule extraction, generalization, or specialization has been performed, check to see if the conditions of any two rules are close enough and thus if the two rules may be combined: If one rule is covered completely by another, put it on the children list of the other. If one rule is covered by another except for one dimension, produce a new rule that covers both. One may choose how frequently the *merge* operation is performed.

In the afore-described rule generalization/specialization, values are added or deleted from an input dimension. Any value from any dimension may be chosen. This is the default. It is also the only way when these dimensions are nominal. On the other hand, if a dimension is ordinal, there are the following options: We can either add/delete an arbitrary value, which results in non-contiguous value ranges for some input dimensions in a rule condition, or we make sure that we have contiguous value ranges by adding/deleting values only at the two ends of the current ranges.

Another option is that when we generalize, we can either add one value at a time to a dimension, or add all the values of that dimension. In case the "all values" option is adopted, the IG calculation as specified before needs to be altered: It should be done with respect to $C' = C$ plus all values. Similarly, when we specialize, we can remove one value at a time from a dimension, or we can remove all but the last value (if there are two or more values) from a dimension. [44] In case we use the "all values" option, the IG calculation specified earlier can be similarly altered, with respect to $C' = C$ minus all values except one (if there are two or more values).

Beside the default IG measure, one may specify alternative IG measures. Similarly, one may also specify alternative positivity criteria used there.

As evident in the discussion above, this (default) extraction method is suitable for specific-to-general rule learning (Bruner et al 1956, Haygood et al 1970). That is, the agent extracts a most specific rule and then tries to generalize it later. An alternative, for general-to-specific rule learning, is to extract a rule with a small condition involving, for example, only one input dimension (or only a few), and then to try to specialize it. In this case, to extract a rule, we do the following:

- *Extraction*: If the current step is positive (according to the current extraction criterion), and if there is no rule that covers this step in the top level (matching the state and the action), set up a rule $C \longrightarrow a$, where C specifies a (randomly) chosen set of values of a (randomly)

[44] If there is only one value left in each dimension, removing that value is allowed, which amounts to deleting the rule. If there are more than one value left in a dimension, we can remove all but one value from that dimension.

chosen input dimension consistent with the current state x and a is the action performed at the current step.

In the current implementation of CLARION, one may choose to use either one, or both, of the extraction methods. [45]

2.9 Appendix: Some Details of Cross-level Integration and Action Coordination

The coordination process is as follows:

I. First select one knowledge type (by using variable or fixed stochastic selection), or combine the two levels (by using a more complex method, such as weighted-sums).
II. Using the selected knowledge type or using the combined values, if there are multiple possible external actions, choose one (or choose all, as described before).
III. Then (with the selected knowledge type or the combined values), select among the chosen external action, the chosen goal action, or the chosen WM action:

Use one of the following two methods:
1. Select one type of action, according to a pre-specified probability distribution. (This method includes "external action first" as a special case.) If the selected action type is not available or indicates *do-nothing*, select one of the remaining two action types stochastically. If one of the two remaining actions is not available or indicates *do-nothing*, then select the sole available action type. If none of the action type is available or indicates anything other than *do-nothing*, do nothing.
2. Perform simultaneously all the chosen actions of each of the three action types. (If one action type is not available or is *do-nothing*, ignore it.)

[45] Regardless of whether the general-to-specific or the specific-to-general method is adopted, once a rule is extracted, generalization and specialization may be applied to it the same way. The only difference is in the initial extraction.

2.10 Appendix: AND/OR Activation Functions

As discussed before, in the simplest case, AND/OR links can be implemented as logical operations: For an AND link, if all of its source nodes are activated (to a strength level of 1), then its target node is activated (to a strength level 1). For an OR link, if any of its source nodes is activated (to a strength level of 1), then its target node is activated (to a strength level 1).

Alternatively, we may implement AND or OR with *min* and *max* respectively. As an example, Figure 2.4 shows an implementation of the following rule:

$$if\ a\ and\ b,\ then\ c\ (with\ confidence\ w)$$

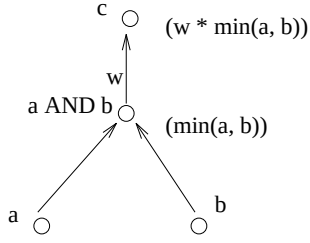

Figure 2.4. A Network Using MAX
(implementing the rule: *if a and b, then c*)

In addition, for an AND link, there may be a weighted-sum activation function that calculate the activation of the chunk node based on partial match of features: that is, based on how many of its feature nodes (dimension-value pairs) are activated (i.e., how many dimensions have an allowable value).

However, potentially, we may need to adjust these links through learning algorithms. Gradient-descent learning algorithms require that activation functions be differentiable. To facilitate the training of a connectionist network involving AND/OR links, we may use a connectionist implementation of them, which is a close approximation of AND/OR operations and makes them differentiable.

A connectionist implementation is as follows. Suppose that one value is specified for each feature dimension in a chunk. If the specified value of a

feature dimension is in positive form, the link from the specified dimensional value to the chunk node carries a positive weight, say w; if the specified value of a feature dimension is in negative form, the link carries a negative weight, say $-w$. The threshold of the chunk node is set to be $n*w-\theta$, where n is the number of incoming links (the number of feature dimensions leading to the chunk represented by this node), and θ is a parameter to make sure that the node has activation approximately equal to 0.9 or above, when all of the specified values of its feature dimensions are activated (to an activation approximately equal to 1), and has activation approximately equal to 0.1 or below, when some of the specified value of its feature dimensions are not activated. That is, θ is used to separate *true* and *false* so that they can be detected easily. This is an implementation of AND.

However, if there are more than one allowable values in a feature dimension, an intermediate node is created for each such dimension. All the allowable values in one feature dimension are linked to the same intermediate node; then, all the intermediate nodes are linked to the chunk node in the same way as stated earlier. The activation function of the intermediate node, however, is different: If a specified dimensional value (among multiple such values in that dimension) is in the positive form, the link from that dimensional value to the intermediate node carries a positive weight, say w; if it is in the negative form, the link carries a negative weight, say $-w$. The threshold of the intermediate node is set to be $n*w-\theta$, where n is the number of incoming links (the number of allowable values in the dimension), and θ is a parameter to make sure that the node has activation approximately equal to 0.9 or above, when any of the allowable value in the dimension is activated (to an activation approximately equal to 1), and has activation approximately equal to 0.1 or below, when none of these allowable values is activated. This is an implementation of OR.

Note that one potential discrepancy between the connectionist implementation and the logical one is that when the logical implementation produces a 1, the connectionist one may produce a slightly different value (e.g., 0.9); when the logical implementation produces a 0, the connectionist one may again produce a slightly different value (e.g., 0.1). Note also that this connectionist implementation does not deal with partial match. If the option of partial match is selected, a weighted-sum activation function should replace the connectionist implementation of AND (while

the connectionist implementation of OR remains the same).

Chapter 3

The Non-Action-Centered Subsystem

The non-action-centered subsystem is for storing and retrieving general (non-action-centered) knowledge of the world in various forms. This subsystem is close to what has been variously referred to as semantic memory or declarative memory in the literature. In its various forms, it has occupied major roles in many previous theories of cognition. Its role is somewhat diminished in CLARION, however. In CLARION, it is mainly used for supplementing and augmenting the action-centered subsystem.

This subsystem can be formed through acquiring and assimilating general (non-action-centered) knowledge, given from external sources (e.g., by the ACS), or from summarizing experiences of the world during action decision making. Let us discuss details below.

3.1 Representation

Let us first look into representations used in this subsystem, both at the top level and at the bottom level. See Figure 3.1 for a diagrammatic view of the NACS.

3.1.1 The Top Level

At the top level, a general knowledge store (GKS) encodes explicit, non-action-centered knowledge. In this structure, chunks (sets of dimension-value pairs) encode co-occurrences of features. Links across chunks (hereby termed *associative rules*) encode (uni- or bi-directional) associations between chunks.

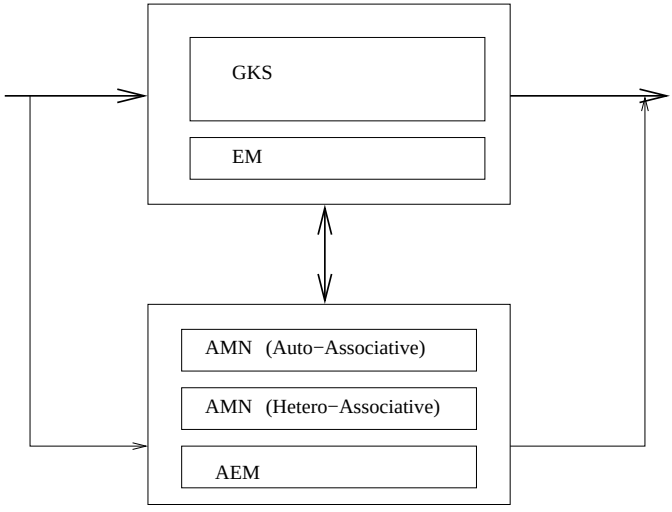

Figure 3.1. The non-action-centered subsystem.

Note that, similar to the ACS, the bottom level of the NACS may be divided into multiple networks, each for a different kind of information: for instance, visuospatial information, auditory/verbal information, and other types of information, respectively. Therefore, correspondingly, the top level of the NACS may also be divided into multiple rule groups in accordance with bottom-level networks.

3.1.1.1 Chunks

Similar to what was described for the ACS, the basic form of a chunk in the NACS is as follows: *chunk-id:* $(dim_{i_1}, val_{i_1})(dim_{i_2}, val_{i_1})$ ..... (dim_{i_n}, val_{i_n}), where *dim* denotes a particular dimension, and *val* specifies the corresponding value of that dimension. For example, *table-1: (size, large) (color, white) (number-of-legs, four)* specifies a large, four-legged, white table. The *chunk-id* may be externally given (if the chunk is presented from external sources) or may be generated (randomly) internally (if the chunk is formed internally; see the discussion of learning later).

A node is set up at the top level (the GKS) to represent a chunk. The chunk node connects to its features in the bottom level. That is, each chunk node is linked to its corresponding dimensional values in the bottom level

(i.e., to the nodes in a corresponding bottom-level network representing different values of different dimensions contained in the chunk). [1]

In the simplest case, such linking can be accomplished using AND/OR functions (the same as action rule condition chunks in the ACS). For example, a node may represent the following chunk of features: *(temp, warm) (rainfall, heavy)*. Thus, it is linked to the nodes at the bottom level that represent *(temp, warm)* and *(rainfall, heavy)*. Through a *bottom-up activation* process, which carries out an AND operation, when all of these dimensional values have a strength level of 1, the strength level of the chunk node can be set to 1. (A *min* function may be used to implement AND.) However, if multiple values in a certain dimension are allowed, an OR operation combines these values: As long as the strength of one of these allowed values is 1, this dimension is considered to have an activation 1. (A *max* function may be used to implement OR.) [2] Conversely, when a chunk is "activated" to a certain strength level, through a *top-down activation* process, its features at the bottom level can be activated as well. [3] The idea is the same as detailed before with regard to the ACS.

A density parameter (d_c) determines the minimum frequency of invocation (encoding, re-encoding, extraction, re-extraction, or activation) necessary in order to keep an existing chunk. For example, if $d_c = 1/100$, then there should be at least one invocation (encoding, re-encoding, extraction, re-extraction, or activation) of a chunk for every 100 steps in order to keep it. The parameter applies to all existing chunks (except those in episodic memory). Note that, if a chunk is removed, all associative rules linked from/to it are removed as well.

[1] Note that these chunk nodes from the top level of the NACS are connected mainly to the feature (dimensional value) nodes at the input side of the bottom-level networks, but the output side may also affect the chunks at the top level of the NACS. See the later section on reasoning regarding the exactly controlled flow of information between the two levels in the NACS as well as the timing of the information flow.

[2] Optionally, when dimensional values are partially activated (with certain partial strength levels), or when only a subset of these values are activated, the process may partially activate the chunk node (with a partial strength level between 0 and 1, inclusive). See the discussion of similarity-based reasoning later.

[3] The features are usually activated to the same strength level as the chunk. However, if one of the dimensions of a chunk has multiple allowable values (i.e., multiple disjuncts), the strength of the chunk is evenly split among these allowable values of the same dimension. If a dimensional value receives activations from multiple chunks, the *max* of these activations is used.

Dimensional values in the bottom level serve as features of a chunk, and the chunk in turn serves to identify and label this set of features as a whole. Each chunk is a concept that can be used for reasoning.

3.1.1.2 Associative Rules

The condition of an associative rule consists of a single chunk or multiple chunks. These chunks are represented as chunk nodes at the top level whereas their dimensional values are represented as individual feature nodes at the bottom level. The conclusion of an associative rule constitutes one chunk, which is represented as a chunk node at the top level. The chunk is connected to the corresponding dimensional values at the bottom level (in the form of individual feature nodes). [4] Given the chunk representation, this form of associative rule is simple, natural, and uniform —representing mappings from a set of dimension-value pairs to another, in an explicit form but without overly complex structures. [5] For details of reasoning with associative rules, see the later section on reasoning.

There are parameters associated with associative rules, similar to the case of action rules in the ACS. In particular, the density parameter (d_a) of an associative rule determines the minimum frequency of invocation (encoding, re-encoding, extraction, re-extraction, or application) necessary in order to keep an existing associative rule. For example, if $d_a = 1/100$, then there should be at least one invocation of an associative rule for every 100 steps in order to keep it.

3.1.1.3 Chunk Strength

Chunks may be "activated", either as a result of applying an associative rule linking an activated chunk to a dormant chunk, or as a result of receiving an input (e.g., from the action of a rule in the ACS).

Specifically, (1) all the chunks whose dimension-value specifications constitute a subset of the input dimension-value pairs [6] are "activated" to

[4] Both condition chunks and conclusion chunks are linked to both the input side and the output side of the bottom level, as will be discussed more later.

[5] Similar to the case of action rules in the ARS, we may divide up associative rules in the GKS into multiple groups with each corresponding to one of the networks in the bottom level.

[6] If the chunk has multiple allowable values in one dimension, only one of them needs

the full strength 1 when the input is received. (2) An action carried out by the ACS may set the strength of a chunk to any (arbitrary) value between 0 and 1, inclusive. (3) An associative rule may "activate" a chunk. (4) At the end of associative mapping in the bottom level of the NACS, the result may go bottom-up to "activate" a chunk. (5) The result of similarity-based reasoning is the "activation" of a chunk to a certain strength level (when similarity-based reasoning is enabled). And so on (the details of these operations will be explained later).

When combining different sources of activation, the strength of a chunk is determined by

$$S_k^c = \max_x S_k^{c,x} \tag{3.1}$$

where x denotes a particular source, and *max* ranges over all possible sources: associative rules, externally given input, ACS (strength setting) actions, and other possible sources (including bottom-up activation from the bottom level). The reason why the *max* function is used here is that it provides a simple, intuitively appealing, and uniform way of combining all sources within the NACS.

Given the possibility of bi-directional connections, care must be taken to avoid a positive "feedback loop" that leads to saturation of strengths of all the chunks involved. [7]

Note that this structure is essentially a simplified form of a semantic network (e.g., Quillian 1968). Notably, there is no deep embedding, as in the case of, for example, one value of one dimension containing a pointer to another chunk that describes that dimension/value. In the GKS, there is only one flat structure. This structure should be sufficient for our purposes, because each value can, although not always, be described by another (separate) chunk. Therefore, although there is no direct embedding, it can be emulated. The result is a simple and uniform representation.

to be in the subset.

[7] The possibility of a positive feedback loop is lessened, due to the fact that if a strength is spread to other chunks from one chunk, then that strength may be attenuated (through the requirement of $\sum_i W_i \leq 1$). The strength may come back to the original chunk (because of a bi-directional link or an indirect feedback loop), but it may come back attenuated.

3.1.1.4 Base-Level Activation

Each chunk has a base-level activation (BLA), which is similar to the base-level activation of action chunks in the ACS, determined by the same equation, although parameters may be differently specified.

Each associative rule has a base-level activation as well, similar to what was described earlier in relation to action rules in the ACS, determined by the same equation, although parameters may be differently specified.

There are of course a number of options as before. We will not repeat the details here (see the chapter on the ACS). Base-level activation can be used for determining response times when NACS entities (chunks and associative rules) are invoked, the same way as in the ACS.

3.1.2 The Bottom Level

At the bottom level, on the other hand, "associative memory" networks (or AMNs for short) encode non-action-centered implicit knowledge.

Specifically, among other possibilities (Rumelhart et al 1986), a backpropagation network (or one of its many variants) can be used for implementing this memory. Associations are formed by mapping an input to an output. An input or an output is in the form of a set of dimension-value pairs. The regular backpropagation learning algorithm can be used to establish associations between the input and the output (between a set of dimension-value pairs representing the input and a corresponding set of dimension-value pairs representing the output).

In CLARION, there are various possibilities of training such a network for the purpose of capturing implicit associations. The first possibility is that sets of the dimension-value pairs (that are observable to the AMNs) may be presented to a network in an auto-associative way—both as the input and as the desired output. Through repeated training trials, the network may be able to develop an internal representation of these sets of dimension-value pairs (in the hidden layer of the network) and thereby extract central tendencies of these input. After training, the network may be used to produce (predict) a complete set of dimension-value pairs based on partial input information. For example, these sets may be various states encountered during the action decision making of the agent. In that case, after training, given certain partial information about a particular state, the

network will output (predict) the most likely state based on that partial information.

Another possibility is to learn to hetero-associate a pattern with another. In this case, a set of dimension-value pairs is presented as the input, and another (different) set of dimension-value pairs (which is to be associated with the former) is presented as the desired output. Through repeated training trials, the network may be able to output the desired pattern when a corresponding input pattern is presented. This is useful for, for example, establishing connections between facts (such as linking climate information of a region to its agricultural products, or linking a person to his profession, and so on).

These different ways of using the AMNs may be implemented as separate networks of the AMNs, if they are needed. They may even be mixed when circumstances permit. (These different operational modes may be decided by the ACS, which often controls the operation of the NACS. See the section on coordination later.)

3.1.3 Integrating the Two Levels

If a chunk in the GKS is "activated" by both a source within the GKS and the result of the AMNs, we need to set its final, combined strength based on integrating the two sources.

When the result from the AMNs (from the output side of the AMNs) is sent bottom-up to the GKS, it "activates" all chunks compatible with it.

In case a full match is required (i.e., no similarity-based reasoning is allowed), only dimensional values in the result of the AMNs that have a full activation (i.e., an activation of 1) will be considered. A chunk at the top level is "activated", if its specified values along the specified dimensions constitute a *subset* of the fully activated dimension-value pairs in the result of the AMNs [8] The "activated" chunk will have a strength level of 1.

In case similarity-based reasoning is enabled, partial match (and partial activation) will be considered, and a weighted-sum is used to determined the strength of an "activated" chunk at the top level. See the later section on similarity-based reasoning for details.

[8] If a chunk has multiple allowable values in one dimension, only one of them needs to be in the subset.

When we calculate the overall strength of a chunk, combining both the AMNs result (through bottom-up activation) with the result from within the GKS (which is either externally given or resulting from GKS inference), a *max* function is used (as mentioned before):

$$S_i^c = \max(S_i^{c,GKS}, S_i^{c,AMN}) \tag{3.2}$$

where $S_i^{c,AMN}$ is the bottom-up strength of chunk i, $S_i^{c,GKS}$ is the strength of chunk i from within the GKS (which is the maximum of all top-level sources, including externally given strengths and strengths resulting from inference within the GKS), and S_i^c is the final, combined strength of chunk i. [9]

Often, when top-level chunks are "activated" but not their corresponding dimensional values at the bottom level (e.g., when a chunk is inferred at the top level but not at the bottom level), a top-down activation process may be needed to activate corresponding dimensional values at the bottom level (usually on the input side of the AMNs). Generally speaking, the activation of a feature (dimensional value) node at the bottom level is set to the strength level of its corresponding chunk. However, in the top-down activation process, when a feature (i.e., a dimensional value) receives activations from several chunks, its strength is set to the maximum of these chunks. If one of the dimensions of a chunk has multiple allowable values (i.e., multiple disjuncts), the strength is evenly split among these allowable values of that dimension.

3.2 Reasoning

The information in associative rules may be used in a variety of ways in reasoning.

Reasoning starts with the input to the NACS. The input to the NACS is normally determined by ACS actions (but see Appendix for other possibilities). The input to the NACS may specify a set of chunks (that exist in the top level of the NACS), or alternatively, specify a set of dimension-value pairs. One may view the input as to the GKS (the top level) if

[9] In case that altering the relative scales of $S_i^{c,AMN}$ and $S_i^{c,GKS}$ is needed, the ACS may obtain the two results separately in two separate steps, and store them in the WM. Then, the ACS may combine them through a weighted sum under the direct control of ACS actions.

it specifies chunks, or as to the AMNs (the bottom level) if it specifies dimensional values. In either cases, the activation levels of input (either in the form of a set of chunks or in the form of a set of dimension-value pairs) are always 1 (the full activation).

