12 Cognitive architectures and agents

Sebastien Helie, Department of Psychological Sciences, Purdue University

Ron Sun, Cognitive Science Department, Rensselaer Polytechnic Institute

Abstract

A cognitive architecture is the essential structures and processes of a domain-generic computational cognitive model used for a broad, multiple-level, multiple-domain, analysis of cognition and behavior. This chapter reviews some of the most popular psychologically-oriented cognitive architectures, namely ACT-R, Soar, and CLARION. For each cognitive architecture, an overview of the model, some key equations, and a detailed simulation example are presented. The example simulation with ACT-R is the initial learning of the past-tense of irregular verbs in English (developmental psychology), the example simulation with Soar is the well-known Missionaries & Cannibals problem (problem solving), and the example simulation with CLARION is a complex mine field navigation task (autonomous learning). This presentation is followed by a discussion of how cognitive architectures can be used in multi-agent social simulations. A detailed cognitive social simulation is presented with CLARION to reproduce results from organizational decision-making. The chapter concludes with a discussion of the impact of neural network modeling on cognitive architectures and a comparison of the different models.

Keywords: Cognitive Architectures, Psychology, ACT-R, CLARION, Soar, Cognitive Social Simulation.

12.1 Introduction

Cognitive theories are often underdetermined by data [12.1]. As such, different theories, with very little in common, can sometimes be used to account for the very same phenomena [12.2]. This problem can be resolved by adding constraints to cognitive theories. The most intuitive approach to adding constraints to any scientific theory is to collect more data. While experimental psychologists have come a long way toward this goal in over a century of psychology research, the gap between empirical and theoretical progress is still significant.

Another tactic that can be adopted toward constraining psychological theories is *unification* [12.1]. Newell argued that more data could be used to constraint a theory if the theory was designed to explain a wider range of phenomena. In particular, these 'unified' (i.e., integrative) cognitive theories could be put to the test against well-known (stable) regularities that have been observed in psychology. So far, these integrative theories have taken the form of *cognitive architectures*, and some of them have been very successful in explaining a wide range of data.

A cognitive architecture is the essential structures and processes of a domain-generic computational cognitive model used for a broad, multiple-level, multiple-domain, analysis of cognition and behavior [12.3]. Specifically, cognitive architectures deal with componential processes of cognition in a structurally and mechanistically well-defined way. Its function is to provide an essential framework to facilitate more detailed exploration and understanding of various components and processes of the mind. In this way, a cognitive architecture serves as an initial set of assumptions to be used for further development. These assumptions may be based on available empirical data (e.g., psychological or biological), philosophical thoughts and arguments, or computationally inspired hypotheses concerning psychological processes. A cognitive architecture is useful and important precisely because it provides a comprehensive initial framework for further modeling and simulation in many task domains.

While there are all kinds of cognitive architectures in existence, in this chapter we are concerned specifically with psychologically-oriented cognitive architectures (as opposed to software engineering-oriented cognitive architectures, e.g., LIDA [12.4], or neurally-oriented cognitive architectures, e.g., ART [12.5]). Psychologically-oriented cognitive architectures are particularly important because they shed new light on human cognition and therefore they are useful tools for advancing the understanding of cognition. In understanding cognitive phenomena, the use of computational simulation on the basis of cognitive architectures forces one to think in terms of processes, and in terms of details. Instead of using vague, purely conceptual theories, cognitive architectures force theoreticians to think clearly. They are therefore critical tools in the study of the mind. Cognitive psychologists who use cognitive architectures must specify a cognitive mechanism in sufficient detail to allow the resulting models to be implemented on computers and run as simulations. This approach requires that important elements of the models be spelled out explicitly, thus aiding in developing better, conceptually clearer theories. It is certainly true that more specialized, narrowly-scoped models may also serve this purpose, but they are not as generic and as comprehensive and thus may not be as useful to the goal of producing general intelligence.

It is also worth noting that psychologically-oriented cognitive architectures are the antithesis of expert systems: Instead of focusing on capturing performance in narrow domains, they are aimed to provide broad coverage of a wide variety of domains in a way that mimics human performance [12.6]. While they may not always perform as well as expert systems, business and industrial applications of intelligent systems increasingly require broadly-scoped systems that are capable of a wide range of intelligent behaviors, not just isolated systems of narrow functionalities. For example, one application may require the inclusion of capabilities for raw image processing, pattern recognition, categorization, reasoning, decision-making, and natural language communications. It may even require planning, control of robotic devices, and interactions with other systems and devices. Such requirements accentuate the importance of research on broadly-scoped cognitive architectures that perform a wide range of cognitive functionalities across a variety of task domains (as opposed to more specialized systems).

In order to achieve general computational intelligence in a psychologically-realistic way, cognitive architectures should include only minimal initial structures and independently learn from their own experiences. Autonomous learning is an important way of developing additional structure, bootstrapping all the way to a full-fledged cognitive model. In so doing, it is important to be careful to devise only minimal initial learning capabilities that are capable of "bootstrapping", in accordance with whatever phenomenon is modeled (e.g., [12.7]). This can be accomplished through environmental cues, structures, and regularities. The avoidance of overly complicated initial structures, and thus the inevitable use of autonomous learning, may often help to avoid overly representational models that are designed specifically for the task to be achieved [12.3]. Autonomous learning is thus essential in achieving generality in a psychologically-realistic way.

12.1.1 Outline

The remainder of this chapter is organized as follows. The next three sections review some of the most popular cognitive architectures that are used in psychology and cognitive science. Specifically, Section 12.2 reviews ACT-R, Section 12.3 reviews Soar, and Section 12.4 reviews CLARION. Each of these sections includes an overview of the architecture, some key equations, and a detailed simulation example. Following this presentation, Section 12.5 discusses how cognitive architectures can be used in multi-agent cognitive social simulations and presents a detailed example with CLARION. Finally, Section 12.6 presents a general discussion and compares the reviewed models.

12.2 Adaptive Control of Thought-Rational (ACT-R)

ACT-R is one of the oldest and most successful cognitive architectures. It has been used to simulate and implement many cognitive tasks and applications, such as the Tower of Hanoi, game playing, aircraft control, and human-computer interactions [12.8]. ACT-R is based on three key ideas [12.9]: (a) rational analysis, (b) the distinction between procedural and declarative memories and, (c) a modular structure linked with communication buffers (see Fig. 12.1). According to the rational analysis of cognition (first key idea; [12.10]), the cognitive architecture is optimally tuned to its environment (within its computational limits). Hence, the functioning of the architecture can be understood by investigating how optimal behavior in a particular environment would be implemented. According to Anderson, such optimal adaptation is achieved through evolution [12.10].

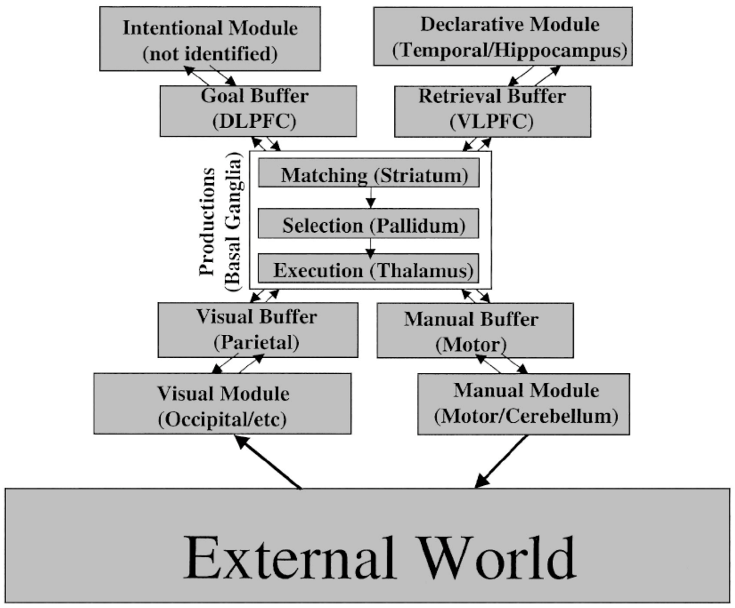

Fig. 12.1. General architecture of ACT-R. DLPFC = Dorsolateral prefrontal cortex; VLPFC = Ventrolateral prefrontal cortex. Reprinted from "An Integrated Theory of the Mind," by J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin, 2004, *Psychological Review*, *111*, p. 1037, Copyright 2004 by the American Psychological Association.

The second key idea, the distinction between declarative and procedural memories, is implemented by having different modules in ACT-R, each with its own representational format and learning rule. Briefly, procedural memory is represented by production rules (similar to a production system) that can act on the environment through the initiation of motor actions. In contrast, declarative memory is passive and uses chunks to represent world knowledge that can be accessed by the procedural memory but does not interact directly with the environment through motor actions.

The last key idea in ACT-R is modularity. As seen in Fig. 12.1, procedural memory (i.e., the production system) cannot directly access information from the other modules: the information has to go through dedicated buffers. Each buffer can hold a single chunk at any time. Hence, buffers serve as information processing ‘bottlenecks’ in ACT-R. This restricts the amount of information available to the production system, which in turn limits the processing that can be done by this module at any given time. Processing within each module is encapsulated. Hence, all the modules can operate in parallel without much interference. The following subsections describe the different ACT-R modules in more details.

12.2.1 The perceptual-motor modules

The Perceptual-Motor modules in ACT-R include a detailed representation of the output of perceptual systems, and the input of motor systems [12.8]. The visual module is divided into the well-established ventral (what) and dorsal (where) visual streams in the primate brain. The main function of the dorsal stream is to find the location of features in a display (e.g., red colored items, curved shaped objects) without identifying the objects. The output from this module is a location chunk, which can be sent back to the central production system. The function of the ventral stream is to identify the object at a particular location. For instance, the central production system could send a request to the dorsal stream to find a red object in a display. The dorsal stream would search the display and return a chunk representing the location of a red object. If the central production system needs to know the identity of that object, the location chunk would be sent to the ventral stream. A chunk containing the object identity (e.g., a fire engine) would then be returned to the production system.

12.2.2 The goal module

The goal module serves as the context for keeping track of cognitive operations and supplement environmental stimulations [12.8]. For instance, one can do many different operations with a pen picked up from a desk (e.g., write a note, store in a drawer, etc.). What operation is selected depends primarily on the goal that needs to be achieved. If the current goal is to clean the desk, the appropriate action is to store the pen in a drawer. If the current goal is to write a note, putting the pen in a drawer is not a useful action.

In addition to providing a mental context to select appropriate production rules, the goal module can be used in more complex problem solving tasks that need subgoaling [12.8]. For instance, if the goal is to play a game of tennis, one first needs to find an opponent. The goal module must create this subgoal that needs to be achieved before moving back to the original goal (i.e., playing a game of tennis). Note that goals are centralized in a unique module in ACT-R and that production rules only have access to the goal buffer. The current goal to be achieved is the one in the buffer, while later goals stored in the goal module are not accessible to the production. Hence, the ‘play a game of tennis’ goal is not accessible to the production rules while the ‘find an opponent’ subgoal is being pursued.

12.2.3 The declarative module

The declarative memory module contains knowledge about the world in the form of chunks [12.8]. Each chunk represents a piece of knowledge or a concept (e.g., fireman, bank, etc.). Chunks can be accessed effortfully by the central production system, and the probability of retrieving a chunk depends on the chunk activation:

$$P_i = \left[1 + e^{-\frac{(B_i + \sum_j W_j S_{ji} - \tau)}{\varepsilon}} \right]^{-1} \qquad (12.1)$$

where P_i is the probability of retrieving chunk i, B_i is the base-level activation of chunk $\underline{i}$, S_{ji} is the association strength between chunks j and i, W_j is the amount of attention devoted to chunk j, τ is the activation threshold, and ε is a noise parameter. It is important to note that the knowledge chunks in the declarative module are passive and do not do anything on their own. The function of this module is to store information so that it can be retrieved by the central production system (which corresponds to procedural memory). Only in the central production system can the knowledge be used for further reasoning or to produce actions.