However, if the input specifies a set of dimension-value pairs (with an activation of 1), all the chunks whose dimensional value pairs constitute a subset of the input) are also "activated" at the top level (via *bottom-up activation* as described before). If the input specifies a set of chunks, the corresponding dimensional values at the bottom level are also activated (via *top-down activation* as described before). So, the net effect of these two forms of input is the same: the full activation of a set of dimension-value pairs and all their compatible chunks.

Furthermore, when similarity-based reasoning is enabled, the bottom-up activation process becomes more complex, in order to implement similarity-based reasoning. The details of similarity-based reasoning will be discussed in a later section.

During a round of NACS reasoning, associative mapping at the bottom level (in the AMNs) starts with the activated feature nodes (dimensional values) on the input side of the AMNs. This enables a round of AMN associative mapping, which activates a set of dimensional values on the output side of the AMNs.

Concurrent with the round of AMN associative mapping, at the top level, an iteration of GKS inference goes on, starting from all the currently "activated" chunks in the GKS (including all previously given, previously inferred, or previously bottom-up activated chunks). All applicable associative rules in the GKS fire simultaneously: There is no competition/selection among associative rules. All the applicable rules are applied, and all possible conclusions are derived. New chunks are inferred in the GKS as a result.

To integrate the outcomes of the GKS and the AMNs, the results from the output side of the AMNs go bottom-up and "activate" chunks in the GKS (via the *bottom-up activation* process as described before). Following the bottom-up activation, *top-down activation* ensues that activates feature nodes (dimensional value nodes) on the input side of the AMNs (details of the top-down activation process has been described before).

After this round of NACS reasoning (with both the GKS and the

AMNs) and integration of outcomes, there may be a new round of NACS reasoning. This process goes on for a pre-specified number of iterations, or until no new conclusion (chunk) can be drawn.

3.2.1 Reasoning Methods

The following reasoning methods are included in the top level of the NACS:

- Forward chaining reasoning: for drawing all possible conclusions (chunks) in a forward direction—from known conditions to new conclusions. This is the primary way of using associative rules (as well as action rules). The process of forward chaining may stop after one pass (or a few passes), or it may be iterated, until no new conclusion (chunk) can be drawn.
- Similarity-based forward chaining reasoning: for drawing all possible conclusions, using rules as well as similarity-based inference (Sun 1994, 1995). This type of mixed rule-based and similarity-based reasoning will be explained in full later. This process may stop after one pass (or a few passes), or it may be iterated, until no new conclusion (chunk) can be drawn.

For these reasoning methods, there is a threshold ($threshold_r$) that determines whether a conclusion is acceptable or not: If it is below the threshold, it will be discarded (that is, as if its strength is zero).

3.2.2 Rule-Based Reasoning

First, let us look into details of rule-based reasoning.

In terms of various measures associated with rule-based reasoning, such as base-level activation, rule support, and rule utility, a few options are available. In the NACS, the default choice is to set rule utility off, but use base-level activation, as well as rule support. Rule support (propagating from a rule condition to its conclusion) may be used for selecting chunks, and base-level activation for determining response times (possibly along with chunk base-level activation).

In case a full match of conditions is required (i.e., similarity-based reasoning is disabled), rule support for associative rules in the GKS may be simply calculated as follows: If all the chunks in the condition of a rule have a strength level of 1, then the support for the rule is 1.

However, when similarity-based reasoning is enabled, the chunks in the condition of a rule may have a strength level below 1. In that case, the rule support should be proportionally lower:

$$S_j^a = \sum_i S_i^c * W_i^a \tag{3.3}$$

where j indicates the jth rule at the top level, S_j^a is the support for associative rule j, S_i^c is the strength of the ith chunk in the condition of the rule, i ranges over all the chunks in the condition of rule j, W_i^a is the weight of the ith chunk in the condition of rule j (which, by default, is $W_i^a = 1/n$, where n is the number of chunks in the condition of the rule).

The conclusion chunk of associative rules has a strength level that is determined by the support from all the relevant rules:

$$S_{c_k}^{c,a} = \max_{j:\text{all associative rules leading to } c_k} S_j^a \tag{3.4}$$

where $S_{c_k}^{c,a}$ is the strength of chunk c_k (resulting from associative rules), and j ranges over all the associative rules pointing to c_k.

3.2.3 Similarity-Based Reasoning

Let us turn to an abstract formulation of similarity-based reasoning first. Similarity-based reasoning (including inheritance reasoning) can be carried out using chunks (see Sun 1995). An agent may compare a known (given or inferred) chunk with another chunk. If the similarity between them is sufficient high, then the latter chunk is inferred. The strength of a chunk c_j as a result of such similarity-based reasoning is:

$$S_{c_j}^{c,s} = \max_i (S_{c_i \sim c_j} \times S_{c_i}^c) \tag{3.5}$$

where $S_{c_j}^{c,s}$ denotes the strength of chunk j resulting from similarity, $S_{c_i \sim c_j}$ measures the similarity from c_i to c_j, $S_{c_i \sim c_j} \times S_{c_i}^c$ measures the support to c_j from the similarity with c_i, i ranges over all chunks.

The default similarity measure in CLARION (as in Sun 1995) is:

$$S_{c_i \sim c_j} = \frac{N_{c_i \cap c_j}}{f(N_{c_j})} \tag{3.6}$$

where $S_{c_i \sim c_j}$ denotes the similarity from c_i to c_j. In the formula, $N_{c_i \cap c_j}$ is the weighted sum of all the identically valued dimensions of c_i

and c_j (among all the specified dimensions of c_j [10]), that is, $N_{c_i \cap c_j} = \sum_{k \in c_j \cap c_i} V_k^{c_j} \times D_k$, where D_k in this case represents dimension k in chunk c_j and has a value of 1. The weights $V_k^{c_j}$ in the above weighted sum are specified with respect to c_j (the target of similarity, not the source of it). The default is that these weights are the same and equal to 1. On the other hand, N_{c_j} is the weighted sum of the specified dimensions of c_j; that is, $N_{c_j} = \sum_{k \in c_j} V_k^{c_j} \times D_k$, where $D_k = 1$ and $V_k^{c_j} = 1$ (by default). f is a superlinear, but close to linear, function, such as $f(x) = x^{1.10}$. Thus, the similarity measure is limited to $[0, 1)$. [11]

Going beyond the above abstract formulation, we need to look into the implementation of similarity-based reasoning. We note that similarity can be automatically computed whenever reasoning involves multiple chunks that are similar to one another. Therefore, there is no need for dedicated representation of similarity between any two chunks (in contrast to other cognitive architectures, such as ACT-R). When similarity-based reasoning is enabled, in order to (approximately) implement the similarity measure specified above, we need a more complex process for bottom-up activation. In this case, the weights for bottom-up activation of a chunk are as follows:

$$W_k^{c_j} = \frac{V_k^{c_j}}{f(\sum_{k \in c_j} V_k^{c_j} \times D_k)} \tag{3.9}$$

where $W_k^{c_j}$ is the weight of the kth feature (at the bottom level) of chunk j, and $V_k^{c_j}$ and D_k are as specified before. In turn, the bottom-up activation $S_{c_j}^{c,s}$ is determined as follows:

$$S_{c_j}^{c,s} = \sum_{k \in c_j} W_k^{c_j} \times A_k = \sum_{k \in c_j} \frac{V_k^{c_j} \times A_k}{f(\sum_{k \in c_j} V_k^{c_j} \times D_k)} \tag{3.10}$$

[10] Here, the term "specified dimension" denotes those dimensions in a chunk that have their values specified as part of the specification of the chunk.

[11] Note that there are many other possible similarity measures. See, for example, Tversky (1977) or Sun (1995) for some of them. Two other possibilities (among many others) are:

$$S_{c_i \sim c_j} = \frac{N_{c_i \cap c_j}}{f(N_{c_i})} \tag{3.7}$$

and

$$S_{c_i \sim c_j} = \frac{N_{c_i \cap c_j}^2}{f(N_{c_j} \times N_{c_i})} \tag{3.8}$$

Compared with these options, the default option described above is preferable (Sun 1995).

where A_k is the activation of feature k (at the bottom level) of chunk c_j (as a result of other chunks that are similar to c_j). [12]

Similarity-based reasoning produces inexact output, as determined by similarity measures between the two chunks involved. We apply the similarity threshold to the conclusion, so that if a conclusion reached has a strength level below $threshold_{sim}$, we discard the conclusion (as if its strength is 0).

Similarity can be utilized whenever reasoning can benefit from similarity, that is, whenever new conclusions may be reached from knowledge associated with similar chunks. Of course, one may choose to use, or not to use, similarity-based reasoning.

Similarity-based and rule-based reasoning can be inter-mixed and as a result, complex patterns of reasoning can emerge (as studied in Sun 1995). When similarity-based reasoning is employed, extending the strength calculation of rule-based reasoning explained earlier, we have:

$$S^c_{c_i} = \max(S^{c,a}_{c_i}, S^{c,s}_{c_i}) \tag{3.11}$$

$$= \max(\max_{j:\text{all associative rules leading to } c_i} S^a_j,$$

$$\max_{j:\text{all chunks similar to } c_i} (S_{c_j \sim c_i} \times S^c_{c_j})) \tag{3.12}$$

where $S_{c_j \sim c_i}$ is the similarity measure. [13] As mentioned before, there may be other sources of chunk activation. When they are present, they should be incorporated into the outer *max* function.

In forward-chaining reasoning with both rule-based and similarity-based processes, the afore-described mixed computation of chunk strengths can be repeatedly applied. The conclusion from one step of reasoning can be used as a starting point of the next step. The iterative process of mixed rule-based and similarity-based reasoning allows all of the following possible sequences (Sun 1995): rule-based reasoning followed by rule-based reasoning, rule-based reasoning followed by similarity-based reasoning,

[12] The minor difference from the abstract specification (with the default similarity measure) lies in the use of A_k (as opposed to D_k times $S^c_{c_i}$ as in the abstract specification).

[13] In case that altering the relative scales of $S^{c,a}_{c_i}$ and $S^{c,s}_{c_i}$ is needed, we may instead alter the similarity measure by a scaling constant, or the rule weights by a scaling constant.

similarity-based reasoning followed by rule-based reasoning, similarity-based reasoning followed by similarity-based reasoning, ans so on. As shown by Sun (1995), these different sequences are important in terms of capturing essential patterns of human everyday (mundane, commonsense) reasoning. Some examples will be presented next.

3.2.4 Examples of Similarity-Based Reasoning

Let us look into a few examples of mixed rule-based and similarity-based reasoning.

One chunk may be the superset of another (i.e., the former contains a superset of dimension-value pairs of the latter), in which case, the former describes a subset of the objects that the latter describes. Conversely, if a chunk is a subset of another chunk, then it describes the superset of objects that the other chunk describes. Based on this reverse containment relationship, much reasoning can be carried out. Inheritance reasoning, for one thing, can be carried out based on the reverse containment relationship.

For one instance, using the default similarity measure above, if chunk i is a superset of chunk j (that is, i describes a subset of what j describes) and if chunk j is known to have an associative rule to chunk k (say, with full strength 1), we infer that chunk i has an association to chunk k as well. Since i is a superset of j, we have

$$S_{i\sim j} = \frac{N_{i\cap j}}{f(N_j)} = \frac{N_j}{f(N_j)} \approx 1 \tag{3.13}$$

where $S_{i\sim j}$ is less than, but close to, 1. Therefore, given i (say, with full strength 1), from $i \sim j$ and $j \longrightarrow k$, we infer k. Furthermore, we infer k with almost full strength 1 (i.e., $S^c_k \approx 1$), if both the known rule $j \longrightarrow k$ and the known chunk i have the full strength (i.e., $S^a_{j\longrightarrow k} = 1$ and $S^c_i = 1$). For instance, i may stand for *sparrow*, j for *bird* (in general), and k for *fly*. From $bird \longrightarrow fly$ and $sparrow \sim bird$, we infer that sparrow flies. This case is an example of "inheritance" (or bottom-up inheritance, i.e., superset-to-subset inheritance; Sun 1995).

Another instance, a case of "cancellation of inheritance" (Sun 1995) is as follows. Suppose that chunk i is a superset of chunk j and that chunk j is known to have an associative rule to chunk k (with full strength 1). Furthermore, suppose that i is known to have an associative rule to $\neg k$

(with full strength 1). For instance, i may stand for *ostrich*, j for *bird* (in general), and k for fly. As before, we might infer that chunk i has an association to chunk k. That is, given i (with full strength 1), we have $S_{i\sim j} = \frac{N_{i\cap j}}{f(N_j)} = \frac{N_j}{f(N_j)} \approx 1 (< 1)$, and thus from $i \sim j$ and $j \longrightarrow k$, we might infer k, with a strength close to, but less than, 1. However, k is canceled in this case by the known associative rule $i \longrightarrow \neg k$. Since this known rule has the full strength 1, its conclusion ($\neg k$) also has the full strength 1 and therefore wins the competition with the other conclusion (k) reached through similarity. That is, in the above example, from $bird \longrightarrow fly$, $ostrich \sim bird$, and $ostrich \longrightarrow \neg fly$, we infer in the end that ostriches do not fly.

On the other hand, if chunk i is a subset of chunk j (that is, i describes a superset of what j describes) and if chunk j has an associative rule to chunk k (e.g., with full strength 1), we can infer, to some extent, that chunk i has an association to chunk k. Since i is a subset of j, we have

$$S_{i\sim j} = \frac{N_{i\cap j}}{f(N_j)} = \frac{N_i}{f(N_j)} < 1 \tag{3.14}$$

Therefore, given i (with full strength), from $i \sim j$ and $j \longrightarrow k$, we infer k, with a partial strength. We may term this case "top-down inheritance" (i.e., subset-to-superset inheritance), as different from regular, bottom-up inheritance (Sun 1995). This type of "inheritance" is of course less reliable. For instance, i may stand for *animal* in general, j for *cat*, and k for *jump*. From $cat \longrightarrow jump$ and $animal \sim cat$, we infer that animals jump.

Moreover, in the above case, if we know that $i \longrightarrow \neg k$ (with full strength 1), then the two conclusions, k and $\neg k$, compete with each other, and $\neg k$ will win due to its full strength. For instance, i may stand for *animal* (in general), j for *bird*, and k for fly. The conclusion is thus "animals in general do not fly". We may describe this case as "cancellation of top-down inheritance" (Sun 1995).

For analysis of other cases and how they can also be handled by this approach, see Sun (1994, 1995). Note that these reasoning patterns are performed with a combination of the top level and the bottom level.

3.2.5 Other Reasoning Methods *

There are, of course, other reasoning methods. They are not implemented as part of the architecture, but they can be implemented within the

architecture using a combination of the ACS and the NACS.

For instance, in contrast to forward chaining reasoning discussed so far, backward chaining reasoning may also be used. It is useful for verifying a particular conclusion using a set of rules. A specialized set of action rules may be specified in the ACS to apply this reasoning method. This special set of action rules may be in the form of "fixed rules" in the ACS. Once invoked, they verify a specific hypothesis in a backward direction. The set of fixed rules uses the goal structure and the working memory to control the progress of backward chaining: The goal structure stores the original and intermediate hypotheses to be verified, and the working memory keeps track of the process of verification.

This method can be used for reasoning within the GKS, or it can be used for verifying action decisions generated by the IDNs, or by a rule set (IRL or RER) in the ARS. An interesting possibility emerged is that this method allows an agent to rely primarily on the bottom level (of the ACS) for action decision making, and use the top level as a verifier for gate-keeping purposes (Libet 1985).

Another possibility is proof by contradiction. A specialized set of action rules (FRs) may be specified in the ACS to apply this type of reasoning. The set of FRs may use the goal structure and the working memory to control the reasoning process.

Yet another possibility is counterfactual reasoning. A specialized set of action rules (FRs in the ACS) may be specified to apply this type of reasoning. They may again use the goal structure and the working memory to control the reasoning process.

3.3 Learning

When it comes to learning non-action-centered knowledge, similar to learning action-centered knowledge, there is also the distinction between top-down learning and bottom-up learning. Of course, learning, first and foremost, occurs within each level separately.

3.3.1 Learning Explicit Knowledge

3.3.1.1 Externally Given Explicit Knowledge

One way for an agent to learn explicit, non-action-centered knowledge, is that the agent takes externally given explicit knowledge and encodes it in the GKS. Explicit knowledge can be given, from external sources, in the form of chunks and associative rules connecting chunks. [14] Then, it can be encoded, in a straightforward way, in the GKS. For example, a piece of externally given knowledge, "a warm, moist region may be a rice-producing region", may be represented by a link between two chunks: *(chunk-54: (temp, warm) (rainfall, medium–heavy)) $\longrightarrow$ (chunk-79: (agri-product, rice))*

Usually, the encoding of externally given knowledge is under the control of the ACS. Once the ACS receives or constructs a piece of knowledge (from its input state and available action-centered knowledge), in the form of a chunk or an associative rule, it sends the information on to the NACS (with a command for the NACS to encode the knowledge). The NACS encodes the knowledge as a chunk or an associative rule.

In the GKS, chunks are also formed in a variety of other ways from external sources, including the following: (1) The given value in each dimension of each state observed by the ACS is encoded as a chunk. (2) So is the chosen value in each dimension of each action recommendation from the ACS. (3) Each state experienced, as a whole, as observed by the ACS, is also encoded as a chunk. [15] (4) So is each action step experienced as a whole (which includes the state, the action in that state, the next state following the action, and the immediate reinforcement following the action).

Note that these above four types are experience-specific chunks, as they are created due to specific experiences and they reflect such experiences. These chunks are encoded as they are experienced. [16]

[14] In general, externally given knowledge may be presented in forms that can be transformed into associative rules and chunks. Here, we will not worry about the transformation of externally given knowledge into chunks and associative rules.

[15] The dimensions and values seen by the NACS are normally the same as, or a subset of, what the ACS sees. In other words, what is filtered out by the ACS is also filtered out by the NACS. There may be the same amount of, or more, filtering in the NACS than in the ACS.

[16] However, they are different from episodic chunks (discussed later): They do not have

The encoding of these above five types of externally given explicit knowledge in the GKS is subject to density parameters and encoding probability parameters. An encoding probability parameter p_a may be specified, which determines how likely an encoding of an associative rule will be successful. Likewise, an encoding probability parameter p_c may be specified, which determines how likely an encoding of a chunk will be successful. As explained before, density parameters (d_a and d_c) may also be specified. [17]

Note that, beside encoding externally given knowledge, other chunks are formed when they are (1) extracted from the bottom level of the NACS, or (2) extracted as a result of RER or IRL rule learning in the ACS (i.e., extracted from the IDNs). Discussions of these types of learning are next.

3.3.1.2 Extraction of Explicit Knowledge

Extraction of Explicit Knowledge from the IDNs in the ACS. First, here is a quick review of extraction of explicit knowledge in the ACS. In the ACS, a chunk may be learned as a result of extracting an action rule: When the condition of an action rule is established, a localist (unitary) encoding of that condition is also established, and thus a new chunk is formed.

Specifically, while learning action rules in the ARS (using RER or IRL), an agent forms chunks: A separate node (a chunk node) is set up to represent the condition of a rule, when a rule is extracted, which connects to its features (values of various dimensions) represented in the bottom level (in the IDNs). A chunk node is linked to its corresponding dimensional values in the bottom level using, for example, AND/OR functions (or more complex activation functions; see the chapter on the ACS). Such chunks, in general, are in the form of: $(dim_{i_1}, val_{i_1})(dim_{i_2}, val_{i_1})$ ..... (dim_{i_n}, val_{i_n}). An arbitrary label (i.e., a *chunk id*) may be given to each of them.

time stamps associated with them and thus they are not episodic in nature.

[17] As explained before, a density parameter d_a can be specified to determine the minimum frequency of the invocation (encoding, re-encoding, extraction, re-extraction, or application) of an associative rule in order to keep that associative rule. Similarly, a density parameter d_c can be specified to determine the minimum frequency of the invocation (encoding, re-encoding, extraction, re-extraction, or activation) of a chunk in order to keep the chunk.

For example, a localist chunk node may be set up in the ARS to represent the following chunk of features: *chunk-564: (temp, warm) (rainfall, heavy)*. The chunk may be extracted in the process of extracting an action rule in the ARS: *(temp, warm)(rainfall, heavy)* $\longrightarrow$ *(action, stay-under–trees)*. In turn, the formed chunk is also set up and used in the GKS.

This is basically a *prototype model* of concepts (Smith and Medin 1981, Rosch 1978). Localist chunk nodes serve as the identification and the encoding of a set of correlated features, in a bottom-up direction. Chunk nodes at the top level also serve to trigger dispersed features at the bottom level, in a top-down direction, once a relevant concept is brought into attention ("activated").

Concepts formed in this way are context-dependent and task-oriented, because they are formed in relation to a task at hand and for the express purpose of exploiting environmental regularities encountered in dealing with the task. Thus, CLARION emphasizes the functional role of concepts and the importance of function in forming concepts: A concept is formed as part of an action rule, which is learned to accomplish a task in a particular environment. As a result, acquired concepts are functional. Task contexts help an agent to determine which set of features in the environment need to be attended to. [18]

Extraction of Explicit Knowledge from the AMNs. Chunks, acquired from external sources, through action rule extraction in the ARS, or through other means, are used to encode, explicitly, knowledge extracted from the AMNs, in the form of associative rules between chunks.

Associative rules are formed (i.e., extracted), when associative mapping is performed in the AMNs—associative links are established between chunks describing the given cue for the mapping and the chunks describing the outcome from the mapping.

Specifically, (1) a cue used in associative mapping by an AMN is encoded as one chunk in the GKS and "activated", upon the presentation of the cue, to a strength level corresponding to the cue (to be defined later), if that strength is above $threshold_{ce}$. [19] Next, (2) through bottom-up activation, the result from the AMN (the dimension-value pairs) is

[18] More description of rule learning may be found in the chapter on the ACS. Sun (2002) provided analyses of concepts formed by agents in learning specific tasks.

[19] If there is no chunk corresponding to the cue at the time when it is presented, with

examined to discern all possible chunks compatible with the result of the AMN (with or without similarity-based reasoning enabled). Those chunks are "activated" to a strength level corresponding to the result from the AMN (to be defined later), if that strength is above $threshold_{ce}$. Furthermore, (3) a separate chunk is set up that corresponds to the result from the ANN as a whole, if such a chunk is not already there. The chunk is "activated" to a strength level corresponding to the result from the AMN (to be defined later), if that strength is above $threshold_{ce}$. In addition, (4) a separate chunk is also set up that corresponds to the cue and the result together as a whole, if such a chunk is not already there. [20] The chunk is "activated" to a strength level corresponding to the result from the AMN (to be defined later), if that strength is above $threshold_{ce}$.

In setting the strength level of an extracted chunk based on the activations of its features, the same bottom-up activation process as discussed before is used. As discussed before, it can be done either with or without enabling similarity-based reasoning.

Similarly, when a result chunk from an AMN is strong enough (i.e., $> threshold_{ae}$, the associative rule extraction threshold), an associative rule corresponding to the associative mapping from the AMN is set up (if it is not there already). That is, an associative link is established that connects the chunk representing the cue with each chunk compatible with the result of the AMN (of a type specified by 2 or 3 above, with or without similarity-based reasoning enabled), if the link is not already there and if the associative rule extraction threshold ($threshold_{ae}$) is exceeded.

There are some parameters concerning explicit knowledge extraction. The extraction threshold for chunks, $threshold_{ce}$, specifies the minimum activation level (concerning the resulting chunk) for chunk extraction to be considered. There is also a probability parameter p_{ce} that determines how likely the extraction of a chunk will be successful (provided that $threshold_{ce}$ is reached). The extraction threshold for associative rules, $threshold_{ae}$, specifies the minimum activation level for associative rule extraction to be

the same set of dimension-value pairs, we set up a chunk in the GKS that contains exactly the same dimensional values as in the cue, and "activate" it to the strength level corresponding to the cue.

[20] The chunk is formed by merging the dimension-value pairs of the cue with those of the result. When there is an overlapping dimension where the cue and the result specify different values, an OR operation may be used to combine them.

considered (where the minimum activation level concerns the strength of the conclusion chunk resulting from associative mapping). There is also a probability parameter p_{ae} that determines how likely the extraction of an associative rule will be successful (provided that threshold $threshold_{ae}$ is reached). [21] [22]

Similar to concept learning in the ACS, concepts (chunks) and associative rules formed in this way are also task-oriented to some extent, because they are formed in relation to a task at hand.

3.3.2 Learning Implicit Knowledge

Assimilating explicit knowledge is accomplished through training an AMN using associations stored in the episodic memory (see the section on the EM later).

At each step, with a certain probability (pre-specified), a (randomly selected) subset of association items from the EM is presented to an AMN. The set may be presented to the AMN one item at a time, in multiple iterations (the number of which is determined by a parameter). As the input and the output respectively, associated chunks are presented to the AMN: $(dim_{i_1}, val_{i_1})(dim_{i_2}, val_{i_2})$ ..... (dim_{i_n}, val_{i_n}) $\Longrightarrow$ $(dim_{j_1}, val_{j_1})(dim_{j_2}, val_{j_2})$ ..... (dim_{j_m}, val_{j_m}), where values of some (not necessarily all) dimensions are specified in each chunk, while values of other (unspecified) dimensions are left blank.

In the AMN, because multiple associations are formed gradually and simultaneously, they are interleaved in the memory. Thus, common patterns and clusters may emerge, which enables some generalization (Rumelhart et al 1986).

An AMN may be trained under the direct control of the ACS: The ACS may specifically provide an association to train a particular AMN.

[21] As discussed earlier, a density parameter d_a can be specified to determine how much invocation (application, extraction, re-extraction, encoding, or re-encoding) of a particular associative rule is necessary in order to keep it. Similarly, a density parameter d_c can be specified, which determines how much invocation (activation, extraction, re-extraction, encoding, or re-encoding) is necessary in order to keep a chunk.

[22] In addition, each associative rule or each chunk in the GKS, once "activated" to a strength level above $threshold_{ae}$ or $threshold_{ce}$ respectively, can also spawn a new corresponding *episodic* chunk, that is, a corresponding chunk with exactly the same makeup but with a time stamp added. They constitute part of the episodic memory.

When such an association is given by the ACS, the AMN is trained right away (for one iteration). The given association is also stored into the EM, to be used later for further assimilation.

The associations stored in the EM include all the associative rules applied, all the associations given externally, all the associations representing the mappings from the input to the NACS and each of the resulting (inferred) chunks (based on all forms of reasoning), and so on. Each association that is alive in the EM might be selected for training an AMN (so that the AMN may assimilate the knowledge). One may specify the type of association from the EM to be used in training an AMN, for example, all associative rules applied, all associations given by the ACS, all associations given by the ACS for this particular AMN, and so on.

After sufficient training, we may use the resulting associations from an AMN. To apply learned associations, a set of dimensions and their corresponding values are presented to an AMN as input: $(dim_{i_1}, val_{i_1})(dim_{i_2}, val_{i_2})$ ..... (dim_{i_n}, val_{i_n}). Then, we read off the resulting association. The result consists of activated dimension-value pairs. [23]

3.4 Coordination of the NACS and the ACS

3.4.1 Action-Directed Reasoning

Amongst many possibilities of coordinating the ACS and the NACS, one distinct possibility is that the NACS may be under the complete control of the ACS. An action command by the ACS may specify performing reasoning in the NACS and obtaining reasoning results from it before taking another action. (Incidentally, this is also the way ACT-R is structured, although it does not have the two separate subsystems. See Anderson 1993.)

The ACS dictates the type of reasoning to be performed by the NACS. Reasoning in the GKS may be carried out using a variety of reasoning methods. Usually, forward chaining or similarity-based forward chaining are used. These choices can be dictated by action commands from the ACS, but they may also be pre-determined by the MCS (see the chapter on the MCS), when there is no direct command from the ACS.

[23] On top of that, we may *explicate* the result of implicit mapping, through bottom-up activation, as described earlier.

Action commands from the ACS may dictate other reasoning parameters for the NACS as well, including number of reasoning iterations, number of AMN passes for each iteration, and so on.

ACS action commands may decide how outcomes from the NACS are to be used. As dictated by an action of the ACS, "filtered" results of the NACS may be retrieved, either (1) in the form of all the "activated" chunks, or (2) in the form of a single "activated" chunk. If the ACS dictates that only one chunk be retrieved, to select one chunk from the NACS, competition (through a Boltzmann distribution) occurs among all the chunks "activated" (with a strength level $> threshold_r$). The winner is sent back to the ACS (through a retrieval buffer). Another possibility is that the ACS dictates that all the "activated" chunks should be retrieved. In that case, all the "activated" chunks (with a strength level $> threshold_r$) are sent back to the ACS (through a retrieval buffer). In either of the above two cases, we may focus on only a certain type of results, for example, only on chunks that are not contained in the original input to the NACS at the beginning of reasoning. These choices are also determined by action commands from the ACS. When such results are sent back to the ACS, they are first stored into the retrieval buffer.

Note that, beside using action-directed reasoning, there may be other possibilities in coordinating the two subsystems. See Appendix for a discussion of some of these other possibilities. Compared with these other possibilities, action-directed reasoning appears to be a reasonable choice for CLARION, useful for a wide variety of simulations (see the simulations later).

3.4.2 Examples of Action-Directed Reasoning

To further explain (and motivate) the use of the non-action-centered subsystem, let us look into an example of reasoning in the service of action decision making. Although this case seems different from what was described above, there is actually no need for a different process; the same process as described above works.

Suppose that an agent is to learn to fly an aircraft. Suppose that, to begin with, the agent is given some rudimentary domain knowledge of flying an aircraft, from, for example, a flight instructor, in the form of: "if

you see x, then you do y", where x is a rough description of a situation (it is rough because it leaves many dimensions unspecified), and y is a set of actions, a rough description of what to do in a given situation x.

Such a rule is accepted and wired up in the top level of the action-centered subsystem. Then, in the ACS, much practice is carried out through interacting with the aircraft. Through practice, assimilation of the top-level knowledge occurs in the ACS. At the same time, implicit learning occurs and implicit action routines are formed in the bottom level of the ACS (incorporating the assimilated explicit knowledge and implicit knowledge gained from interacting with the world). Through practice, in addition to forming implicit action routines in the bottom level, at the top level, rule learning, and associated concept formation, occurs through bottom-up processes. Without loss of generality, suppose that the following rules are extracted: "if see x_1, do y_1" and "if see x_2, do y_2". They can be considered the refinements of the externally given rule mentioned earlier.

At the same time, in the NACS, association formation and knowledge extraction occur. Suppose that some chunks (concepts), x_3, x_4, and x_5, are externally given (e.g., by an instructor, when these chunks represent important situations). Suppose also that an associative rule among two chunks is given: "if x_3, then x_1". Suppose that x_5 appears similar to x_2, judging from their respective dimensional values. So we have "x_5 is similar to x_2" (as indicated by the representation of chunks x_5 and x_2).

Then, the agent combines knowledge from the two subsystems in action decision making. For example, we may examine the following two scenarios. When x_3 is encountered, the agent chooses action y_1, which is derived because "if x_3, then x_1" and "if see x_1, then do y_1". To accomplish this, the agent selects an action in the ACS that dictates reasoning using the NACS. Based on the existing knowledge in the NACS, x_1 is derived and stored into the working memory, and then another around of decision making in the ACS based on x_1 leads to taking action y_1. On the other hand, when x_5 is encountered, the agent chooses action y_2, which is derived because "x_5 is similar to x_2" and "if see x_2, then do y_2". Again, the existing knowledge in the NACS is used to derive x_2, which in turn leads to action y_2 in the ACS.

Of course, in addition to these two indirect action decision making scenarios (as they involve the NACS), there are also directly available

decision rules, including "if see x_1, then do y_1" and "if see x_2, then do y_2". There are also IDNs that can recommend action decisions. These rules and IDNs can be applied in pertinent situations, when a proper situation (such as x_1 or x_2) appears as the sensory input.

On the other hand, there is also another possible scenario: An original, externally given rule "if see x, then do y" may have been eliminated due to its imperfections (through RER or IRL), or given low priority due to its low utility measure.

3.5 Episodic Memory and Abstract Episodic Memory *

3.5.1 Episodic Memory *

Episodic memory (or EM for short) is a special part of the GKS. It stores recent experiences in various forms. The EM, first and foremost, stores action-oriented experiences involving stimulus, response, and consequence, along with time stamps of when those occurred. In addition to action-oriented experiences, it also stores non-action-oriented experience, in the form of chunks and associative rules (from the GKS), along with time stamps.

There are the following types of episodic chunks: (1) Each state (at each step), as a whole, as observed by the ACS, is represented in the EM, along with a time stamp. (2) Each step performed by the ACS, as a whole (including the state, the action in that state, the next state following the action, and the immediate reinforcement following the action), is represented as an EM item, along with a time stamp. Note that these chunks are tied to specific prior experiences, along with a time dimension, and thus they are episodic in nature and constitute part of the episodic memory.

In addition, (1) each action rule, (2) each associative rule, (3) each action chunk, (4) each NACS chunk, (5) each association given by the ACS to the NACS, and (6) each association inferred by the NACS, once invoked, may spawn a new corresponding EM item. That is, once such an entity (a chunk or a rule) is "activated" to a certain strength level ($threshold_{ce}$ or $threshold_{ae}$ respectively), it leads immediately to the creation of a corresponding EM item with exactly the same makeup but with a time stamp added.

Information in the EM is recency-filtered. A base-level activation is associated with each item in the EM. The base-level activation mechanism of an EM item is the same as that of other chunks, except possibly its parameters. The BLA decays gradually. Eventually, when the BLA of an item falls below $threshold_{EM}$, the item is removed from the EM.

EM items are not subject to the chunk density parameter or the associative rule density parameter (because their durations in the NACS are determined by their BLAs as explained above). However, they are subject to the chunk encoding probability parameter or the associative rule encoding probability parameter, respectively (as has been described before).

Note that EM chunks are linked to a corresponding representation in the bottom level. The bottom-level representation of EM chunks are separate from other bottom-level representations. The time stamps in EM chunks may be viewed as a special (ordinal) dimension with many values.

Part of this memory may be used for helping learning, because it provides additional opportunities for practicing. Stored instances in the form of "state, action, new state, immediate reinforcement" (i.e., stimulus, response, and consequence) may be used for learning at both levels of the ACS, as if they were real experiences encountered. Therefore, exactly the same learning processes may be applied. The cognitive justification for this use of the EM is that memory consolidation is known to improve learning performance, and this use of the EM serves as a memory consolidation process (McClelland et al 1995).

In the current implementation of CLARION, one may specify the probability with which this part of the EM (which consists of state-action-new-state-reinforcement tuples) may be used for training the ACS at each step, and the number of such EM items to be (randomly) chosen for this purpose at each step, and the number of training iterations at each step.

The EM may also help the learning of the AMNs. See the section on learning for details.

3.5.2 Abstract Episodic Memory *

As opposed to the (regular) EM, which stores specific experiences (in the forms specified above), the abstract episodic memory (the AEM) *summarizes* information regarding past episodes experienced by the ACS,

so that summary information of past episodes becomes readily usable, without the need to examine details of each individual step. The summary information can be useful (1) in helping learning and (2) in extracting explicit knowledge from the IDNs (Sun and Sessions 2000). Judging from its form and content, this memory may be viewed as part of the AMNs (i.e., part of implicit, non-action-centered knowledge representation).

The AEM is constituted by an action frequency network (AFN) and a state frequency network (SFN). The AFN maps the state to the frequency distribution of actions. The SFN maps the state/action pair to the frequency distribution of succeeding states, as well as the frequency distribution of (immediate) reinforcement. States, different from the previous specification of encoding with dimension-value pairs, are coded in this case using localist (unitary) encoding in order for the two networks to output state transition frequencies: Each state is represented by one individual node (at the input or the output layer of a network). States are identified one by one, as these states are experienced, and coded localistically (in a unitary manner) by the two networks. [24] Actions are also coded using localist (unitary) encoding (for the same reason as that for localist state encoding): Each action is represented by one node (at the input or the output layer of a network). [25] Actions are identified one by one as they are experienced (i.e., performed by the agent). [26] Reinforcement is quantized into a certain number of intervals and each of these intervals is coded localistically (in a unitary manner). In this way, a frequency distribution of reinforcement can be output.

Both networks are backpropagation networks, but trained with an alternative algorithm: the probabilistic backpropagation learning algorithm, which is more suitable for learning frequency distributions. The training is based on episodic data from the (regular) EM. See Appendix for details.

The AEM, the same as the (regular) EM, may be used to help speed up the learning of the ACS. That is, this memory can be used to generate synthetic "experience" for training the ACS, through iteratively doing the following: randomly selecting a state, and then sampling the

[24] Thus, the maximum number of states must be specified a priori.
[25] Likewise, the maximum number of actions must be specified a priori.
[26] Each action here is a conjunction of a set of dimension-value pairs (covering relevant action dimensions).

action distribution, the succeeding state distribution, and the reinforcement distribution based on the selected state. The generated synthetic experience (a 4-tuple of state, action, new state, and immediate reinforcement) can then be used to train the ACS, as if it is real experience.

This memory can also be used in other, more complex ways. For example, this memory can be used to improve the Q-learning process. For details, see Appendix. It can also be used in *plan extraction* from the IDNs, as discussed in Sun and Sessions (2000). For details, see Sun and Sessions (2000).

entity	evidential support	selection	response time determin
associative rule	rule support (w/ or w/o SBR)	n/a	bla
chunk	strength	strength	bla

Figure 3.2. Summary of mechanisms in the NACS.

3.6 Appendix: Learning in the AEM

Specifically, assume a two-layer network is used (without any hidden layer). The off-line (cumulative) learning rule is:

$$\Delta w_{jk} = \alpha \sum_i (d_j(x_i) - h_j(x_i)) x_{jk}(x_i) \tag{3.15}$$

where x_i is the input to the network, $d_j(x_i)$ is the given target output for x_i, $h_j(x_i)$ is the actual output (which represents the frequency), and $x_{jk}(x_i)$ is the kth input (from the kth input unit) to the jth output unit given input x_i. An on-line version of the learning rule is:

$$\Delta w_{jk} = \alpha (d_j(x_i) - h_j(x_i)) x_{jk}(x_i) \tag{3.16}$$

after the presentation of each input x_i. The network (either the SFN or the AFN) may be trained with experience from the EM (4-tuples of state, action, new state, and reinforcement), which provide the input (x_i) and the target output ($d_j(x_i)$). A probability may be specified that determines the likelihood that items from the EM will be used to train the AEM. One may then randomly choose a pre-specified number of such items from the EM, at each step, for training the AEM.

3.7 Appendix: Using the AEM in Q-learning

Q-learning can incorporate the frequency estimates represented by the AEM. The Q-learning equation can be changed to:

$$\Delta Q(x, a) = \alpha (r + \gamma \sum_y Pr(x, a, y) \times e(y) - Q(x, a)) \tag{3.17}$$

where $Pr(x, a, y)$ is the transition probability going from (x, a) to y (which is obtained from the SFN), and the summation is over all possible next states. It is believed that this enhanced Q-learning equation can speed up learning and improve convergence (Barto et al 1995, Bertsekas and Tsitsiklis 1996).

3.8 Appendix: The Algorithm for Action-Directed Reasoning

Here is a sketch of the reasoning process in the NACS under the direction of the ACS:

1. Observe the current state x and store it in the current state buffer.
2. Compute in the bottom level of the ACS (the IDNs) the "value" of each of the possible actions (a_i's) in the state x: $Q(x, a_1)$, $Q(x, a_2)$, ..., $Q(x, a_n)$.
3. Find out all the possible actions (b_1, b_2, ..., b_m) at the top level of the ACS (the ARS), based on the the current state information x and the existing action rules at the top level (the ARS).
4. Choose an appropriate action a through integrating the recommendations of a_i's and b_j's
5. Perform the action a.
6. If the action chosen involves reasoning in the NACS, reasoning takes place in the NACS, based on the information provided by that adopted action, either a specific set of dimensional values or a set of chunks. Repeat the following steps:

6.1. As directed by the ACS action, perform reasoning in the GKS using associative rules (if any). A reasoning method may be specified by the action too. Reasoning continues either until there is no more new conclusion that can be drawn, or until a fix time limit is reached. The time limit may be specified by the adopted action as well.

6.2. As directed by the ACS action, perform in the AMNs associative mapping. Furthermore, optionally, the AMNs may perform further associative mapping from the result of previous associative mapping (one step at a time).

6.3. As directed by the ACS action, combine the results of associative mapping at the bottom level (which is sent up to the top level), with the results of the top-level inference.

7. As determined by the adopted action of the ACS, output the "filtered" results, (1) in the form of a set of chunks (that are compatible with the final result), or (2) in the form of one chunk (selected through a Boltzmann distribution based on chunk strengths).
8. After performing the action, observe the next state y and (possibly) the reinforcement r.

8.1. Update the IDNs in accordance with QBP.
8.2. Update the ARS using RER and/or IRL.
8.3. If the NACS was used at this step, update the GKS through extracting chunks and associative rules corresponding to the results from the AMNs, if any mapping occurred in the AMN and if there are no corresponding rules in the GKS.
8.4. If the NACS was used at this step, optionally, update the AMNs, through presenting associations to the network (including the auto-association in the AAM, and the hetero-association in the HAM).
9. Go back to Step 1.

3.9 Appendix: Other Possibilities in Coordinating the ACS and the NACS

In case of reasoning in the service of action decision making, both action rules from the ACS and associative rules from the NACS may be used. However, the conclusion from these two components must be combined with the result of the bottom level of the ACS, using one of the integration methods discussed in the chapter on the ACS, to reach a final decision on an action.

On the other hand, in case of pure reasoning (not in the service of action decision making), both action rules from the ACS and associative rules from the NACS may be used. A state is presented to the agent. Then, reasoning takes place. The reasoning process produces conclusions, in the form of a number of chunks, from action rules and associative rules. Retrieval from the AMNs (which is also presented with the information from the current state buffer) may also be used. The results from the AMNs and the rules (action rules and associative rules) may be combined.