12.2.4 The procedural memory

The procedural memory is captured by the central production rule module and fills the role of a central executive processor. It contains a set of rules in which the conditions can be matched by the chunks in all the peripheral buffers and the output is a chunk that can be placed in one of the buffers. The production rules are chained serially, and each rule application takes a fixed amount of psychological time. The serial nature of central processing constitutes another information processing bottleneck in ACT-R.

Because only one production rule can be fired at any given time, its selection is crucial. In ACT-R, each production rule has a utility value that depends on: a) its probability of achieving the current goal, b) the value (importance) of the goal and, c) the cost of using the production rule [12.8]. Specifically,

$$U_i = P_i G - C_i \qquad (12.2)$$

where U_i is the utility of production rule i, P_i is the (estimated) probability that selecting rule i will achieve the goal, G is the value of the current goal, and C_i is the (estimated) cost of rule i. The most useful rule is always selected in every processing cycle, but the utility values can be noisy, which can result in the selection of sub-optimal rules [12.11]. Rule utilities are learned online by counting the number of times that applying a rule has achieved the goal. Anderson [12.10] has shown that the selection according to these counts is optimal in the Bayesian sense. Also, production rules can be made more efficient by using a process called *production compilation* [12.11]-[12.12]. Briefly, if two production rules are often fired in succession and the result is positive, a new production rule is created which directly links the conditions from the first production rule to the action following the application of the second production rule. Hence, the processing time is cut in half by applying only one rule instead of two.

12.2.5 Simulation example: Learning verb past tenses

General computational intelligence requires the abstraction of regularities to form new rules (e.g., rule induction). A well-known example in psychology is children learning of English verb past tenses [12.12]. This classical result shows that children's accuracy in producing the past tense of irregular verbs follows a U-shaped curve [12.13]. Early in learning, children have a separate memory representation for the past tense of each verb (and no conjugation rule). Hence, the past tenses of irregular verbs are used mostly correctly (Phase 1). After moderate training, children notice that most verbs can be converted to their past tense by adding the suffix '-ed'. This leads to the formulation of a default rule (e.g., to find the past tense of a verb, add the suffix '-ed' to the verb stem). This rule is a useful heuristic and works for all regular verbs in English. However, children tend to over-generalize and (incorrectly) apply the rule to irregular verbs. This leads to errors and the low point of the U-shaped curve (Phase 2). Finally, children learn that there are exceptions to the default rule, and memorize the past tense of irregular verbs. Performance improves again (Phase 3).

In ACT-R, the early phase of training uses instance-based retrieval (i.e., retrieving the chunk representing each verb's past tense using Eq. 12.1). The focus of the presentation is on the induction of the default rule, which is over-generalized in Phase 2 and correctly applied in Phase 3. This is accomplished by joining two production rules. First, consider the following memory retrieval rule used in Phase 1 [12.11]:

(1) Retrieve-past-tense

IF the goal is to find the past tense of a word *w*:

THEN issue a request to declarative memory for the past tense of *w*.

If a perfect match is retrieved from declarative memory, a second rule is used to produce the (probably correct) response. However, if Rule 1 fails to retrieve a perfect match, the verb past tense is unknown and an analogy rule is used instead [12.11]:

(2) Analogy-find-pattern

IF the goal is to find the past tense of word *w*1;

AND the retrieval buffer contains past tense *w*2-suffix of *w*2:

THEN set the answer to *w*1-(*w*2-suffix).

This rule produces a form of generalization using an analogy. Because Rule 2 always follows Rule 1, they can be combined using production compilation [12.11]. Also, *w*2 is likely to be a regular verb, so *w*2-suffix is likely to be '-ed'. Hence, combining Rules 1 and 2 yields [12.11]:

(3) Learned-rule

IF the goal is to find the past tense of word *w*:

THEN set the answer to *w*-ed.

which is the default rule that can be used to accurately find the past tense of regular verbs.

The U-shaped curve representing the performance of children learning irregular verbs can thus be explained with ACT-R as follows [12.12]: In Phase 1, Rule 3 does not exist and Rule 1 is applied to correctly conjugate irregular verbs. In Phase 2, Rule 3 is learned and has proven useful with regular verbs (thus increasing P_i in Eq. 12.2). Hence, it is often selected to incorrectly conjugate irregular verbs. In phase 3, the irregular verbs become more familiar as more instances have been encountered. This increases their base-level activation in declarative memory (B_i in Eq. 12.1), which facilitates retrieval and increases the likelihood that Rule 1 is selected to correctly conjugate the irregular verbs. More details about this simulation can be found in [12.12].

12.3 Soar

Soar was the original unified theory of cognition proposed by Newell [12.1]. Soar has been used successfully in many problem solving tasks such as Eight Puzzle, the Tower of Hanoi, Fifteen Puzzle, Think-a-dot, and Rubik Cube. In addition, Soar has been used for many military applications such as training models for human pilots and mission rehearsal exercises. According to the Soar theory of intelligence [12.14], human intelligence is an approximation of a knowledge system [12.9]. Hence, the most important aspect of intelligence (natural or artificial) is the use of all the available knowledge [12.15], and failures of intelligence are failures of knowledge [12.16].

All intelligent behaviors can be understood in terms of problem solving in Soar [12.9]. As such, Soar is implemented as a set of *problem-space computational models* (PSCM) that partition the knowledge into goal relevant ways [12.15]. Each PSCM implicitly contains the representation of a problem space defined by a set of states and a set of operators that can be visualized using a decision tree [12.16]. In a decision tree representation, the nodes represent the states, and one moves around from state to state using operators (the branches/connections in the decision tree). The objective of a Soar agent is to move from an initial state to one of the goal states, and the best operator is always selected at every time step [12.15]. If the knowledge in the model is insufficient to select a single best operator at a particular time step, an impasse is reached, and a new goal is created to resolve the impasse. This new goal defines its own problem space and set of operators.

12.3.1 Architectural representation

The general architecture of Soar is shown in Fig. 12.2. The main structures are a working memory and a long-term memory. Working memory is a blackboard where all the relevant information for the current decision cycle is stored [12.16]. It contains a goal representation, perceptual information, and relevant knowledge that can be used as conditions to fire rules. The outcome of rule firing can also be added to the working memory to cause more rules to fire. The long-term memory contains associative rules representing the knowledge in the system (in the form of "IF → THEN" rules). The rules in long-term memory can be grouped/organized to form operators.