Beside the above two methods (pure action decision making vs. pure reasoning), reasoning and action decision making may go on simultaneously and separately. There are a number of possibilities in this regard. One possibility is to set the relative timing of the two processes, and the frequency with which to update the input of the NACS from the external environment (which is changing, in part because of the actions performed by the ACS). However, a better alternative is to use the actions of the ACS to coordinate the two subsystems, as discussed before. In the latter case,

the reasoning in the NACS is completely under the control of the ACS, ensuring better coordination.

Chapter 4

The Motivational and Meta-Cognitive Subsystems

4.1 Introduction

We are now ready to move one step further, addressing more complex kinds of agent/environment interaction in which motivational and meta-cognitive issues arise. Modeling motivational and meta-level processes (and their incorporation into a cognitive architecture) is a topic of significant interest to both cognitive science and artificial intelligence (especially robotics).

To begin with, it is not too far-fetched to posit that cognitive agents must meet the following criteria in their activities, among many others:

- Sustainability: An agent must attend to its basic needs, such as hunger and thirst. The agent must also know to avoid danger and so on (Toates 1986).
- Purposefulness: The action of an agent must be chosen in accordance with some criteria, instead of completely randomly (Hull 1943, Anderson 1993). Those criteria are related to enhancing sustainability of an agent (Toates 1986).
- Focus: An agent must be able to focus its activities in some ways, with respect to particular purposes. Its actions need to be consistent, persistent, and contiguous, in order to fulfill its purposes (Toates 1987). However, an agent needs to be able to give up some of its activities, temporally or permanently, when necessary (Simon 1967, Sloman 1986).

- Adaptivity: An agent must be able to adapt its behavior (i.e., to learn) to improve its purposefulness, sustainability, and focus.

In order to meet these criteria, motivational and meta-cognitive processes are necessary, especially to deal with issues of purpose and focus.

To develop supervisory systems of motivational and meta-cognitive processes, many questions relevant to the understanding of mechanistic processes of motivation and meta-cognition need to be asked. For example, how can the internal drives, needs, and desires of an agent be represented? Are they explicitly represented, as symbolic/logicist AI would suggest, or are they implicitly represented (in some ways)? (See the two contrasting quotes at the beginning of this chapter.) Are they transient, or are they relatively invariant temporally? How do contexts affect their status? How do their variations affect performance? How can an agent exert control over its own cognitive processes? What factors determine such control? How is the control carried out? Is the control explicit or implicit?

In our view, the supervisory processes are made up of two components (two subsystems): the motivational subsystem and meta-cognitive subsystem. The motivational subsystem (the MS) is concerned with drives and their interactions (Toates 1986). That is, it is concerned with why an agent does what it does—how an agent chooses the actions it takes. Simply saying that an agent chooses actions to maximizes gains, rewards, or payoffs leaves open the question of what determines gains, rewards, or payoffs. The relevance of the motivational subsystem to the main component of an agent, the ACS, lies primarily in the fact that it provides the context in which the goal and the payoff of the ACS is set. It thereby influences the working of the ACS, and by extension, the working of the NACS. [1]

Closely tied to the motivational subsystem, there is also the meta-cognitive subsystem (the MCS), which also needs to be incorporated into cognitive architectures. The MCS is for controlling and regulating cognitive processes, for the sake of improving cognitive performance. The control and regulation may be in the forms of setting goals for the ACS, setting

[1] There have been interesting ideas concerning motivations from ethology (i.e., studies of animal behavior), such as those discussed by Toates (1986), McFarland (1989), and others. There has also been computational work that demonstrates the advantages of incorporating such a motivational subsystem in terms of enhancing the performance (e.g., survival) of agents, for example, Tyrell (1993) and Scheutz and Sloman (2001).

essential parameters of the ACS and the NACS, interrupting and changing on-going processes in the ACS and the NACS, and so on. The control and regulation may also be carried out through setting reinforcement functions (as discussed in the chapter on the ACS). The reinforcement functions of an agent may be determined by the MCS on the basis of motivational states, along with the goal of the system.

Many academic fields are relevant to the undertaking of understanding and developing the MS and the MCS. In particular, the following fields are pertinent: (1) artificial intelligence and robotics, especially the study of robotic agents, as well as hybrid connectionist/symbolic models; (2) cognitive modeling, including computational as well as mathematical modeling, that links cognition and motivation to the control of goal-directed behaviors; (3) formal approaches to mental states and their inter-relations, such as logics of beliefs, intention, and action, which are useful in providing insight into motivational and meta-cognitive issues; (4) ethology, which studies animal behaviors in their natural context and thus provides detailed pictures of motivational matters.

To foster integrative work to counteract the tendency of fragmentation in cognitive science into narrow and isolated sub-disciplines, it is necessary to consider seriously the overall architecture of the mind that incorporates, rather than excludes, important elements such as motivations and meta-cognition. Furthermore, it is beneficial to translate into architectural terms the understanding that has been achieved of the inter-relations among cognitive, meta-cognitive, motivational, and emotional aspects of the mind (Maslow 1962, 1987, Simon 1967, Toates 1986, Weiner 1992). By so doing, we may create a more complete picture of the structuring of the mind, and an overall understanding of the interaction among cognition, motivation, meta-cognition, and emotion, in the context of the overall structuring of the mind.

In the remainder of this chapter, details of these two subsystems will be developed. First, the motivational subsystem will be developed, drawing ideas from ethology, as well as from human cognition. The result is a system with dual representation. Later on, the meta-cognitive subsystem will also be developed. The design of meta-cognitive processes again mirrors the two-level architecture of CLARION, based on prior work on meta-cognition.

4.2 The Motivational Subsystem

In this section, the question of a proper motivational system that guides an agent's interaction with its environment and guarantees characteristics of motivated behavior will be examined. This system distinguishes motivational agents from simpler reactive agents.

4.2.1 Considerations Concerning Dual Representation

Let us examine the issue of explicit versus implicit representation of these internal mental structures. On the one hand, it is hard to imagine that there is no explicit representation of *goals* in a cognitive agent, since all the evidence points to the contrary (Anderson 1993, Anderson and Lebiere 1998, Rosenbloom et al 1993). But, on the other hand, the process of generating drives, needs, or desires are certainly not explicit and not readily accessible cognitively (Hull 1943). So, it seems reasonable to assume that, again, the idea of dual representation is applicable and, relatedly, implicit motivational processes are primary and more essential than explicit representation of goals.

Evidently, some combinations of explicit and implicit representations of goals and drives are needed. Reinforcement is determined from these (implicit or explicit) representations. We further hypothesize that the explicit representation consists mainly of explicit goals of an agent (used in the ACS, as explained before). It is necessary to the ACS because of the requirement that an agent tailors its action (within the ACS) to the achievement of a goal. Without an explicit representation, it would be more difficult to achieve that. Explicit goals provide specific and tangible motivations for actions. Explicit goal setting also allows more behavioral flexibility (or "individuation" as termed by Epstein 1982), and formation of expectancies (Epstein 1982), all within the ACS. Moreover, it may sometimes be necessary to compute a match of a state of the world to the goal, so as to discern the progress in achieving the goal and to generate context-dependent reinforcement signals (all within the MCS, to be explained later). This match may be facilitated by using an explicit representation of goals in an agent. Furthermore, drive states change from moment to moment, but explicit goal representation is more persistent and longer lasting. In many circumstances, persistence in goal

attainment is needed (as discussed in section 1). In addition, explicit representation enables explicit cognitive processes to work on these goals and their attainment, in addition to implicit processes. As we discussed before, explicit representation of goals is implemented with the use of the "goal structure" (see the earlier chapter on the ACS).

A bipartite system of motivational representation may be as follows. The (explicit) goals (such as "finding food") of an agent (which is tied to the working of the ACS, as explained before) may be generated based on (past and current) internal drive states (for example, "being hungry") of the agent (within the MCS, to be explained later). This explicit representation of goals derives from, and hinges upon, (implicit) drive states.

This duality is nicely expressed by the two contrasting quotes at the beginning of this chapter. The dual motivational process is, to some extent, innate, molded by long evolutionary processes, but with the possibility of adjustments/adaptation from the existential experience of an individual agent. [2] It is pertinent to note that in robotics research, the development of motivational models has been in the direction of building hierarchical structures to provide more advanced functions, using reactive behavior as a foundation (Savage 2003). This, to some extent, is consistent with our dual-representation approach.

4.2.2 Considerations Concerning Drives

Along the line of Toates' (1986) model of motivation, Tyrell (1993) identified a set of considerations concerning drives that the design of the motivational subsystem of an agent must take into account. These considerations include:

- *Proportional activation.* The activation of a drive should be proportional to corresponding offsets, or deficits, in related aspects (such as food or water).
- *Opportunism.* An agent needs to incorporate considerations concerning opportunities, when calculating desirability of alternatives in choosing its actions. For example, the availability of water may lead to prefer

[2] Note that, although drive states may be identified individually as we have done above, such identifications are approximate. They do not represent the full complexity of the matter. Thus, we view drive states as implicit nevertheless.

drinking water over gathering food, provided that food deficits is not too great.

- *Contiguity of actions.* There should be a tendency to continue the current action sequence, rather than switching to a different sequence, to avoid the overhead of switching.
- *Persistence.* Similarly, actions to satisfy a drive should persist beyond minimum satisfaction, that is, beyond a level of satisfaction barely enough to reduce the most urgent drive to be slightly below some other drives. For example, an agent should not run toward a water source and drink only a minimum amount, and then run toward a food source and eat a minimum amount, then going back to the water source to repeat the cycle.
- *Interruption when necessary.* However, when a more urgent drive arises (such as "avoid-danger"), actions for a lower-priority drive (such as "get-sleep") may be interrupted.
- *Combination of preferences.* The preferences resulting from different drives should be combined to generate a somewhat higher overall preference. Thus, a compromise candidate may be generated that is not the best for any single drive but the best in terms of the combined preference.

Let us see how these considerations may be fulfilled. First of all, the first two considerations point to the use of products, such as *food-deficit * food-stimulus*, in determining drives, which takes into consideration both need and availability (Tyrell 1993).

The next two considerations necessitate a persistent goal structure, which can be set and then persist until an interruption by a much more urgent drive (such as "avoid-danger" when danger is close by). In this way, we may avoid "thrashing": switching back and forth among two or more alternative tasks that are demanded by drives with comparable drive strengths, while preserving the possibility of interruption when a much more urgent need arises.

Combination of preferences is an issue that deserves careful consideration. It is believed that combination should be carried out by the resemblance of a multi-vote voting system whereby a goal emerges from tallying the votes cast by different drives (cf. Tyrell 1993). The problem with the single-vote approach is that only the top-priority goal

of each drive is taken into consideration, but lesser goals may be ignored, which may nevertheless make excellent compromise candidates. The multi-vote approach takes into consideration multiple preferences. Following this approach, we may implement combination in a variety of ways. For example, a connectionist network may be used to implement a multi-vote approach, which leads to the setting of a goal based on all the preferences of all the active drives. See the later discussion of the MCS, which is responsible for actual goal setting (as well as other functions).

Let us now turn to examine in detail a set of drives and how we may deal with them to satisfy the afore-discussed considerations.

4.2.3 Primary Drives: Low-Level

There have been a variety of taxonomies of drives being proposed by various researchers in the past. For example, the work of Lewin (1936), Hull (1943), and McClelland (1951) is well known. Sloman (1986) hypothesized a taxonomy of motives and their working in terms of goal setting. Tyrell (1993) came up with a set of basic drives, and their working mechanisms.

Although Tyrell's (1993) system was mainly for simple organisms, we can build more complex drive systems on its basis. Here, we refer to as *primary drives* those drives that are essential to an agent and are most likely built-in (hard-wired) to begin with. A default set of low-level primary drives and their interactions are as follows (cf. Tyrell 1993):

Get-food. The strength of this drive is proportional to *0.95* * max *(food-deficit, food-deficit * food-stimulus)*. The maximum strength of this drive is 0.95. The actual strength is determined by two factors: *food-deficit* felt by the agent, and the *food-stimulus* perceived by it. The use of *food-stimulus* (as well as *water-stimulus*, *mate-stimulus*, and so on, to be detailed later) is necessary, because otherwise an agent may be dominated by one slightly larger deficit and ignore accessibility issues all together. [3] Thus, the product of *food-deficit * food-stimulus* is included above. However, *food-deficit* alone needs to be taken into account too, because, otherwise an agent may starve to death if *food-stimulus* is not available at all (while,

[3] For example, when water is nearby and easily accessible, and *food-deficit* is slightly larger than *water-deficit*, but food-stimulus is not available, indicating the unavailability of food, the agent should consider satisfy the *water-deficit* first.

e.g., *water-stimulus* is abundantly available). [4]

Get-water. The strength of this drive is proportional to *0.95 ** max *(water-deficit, water-deficit * water-stimulus)*. This situation is similar to *get-food*. For exactly the same reason as described above with regard to *get-food*, both *water-deficit* and *water-deficit * water-stimulus* are included in the determination of the strength of this drive.

Avoid-danger. The strength of this drive is proportional to *0.98 * danger-stimulus * danger-certainty*. The maximum strength of this drive is 0.98. It is proportional to the danger signal: its distance, severity (disincentive value), and certainty. The first two factors are captured by *danger-stimulus* (which is determined by distance and severity), and the third factor by *danger-certainty*.

Get-sleep. The strength of this drive is proportional to *0.95 ** max *(night-proximity, exhaustion)*. This formula is self-explanatory.

Reproduce. The strength of this drive is determined by *0.30 + (0.60 * mate-stimulus)*. This drive is always present to a certain extent (at the level of 0.30), and intensified when *mate-stimulus* is present, proportional to the intensity of *mate-stimulus* (up to the level of $0.30 + 0.60 = 0.90$).

There also exists a set of "avoid saturation" drives. For example, **avoid-water-saturation** gets turned on when there is too much water in the body; **avoid-food-saturation** gets turned on when there is too much food in the body. Their strengths increase with the increase of the excess amount of water or food. But, their strengths are not as strong as the corresponding "get" drives. Possible equations for them are: **avoid-water-saturation** = *0.20 * water-excess*, and **avoid-food-saturation** = *0.20 * food-excess*.

There may be other primary drives, such as **curiosity** and **avoid-boredom**. Although we will not describe these drives here, one may easily add them into the afore-described framework.

4.2.4 Primary Drives: High-Level

Beyond these basic drives concerning physiological needs, there are also higher-level (and yet primary, in the sense of being "hardwired") drives. Maslow (1987) has developed a set of these drives in the form of a "need

[4] Note that *food-stimulus* captures both the "incentive" value of a food item as well as its accessibility (its distance). See Hull (1951).

hierarchy". A few relevant high-level drives, on the basis of Maslow's formulation of needs, are as follows:

Belongingness. As Maslow put it, it denotes "our deep animal tendencies to herd, to flock, to join, to belong." Clearly, this drive is species-specific—not all animal species have an equally strong need for social belongingness.

Esteem. Maslow claimed that "all people in our society .... have a need or desire for a stable, firmly based, usually high evaluation of themselves, for self respect or self esteem, and for the esteem of others". It includes the desire for competence, adequacy, recognition, attention, and domination. All of the above facets lead to self esteem and/or esteem by others.

Self-actualization. This term is used by Maslow to refer to the tendencies for humans to become actualized in what they are potentially, that is, the desires to become more of what one idiosyncratically is (or to become everything that one is destined to become).

Often, the degree to which each of these drives is significant is variable. It may be modulated, to some extent, by cultural differences, individual differences, or physical/physiological conditions. Nevertheless, the existence of these drives and their primary importance to the functioning of humans and human societies are believed to be universal, as has been discussed by many clinical psychologists and psychoanalysts (in the tradition of Sigmund Freud, Alfred Adler and Carl Jung), and is fairly well established.

Due to the "hierarchical" nature of these drives, that is, the fact that more basic needs must be satisfied first (Maslow 1987), the deficit of more basic drives must weigh more heavily than others. Based on this consideration, the drive strength for belongingness may be set at *0.20 + 0.50 * belongingness-deficit*. The maximum strength of this drive is 0.70. The strength is determined by two factors: the pertinent deficit felt by the agent, and an ever-present component (0.20). On the other hand, the drive strength for esteem may be set at *0.15 + 0.45 * esteem-deficit*, which is analogous to belongingness. Finally, the drive strength for self-actualization may be, likewise, set at *0.10 + 0.40 * self-actualization-deficit*. Given the high-level nature of these drives, the parameters for these drives may well vary from individual to individual.

4.2.5 Derived (Secondary) Drives

Besides these primary drives, which are built-in and relatively unalterable, there are also "derived" drives, which are secondary, changeable, and acquired mostly in the process of satisfying primary drives.

Derived drives may include the following types: (1) gradually acquired drives, through "conditioning"; that is, through the association of a (secondary) goal with a (primary) drive, the (secondary) goal becomes a drive by itself (Hull 1951); (2) externally set drives, through externally given instructions; for example, due to the transfer of the desire to please instructors or superiors into a specific desire to conform to the instructions.

Although we do not describe derived drives in detail, one may, however, use such drives in simulation. One may (1) set up additional drives (by hand) to simulate the *effect* of derived drives (see the later discussion of simulations), or (2) add the process of acquiring derived drives to the architecture.

4.2.6 Structure of the Motivational Subsystem

The structure of the motivational subsystem (the MS) is as shown in Figure 4.1. This subsystem is not standalone—it is closed tied to the meta-cognitive subsystem (e.g., for the purpose of goal setting by the MCS) and the ACS (to set, to change, and to carry out goals).

In this subsystem, the goal structure has been described as belonging to the ACS (see the chapter on the ACS)—in fact, it is an integral part of both subsystems, as well as closely tied to the MCS. So it is at the center of the whole system. The goal structure constitutes an explicit representation of motivation, and drives an implicit one. However, it is not necessarily the case that the two types of representations directly correspond to each other (e.g., one being extracted from the other), as in the case of the ACS or the NACS.

The mapping between the state of the world (as perceived by an agent, including the sensing of various deficits) and the drives can be implemented, in accordance with the afore-specified value ranges and relations, by backpropagation networks. The networks can be used to identify relevant features (and their intensities), such as *food-stimulus* or *mate-stimulus*, from raw sensory input. The output of such a network may

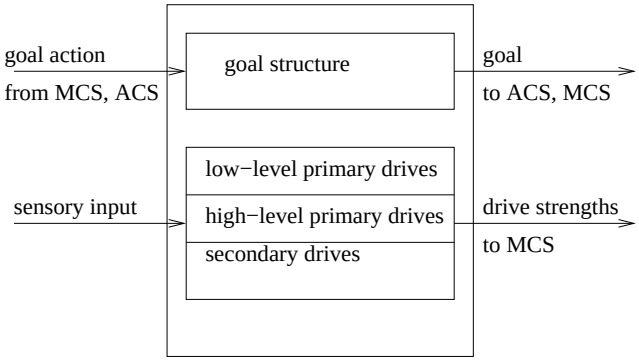

Figure 4.1. Structure of the motivational subsystem.

be the strengths of drives.

In advance of cognitive modeling of specific tasks, a drive network may be trained off-line as follows: The input to the net consist of raw sensory input, that is, without pre-processing that identifies various deficits and stimuli (although such pre-processing is possible and may make learning easier). The output are proper drive strengths, for example, as calculated from the afore-given formulas (although this is not necessarily the case). Through backpropagation learning, the network *learns* to identify relevant deficits and stimuli through its three-layered weight structure and to output proper (pre-determined) drive strengths. It may be hypothesized that this training process is a (very rough) approximation of a long evolutionary process that gradually shapes a drive system, with individual adjustment and adaptation to some degree by individual agents during their lifetime.

This is the preferred alternative to using the afore-specified formulas directly, which would require various deficits to be given as input individually.

Note that beyond the afore-described simple drive system, more complex sets of drives may be devised. More complex drive systems may include truly hierarchical structures, enabling the modeling of more complex relationships among drives (although there have been criticisms of strictly hierarchical decision making with regard to drives; see Tyrell 1993). For an existing example, see Timberlake and Lucas (1989).

4.3 The Meta-Cognitive Subsystem

The drive states and the goals derived from these drive states lead to the need for meta-cognitive control, which regulates not only goal structures but also cognitive processes per se, for the sake of facilitating the achievement of the goals. [5]

4.3.1 Considerations

According to Flavell (1976), meta-cognition refers to "one's knowledge concerning one's own cognitive processes and products, or anything related to them." Meta-cognition also includes "the active monitoring and consequent regulation and orchestration of these processes in relation to the cognitive objects or data on which they bear, usually in the service of some concrete goal or objective." This notion of meta-cognition is operationalized within CLARION.

The meta-cognitive subsystem (the MCS) is comprised of two levels of implicit and explicit processes, the same as the overall CLARION architecture. In this subsystem, the two levels are both action-centered, and are very similar to the ACS. The bottom level consists of implicit decision networks. The top level consists of groups of rules.

In this subsystem, mostly, the bottom level takes effective control. The top level has relatively little impact on outcomes. This is because meta-cognitive control is, usually, fast and effortless (Reder and Schunn 1996). Thus, it is mostly implicit. This can be accomplished through setting cross-level integration parameters, within the MCS, to strongly favor the bottom level. However, under some circumstances, the top level can also exert strong influence (Forrest-Pressley and Waller 1984). Furthermore, different modules of the MCS may have different degrees of reliance on the bottom level (as opposed to the top level). Thus, different integration parameters may be used for different modules.

As noted before, there is a competition among drives. The competition helps to set an appropriate goal for the agent and generate the corresponding reinforcement to the agent. There is not just competition

[5] Meta-cognition has been extensively studied in cognitive psychology; see, for example, Schwartz and Shapiro (1986), Metcalfe and Shimamura (1994), Reder (1996), Mazzoni and Nelson (1998), Lovett, Daily and Reder (2000), and so on.

among drives, but also complementarity among drives; for example, a food drive and a water drive together may point to a goal to go to a location where both can be found. Thus, multiple drives may lead to the setting of an appropriate, balanced goal taking into consideration different drives to different extent (in some accordance with drive strengths). In other words, the mapping from drives to goals is highly context-dependent (involving the context of multiple drives and the current state).

4.3.2 Structures and Functions

In this subsystem, several types of meta-cognitive processes are available, for various meta-cognitive control purposes. Among them, there are the following types:

(1) behavioral aims:
setting of reinforcement functions
setting of goals

(2) information filtering:
focusing of input dimensions in the ACS
focusing of input dimensions in the NACS
selection of input values (within certain input dimensions) in the ACS
selection of input values (within certain input dimensions) in the NACS

(3) information acquisition:
selection of learning methods in the ACS
selection of learning methods in the NACS

(4) information utilization:
selection of reasoning methods in the top level of the ACS
selection of reasoning methods in the top level of the NACS

(5) outcome selection:
selection of output dimensions in the ACS
selection of output dimensions in the NACS
selection of output values (within certain output dimensions) in

the ACS
selection of output values (within certain output dimensions) in the NACS

(6) cognitive modes:
selection of explicit processing, implicit processing, or a combination thereof (with proper integration parameters), in the ACS
selection of explicit processing, implicit processing, or a combination thereof (with proper integration parameters), in the NACS

(7) parameters of the ACS and the NACS:
setting of parameters for the IDNs
setting of parameters for the ARS
setting of parameters for the AMNs
setting of parameters for the GKS

Structurally, the subsystem may be subdivided into a number of modules. The bottom level consists of the following (separate) networks: the goal setting network, the reinforcement network, the input selection network, the output selection network, the parameter setting network (for setting learning rates, temperatures, action cost, etc.), and so on. In a similar and correlated fashion, the rules at the top level (if they exist there) can be correspondingly subdivided into these groups: the goal setting rule group, the reinforcement rule group, the input selection rule group, the output selection rule group, the parameter setting rule group, and so on. See Figure 4.2 for a diagram of the structure of the MCS.

The monitoring buffer may be subdivided into several sections: the ACS performance section, the NACS performance section, the ACS learning section, the NACS learning section, and other sections. Each section contains information about the bottom level and the top level of a subsystem.

In each performance section, the information about each level of a subsystem includes the *relative strength* of the top few conclusions, [6] which

[6] A relative strength is defined as:

$$rs = \frac{\sum_{i \in top} S_i}{\sum_i S_i} \tag{4.1}$$

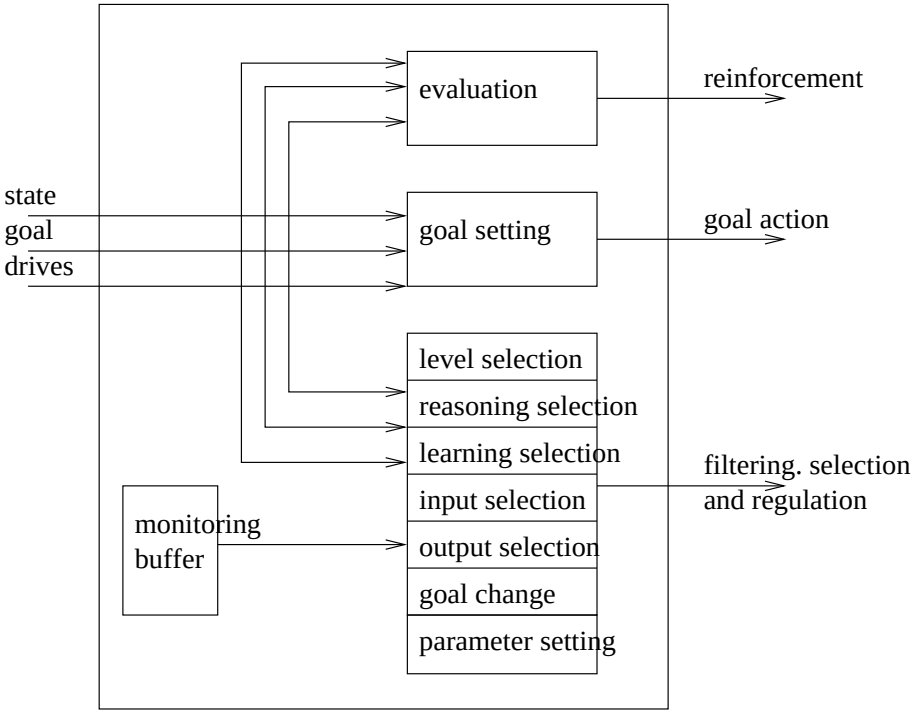

Figure 4.2. The structure of the meta-cognitive subsystem.

concerns how distinguished or certain the top conclusions are in relation to other competing ones. The numbers of top conclusions tracked are determined (for each component) by parameters (see the appendix chapter on options and parameters).

In each learning section, the performance of each subsystem is tracked for up to a certain number of episodes backwards (where "episodes" can be specified on a case-by-case basis). This information shows the improvement of performance (or the lack of it), namely learning, over time. In either level of the ACS, for the sake of gauging learning, each episode may be measured by *sr* (as explained in the chapter on the ACS). In the NACS, for the sake of gauging learning, the *number of chunks inferred* (combining the results of the two levels) may be counted. For each subsystem, a

where the size of the set *top* is determined by a parameter for each level of each subsystem.

parameter determines the number of episodes to keep track of (see the appendix chapter on options and parameters). [7]

4.3.3 Goal Setting by Drives

Theoretically, there are two possible ways in which goal setting may be carried out (cf. Tyrell 1993): (1) Balance-of-interests: Each drive votes for multiple goals, with different numerical values (e.g., ranging from 0 to 9), indicating priority or desirability. The votes from different drives are tallied, and the goal that receives the highest total score becomes the winning new goal. This is preferrable to a single-vote approach as discussed before. (2) Winner-take-all: In this case, the drive that has the highest strength wins the competition. The new goal is chosen for the sole purpose of reducing the winning drive. [8] As discussed earlier, we prefer a multi-vote approach, because it allows multiple considerations and different degrees of preferences to be taken into consideration. More importantly, the approach satisfies the requirement of "combination of preferences" as stated earlier. [9]

The actual process of goal selection is accomplished by the two levels of the MCS (including both implicit decision networks and explicit rules). Each of them maps the strengths of all the drives, along with the current state, to the current goal to be set, in accordance with the multi-vote

[7] The improvement of performance is measured by

$$m_1 = \sum_{i=t-c-1}^{t-1} (p_{i+1} - p_i)^2 \tag{4.2}$$

and

$$m_2 = p_t - p_{t-c-1} \tag{4.3}$$

where c is the number of step to be kept track of, t is the current time step, and p is the measure of a subsystem (used for gauging learning).

[8] In Tyrell (1993), this difference was referred to as free-flow decision-making versus "hierarchical" decision-making. The free-flow approach, with balance-of-interests, was justified in Tyrell (1993) through examples from behaviors of a variety of animal species.

[9] The approach adopted in CLARION is:

$$V_g^{total} = \sum_i V_g(S_i^d, state) \tag{4.4}$$

where g denotes a goal, i denotes a drive, S_i^d is the strength of drive i, V_g is a (predetermined) function of votes for goal g from drive i (S_i^d), and V_g^{total} is the total votes for goal g.

approach. The implicit network may be pre-trained in accordance with the afore-discussed formulas, using pre-generated training data. The explicit rules may be pre-set, or pre-trained using, for example, RER.

There are a few possibilities concerning the timing of setting new goals: (1) Setting a new goal whenever a different goal emerged through competition. This approach does not satisfy the requirements of persistence and contiguity, but it does enable prompt interruption. (2) Setting a new goal at fixed time interval, for example, at an interval of 10 time steps. In this way, to some extent, persistence and contiguity are maintained, and the "thrashing" problem ameliorated. But it does not enable prompt interruption. (3) Setting a new goal when the strength of one of the other drives (other than the original drive that set the current goal) is above a certain threshold (specified by a parameter), which signals an urgent situation. In this case, this high drive gets a chance to influence the setting of a new goal (along with other drives) through competition. To some limited extent, this approach alleviates the "thrashing" problem and satisfies the requirements of persistence and contiguity (when there are not two urgent drives simultaneously). Prompt interruption is enabled only when a drive is urgent. (4) Setting a new goal when the difference between the strengths of the currently most highly activated drive and the previously dominating drive (which helped to set the current goal) is above a certain threshold (specified by a parameter), which signals a significantly changed situation. This approach satisfies the requirements of persistence and contiguity. It also enables prompt interruption. Therefore, this approach is the default.

There are two possibilities concerning the handling of the relationship between old and new goals. When a new goal is decided on, the change to the goal structure may involve either (1) the removal of the current goal and the setting of the new goal, or (2) the setting of the new goal without removing the old goal. When the former approach is adopted, there can be no goal stack at all. When the latter approach is adopted, a stack structure may (or may not) emerge. Therefore, the latter approach is more versatile, and therefore it is adopted. [10]

[10] The handling of goal structure in CLARION is clearly different from Tyrell (1993), in which there is no explicit goal structure. Judging from various experimental results, it is highly unlikely that simple drive combination and action activation models, such as those described in Tyrell (1993), will be able to efficiently, effectively, and conveniently handle

Note that goal setting is not limited to the MS. Goals may be set by the ACS itself, through its actions ("goal-setting actions"; see the chapter on the ACS). The goal structure is there mainly for the purpose of achieving persistence and contiguity, as discussed before.

4.3.4 Reinforcement from Drives

As pointed by researchers in studying machine learning, one problem facing reinforcement learning is how to come up with an appropriate reinforcement signal (see the chapter on the ACS). In general, the world in which an agent lives does not readily provide a simple, scalar reinforcement signal, as usually assumed in the reinforcement learning literature (Barto and Sutton 1998, Bertsekas and Tsitsiklis 1996). Instead, it simply "changes" into a new "state", after an action is performed by the agent, which may (or may not) be observable to the acting agent. In such a real world, an appropriate reinforcement signal has to be determined *internally* within the agent, through synthesizing various information.

We may posit that such a signal is internally determined from the drives and the goals of the agent. More specifically, reinforcement signals are derived from measuring the degree of satisfaction of the drives and the goals of the agent.

Let us look into mechanisms for the determination of reinforcement signals from the current state (including sensory input, drives, and the goal). Reinforcement generation can be accomplished by a module of the MCS that produces an evaluation of the current state in terms of the current goal, in the context of the currently active drives: whether it satisfies the goal or not (that is, a binary output), or alternatively, how much it satisfies the goal (that is, a graded output). Environmental and internal sensory information, drives, and the goal (from the GS) are all input to the module. That is, this module maps (environmental and internal) sensory information, along with drives and the goal, to an evaluation that is used as reinforcement to the agent.

complex situations that a complex cognitive agent will have to deal with (for example, juggling various demands, constraints, and opportunities), while satisfying all of the afore-mentioned considerations (including persistence, contiguity, and interruption). It appears evident that something more complex than direct drive-to-action activations is needed.

It is not necessary for this module to produce an evaluation for every state (a combination of the sensory input, the current goal, and the drives) at every step. It is only necessary to produce an evaluation (a reinforcement signal) for a state that directly satisfies a goal in some way. This is because the sequential decision learning mechanism, such as Q-learning, can automatically propagate back an evaluation to temporally adjacent states and actions, in proportion to the likelihood of leading to receiving the evaluation, i.e., to the satisfaction of the goal. [11] Of course, intermediate evaluations (intermediate reinforcement), when available, can be used beneficially as part of a reinforcement function. Intermediate reinforcement may be based on the evaluation of the progress towards achieving a goal.

In general, a reinforcement function is accomplished both implicitly and explicitly, with both a top level and a bottom level, the same as other modules of the MCS. In the current implementation, reinforcement functions may be specified in a variety of ways. One way is specifying a reinforcement function explicitly as a set of simple rules (mapping from certain states to reinforcement signals; that is, in the form of a table). This is a common choice, as discussed earlier. Another possibility is using a (pre-trained) connectionist network to generate reinforcement signals for certain states. The current implementation of the architecture allows the specification of a reinforcement function either in the form of a network or as a set of rules (or both).

In biological organisms, there are, conceivably, pre-wired "automatic" evaluations and resulting "automatic" reinforcement, such as those concerning satisfying hunger (see Sun 2002). On the other hand, for humans and other sufficiently complex organisms, reinforcement may be (partially) learned as well, on the basis of interacting with the world. In CLARION, supervised learning (or simplified Q-learning) may be used for that purpose (when a backpropagation network is used). Alternatively, a set of rules may be set up or learned to generate reinforcement. The requisite target output of reinforcement functions may be from a variety of sources. For example, for humans and some other social animals, there are sociocultural sources.

[11] It is true that there are models that requires reinforcement signals every step of the way (i.e., for every state after every action), but this is the kind of setup that we should avoid due to the difficulty of generating proper reinforcement signals for every step.

[12] Imitative learning, as discussed in the chapter on the ACS, is a simple example of this.

Note that an alternative to the above scheme is to have a single connectionist network that receives as its input the internal drive states (e.g., being hungry) and the sensory information (past and present) and produces evaluations of them, instead of involving two networks: one mapping from drive states to the goal and another mapping from the goal (as well as drives and sensory information) to reinforcement. This network may be tuned through experience of the agent. On this view, there is, fundamentally, no need for explicit representation of goals, which, consequently, are merely a way of interpreting state changes, and action propensities that bring about such changes, in a post hoc fashion, as a retrospective rationalization. This is an alternative that is not endorsed by CLARION, for reasons discussed earlier (in relation to the essential considerations of motivational structures). [13]

4.3.5 Filtering, Selection, and Regulation

There are several rather separate aspects in this regard. These aspects include: focusing of input dimensions, focusing of output dimensions, selection of reasoning methods, and selection of learning methods.

First of all, attention focusing, either on input (information filtering on either specific dimensions or specific values) or on output (regulation of either dimensions or values), is carried out by the MCS, through meta-cognitive actions that suppress certain dimensions and/or values. The information based on which this type of decision is made lies mainly in the current sensory input, the current goal, the drives, the working memory, and the on-going performance of the ACS and the NACS (registered in the monitoring buffer of the MCS). [14] Based on such

[12] From sociocultural sources, an agent may learn to come up with complex, indirect forms of reinforcement, for the sake of evaluating complex sociocultural situations, as well as learn to adjust (to some extent) the evaluation of simple, direct, or even bodily states.

[13] Nevertheless, if one likes, this alternative may be viewed as a special case of the above approach adopted in CLARION.

[14] External instructions or hints, which may be captured in goals on the goal structure, can have significant impact on attention focusing.

information, certain dimensions are suppressed, which is accomplished by the MCS through sending a zero-activation signal to these dimensions that prohibits the transmission of information through multiplicative connections. Other dimensions receive a high activation (=1) in their multiplicative connections, and thus their information is transmitted forward. Attention focusing is carried out differently and separately on the ACS and on the NACS. Additionally, focusing can be done separately for individual components within a subsystem (such as for each IDN or for each rule set in the ARS), as discussed in the chapter on the ACS.

Similarly, selection of reasoning methods is carried out by the MCS, using its meta-cognitive actions that enable certain methods and disable some others. The basis for this type of decision, again, lies in the current goal, the drives, the working memory, the sensory information, and the ACS/NACS performance information (registered in the monitoring buffer of the MCS). The selection can be made separately for the top levels of the ACS and the NACS, as discussed before in respective chapters.

Likewise, selection of learning methods is carried out by the MCS, using its output to enable certain methods and disable some others. The basis for the decision consists of the monitoring of on-going learning performance (performance improvement or the lack of it) as registered in the monitoring buffer of the MCS, in addition to the sensory information, the current goal, the working memory, the drives, and so on. Again, selection can be made separately for different components (modules) of the ACS and the NACS, as discussed before. For example, possible selections for a bottom-level module of the ACS include: Q-learning with backpropagation, simplified Q-learning, supervised learning with backpropagation, and so on (see the chapter on the ACS).

The selection of which level to carry out the processing of a task is a key aspect of CLARION. It may be decided in a variety of ways in CLARION. Normally, the selection is determined based on a (fixed or variable) probability distribution (when stochastic selection is used, as explained earlier in the chapter on the ACS). Under some circumstances, the MCS may override the default selection, through designating the top level or the bottom level specifically. For example, in a highly analytical situation (such as during a logical analysis), one would rely more heavily, or even completely, on the top level. On the other hand, in a routine

(automatized) situation, one may rely (almost) completely on the bottom level. Generally, the relative reliance on either level can be altered by changing the probability of selecting a level by the MCS. Changes can be dictated by the MCS, by a probability value (e.g., a 0.85 probability, or a 1.0 probability, for the top level). Changes can also be made indirectly, by changing β parameters. [15]

The setting and changing of other parameters involved in the ACS and the NACS can also be carried out by the MCS. These parameters include, for example, bottom-level learning rate, temperature in stochastic decision making, rule learning thresholds for RER or IRL, and many others.

Note that not all selections of methods, strategies, and parameters are carried out by the MCS. Some of the selections of specific methods or strategies may be carried out by regular processes in the ACS. For example, the decision of "retrieve versus compute" when faced with a numerical problem (Reder and Schunn 1996) may be carried out by a module in the ACS (either explicitly or implicitly), before regular computation or retrieval is carried out by other modules in the ACS. As explained before, the ACS is divided into multiple modules (each including an IDN and an ARS). Some of these (higher-level) modules may select strategies that can be used in other (lower-level) modules. There may be a hierarchical organization of modules that facilitate hierarchical, increasingly more detailed processing from the coarsest level down. We hypothesize that only selection and regulation at the highest level must be carried out by the MCS. In a way, the selection carried out by the MCS may be viewed as being at the very top of hierarchical decision making based on a hierarchical organization of modules (which, mostly, reside in the ACS).

In the same vein, we may note that some knowledge evoked in various existing meta-cognitive experimental work, such as the "feeling of knowing" judgment and related strategy selection (Reder and Schunn 1996), may conceivably be found in the ACS or the NACS. For example, the "feeling of knowing" judgment may be assessed in the NACS through similarity-based reasoning processes. Some other types of meta-cognitive knowledge, such as the "warmth" judgment (regarding how close to a solution one is, or how certain a solution is; Metcalfe 1986), may be registered in the monitoring

[15] In case a more complex integration method is used, similar meta-cognitive control applies.

buffer of the MCS. Meta-cognitive control, on the basis of such information, can be carried out either by the MCS, or by the ACS (when decisions to be made are at a relatively low level).

4.4 Summary

In this chapter, we first discussed the details of the motivational subsystem, focusing on the representation of motivational states. We then moved on to discuss the details of the MCS, which went from the motivational representation to the control of behavior (in the ACS and the NACS). We also touched upon a thorny issue, emotion, which was important nevertheless both theoretically and practically. Finally, we briefly reviewed the issue of personality.

In our construction of motivational and meta-cognitive subsystems, we combined reactive accounts of behavior (Brooks 1991, Bickhard 1993, Savage 2003) with motivational accounts (Toates 1986, Weiner 1992, Tyrell 1993). At its core, CLARION is reactive, without the necessity of relying on motivational constructs, as in the ACS, which can run by itself (assuming the input from the MS and the MCS are not available or are constants). On the other hand, CLARION can also incorporate complex motivational constructs, including a bipartite representation of goals and drives. Thus, it, in a way, synthesized the two world views.

Although this chapter contained quite a lot of theoretical speculations, the focus, however, was clearly the same as the preceding chapter. That is, the focus has been on the construction of processes and representations for the sake of explaining cognitive data in a generalized way. (The simulation of data using these subsystems will follow in later chapters.)

Chapter 5

Determining Response Times

For a variety of simulations of cognitive processes, response times of a model under various circumstances may need to be determined. This is because in many cognitive tasks and experiments, response times are used as an important measure of human performance. To avoid ad hoc and sometimes arbitrary methods of calculating times, we need to specify an approximate, but justifiable and systematic, way of determining response times.