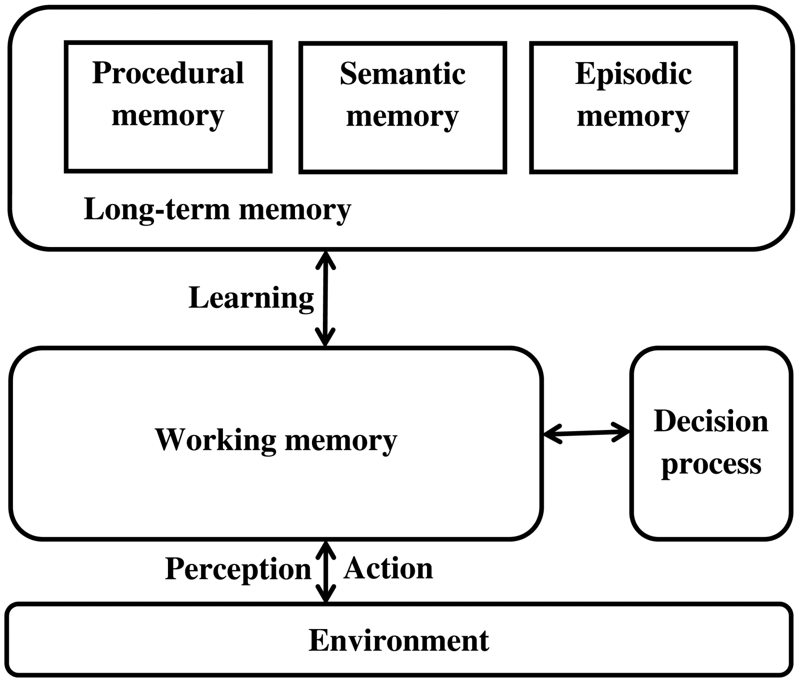

Fig. 12.2. The general architecture of Soar. The subdivision of long-term memory is a new addition in Soar 9.

12.3.2 The Soar decision cycle

In every time step, Soar goes through a six-step decision cycle [12.16]. The first step in Soar is to receive an input from the environment. This input is inserted into working memory. The second step is called the *elaboration phase*. During this phase, all the rules matching the content of working memory fire in parallel, and the result is put into working memory. This in turn can create a new round of parallel rule firing. The elaboration phase ends when the content of working memory is stable, and no new knowledge can be added in working memory by firing rules.

The third step is the proposal of operators that are applicable to the content in working memory. If no operator is applicable to the content of working memory, an impasse is reached. Otherwise, the potential operators are evaluated and ordered according to a symbolic preference metric. Step four is the selection of a single operator. If the knowledge does not allow for the selection of a single operator, an impasse is reached. The fifth step is to apply the operator. If the operator does not result is a change of state, an impasse is reached. Finally, step six is the output of the model, which can be an external (e.g., motor) or an internal (e.g., more reasoning) action.

12.3.3 Impasses

When the immediate knowledge is insufficient to reach a goal, an impasse is reached and a new goal is created to resolve the impasse. Note that this subgoal produces its own problem space with its own set of states and operators. If the subgoal reaches an impasse, another subgoal is recursively created to resolve the second impasse, and so on. There are four main types of impasses in Soar [12.16]:

(1) Impasse: No operator is available in the current state.

New goal: Find an operator that is applicable in the current state.

(2) Impasse: An operator is applicable in the current state, but its application does not change the current state.

New goal: Modify the operator so that its application changes the current state. Alternatively, the operator could be modified so that it is no longer deemed applicable in the current state.

(3) Impasse: Two or more operators are applicable in the current state but neither one of them is preferred according to the symbolic metric.

New goal: Further evaluate the options and make one of the operators preferred to the others.

(4) Impasse: More than one operator is applicable, and there is knowledge in working memory favoring two or more operators in the current state.

New goal: Resolve the conflict by removing from working memory one of the contradictory preferences.

Regardless of which type of impasses is reached, resolving an impasse is an opportunity for learning in Soar [12.16]. Each time a new result is produced while achieving a subgoal, a new rule associating the current state with the new result is added in long-term memory to ensure that the same impasse will not be reached again in the future. This new rule is called a 'chunk' to distinguish it from rules that were precoded by the modeler (and learning is called 'chunking').

12.3.4 Extensions

Unlike ACT-R (and CLARION, as described next), Soar was originally designed as an artificial intelligence model [12.15]. Hence, initially, more attention has been paid to functionality and performance than to psychological realism. However, Soar has been used in psychology and Soar 9 has been extended to increase its psychological realism and functionality [12.17]. This version of the architecture is illustrated in Fig. 12.2. First, the long-term memory has been further subdivided in correspondence with psychology theory [12.18]. The associative rules are now part of the procedural memory. In addition to procedural memory, long-term memory now also includes a semantic and an episodic memory. Semantic memory contains knowledge structures representing factual knowledge about the world (e.g., the earth is round), while episodic memory contains a snapshot of working memory representing an 'episode' (e.g., Fido the dog is now sitting in front of me).

At the subsymbolic level, Soar 9 includes activations in working memory to capture recency/usefulness (as in ACT-R). In addition, Soar 9 uses non-symbolic (numerical) values to model operator preferences. These are akin to utility functions and are used when symbolic operators as insufficient to select a single operator [12.19]. When numerical preferences are used, an operator is selected using a Boltzmann distribution:

$$P(O_i) = \frac{e^{S(O_i)/\tau}}{\sum_j e^{S(O_j)/\tau}} \tag{12.3}$$

where $P(O_i)$ is the probability of selecting operator i, $S(O_i)$ is the summed support (preference) for operator i, and τ is a randomness parameter. Numerical operator preferences can be learned using reinforcement learning.

Finally, recent work has been initiated to include a clustering algorithm that would allow for the creation of new symbolic structures and a visual imagery module to facilitate symbolic spatial reasoning (not shown in Fig. 12.2). Also, the inclusion of emotions is now being considered (via appraisal theory; [12.20]).

12.3.5 Simulation example: Learning problem solving

Nason and Laird [12.19] proposed a variation of Soar that includes a reinforcement learning algorithm (following the precedents of CLARION and ACT-R) to learn numerical preference values for the operators. In this implementation (called Soar-RL), the preferences are replaced by *Q*-values [12.21] that are learned using environmental feedback. After the *Q*-value of each relevant operator has been calculated (i.e., all the operators available in working memory), an operator is stochastically selected (as in Eq. 12.3).

Soar-RL has been used to simulate the Missionaries & Cannibals problem. The goal in this problem is to transport three missionaries and three cannibals across a river using a boat that can carry at most two persons at a time. Several trips are required, but the cannibals must never outnumber the missionaries on either riverbank. This problem has been used as a benchmark in problem solving research because, if the desirability of a move is evaluated in term of the number of peoples that have crossed the river (which is a common assumption), a step backward must be taken midway in solving the problem (i.e., a move that reduces the number of peoples that crossed the river must be selected).

In the Soar-RL simulation, the states were defined by the number of missionaries and cannibals on each side of the river and the location of the boat. The operators were boat trips transporting people, and the *Q*-values of the operators were randomly initialized. Also, to emphasize the role of reinforcement learning in solving this problem, chunking had been disengaged. Hence, the only form of adaptation was the adjustment of the operator *Q*-values. Success states (i.e., all peoples crossed the river) were rewarded, failure states (i.e., cannibals outnumbering missionaries on a riverbank) were punished, and all other states received neutral reinforcement.

Using this simulation methodology, Soar-RL generally learned to solve the Missionaries & Cannibals problem. Most errors resulted from the stochastic decision process [12.19]. Nason and Laird also showed that the model performance can be improved five-fold by adding a symbolic preference preventing an operator at time t from undoing the result of the application of an operator at time t-1. More details on this simulation can be found in [12.19].

12.4 CLARION

CLARION is an integrative cognitive architecture consisting of a number of distinct subsystems with a dual representational structure in each subsystem (implicit versus explicit representations). CLARION is the newest of the reviewed architectures, but it has already been successfully applied to several tasks such navigation in mazes and mine fields, human reasoning, creative problem solving, and cognitive social simulations. CLARION is based on the following basic principles [12.22]. First, humans can learn with or without much a priori specific knowledge to begin with, and humans learn continuously from on-going experience in the world. Second, there are different types of knowledge involved in human learning (e.g., procedural vs. declarative, implicit vs. explicit; [12.22]), and different types of learning processes are involved in acquiring different types of knowledge. Third, motivational processes as well as meta-cognitive processes are important and should be incorporated in a psychologically realistic cognitive architecture. According to CLARION, all three principles are required to achieve general computational intelligence. An overview of the architecture is shown in Fig. 12.3.

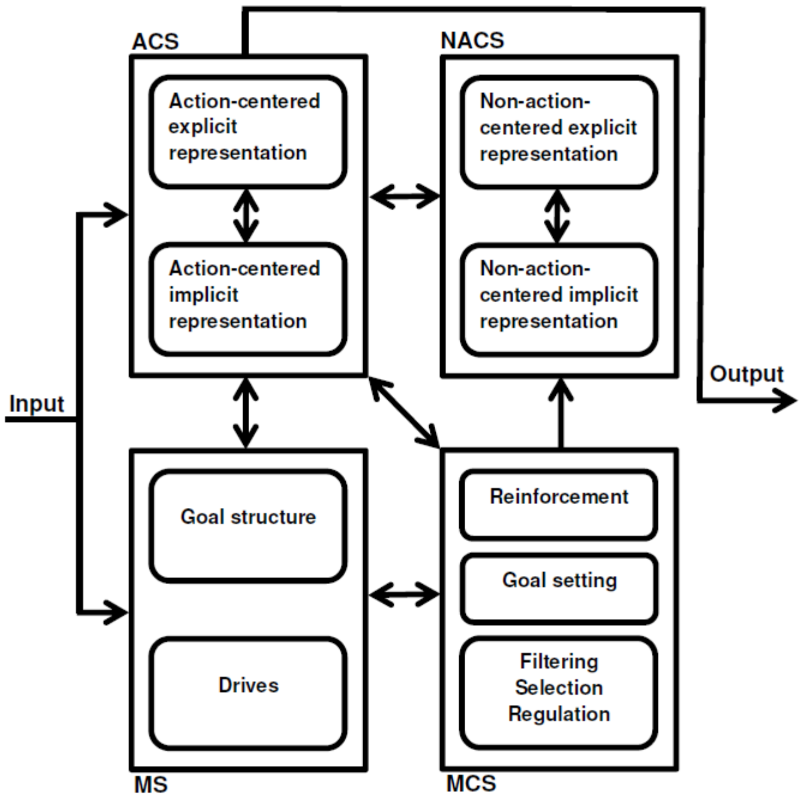
ACS
NACS
Action-centered explicit representation
Action-centered implicit representation
Non-action-centered explicit representation
Non-action-centered implicit representation
Input
Output
Goal structure
Drives
Reinforcement
Goal setting
Filtering Selection Regulation
MS
MCS

Fig. 12.3. The CLARION architecture. ACS stands for the action-centered subsystem, NACS the non-action-centered subsystem, MS the motivational subsystem, and MCS the meta-cognitive subsystem. This Figure is reprinted from [12.42].

The CLARION subsystems include the action-centered subsystem (the ACS), the non-action-centered subsystem (the NACS), the motivational subsystem (the MS), and the meta-cognitive subsystem (the MCS). The role of the ACS is to control actions, regardless of whether the actions are for external physical movements or for internal mental operations. The role of the NACS is to maintain general knowledge. The role of the MS is to provide underlying motivations for perception, action, and cognition, in terms of providing impetus and feedback (e.g., indicating whether outcomes are satisfactory or not). The role of the MCS is to monitor, direct, and modify dynamically the operations of the other subsystems.