5.1 Response Times of the ACS

Clearly, the overall response time (RT) is a function of the response times of the bottom level and the top level. That is,

$$RT = f(RT_{BL}, RT_{TL}) \tag{5.1}$$

If we use *stochastic selection*, the overall response time (RT) is determined by which level is being used to generate a final output. If the output from the bottom level is chosen, the overall response time is the bottom-level response time. If the output from the top level is chosen, the overall response time is the top-level response time.

If *bottom-up rectification* or *top-down guidance* is used (including using the *weighted-sum* method), the overall response time is the sum of the two response times from the two levels.

The response time of the bottom level is determined by

$$RT_{BL} = f_{BL}(PT_{BL}, DT_{BL}, AT_{BL}) \tag{5.2}$$

where RT_{BL} is the bottom-level response time, PT_{BL} is the bottom-level perceptual time, DT_{BL} is the bottom-level decision time, and AT_{BL} is the bottom-level actuation (action) time. Here f_{BL} is a function that combines the three variables. For example, a simple (and probably simplistic) way of constructing this function may be: $RT_{BL} = PT_{BL} + DT_{BL} + AT_{BL}$.

The response time of the top level is determined by

$$RT_{TL} = f_{TL}(PT_{TL}, DT_{TL}, AT_{TL}) \tag{5.3}$$

where RT_{TL} is the top-level response time, PT_{TL} is the top-level perceptual time, DT_{TL} is the top-level decision time, and AT_{TL} is the top-level actuation (action) time. f_{TL} is a function. For example, $RT_{TL} = PT_{TL} + DT_{TL} + AT_{TL}$.

Let us look into the parameters. First of all, we may use (as the default):

$$PT_{TL} = PT_{BL} + 100ms \tag{5.4}$$

where PT_{BL} is set at 200 *ms* (as the default). This is because it is known that before conscious awareness of perceptual information happens, a great deal of unconscious pre-processing goes on (Marcel 1983). Therefore it is reasonable to assume that it takes more time to consciously access perceptual information than to access it at an unconscious level. Likewise, it has been shown experimentally that for a unconsciously initiated intention to become conscious, it takes several hundred *ms* (Libet 1985). Synthesizing these above viewpoints, we set the default values of these parameters at the above values.

Second, DT_{BL} is assumed to be a fixed value. This is because this kind of unconscious decision making is rather direct (from state to action directly) and therefore fast. As a default, we set $DT_{BL} = 350ms$. This is roughly based on the fact that, according to Libet (1985), more than $350ms$ time lag leads to conscious decision making; we set DT_{BL} to that limit. [1]

Next, for determining DT_{TL}, we may do the following. Assuming that response time of using an action rule or an action chunk is proportional to the odds (not probabilities) of that rule or chunk being needed based on

[1] An alternative (which we have also used in simulations) is to set DT_{BL} to approximately half of that—that is, $DT_{BL} = 200ms$.

past use (as per Anderson 1993), we therefore have

$$DT_{TL} = \text{operation-time} + t_0 \times 1/\text{rule-bla} + t_1 \times 1/\text{chunk-bla} \qquad (5.5)$$

where t_0 and t_1 are constants, and *operation-time* is determined by the mental operation dictated by a particular action rule. This formula indicates two kinds of priming: priming of the rule applied, and priming of the action chunk selected. As described earlier, the base-level activation represents the odds of needing a piece of information based on the history of its past use (Anderson 1993). The base-level activation of a rule represents the odds that the rule is needed at the current step. The base-level activation of a chunk represents the odds that the chunk is needed at the current step. [2]

Note, however, that different from Anderson (1993), the base-level activations of working memory items are not included in this formula (even if WM items are involved in a rule). [3] This is because working memory items represent results of past retrieval from long-term memory. In CLARION, the retrieval of these items from long-term memory is done separately by specific applications of retrieval rules. Therefore, in CLARION, the retrieval time of an item has been calculated into the time of applying a retrieval rule. Consequently, there is no need to take into account the retrieval times of working memory items at the time of action decision making using these items.

On the other hand, AT_{TL} and AT_{BL} are variables. Their values are dependent on a host of factors, including the response modality used (e.g., verbal responding, mouse clicking, button pressing, etc.). As an instance, for a verbal response or a mouse clicking, we may set $AT_{TL} = AT_{BL} = 500ms$ (Anderson and Lebiere 1998).

[2] Note that, if multiple rules matching the current input, we select one rule out of these rules, using one of the methods discussed in the chapter on the ACS, then the selected rule is applied. The response time of the applied rule is caculated as above.

[3] However, it is true that sometimes decision times may be dependent on retrieval times of certain items from memory. This situation is handled in CLARION by first retrieving an item (a chunk) from the NACS (see the chapter on the NACS), the time of which is indeed dependent on the base-level activation of the retrieved chunk, and then storing it in the WM and use it in action decision making through the WM. We believe that this is a justifiable approach, because working memory, as defined by Baddeley (1986), is supposed to be immediately available. That is why we use the notion of working memory in the first place—to distinguish it from regular memory, which may indeed involve complex retrieval processes.

The above framework for determining RTs can be extended to deal with a variety of issues related to RTs. For example, the well known speed/accuracy trade-off may be modeled within the framework. To do so, we need a few assumptions. Suppose that stochastic selection is used. We assume that DT_{TL} is a monotonically increasing function of t_0 and t_1. We also assume that the temperature in rule or action selection is a function of t_0 and t_1:

$$\tau_{TL} = g_1(t_0, t_1) \tag{5.6}$$

where g_1 is a monotonically decreasing function of t_0 and t_1 (which can be specified in a domain specific way). Under these assumptions, if t_0 and t_1 are reduced, that is, if the decision time (i.e., DT_{TL}) is reduced, then, the temperature (i.e., the randomness) of decision making is increased and consequently the accuracy of performance is decreased. Hence the speed/accuracy trade-off. Similarly, for the bottom level, we assume that DT_{BL} is variable, and that the temperature of action selection is a function of DT_{BL}:

$$\tau_{BL} = g_2(DT_{BL}) \tag{5.7}$$

where g_2 is a monotonically decreasing function of DT_{BL} (which can be specified by the modeler). Thus, if DT_{BL} is reduced, the temperature is increased and consequently the accuracy of performance is decreased. [4]

Note that, for the sake of simplicity and uniformity, we calculate response times of a sequence of steps as a series of time lags each of which is determined by the sum of the perception time, the decision time and the actuation time of that step. That is, response times are calculated in a step-by-step fashion, with no interaction between steps. However, in reality, there may be complex interaction and interleaving, which are not taken into consideration in the above simplified scheme.

5.2 Response Times of the NACS

First, let us look into the top level. Chunk retrieval time is inversely proportional to the base-level activation of the chunk in question. That

[4] In case of using the weighted-sum or a more complex integration method, τ can be similarly specified as a function of speed.

is, the retrieval time is

$$t_c = t_2 + t_3 \times 1/\text{chunk-bla} \tag{5.8}$$

where t_2 and t_3 are two constants.

Associative rule application time is inversely proportional to the base-level activation of the associative rule in question. That is, the retrieval time is

$$t_a = t_4 + t_5 \times 1/\text{associative-rule-bla} \tag{5.9}$$

where t_4 and t_5 are two constants. Application of associative rules may also involve the retrieval of result chunks. Therefore, in that case, we need to add in chunk retrieval times as well (as have been described above).

Since all of the applicable associative rules are applied in parallel in the GKS, the total associative rule application time is the maximum of individual associative rule application and result chunk retrieval times:

$$t_{GKS} = \max_{\text{all applicable rules}} (t_a + t_c) \tag{5.10}$$

Second, let us look into the bottom level. The time spent on one iteration of associative mapping from the AMNs (one pass going from input to output) is a constant, the same way as the IDN decision time is a constant. This is because we view implicit processes as direct mappings from input to output, and therefore, there is no complex process involved and all mappings are equal in terms of time. We denote that constant as t_{AMN}, the default value of which is $350ms$.

Any of these time periods can be viewed as part of AT_{TL} or AT_{BL}, when the ACS (either its top level or its bottom level) is used to direct chunk retrieval or associative rule application. In such cases, chunk retrieval or associative rule application is part of the execution of a relevant ACS decision, and thus part of the actuation time of the ACS. That is, if a rule from the ACS performs a retrieval action of chunks from the NACS, the chunk retrieval time from the NACS may be added to the actuation time (AT_{TL}) of that retrieval rule.

The failure of retrieval of a chunk in the GKS takes a certain amount of time to detect. It can be determined by a retrieval failure time parameter: t_f.

5.3 Response Times of the MS and the MCS

The response time of the MS and the MCS is determined in the same way as the ACS (since they are highly similar). The same equations as those used in the ACS are used in MS and the MCS. However, parameters and options different from those of the ACS may be adopted for the MS and for the MCS.

In addition, parameters and options may be set differently for the MS and for the MCS, since they are viewed as two separate subsystems. Their RT parameters need to be set separately. Note that the MS has no top level, but only a bottom level.

It is worth pointing out that the MS, the MCS, and the ACS operate in parallel. (The NACS is under the control of the ACS, under normal circumstances, so it can be considered as part of the ACS in this regard.) One would expect that the MS and the MCS are faster compared with the ACS and the NACS. The changes to drives, goals, parameters and so on, brought by the MS and the MCS, can take effect rapidly, while the ACS (and the NACS) are working. [5]

5.4 A General Note

To summarize the above calculation, RTs are determined by the following factors: level (top versus bottom), BLA (chunks and/or rules), and subsystem (ACS versus NACS). In a way, a response time is a linear function of the BLAs of all entities involved. This calculation apparently involves quite a lot of parameters, which are nevertheless highly redundant. That is, most of them can be merged, which can simplify the computation significantly. Nevertheless, they are kept separate here for the sake of maintaining conceptual clarity. As a result, the exact values of most of these parameters do not matter that much. What is important is the calculation of BLAs of chunks and rules, as well as *operation-time* sometimes, which are most relevant to determining RTs.

[5] It is generally the case that the ACS and the NACS will use whatever information and parameters are available at the beginning of the current step (the current perceptual-decision-actuation cycle). Changes occurred during the current step will take effect at the beginning of the next step.

5.5 More Precise Determination

For the sake of simplicity and uniformity, we may calculate the response time of a sequence of steps as the sum of a series of time lags each of which is determined by the sum of perception time, decision time and actuation time of that step. But this picture is a rather rough approximation.

An important issue in this regard is whether more detailed capturing of components of response times is necessary. For example, a more detailed capturing of RTs may involve modeling, in detail, perceptual processes and motor processes, the same as modeling decision processes (as done above). In order to more precisely determine response times, we may borrow some ideas from Kieras and Myer (1997).

Furthermore, we need to take into consideration the interaction and the interweaving of these three types of processes, as demonstrated by Kieras and Meyer (1998), Byrne and Anderson (1998), and others. In reality, perception, decision, and actuation of a task form a pipeline—a serial flow from one stage to the next. However, the actuation of one task may overlap with the perception and decision of another: That is, they may be concurrent (overlapping in time) (Kieras and Meyer 1998). Actuation of different modalities (e.g., left hand movement versus right hand movement) may also be parallelized to a significant extent. This is true whether these different actions of different modalities are for the same task or for different tasks. In this regard, we need to deal with many fine-grained issues: Perception time and decision time may or may not overlap (which may be domain-specific). If actuation time of one step is long, actuation of the next step may or may not have to wait for its end (depending, to some extent, on whether the same modality is involved or not). Preparation time for motor movements is lumped into decision time here, which may or may not be appropriate. And so on.

Such details may be interesting in their own right. However, the exploration of them is unlikely to shed additional light on the interaction between implicit and explicit knowledge, or the interaction between the ACS and the NACS, both of which we specifically focused on in this work.

Chapter 6

Appendix: Options and Parameters

6.1 The Major Subsystems

First of all, here are the options for the three major subsystems of CLARION. The finer options within each of them and the associated parameters will be described later.

Major subsystems involved (each on or off):
the ACS (default=on)
the NACS (default=off)
the MS/MCS (default=off)

6.2 Options of the ACS

The following options and their associated parameters are concerned with the ACS.

Components of the ACS (specify each):
goal stack, goal list, or neither (default = neither)
working memory (on or off; default =off)

Specifications of actions that operate within the system
WM actions (specify details)
goal actions (specify details)
external actions for reporting contents of WM or GS (specify details)

external actions for controlling the NACS (specify details)
(Note that each type above has a set of default actions as described before, and additional ones may be defined by users)

6.2.1 Options of the IDNs

There are the following options and parameters concerning the IDNs:

Bottom-level representation:
the working memory action network (on or off, if "WM" is on; default=off)
the goal action network (on or off, if a goal structure is selected; default=off)
the external action network(s) (specify number of networks; default=1)

Details of bottom-level representation:
i/o dimensions and values of external action network(s) (specify for each network)
eligibility conditions of external action network(s) (specify for each network, if the number of external action network is more than one)
i/o dimensions and values of the working memory action network (specify, if "WM action network" is on)
i/o dimensions and values of the goal action network (specify, if "goal action network" is on)
number of hidden units for each of these above networks
(Note that in specifying action dimensions and their values a number of commonly used actions, especially for controlling the NACS, are pre-set and available for use)

Bottom-level learning (choose one for each bottom-level network):
(1) Q-learning with backpropagation (default)
(2) simplified Q-learning with backpropagation
(3) backpropagation (supervised learning)
(4) top-down rule assimilation (if on, specify frequency x/y)

(5) imitative learning
(6) fixed
(Note that only one learning method can be used at any particular moment, for any particular network)

Parameters for BP and Q-learning (specify for each bottom-level network):
the BP learning rate (default: $\alpha = 0.1$)
the BP momentum (default: $mom = 0.1$)
the Q value discount rate (default: $\gamma = 0.99$)
the Boltzmann temperature for bottom-level action selection (if stochastic selection is used; default: $\tau_{BL} = 0.1$),

Reinforcement function specification (choose one):
(1) one function for all (default)
(2) one function for each network

One reinforcement function for all networks (specify, if "one function for all" is chosen):
reinforcement function in the form of $r_1(x)$

Reinforcement functions for each network (specify for each network, if "one function for each network" is chosen):
a reinforcement function for each external action network (in the form of $r_1(x)$)
a reinforcement function for goal actions, if the GS is selected (in the form of $r_2(x)$)
a reinforcement function for working memory actions, if the WM is selected (in the form of $r_3(x)$)
(Note that these functions may be determined by the MCS on the fly. They may also incorporate the input from the MS.)

6.2.2 Options of the ARS

There are the following options and parameters concerning rules in the ARS of the ACS. The options and their associated parameters are specified with respect to each rule group corresponding to an ACS bottom-level network

(including each external action network, the goal action network, and the WM action network).

Rule learning components (on or off, for each rule group):
RER (default=on)
IRL (default=off)
FR (default=off)
imitative learning (default=off)
plan extraction (default=off) (specify frequency if on)

6.2.2.1 Options for Rule and Chunk Representation

The following options and associated parameters are specified with respect to each rule group (corresponding to an ACS bottom-level network) and each rule set within.

Details of top-level representation (specify each):
i/o dimensions and values of working memory action rules (specify for each rule set)
i/o dimensions and values of goal action rules (specify for each rule set)
i/o dimensions and values of external action rules (specify for each rule set in each rule group corresponding to a bottom-level network)
(Note that they must be the subsets of the corresponding bottom-level representation.)

Action rule and chunk representation (each on or off):
action rule utility (default=on)
action rule base-level activation (default=on)
action chunk base-level activation (default=off)
partial match in rule support (default=on; must be on if "forward chaining with partial match" is selected)

Weights of a chunk representing an action rule condition (specify, when "partial match in rule support" is on):

$\forall i$: W_i^c (the weight of the ith dimension in the chunk representing the condition of an action rule) (default: $W_i^c = 1/n$, where n is the number of dimensions specified in the chunk)

The weight of an action rule (specify, when "partial match in rule support" is on):

W^r (the weight of the chunk representing the condition of an action rule) (default: $W^r = 1$)

Action rule base-level activation (choose one, when "action rule base-level activation" is on; default is 2):

(1) constant base-level activation (default: $B_j^r = 1$)

(2) recency-based base-level activation: default is

$$B_j^r = iB_j^r + c * \sum_{l=1}^{n} t_l^{-d}$$

where j indicates the jth rule at the top level, l indicates the lth encoding/use of that item, t is measured in ms (default: $c = 2, d = 0.5, iB_j^r = 0$)

Rule utility (choose one, when "action rule utility" is on; default is 2; may be specified with respect to individual FRs or individual IRL subsets):

(1) constants (if on, specify constants; default: $benefit_j = 2, cost_j = 1, \nu = 1$, and $U_j^r = benefit_j - \nu * cost_j$)

(2) functions (if on, specify functions; default: $benefit_j = \frac{c_7 + PM}{c_8 + PM + NM}$, where $c_7 = 1$ and $c_8 = 2$; $cost_j = \frac{execution-time-of-rule-j}{average-execution-time}$; and $U_j^r = benefit_j - \nu \times cost_j$, where $\nu = 1$)

Chunk base-level activation (choose one, if "action chunk BLA" is on; default is 2):

(1) constant base-level activation (default: $B_i^c = 1$)

(2) recency-based base-level activation: default is

$$B_i^c = iB_i^c + c * \sum_{l=1}^{n} t_l^{-d}$$

where i indicates the ith chunk, l indicates the lth encoding/use of the chunk (by the input, the bottom level, or other chunks via rules), t is measured in ms (default: $c = 2, d = 0.5, iB_i^c = 0$)

Parameters for action decision making (specify):
the temperature for top-level action selection (if stochastic selection is used) (default: $\tau_{TL} = 0.1$),

Use of action rules in action decision making (choose one; default is 1)
(1) forward chaining
(2) forward chaining with partial match (if "partial match in rule support" is on; if on, specify also a partial match threshold $threshold_{PM}$)
(Note that only one method can be used at any particular step)

6.2.2.2 Options for Rule Acquisition

The following options and associated parameters are concerned with the top level, and specified with respect to each rule group corresponding to each ACS bottom-level network (e.g., each external action network, the goal action network, and the WM action network).

Fixed rule sets (specify, if "FR" is on)

IRL details (if "IRL" is on):
the IRL rule subsets, in the order to be generated and tested (specify)
whether generalization and specialization shall be performed on IRL rules being tested (on or off; default=off; if on, specify relevant parameters below)

RER parameters (specify each, if "RER" is on):
the density parameter d_r (default = 1/100)
the rule extraction probability parameter p_{re} (default= 1)
specific-to-general (default=on),

general-to-specific (default=off)

Rule positivity criterion (specify each, if a corresponding option, RER or IRL, is on):

the positivity criterion used in RER (default criterion: a rule is positive, if $\gamma \max_b Q(y,b) - Q(x,a) + r > threshold_{RER}$, in which x and a match the rule condition and action respectively, and $threshold_{RER} = 0.08$ by default)

the positivity criterion used in IRL (default criterion: a rule is positive, if $\gamma \max_b Q(y,b) - Q(x,a) + r > threshold_{IRL}$, in which x and a match the rule condition and action respectively, and $threshold_{IRL} = 0.08$ by default)

IG measures for rule learning (specify each):

the IG measure for RER rule learning (if "RER" is on) (default: $IG(C, all) = log_2 \frac{PM_a(C)+c_1}{PM_a(C)+NM_a(C)+c_2} - log_2 \frac{PM_a(all)+c_1}{PM_a(all)+NM_a(all)+c_2}$)

the IG measure for IRL rule learning (if "IRL" is on) (default: $IG(C) = log_2 \frac{PM(C)+c_5}{PM(C)+NM(C)+c_6}$)

Rule learning thresholds (specify):

for RER (if "RER" is on): $threshold1$ (default = 4.0), and $threshold2$ (default = 1.0).

for IRL (if "IRL" is on): $threshold4$ (default =???), as well as $threshold5$ (default = 4.0) and $threshold6$ (default = 1.0) if generalization/specialization is used for IRL.

Action rule generalization/specialization style (choose one, for RER and IRL separately):

(1) contiguous value range

(2) non-contiguous value range (default)

(3) one value or all values

6.2.3 Goal Structure

Details of the goal structure:

the set of goal symbols (specify)

the size of the goal chunk (number of dimensions, including the goal dimension) $size_{gc}$ (specify; default: $size_{gc} = 1$)

each parameter dimension and its values (beside the goal dimension) (specify)

the total capacity of the goal structure $size_g$ (specify; default: $size_g = 7$)

the goal threshold $threshold_g$ (if goal list is selected) (default: $threshold_g$ =????)

goal actions by the top level only (using only FRs), or by both levels (choose one; default="top level only")

Goal base-level activation (choose one, if "goal list" is selected; default is 2):

(1) constant activation (default: $B_i^g = 1$)

(2) recency-based base-level activation: default is

$$B_i^g = iB_i^g + c * \sum_{l=1}^{n} t_l^{-d}$$

where i indicates the ith item in the goal list, l indicates the lth setting of that item (default: $c = 2, d = 0.5, iB_i^g = 0$)

6.2.4 Working Memory

Details of the WM:

the total capacity of the working memory $size_{wm}$ (specify; default: $size_{wm} = 7$)

the working memory threshold $threshold_{WM}$ (specify; default: $threshold_{WM} = 0.1$???)

flags in the WM (on or off; default=off; if yes, specify number of flags)

WM actions by the top level only (using only FRs), or by both levels (choose one; default="top level only")

flag actions by the top level only (using only FRs) or by both levels (choose one, if "flags"= on; default="top level only")

Working memory base-level activation (choose one, if "WM" is on; default is 2):

(1) constant base-level activation (default: $B_i^w = 1$)
(2) recency-based base-level activation: default is

$$B_i^w = iB_i^w + c * \sum_{l=1}^{n} t_l^{-d}$$

where i indicates the ith item in the working memory, l indicates the lth setting of that item (default: $c = 2, d = 0.5, iB_i^w = 0$)

6.2.5 Options for Integration and Coordinations

The method of integration is specified with respect to each module (a pair of an IDN and its corresponding rule group).