Each of these interacting subsystems consists of two “levels” of representations. In each subsystem, the top level encodes explicit (e.g., verbalizable) knowledge and the bottom level encodes implicit (e.g., non-verbalizable) knowledge. The two levels interact, for example, by cooperating through a combination of the action recommendations from the two levels respectively, as well as by cooperating in learning through bottom-up and top-down processes. Essentially, it is a dual-process theory of mind [12.22].

12.4.1 The Action-Centered Subsystem

The ACS is composed of a top and a bottom level. The bottom level of the ACS is modular. A number of neural networks co-exist, each of which is adapted to a specific modality, task, or group of input stimuli. These modules can be developed in interacting with the world (computationally, through various decomposition methods; see, e.g., [12.23]). However, some of them are formed evolutionarily, reflecting hardwired instincts and propensities [12.24]. Because of these networks, CLARION is able to handle very complex situations that are not amenable to simple rules.

In the top level of the ACS, explicit symbolic conceptual knowledge is captured in the form of explicit symbolic rules (for details, see [12.22]). There are many ways in which explicit knowledge may be learned, including independent hypothesis testing and bottom-up learning. The basic process of bottom-up learning is as follows: if an action implicitly decided by the bottom level is successful, then the model extracts an explicit rule that corresponds to the action selected by the bottom level and adds the rule to the top level. Then, in subsequent interactions with the world, the model verifies the extracted rule by considering the outcome of applying the rule. If the outcome is not successful, then the rule should be made more specific; if the outcome is successful, the agent may try to generalize the rule to make it more universal [12.25]. After explicit rules have been learned, a variety of explicit reasoning methods may be used. Learning explicit conceptual representations at the top level can also be useful in enhancing learning of implicit reactive routines at the bottom level (e.g., [12.7]). The action-decision cycle in the ACS can be described by the following steps:

1. Observe the current state of the environment;
2. Compute the value of each possible action in the current state in the bottom level;
3. Compute the value of each possible action in the current state in the top level;
4. Choose an appropriate action by stochastically selecting or combining the values in the top and bottom levels;
5. Perform the selected action;
6. Update the top and bottom levels according to the received feedback (if any);
7. Go back to Step 1.

12.4.2 The Non-Action-Centered Subsystem

The NACS may be used for representing general knowledge about the world (i.e., the semantic memory and the episodic memory), and for performing various kinds of memory retrievals and inferences. The NACS is also composed of two levels (a top and a bottom level) and is under the control of the ACS (through its actions).

At the bottom level, "associative memory" networks encode non-action-centered implicit knowledge. Associations are formed by mapping an input to an output (such as mapping "2+3" to "5"). The backpropagation [12.7], [12.26] or Hebbian [12.27] learning algorithms can be used to establish such associations between pairs of inputs and outputs.

At the top level of the NACS, a general knowledge store encodes explicit non-action-centered knowledge [12.27]-[12.28]. In this network, chunks (passive knowledge structures, similar to ACT-R) are specified through dimensional values (features). A node is set up in the top level to represent a chunk. The chunk node connects to its corresponding features represented as individual nodes in the bottom level of the NACS (see, e.g., [12.27]-[12.28]). Additionally, links between chunk nodes encode explicit associations between pairs of chunks, known as 'associative rules'. Explicit associative rules may be learned in a variety of ways [12.22].

During reasoning, in addition to applying associative rules, similarity-based reasoning may be employed in the NACS. Specifically, a known (given or inferred) chunk may be automatically compared with another chunk. If the similarity between them is sufficiently high, then the latter chunk is inferred. The similarity between chunks *i* and *j* is computed by using:

$$s_{c_i \sim c_j} = \frac{n_{c_i \cap c_j}}{f\left(n_{c_j}\right)} \tag{12.4}$$

where $s_{ci\sim cj}$ is the similarity from *i* to chunk *j*, $n_{ci\cap cj}$ counts the number of features shared by chunks *i* and *j* (i.e., the feature overlap), n_{cj} counts the total number of features in chunk *j*, and $f(x)$ is a slightly super-linear, monotonically increasing, positive function [by default, $f(x) = x^{1.1}$]. Thus, similarity-based reasoning in CLARION is naturally accomplished using (1) top-down activation by chunk nodes of their corresponding bottom-level feature-based representations, (2) calculation of feature overlap between any two chunks (as in Eq. 12.4), and (3) bottom-up activation of the top-level chunk nodes. This kind of similarity calculation is naturally accomplished in a multi-level cognitive architecture and represents a form of synergy between the explicit and implicit modules. Each round of reasoning in the NACS can be described by the following steps:

1. Propagate the activation of the activated features in the bottom level;
2. Concurrently, fire all applicable associative rules in the top level;
3. Integrate the outcomes of top- and bottom-level processing;
4. Update the activations in the top and bottom levels (e.g., similarity-based reasoning);
5. Go back to Step 1 (if another round of reasoning is requested by the ACS).

12.4.3 The Motivational and Meta-Cognitive Subsystems

The motivational subsystem (the MS) is concerned with drives and their interactions [12.29], which leads to actions. It is concerned with why an agent does what it does. Simply saying that an agent chooses actions to maximize gains, rewards, reinforcements, or payoffs leaves open the question of what determines these things. The relevance of the MS to the ACS lies primarily in the fact that it provides the context in which the goal and the reinforcement of the ACS are set. It thereby influences the working of the ACS, and by extension, the working of the NACS.

Dual motivational representations are in place in CLARION. The explicit goals (such as “finding food”) of an agent may be generated based on internal drives (for example, “being hungry”; see [12.30] for details). Beyond low-level drives (concerning physiological needs), there are also higher-level drives. Some of them are primary, in the sense of being “hard-wired”, while others are secondary (“derived”) drives acquired mostly in the process of satisfying primary drives.

The meta-cognitive subsystem (the MCS) is closely tied to the MS. The MCS monitors, controls, and regulates cognitive processes for the sake of improving cognitive performance [12.31]-[12.32]. Control and regulation may be in the forms of setting goals for the ACS, setting essential parameters of the ACS and the NACS, interrupting and changing on-going processes in the ACS and the NACS, and so on. Control and regulation can also be carried out through setting reinforcement functions for the ACS. All of the above can be done on the basis of drive activations in the MS. The MCS is also made up of two levels: the top level (explicit) and the bottom level (implicit).

12.4.4 Simulation example: Minefield navigation

Sun, Merrill, and Peterson [12.7] empirically tested and simulated a complex minefield navigation task. In the empirical task, the subjects were seated in front of a computer monitor that displayed an instrument panel containing several gauges that provided current information on the status/location of a vehicle. The subjects used a joystick to control the direction and speed of the vehicle. In each trial, a random mine layout was generated and the subjects had limited time to reach a target location without hitting a mine. Control subjects were trained for several consecutive days in this task. Sun and colleagues also tested three experimental conditions with the same amount of training but emphasizing verbalization, over-verbalization, and dual-tasking (respectively). The human results show that learning was slower in the dual-task condition than in the single-task condition, and that a moderate amount of verbalization speeds up learning. However, the effect of verbalization is reversed in the over-verbalization condition; over-verbalization interfered with (slowed down) learning.

In the CLARION simulation, simplified (explicit) rules were represented in the form “State → Action” in the top level of the ACS. In the bottom level of the ACS, a backpropagation network was used to (implicitly) learn the input-output function using reinforcement learning. Reinforcement was received at the end of every trial. The bottom-level information was used to create and refine top-level rules (with bottom-up learning). The model started out with no specific a priori knowledge about the task (the same as a typical subject). The bottom level contained randomly initialized weights. The top level started empty and contained no a priori knowledge about the task (either in the form of instructions or rules). The interaction of the two levels was not determined a priori either: there was no fixed weight in combining outcomes from the two levels. The weights were automatically set based on relative performance of the two levels on a periodic basis. The effects of the dual task and the various verbalization conditions were modeled using rule-learning thresholds so that more/less activities could occur at the top level. The CLARION simulation results closely matched the human results [12.7]. In addition, the human and simulated data were input into a common ANOVA and no statistically significant difference between human and simulated data was found in any of the conditions. Hence, CLARION did a good job of simulating detailed human data in the minefield navigation task. More details about this simulation can be found in [12.7].

12.5 Cognitive architectures as models of multi-agent interaction

Most of the work in social simulation assumes rudimentary cognition on the part of agents. Agent models have frequently been custom-tailored to the task at hand, often with a restricted set of highly domain-specific rules. Although this approach may be adequate for achieving the limited objectives of some social simulations, it is overall unsatisfactory. For instance, it limits the realism, and hence applicability of social simulation, and more importantly it also precludes any possibility of resolving the theoretical question of the micro-macro link.

Cognitive models, especially cognitive architectures, may provide better grounding for understanding multi-agent interaction. This can be achieved by incorporating realistic constraints, capabilities, and tendencies of individual agents in terms of their psychological processes (and maybe even in terms of their physical embodiment) and their interactions with their environments (which include both physical and social environments). Cognitive architectures make it is possible to investigate the interaction of cognition/motivation on one hand and social institutions and processes on the other, through psychologically realistic agents. The results of the simulation may demonstrate significant interactions between cognitive-motivational factors and social-environmental factors. Thus, when trying to understand social processes and phenomena, it may be important to take the psychology of individuals into consideration given that detailed computational models of cognitive agents that incorporate a wide range of psychological functionalities have been developed in cognitive science.

For example, Sun and Naveh simulated an organizational classification decision-making task using the CLARION cognitive architecture [12.33]. In a classification decision-making task, agents gather information about problems, classify them, and then make further decisions based on the classification. In this case, the task is to determine whether a blip on a screen is a hostile aircraft, a flock of geese, or a civilian aircraft. In each case, there is a single object in the airspace. The object has nine different attributes, each of which can take on one of three possible values (e.g., its speed can be low, medium, or high). An organization must determine the status of an observed object: whether it is friendly, neutral, or hostile. There are a total of 19,683 possible objects, and 100 problems are chosen randomly from this set.

Critically, no one single agent has access to all the information necessary to make a choice. Decisions are made by integrating separate decisions made by different agents, each of which is based on a different subset of information. In terms of organizational structures, there are two major types of interest: teams and hierarchies. In teams, decision-makers act autonomously, individual decisions are treated as votes, and the organization decision is the majority decision. In hierarchies, agents are organized in a chain of command, such that information is passed from subordinates to superiors, and the decision of a superior is based solely on the recommendations of his/her subordinates. In this task, only a two-level hierarchy with nine subordinates and one supervisor is considered.

In addition, organizations are distinguished by the structure of information accessible to each agent. There are two types of information access: distributed access, in which each agent sees a different subset of three attributes (no two agents see the same subset of three attributes), and blocked access, in which three agents see exactly the same subset of attributes. In both cases, each attribute is accessible to three agents.

The human experiments by Carley et al. [12.34] were done in a 2 × 2 fashion (organization × information access). The data showed that humans generally performed better in team situations, especially when distributed information access was in place. Moreover, distributed information access was generally better than blocked information access. The worst performance occurred when hierarchical organizational structure and blocked information access were used in conjunction.

The results of the CLARION simulations closely matched the patterns of the human data, with teams outperforming hierarchal structures, and distributed access being superior to blocked access. As in the human data, the effects of organization and information access was present, but more importantly the interaction of these two factors with length of training was reproduced. These interactions reflected the following trends: (1) the superiority of team and distributed information access at the start of the learning process and, (2) either the disappearance or reversal of these trends towards the end. These trends persisted robustly across a wide variety of settings of cognitive parameters, and did not critically depend on any one setting of these parameters. Also, as in humans, performance was not grossly skewed towards one condition or the other.

One advantage of using a more "cognitive" agent in social simulations is that we can address the question of what happens when cognitive parameters are varied. Because CLARION captures a wide range of cognitive processes, its parameters are generic (rather than task-specific). Thus, one has the opportunity of studying social and organizational issues in the context of a general theory of cognition. Below we present some of the observed results (details can be found in [12.33]).