Cross-level integration methods (choose one)
(1) stochastic selection (default)
(2) top-down guidance
(3) bottom-up rectification

Stochastic selection (if "stochastic selection" is chosen, choose one of the following two ways):
(1) variable (default) (if on, specify $\beta_{BL}, \beta_{RER}, \beta_{IRL}, \beta_{FR}$; default: $\beta_{BL} = 0.7, \beta_{RER} = 0.15, \beta_{IRL} = 0.15, \beta_{FR} = 0$) (default: $c_3 = 1.0$ and $c_4 = 2.0$)
(2) fixed (if on, specify $P_{BL}, P_{RER}, P_{IRL}, P_{FR}$; default: $P_{BL} = 70\%, P_{RER} = 15\%, P_{IRL} = 15\%, P_{FR} = 0$)

Bottom-up rectification (if "bottom-up rectification" is chosen, choose one of the following; default=1):
(1) variable weighted-sum combination (if on, specify $\beta_{BL}, \beta_{RER}, \beta_{IRL}, \beta_{FR}$; default: $\beta_{BL} = 0.7, \beta_{RER} = 0.15, \beta_{IRL} = 0.15, \beta_{FR} = 0$; alternatively, specify separate measures if multiple bottom-level networks and multiple rule groups are used: $\beta_{BL-EXT_1}, ..., \beta_{BL-EXT_n}, \beta_{BL-GS}, \beta_{BL-WM}$, and so on) (default: $c_3 = 1.0$ and $c_4 = 2.0$)
(2) fixed weighted-sum combination (if on, specify $w_{BL}, w_{RER}, w_{IRL}, w_{FR}$; default: $w_{BL} = 0.70, w_{RER} = 0.15, w_{IRL} = 0.15, w_{FR} =$

0; alternatively, specify separate weights if multiple rule groups/networks are used: $w_{BL-EXT_1}, ..., w_{BL-EXT_n}, w_{BL-GS}, w_{BL-WM}$ and so on)

(3) top-level correction (if on, specify details)

Top-down guidance (if "top-down guidance" is chosen, choose one of the following; default=1):

(1) variable weighted-sum combination (if on, specify $\beta_{BL}, \beta_{RER}, \beta_{IRL}, \beta_{FR}$; default: $\beta_{BL} = 0.7, \beta_{RER} = 0.15, \beta_{IRL} = 0.15, \beta_{FR} = 0$; alternatively, specify separate measures if multiple bottom-level networks and multiple rule groups are used: $\beta_{BL-EXT_1}, ..., \beta_{BL-EXT_n}, \beta_{BL-GS}, \beta_{BL-WM}$, and so on) (default: $c_3 = 1.0$ and $c_4 = 2.0$)

(2) fixed weighted-sum combination (if on, specify $w_{BL}, w_{RER}, w_{IRL}, w_{FR}$; default: $w_{BL} = 0.70, w_{RER} = 0.15, w_{IRL} = 0.15, w_{FR} = 0$; alternatively, specify separate weights if multiple rule groups/networks are used: $w_{BL-EXT_1}, ..., w_{BL-EXT_n}, w_{BL-GS}, w_{BL-WM}$ and so on)

(3) bottom-level guidance taking (if on, specify details)

The Boltzmann temperature for overall action selection (if a weighted sum is used):

the Boltzmann temperature (default: $\tau = 0.1$)

The followings are specified uniformly (with regard to all IDNs and all of their corresponding rule groups, together).

Coordinating multiple external action networks/groups (choose one):

(1) perform all (default)

(2) perform only one based on eligibility (i.e., randomly choose one network/group if there are multiple eligible ones)

Coordination of external vs. goal vs. WM actions (choose one):

(1) random selection of one action type (default; if on, specify P_{EXT}, P_{GS}, P_{WM}; default: $P_{EXT} = 1$, $P_{GS} = 0$, $P_{WM} = 0$)

(2) perform all three action types

6.3 Options of the NACS

The following options and associated parameters are concerned with the non-action-centered subsystem:

Components of the NACS:
the GKS (on or off; default=on)
number of AMNs (specify; default=1)
the EM (on or off; default= off)
the AEM (on or off; default= off)

6.3.1 Options of GKS Representation

Details for setting up the GKS (specify each, if "GKS" is on):
dimensions and values of the GKS (not including the EM and the AEM) (specify) [1]
experience-specific chunks (on or off; default=on) [2]

Temperature for chunk selection in the GKS:
temperature for chunk selection (specify; default: $\tau_c = 0.1$)

Associative rule and chunk representation (each on or off):
associative rule base-level activation (default=on)
chunk base-level activation (default=on)
similarity (default=off)
(Note that rule support and chunk strength are always on)

Weights of associative rule condition chunks (specify, when "similarity" is on):

[1] The dimensions and values of the states and the actions seen by the NACS are normally the same as, or a subset of, what the ACS sees.
[2] Experience-specific chunks refer to chunks that are created due to specific experiences, such as chunks for experienced states, chunks for state and action pairs experienced, or chunks for action recommendations that are actually performed. The difference between experience-specific chunks and episodic chunks is that episodic chunks always have time stamps associated with them.

$\forall i$: W_i^a (the weight of the ith chunk in the condition of an associative rule) (default: $W_i^a = 1/n$, where n is the number of chunks in the condition of the rule)

Weights of dimensions in a chunk (specify, when "similarity" is on):

$\forall k$: V_k^c (the weight of the kth dimension specified in a chunk) (default: $V_k^c = 1$)

Associative rule base-level activation (choose one; default is 2):

(1) constant base-level activation (default: $B_j^r = 1$)

(2) recency-based base-level activation: default is

$$B_j^a = iB_j^a + c * \sum_{l=1}^{n} t_l^{-d}$$

where j indicates the jth rule at the top level, l indicates the lth encoding/use of that item, t is measured in ms (default: $c = 2, d = 0.5, iB_j^a = 0$)

Chunk base-level activation (choose one; default is 2):

(1) constant base-level activation (default: $B_i^c = 1$)

(2) recency-based base-level activation: default is

$$B_i^c = iB_i^c + c * \sum_{l=1}^{n} t_l^{-d}$$

where i indicates the ith chunk, l indicates the lth encoding/use of the chunk, t is measured in ms (default: $c = 2, d = 0.5, iB_i^c = 0$)

Base-level activation for the action-oriented EM (choose one, if "EM" is on; default is 2):

(1) fixed base-level activation (default: $B_i^e = 1$)

(2) recency-based base-level activation: default is

$$B_i^e = iB_i^e + c * \sum_{l=1}^{n} t_l^{-d}$$

where i indicates the ith item, l indicates the lth encoding of the item [3] (default: $c = 2, d = 0.5, iB_i^e = 0$)

Base-level activation for the non-action-oriented EM (choose one, if "EM" is on; default is 2):

(1) fixed base-level activation (default: $B_i^e = 1$)

(2) recency-based base-level activation: default is

$$B_i^e = iB_i^e + c * \sum_{l=1}^{n} t_l^{-d}$$

where i indicates the ith item, l indicates the lth encoding of the item [4] (default: $c = 2, d = 0.5, iB_i^e = 0$)

The EM threshold (specify, if "EM" is on):

the EM threshold (default: $threshold_{EM} = 0.0001$)

Use of the EM by the ACS (specify each, if "EM" is on):

probability of using the EM for off-line training of the ACS at each step (default= 0.1)

number of items selected for off-line training of the ACS at each step (default=5)

number of iterations of off-line training of the ACS at each step (default= 5)

Similarity-based reasoning (SBR):

the similarity measure (specify, if "similarity" is on; default: $S_{c1 \sim c2} = \frac{N_{c1 \cap c2}}{f(N_{c2})}$)

the similarity threshold $threshold_{sim}$ (specify, if "similarity" is on; default= 0.3)

6.3.2 Options for GKS Learning

Non-action-centered learning methods (on or off for each):

encoding associative rules dictated by the ACS (default = on)

[3] Here, storage is what matters, not retrieval as in the WM.

[4] Again, storage is what matters, not retrieval as in the WM.

encoding chunks dictated by the ACS (default = on)
associative rule extraction from the AMNs (default = off)
chunk extraction from the AMNs (default = off)

Parameters for explicit knowledge encoding (from external sources, such as the ACS) (specify each):
the chunk encoding probability p_c (default=1)
the associative rule encoding probability p_a (default=1)

Parameters of explicit knowledge extraction from the AMNs (specify each):
the extraction threshold for associative rules $threshold_{ae}$ (default=0.5)
the extraction threshold for chunks $threshold_{ce}$ (default=0.5)
the extraction probability for associative rules p_{ae} (default=1)
the extraction probability for chunks p_{ce} (default=1)

Density parameters (specify each):
the density parameter for associative rules d_a (default = 1/100)
the density parameter for NACS chunks d_c (default = 1/100)

Use of the GKS:
GKS reasoning method (forward-chaining, forward chaining with similarity-based reasoning, etc.) (choose one for GKS; default=forward-chaining; forward chaining with SBR may be selected only when a corresponding representation was selected)
number of iterations of reasoning (specify: 1, 2, ..., unlimited; default=1)
the reasoning threshold (specify; default: $threshold_r = 0.1$???)
(Note that these may be determined by the ACS on the fly)

6.3.3 Options for AMN Representation and Learning

Details for setting up the AMNs (specify for each AMN, if the number of AMNs is greater than 0):
i/o dimensions and values of each AMN

the number of hidden units for each network

Use of the AMNs (choose for each AMN):
one pass vs. multiple pass (default="one pass") (if "multiple pass", the exact number of iterations is determined by number of steps allowed for the GKS)

Parameters for AMN assimilation (specify for each AMN):
the probability of AMN training at each step (default=1)
the number of items from the EM to use for training this AMN at each step (default=5)
the type of items from the EM to use for training this AMN at each step (e.g., all associative rules applied, all associations given by the ACS, all associations given by the ACS for this particular AMN, all associations inferred by the GKS, auto-associations of all states experienced by the ACS, auto-associations of all state transitions experienced by the ACS (i.e., tuples of state, action, and new state), auto-associations of the associations given by the ACS, and so on; default = "all associations given for this AMN")
number of training iterations for the chosen training set at each step (default=5)

Parameters for BP learning in the AMNs (specify for each bottom-level network):
the BP learning rate (default: $\alpha = 0.1$)
the BP momentum (default: $mom = 0.1$)

Parameters for training the AEM (specify each, if "AEM" is on):
the maximum number of states that can be coded (default =50)
the maximum number of actions (each a set of action dimension-value pairs) that can be coded (default =50)
the range and increment for quantization of reinforcement (default: 0..1; 0.1)
the probability of training the AEM at each step (default=1)
the number of action-centered EM items to use for training the AEM (at each step) (default=5)

the number of times each selected item is presented to the AEM at each step (default=5)
the BP learning rate for training the AEM (default: $\alpha = 0.1$)
the BP momentum for training the AEM (default: $mom = 0.1$)

Parameters for using the AEM to train the ACS (specify each):
probability of using the AEM to train the ACS at each step (default=0.1)
number of AEM samples to use for training the ACS at each step (default=5)
the number of times each selected item is presented to the ACS for training the ACS (at each step) (default=5)
the temperature for AEM output selection (default: $\tau_{AEM} = 0.1$)

6.4 Options of the MS and the MCS

Components (each on or off):
the MS (default=off)
the MCS (default=off)

Drives in the MS (choose one):
(1) the default drive set
(2) a non-default drive set (if chosen, specify details)

Drive representation in the MS (choose one; default is 2):
(1) drive computation using the given set of equations (see the text) (deficits must be individually provided, among sensory input)
(2) drive computation using a pre-trained neural network (if chosen, specify details of the network)

MCS modules (specify on or off for each):
reinforcement function (default=on) (if on, specify details of the pre-trained reinforcement module)
goal setting based on drives (default=off) (if on, specify details of the pre-trained goal setting module)

input dimension selection

a. for the ACS (default=off) (if on, specify details of the pre-trained modules for each rule set/type in each rule group and for each bottom-level network)
b. for the NACS (default=off) (if on, specify details of the pre-trained modules for the GKS and for each AMN)

output dimension selection

a. for the ACS (default=off) (if on, specify details of the pre-trained modules for each rule set/type in each rule group and for each bottom-level network)
b. for the NACS (default=off) (if on, specify details of the pre-trained modules for the GKS and for each AMN)

input value selection (default=off)

a. for the ACS (default=off) (if on, specify details of the pre-trained modules for each rule set/type in each rule group and for each bottom-level network)
b. for the NACS (default=off) (if on, specify details of the pre-trained modules for the GKS and for each AMN)

output value selection (default=off)

a. for the ACS (default=off) (if on, specify details of the pre-trained modules for each rule set/type in each rule group and for each bottom-level network)
b. for the NACS (default=off) (if on, specify details of the pre-trained modules for the GKS and for each AMN)

selection of learning methods

a. for the ACS (default=off) (if on, specify details of the pre-trained learning method selection module for each network and each corresponding rule group of the ACS)

b. for the NACS (default=off) (if on, specify details of the pre-trained learning method selection module for each AMN and for the GKS)

selection of reasoning method

a. for the ACS top level (default=off) (if on, specify details of the pre-trained selection module for each rule set/type in each rule group of the ACS)
b. for the NACS top level (default=off) (if on, specify details of the pre-trained selection module for the top level of the NACS)

cross-level integration

a. for the ACS (default=off) (if on, specify details of the pre-trained integration module)

setting of parameters

a. for the ACS (default=off) (if on, specify details of the pre-trained parameter setting module)
b. for the NACS (default=off) (if on, specify details of the pre-trained parameter setting module)

Regulation of cross-level integration in the ACS (choose one):
(1) through P parameters directly (default)
(2) through β parameters indirectly

Timing of goal setting (choose one; default is 2):
(1) drive threshold (if on, specify $threshold_{d1}$)
(2) drive difference threshold (if on, specify $threshold_{d2}$)

Measures of performance (for the performance section of the monitoring buffer; specify each):
performance measure of the ACS (default: rs)
performance measure of the NACS (default: rs)

Numbers of conclusions monitored (in the monitoring buffer; specify each):

number of top conclusions monitored in the top level of the ACS (default: 1)
number of top conclusions monitored in the top level of the NACS (default: 1)
number of top conclusions monitored in the bottom level of the ACS (default: 1)
number of top conclusions monitored in the bottom level of the NACS (default: 1)

Measures for gauging learning (for the learning section of the monitoring buffer; specify each):
the measure of the ACS (default: *sr*)
the measure of the NACS (default: number of chunks inferred)

Numbers of episodes monitored (in the monitoring buffer):
number of previous episodes monitored in the top level of the ACS (default: 2)
number of previous episodes monitored in the top level of the NACS (default: 2)
number of previous episodes monitored in the bottom level of the ACS (default: 2)
number of previous episodes monitored in the bottom level of the NACS (default: 2)

Definition of episodes (choose one):
(1) by number of steps (default; if on, specify the number of steps)
(2) by the termination of one iteration of the task (signaled by the task simulator)

Cross-level integration in the MCS (specify both for each module):
P_{TL} (default=0).
P_{BL} (default=1).

6.5 Options in Response Time Determination

In addition to the options and parameters discussed so far, there are the response time options and their associated parameters.

Overall response time:
specify f in $RT = f(RT_{BL}, RT_{TL})$ (default: as described in the chapter on RTs)

Response times of the bottom level of the ACS:
specify f_{BL} in $RT_{BL} = f_{BL}(PT_{BL}, DT_{BL}, AT_{BL})$ (default: $RT_{BL} = PT_{BL} + DT_{BL} + AT_{BL}$)

Response time of the top level of the ACS:
specify f_{TL} in $RT_{TL} = f_{TL}(PT_{TL}, DT_{TL}, AT_{TL})$ (default: $RT_{TL} = PT_{TL} + DT_{TL} + AT_{TL}$)

RT Parameters for the ACS (specified with respect to each IDN and each rule group/set, as well as with respect to some other entities in some cases):
PT_{BL} (default= 200 ms)
DT_{BL} (default= 350 ms)
AT_{BL} (default=500 ms; must be specified with respect to a particular action output)
PT_{TL} (default: PT_{BL} + 100 ms; may be specified individually with respect to individual FRs or individual IRL subsets))
DT_{TL} (default: operation-time+$t_0 \times 1$/rule-bla+$t_1 \times 1$/chunk-bla, where t_0, t_1, and *operation-time* must be specified; default: ???)
AT_{TL} (default: 500 ms; must be specified with respect to a particular action output; may also be specified individually with respect to individual FRs or individual IRL subsets)
(Note that these parameters may be specified as functions rather than constants, e.g., for the sake of dealing with the speed/accuracy trade-off)

Associative rule application time in the GKS (specify, for the application of one associative rule)
associative rule application time t_a (default: $t_a = t_4 + t_5 \times 1$/rule-bla, where t_4 and t_5 need to be specified; default: ???)

Chunk retrieval time in the GKS (specify, for the retrieval of one chunk)

chunk retrieval time t_c (default: $t_c = t_2 + t_3 \times 1/\text{chunk-bla}$, where t_2 and t_3 need to be specified; default: ???)

one-pass mapping time for the AMNs (specify):
the one-pass time for the AMNs t_{AMN} (default: $t_{AMN} = 350ms$, for each iteration of mapping input to output)

Response time of the MS/MCS: The response time of the MS and the MCS is determined in the same way as the ACS: The same equations as those used in the ACS may be used, although parameters and options different from those of the ACS may be adopted. In addition, parameters and options may be set differently for the MS and for the MCS.

References

M. Ahlum-Heath and F. DiVesta, (1986). The effect of conscious controlled verbalization of a cognitive strategy on transfer in problem solving. *Memory and Cognition.* 14, 281-285.

E. Altmann and J. Trafton, (2002). Memory for goals: An activation-based model. *Cognitive Science*, 26, 39-83.

J. R. Anderson, (1983). *The Architecture of Cognition.* Harvard University Press, Cambridge, MA

J. R. Anderson, (1993). *Rules of the Mind.* Lawrence Erlbaum Associates. Hillsdale, NJ.

J. Anderson and C. Lebiere, (1998). *The Atomic Components of Thought*, Lawrence Erlbaum Associates, Mahwah, NJ.

M. Arbib, (1980). *From Schema Theory to Language.* Oxford University Press. New York.

A. Baddeley, (1986). *Working Memory.* Oxford University Press, New York.

G. Baerends, (1976). The functional organization of behavior. *Animal Behavior*, 24, 726-735.

A. Barto and R. Sutton, (1998). *Reinforcement Learning.* MIT Press, Cambridge, MA.

H. Ben-Zur, (1998). Dimensions and patterns in decision-making models and the controlled/automatic distinction in human information processing. *The European Journal of Cognitive Psychology*, 10 (2), 171-189.

D. Berry, (1983). Metacognitive experience and transfer of logical reasoning. *Quarterly Journal of Experimental Psychology*, 35A, 39-49.

D. Berry, (1991). The role of action in implicit learning. *Quarterly Journal of Experimental Psychology*, 43A, 881-906.

D. Berry and D. Broadbent, (1984). On the relationship between task performance and associated verbalizable knowledge. *Quarterly Journal of Experimental Psychology.* 36A, 209-231.

D. Berry and D. Broadbent, (1988). Interactive tasks and the implicit-explicit distinction. *British Journal of Psychology.* 79, 251-272.

D. Bertsekas and J. Tsitsiklis, (1996). *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA.

K. Bowers, G. Regehr, C. Balthazard, and Parker, (1990). Intuition in the context of discovery. *Cognitive Psychology.* 22. 72-110.

M. Braine and D. O'Brien, (eds.) (1998). *Mental Logic.* Lawrence Erlbaum Associates, Mahwah, NJ.

D. Broadbent, P. Fitsgerald, and M. Broadbent, (1986). Implicit and explicit knowledge in the control of complex systems. *British Journal of Psychology.* 77, 33-50.

J. Bruner, J. Goodnow, and J. Austin, (1956). *A Study of Thinking.* Wiley, New York.

J. Busemeyer and I. Myung, (1992). An adaptive approach to human decision making: Learning theory, decision theory, and human performance. *Journal of Experimental Psychology: General*, 121 (2), 177-194.

M. Byrne and J. Anderson, (1998). Perception and action. In: J. Anderson and C. Lebiere (eds.), *The Atomic Components of Thought*, pp.167-200. Lawrence Erlbaum Associates, Mahwah, NJ.

S. Chaiken and Y. Trope (eds.), (1999). *Dual Process Theories in Social Psychology.* Guilford Press, New York.

M. Chi, M. Bassok, and M. Lewis, P. Reimann, and P. Glaser, (1989). Self-explanation: How students study and use examples in learning to solve problems. *Cognitive Science.* 13. 145-182.

A. Clark and A. Karmiloff-Smith, (1993). The cognizer's innards: A psychological and philosophical perspective on the development of thought. *Mind and Language.* 8 (4), 487-519.

A. Cleeremans, (1997). Principles for implicit learning. In D. Berry (Ed.),

How implicit is implicit learning? pp. 195-234. Oxford University Press, Oxford, UK.

A. Cleeremans and J. McClelland, (1991). Learning the structure of event sequences. *Journal of Experimental Psychology: General.* 120, 235-253.

A. Cleeremans, A. Destrebecqz and M. Boyer, (1998). Implicit learning: News from the front. *Trends in Cognitive Sciences*, Volume 2, Issue 10, 406-416.

A. Cohen, R. Ivry, and S. Keele, (1990). Attention and structure in sequence learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition.* 16, 17-30.

A. Collins and R. Michalski, (1989). The logic of plausible reasoning. *Cognitive Science*, 13(1), 1-49.

N. Cowan, (1993). Activation, attention, and short-term memory. *Memory and Cognition*, 21 (2), 162-167.

J. Dewey, (1958). *Experience and Nature.* Dover, New York.

Z. Dienes, (1992). Connectionist and memory-array models of artificial grammar learning. *Cognitive Science.* 16. 41-79.

Z. Dienes and R. Fahey, (1995). The role of specific instances in controlling a dynamic system. *Journal of Experimental Psychology: Learning, Memory and Cognition.* 21, 848-862.

Z. Dienes and J. Perner, (1999). A theory of implicit and explicit knowledge. *Behavioral and Brain Sciences*, 22, 735-808.

P. Domingos, (1996). Unifying instance-based and rule-based induction. *Machine Learning*, 24, 141-168.

R. Dominowski, (1972). How do people discover concepts? In: R. L. Solso (Ed.), *Theories in Cognitive Psychology: The Loyola Symposium*, 257-288. Lawrence Erlbaum, Potomac, MD.

H. Dreyfus and S. Dreyfus, (1987). *Mind Over Machine: The Power of Human Intuition.* The Free Press, New York.

H. Dreyfus, (1992). *Being-In-The-World.* MIT Press, Cambridge, MA.

D. Dulaney, R. Carlson, and G. Dewey, (1984). A case of syntactic learing and judgment: How conscious and how abstract. *Journal of Experimental Psychology: General.* 113, 541-555.

J. Dewey, (1958). *Experience and Nature.* Dover, New York.

J. Elman, (1990). Finding structures in time. *Cognitive Science.* 14, 179-211.

A. Epstein, (1982). Instinct and motivation as explanations for complex behavior. In: D. W. Pfaff (ed.), *The Physiological Mechanisms of Motivation.* SpringerVerlag.

M. Erickson and J. Kruschke, (1998). Rules and exemplars in category learning. *Journal of Experimental Psychology: General.* 127, 107-140.

K. Ericssoon and W. Kintsch, (1995). Long-term working memory. *Psychological Review*, 102, 211-245.

J. Fodor, (1983). *The Modularity of Mind.* MIT Press, Cambridge, MA.

J. Fodor and Z. Pylyshyn, (1988). Connectionism and cognitive architecture: A critical analysis. In: Pinker and Mehler (eds.), *Connections and Symbols.* MIT Press, Cambridge, MA.

R. Gagne and E. Smith, (1962). A study of the effects of verbalization on problem solving. *Journal of Experimental Psychology*, 63, 12-18.

E. Gat, (1998). On three-layered architecture. In: D. Kortenkamp, R. Bonasso, and R. Murphy (eds.), *Artificial Intelligence and Mobile Robots.* AAAI Press, Menlo Park, Calif.

J. Gelfand, D. Handelman and S. Lane, (1989). Integrating knowledge-based systems and neural networks for robotic skill acquisition, *Proc.IJCAI*, pp.193-198. Morgan Kaufmann, San Mateo, CA.

J. Gibson, (1979). *The Ecological Approach to Visual Perception.* Houghton Mifflin, Boston.

M. Gluck and G. Bower, (1988). From conditioning to category learning. *Journal of Experimental Psychology: General.* 117 (3), 227-247.

R. Hadley, (1995). The explicit-implicit distinction. *Minds and Machines.* 5, 219-242.

U. Hahn and N. Chater, (1998). Similarity and rules: distinct? exhaustive? empirically distinguishable? *Cognition*, 65, 197-230.

J. Hasher and J. Zacks, (1979). Automatic and effortful processes in memory. *Journal of Experimental Psychology: General.* 108. 356-358.

N. Hayes and D. Broadbent, (1988). Two modes of learning for interactive tasks. *Cognition*, 28 (3), 249-276.

R. Haygood and L. Bourne, (1965). Attribute and rule learning aspects of conceptual behavior. *Psychological Review*, 72 (3), 175-195.

M. Heidegger, (1927). *Being and Time.* English translation published by Harper and Row, New York. 1962.

D. Hintzman, (1986). Schema abstraction in a multiple-trace memory model. *Psychological Review.* 93, 528-551.

D. Hintzman, (1990). Human learning and memory: Connections and dissociations. *Annual Review of Psychology*, Vol.41, 109-139. Annual Review Inc.

C. Hull, (1943). *Principles of Behavior: An Introduction to Behavior Theory.* D. Appleton-Century Company.

C. Hull, (1951). *Essentials of Behavior.* Yale University Press, New Haven, CT.

M. Humphreys, J. Bain, and R. Pike, (1989), Different ways to cue a coherent memory system: a theory for episodic, semantic, and procedural tasks. *Psychological Review*, Vol.96, No.2, 208-233

E. Hunt and M. Lansman, (1986). Unified model of attention and problem solving. *Psychological Review.* 93 (4), 446-461.

E. Husserl, (1970). *Logical Investigation.* Routledge and Kegan Paul, London.

R. Jackendoff, (2002), *Foundations of Language.* ????

W. James, (1890). *The Principles of Psychology.* Dover, New York.

A. Karmiloff-Smith, (1986). From meta-processes to conscious access: Evidence from children's metalinguistic and repair data. *Cognition.* 23. 95-147.

F. Keil, (1989). *Concepts, Kinds, and Cognitive Development.* MIT Press, Cambridge, MA.

J. Kruschke, (1992). ALCOVE: An examples-based connectionist model of category learning. *Psychological Review.* 99, 22-44.

M. Kushner, A. Cleeremans, and A. Reber, (1992). Implicit detection

of event interdependencies and a PDP model of the process. *Proc. of Cognitive Science Society Annual Conference*, 215-220.

N. Lavrac and S. Dzeroski, (1994). *Inductive Logic Programming.* Ellis Horword, New York.

P. Lewicki, (1986). Processing information about covariations that cannot be articulated. *Journal of Experimental Psychology: Learning, Memory, and Cognition.* 12, 135-146.

J. Lewis, (1970). Semantic processing of unattended messages using dichotic listening. *Journal of Experimental Psychology.* 85, 220-227.

B. Libet, (1985). Unconscious cerebral initiative and the role of conscious will in voluntary action. *Behavioral and Brain Sciences.* 8, 529-566.

G. Logan, (1988). Toward an instance theory of automatization. *Psychological Review.* 95 (4), 492-527.

K. Lorenz, (1950). The comparative method in studying innate behavior patterns. *Symp. Soc. Exp. Biol.*, 4, 221-268.

M. Lovett, L. Daily, and L. Reder, (2000). A source activation theory of working memory: cross-task prediction of performance in ACT-R. *Cognitive Systems Research*, 1, 2, 99-118.

R. D. Luce, (1959). *Individual Choice Behavior: A Theoretical Analysis.* Wiley, New York.

R. D. Luce, (2000). *The Utility of Gains and Losses.* Lawrence Erlbaum Associates, Mahwah, NJ.

J. Mandler, (1992). How to build a baby. *Psychological Review.* 99, 4. 587-604.

A. Maslow, (1962). *Toward a Psychology of Being.* Van Nostrand, Princeton, NJ.

A. Maslow, (1987). *Motivation and Personality.* 3rd Edition. Harper and Row, New York.

M. Mataric, (2001). Learning in behavior-based multi-robot systems: Policies, models, and new behaviors. *Cognitive Systems Research*, special issue on multi-agent learning (ed. R. Sun). Vol.2, No.1, ???.

R. Mathews, R. Buss, W. Stanley, F. Blanchard-Fields, J. Cho, and B. Druhan, (1989). Role of implicit and explicit processes in learning

from examples: A synergistic effect. *Journal of Experimental Psychology: Learning, Memory and Cognition.* 15, 1083-1100.

G. Mazzoni and T. Nelson (eds.), (1998). *Metacognition and Cognitive Neuropsychology.* Erlbaum, Mahwah, NJ.

J. McClelland, B. McNaughton and R. O'Reilly, (1995). Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102 (3), 419-457.

D. McFarland, (1989). *Problems of Animal Behaviour.* Longman Publishing, Singapore.

M. Merleau-Ponty, (1963). *The Structure of Behavior.* Beacon Press, Boston.

J. Metcalfe and A. Shimamura (eds.), (1994). *Metacogntion: Knowing about Knowing.* MIT Press, Cambridge, MA.

D. Meyer and D. Kieras, (1997). A computational theory of executive cognitive processes and human multiple-task performance: Part 1, basic mechanisms. *Psychological Review.* 104 (1), 3-65.

W. McDougall, (1932). *The Energies of Man.* London: Methuen.

R. Michalski, (1983). A theory and methodology of inductive learning. *Artificial Intelligence.* Vol.20, pp.111-161.

M. Minsky, (1981). A framework for representing knowledge. In: J. Haugeland (ed.), *Mind Design*, 95-128. MIT Press, Cambridge, MA.

M. Mishkin, B. Malamut, and J. Bachevalier, (1984). Memories and habits: Two neural systems. In: *Neurobiology of Human Learning and Memory.* Guilford Press, New York.

M. Moscovitch and C. Umilta, (1991). Conscious and unconscious aspects of memory. In: *Perspectives on Cognitive Neuroscience.* Oxford University Press, New York.

G. Murphy and D. Medin, (1985). The role of theories in conceptual coherence. *Psychological Review.* 92, 289-316.

A. Newell, (1990). *Unified Theories of Cognition.* Harvard University Press, Cambridge, MA.

R. Nisbett and T. Wilson, (1977). Telling more than we can know: Verbal reports on mental processes. *Psychological Review.* 84 (3), 1977.

M. Nissen and P. Bullemer, (1987). Attentional requirements of learning: Evidence from performance measures. *Cognitive Psychology.* 19, 1-32.

R. Nosofsky, T. Palmeri, and S. McKinley, (1994). Rule-plus-exception model of classification learning. *Psychological Review.* 101 (1), 53-79.

J. Pearl, (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufman, San Mateo, CA.

R. W. Pew and A. S. Mavor (eds), (1998). *Modeling Human and Organizational Behavior: Application to Military Simulations.* National Academy Press, Washington, DC.

J. Piaget, (1971). *Biology and Knowledge.* Edinburgh University Press, Edinburgh, UK.

S. Pinker, (1994). *The Language Instinct.* W. Morrow and Co., New York.

M. Posner, G. DiGirolamo, and D. Fernandez-Duque, (1997). Brain mechanisms of cognitive skills. *Consciousness and Cognition.* 6, 267-290.

D. Premack, (1988). Minds with and wihtout language. In: L. Weiskrantz (ed.), *Thought without Language.* Clarendon Press, Oxford, UK.

R. Proctor and A. Dutta, (1995). *Skill Acquisition and Human Performance.* Sage Publications, Thousand Oaks, CA.

M. R. Quillian, (1968). Semantic memory. In: M. Minsky (ed.), *Semantic Information Processing.* MIT Press, Cambridge, MA. pp.227-270.

M. Rabinowitz and N. Goldberg, (1995). Evaluating the structure-process hypothesis. In: F. Weinert and W. Schneider (eds.), *Memory Performance and Competencies.* Lawrence Erlbaum, Hillsdale, NJ.

A. Reber, (1989). Implicit learning and tacit knowledge. *Journal of Experimental Psychology: General.* 118 (3), 219-235.

R. Rescorla and A. Wagner, (1972). A theory of Pavlovian conditioning. In: A. Black and W. Prokasy (eds.), *Classical Conditioning II: Current Research and Theory*, 64-99. Appleton-Century-Crofts, New York.

S. Roberts and H. Pashler, (2000). How persuasive is a good fit? A comment on theory testing. *Psychological review*, 107 (2), 358-367.

P. Rosenbloom, J. Laird, and A. Newell, (1993). *The SOAR Papers: Research on Integrated Intelligence.* MIT Press, Cambridge, MA.

D. Rumelhart, J. McClelland and the PDP Research Group, (1986). *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, MIT Press, Cambridge, MA.

T. Savage, (2003). The grounding of motivational constructs in artificial animals: Indices of motivational behavior. *Cognitive Systems Research.* 4, 2.

D. Schacter, (1987). Implicit memory: History and current status. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 13, 501-518.

D. Schacter, (1990). Toward a cognitive neuropsychology of awareness: Implicit knowledge and anosagnosia. *Journal of Clinical and Experimental Neuropsychology.* 12 (1), 155-178.

M. Scheutz and A. Sloman, (2001). Affect and agent control: experiments with simple affective states. *IAT'01*, World Scientific, Singapore.

W. Schneider and W. Oliver (1991), An instructable connectionist/control architecture. In: K. VanLehn (ed.), *Architectures for Intelligence*, Erlbaum, Hillsdale, NJ.

J. Schooler, S. Ohlsson, and K. Brooks, (1993). Thoughts beyond words: When language overshadows insight. *Journal of Experimental Psychology: General*, 122 (2), 166-183.

J. Schraagen, (1993). How experts solve a novel problem in experimental design. *Cognitive Science.* 17, 285-309.

C. Schunn and D. Wallach, (2001). In defence of goodness-of-fit in comparison of models to data. Manuscript.

C. Seger, (1994). Implicit learning. *Psychological Bulletin.* 115 (2), 163-196.

M. Seidenburg and J. McClelland, (1989). A distributed, developmental model of work recognition and naming. *Psychological Review*, 96 (4), 523-568.

T. Shallice, (1972). Dual functions of consciousness. *Psychological Review.* 79 (5), 383-393.

D. Shanks, and M. St.John, (1994). Characteristics of dissociable learning systems. *Behavioral and Brain Sciences*, 17, 367-394.

R. Shiffrin and W. Schneider, (1977). Controlled and automatic human information processing II. *Psychological Review*. 84. 127-190.

R. Siegler and E. Stern, (1998). Conscious and unconscious strategy discovery: A microgenetic analysis. *Journal of Experimental Psychology: General*, 127 (4), 377-397.

H. Simon, (1967). Motivational and emotional controls of cognition. *Psychological Review*, 74, 29-39.

A. Sloman, (1986). Motives mechanisms and emotions. Cognitive science research papers, CSRP 062. University of Sussex, Falmer, UK.

E. Smith and J. DeCoster, (2000). Dual process models in social and cognitive psychology: Conceptual integration and links to underlying memory systems. *Personality and Social Psychology Review*, 4 (2), 108-131.

M. Smithson and G. C. Oden, (1999). Fuzzy set theory and applications in psychology. In: H. Zimmermann (ed.), *Practical Applications of Fuzzy Technologies* (Handbooks of Fuzzy Sets Series). Kluwer Academic Publishers. Dordrecht, The Netherlands.

P. Smolensky, (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11 (1), 1-74.

M. Stadler and P. Frensch, (1998). *Handbook of Implicit Learning*. Sage Publication, Thousand Oaks, CA.

W. Stanley, R. Mathews, R. Buss, and S. Kotler-Cope, (1989). Insight without awareness: On the interaction of verbalization, instruction and practice in a simulated process control task. *Quarterly Journal of Experimental Psychology*. 41A (3), 553-577.

R. Sun, (1992). On variable binding in connectionist networks, *Connection Science*, Vol.4, No.2, pp.93-124.

R. Sun, (1993). An efficient feature-based connectionist inheritance scheme. *IEEE Transactions on System, Man and Cybernetics*, Vol.23, No.2. pp.512-522.

R. Sun, (1994). *Integrating Rules and Connectionism for Robust*

Commonsense Reasoning. John Wiley and Sons, New York, NY.

R. Sun, (1995). Robust reasoning: Integrating rule-based and similarity-based reasoning. *Artificial Intelligence.* 75, 2. 241-296.

R. Sun, (1997). Learning, action, and consciousness: A hybrid approach towards modeling consciousness. *Neural Networks*, special issue on consciousness. 10 (7), pp.1317-1331.

R. Sun, (1999). Accounting for the computational basis of consciousness: A connectionist approach. *Consciousness and Cognition*, Vol.8, 529-565.

R. Sun, (2000). Symbol grounding: A new look at an old issue. *Philosophical Psychology*, Vol.13, No.3, 403-418.

R. Sun, (2002). *Duality of the Mind.* Lawrence Erlbaum Associates, Mahwah, NJ.

R. Sun and L. Bookman (eds.), (1994). *Computational Architectures Integrating Neural and Symbolic Processes.* Kluwer Academic Publishers, Norwell, MA.

R. Sun and C. Sessions, (2000). Self-segmentation of sequences: Automatic formation of hierarchies of sequential behaviors. *IEEE Transactions on Systems, Man, and Cybernetics: Part B, Cybernetics*, Vol.30, No.3, pp.403-418.

R. Sun and C. Sessions, (2000 b). Learning plans without a priori knowledge. *Adaptive Behavior*, Vol.8, No.3/4, 225-253.

R. Sun and T. Peterson, (1998). Autonomous learning of sequential tasks: Experiments and analyses. *IEEE Transactions on Neural Networks*, Vol.9, No.6, 1217-1234.

R. Sun, E. Merrill, and T. Peterson, (1998). A bottom-up model of skill learning. *Proceedings of 20th Cognitive Science Society Conference*, 1037-1042, Lawrence Erlbaum Associates, Mahwah, NJ.

R. Sun, E. Merrill, and T. Peterson, (2001). From implicit skills to explicit knowledge: A bottom-up model of skill learning. *Cognitive Science.* Vol.25, No.2, 203-244.

W. Timberlake and G. Lucas, (1989). Behavior systems and learning: From misbehavior to general principles. In: S. B. Klein and R. R. Mowrer (Eds.), *Contemporary Learning Theories: Instrumental Conditioning Theory and*

the Impact of Biological Constraints on Learning, 237-275. Erlbaum, Hillsdale, NJ.

N. Tinbergen, (1951). *The Study of Instinct.* Oxford University Press, London.

F. Toates, (1986). *Motivational Systems.* Cambridge University Press, Cambridge, UK.

E. Tulving, (1972). Episodic and semantic memory. In: E. Tulving and W. Donaldson (eds.), *Organization of Memory*, 381-403. Academic Press, New York.

E. Tulving, (1983). *Elements of Episodic Memory.* Clarendon Press, Oxford.

E. Tulving, (1985). How many meomry systems are there? *American Psychologist*, 40, 385-398.

A. Tversky, (1977). Features of similarity. *Psychological Review*, 84(4), 327-352.

A. Tversky and D. Kahneman, (1983). Extensional versus intuitive reasoning: The conjunction fallacy in probability judgment. *Psychological Review*, 439-450.

T. Tyrell, (1993). *Computational Mechanisms for Action Selection.* Ph.D Thesis, Oxford University, Oxford, UK.

A. Vandierendonck, (1995). A parallel rule activation and rule synthesis model for generalization in category learning. *Psychonomic Bulletin and Review.* 2 (4), 442-459.

K. VanLehn, (1995). Cognitive skill acquisition. *Annual Review of Psychology*, Vol.47. Annual Reviews Inc, Palo Alto, CA.

M. Velmans, (1991). Is human information processing conscious? *Behavioral and Brain Sciences.* 14, 651-726.

L. Vygotsky, (1962). *Thought and Language.* MIT Press, Cambridge, MA.

C. Watkins, (1989). *Learning with Delayed Rewards.* Ph.D Thesis, Cambridge University, Cambridge, UK.

B. Weiner, (1992). *Human Motivation: Metaphors, Theories, and Research.* Sage, Newbury Park, CA.

D. Willingham, M. Nissen, and P. Bullemer, (1989). On the development of procedural knowledge. *Journal of Experimental Psychology: Learning, Memory, and Cognition.* 15, 1047-1060.

E. Wisniewski and D. Medin, (1994). On the interaction of data and theory in concept learning. *Cognitive Science.* Vol.18, 221-281.

L. Zadeh, (1988). Fuzzy Logic. *Computer*, Vol.21, No.4, 83-93