Varying the parameter controlling the probability of selecting implicit versus explicit processing in CLARION interacted with the length of training. Explicit rule learning was far more useful at the early stages of learning, when increased reliance on rules tended to boost performance (compared with performance toward the end of the learning process). This is because explicit rules are crisp guidelines that are based on past success, and as such, they provide a useful anchor at the uncertain early stages of learning. However, by the end of the learning process, they become no more reliable than highly trained networks. This corresponds to findings in human cognition, where there are indications that rule-based learning is more widely used in the early stages of learning, but is later increasingly supplanted by similarity-based processes and skilled performance [12.35]-[12.36]. Such trends may partially explain why hierarchies did not perform well initially: Because a hierarchy's supervisor was burdened with a higher input dimensionality, it took a longer time to encode rules (which were nevertheless essential at the early stages of learning).

Another interesting result was the effect of varying the generalization threshold. The generalization threshold determines how readily an agent generalizes a successful rule. It was better to have a higher rule generalization threshold than a lower one (up to a point). That is, if one restricts the generalization of rules to those rules that have proven relatively successful, the result is a higher-quality rule set, which leads to better performance in the long run.

This CLARION simulation showed that some cognitive parameters (e.g., learning rate) had a monolithic, across-the-board effect under all conditions, while in other cases, complex interactions of factors were at work (see [12.33] for full details of the analysis). This illustrates the importance of limiting one's social simulation conclusions to the specific cognitive context in which human data were obtained (in contrast to the practice of some existing social simulations). By using CLARION, Sun and Naveh [12.33] have been able to accurately capture organizational performance data and, moreover, to formulate deeper explanations for the results observed. In cognitive architectures, one can vary parameters and options that correspond to cognitive processes and test their effects on collective performance. In this way, cognitive architectures may be used to predict human performance in social/organizational settings and, furthermore, to help to improve collective performance by prescribing optimal or near-optimal cognitive abilities for individuals for specific collective tasks and/or organizational structures.

12.6. General Discussion

This chapter reviewed the most popular psychologically-oriented cognitive architectures with some example applications in human developmental learning, problem solving, navigation, and cognitive social simulations. ACT* (ACT-R's early version; [12.37]) and Soar [12.14] were some of the first cognitive architectures available and have been around since the early eighties while CLARION was first proposed in the mid-nineties [12.28]. This chronology is crucial when exploring their learning capacity. ACT* and Soar have been developed before the connectionist revolution of the PDP Research Group [12.38], and were therefore implemented using knowledge-rich production systems [12.39]. In contrast, CLARION was proposed after the connectionist revolution and was implemented using neural networks. While some attempts have been made to implement ACT-R [12.40] and Soar [12.41] with neural networks, these architectures remain mostly knowledge rich production systems grounded in the artificial intelligence tradition. One of the most important impacts of the connectionist revolution has been data-driven learning rules (e.g., backpropagation) that allows for autonomous learning. CLARION was created within this tradition, and every component in CLARION has been implemented using neural networks. For instance, explicit knowledge may be implemented using linear two-layer neural networks (e.g., [12.7], [12.26]-[12.27], [12.42]) while implicit knowledge has been implemented using nonlinear multilayer backpropagation networks in the ACS (e.g., [12.7], [12.26]) and recurrent associative memory networks in the NACS (e.g., [12.27], [12.42]). This general philosophy has also been applied to modeling the MS and MCS using linear (explicit) and nonlinear (implicit) neural networks [12.43]. As such, CLARION requires less pre-coded knowledge to achieve its goals, and can be considered more autonomous.

While the different cognitive architectures were motivated by different problems and took different implementation approaches, they share some theoretical similarities. For instance, Soar is somewhat similar to the top levels of CLARION. It contains production rules that fire in parallel and cycles until a goal is reached. In CLARION, top-level rules in the NACS fire in parallel in cycles (under the control of the ACS). However, CLARION includes a distinction between action-centered and non-action-centered knowledge. While this distinction has been added in Soar 9 [12.17], the additional distinction between explicit and implicit knowledge (one of the main assumptions in CLARION) was not. The inclusion of implicit knowledge in the bottom level of CLARION allows for an "automatic" representation of similarity-based reasoning, which is absent in Soar. While Soar can certainly account for similarity-based reasoning, adding an explicit (and *ad hoc*) representation of similarity can become cumbersome when a large number of items are involved.

ACT-R initially took a different approach. Work on the ACT-R cognitive architecture has clearly focused on psychological modeling from the very beginning and, as such, it includes more than one long-term memory stores, distinguishing between procedural and declarative memories (similar to CLARION). In addition, ACT-R has a rudimentary representation of explicit and implicit memories: explicit memory is represented by symbolic structures (i.e., chunks and production rules) while implicit memory is represented by the activation of the structures. In contrast, the distinction between explicit and implicit memories in CLARION is one of the main focuses of the architecture, and a more detailed representation of implicit knowledge has allowed for a natural representation of similarity-based reasoning as well as natural simulations of many psychological data sets [12.7], [12.26]-[12.27]. Yet, ACT-R memory structures have been adequate for simulating many data sets with almost thirty years of research. Future work should be devoted to a detailed comparison of ACT-R, Soar, and CLARION using a common set of tasks to more accurately compare their modeling paradigms, capacities, and limits.

Acknowledgments

This research was supported by the ONR grant N00014-08-1-0068 to the second author. Requests for reprints should be addressed to Sébastien Hélie, Department of Psychological Sciences, Purdue University, West Lafayette, IN 47907-2081, e-mail: shelie@purdue.edu; or to Ron Sun, Department of Cognitive Science, Rensselaer Polytechnic Institute, email: rsun@rpi.edu.

References

[**12**.1] A. Newell: *Unified Theories of Cognition* (Harvard University Press, Cambridge 1990)

[**12**.2] S. Roberts, H. Pashler: How persuasive is a good fit? A comment on theory testing, Psychological Review **107**, 358-367 (2000)

[**12**.3] R. Sun: Desiderata for cognitive architectures. Philosophical Psychology **17**, 341-373 (2004)

[**12**.4] S. Franklin, F.G. Patterson, Jr: The Lida architecture: Adding new modes of learning to an intelligent, autonomous, software agent, Integrated Design and Process Technology, IDPT-2006, San Diego, 2006 (Society for Design and Process Science 2006)

[**12**.5] G.A. Carpenter, S. Grossberg, S: A massively parallel architecture for a self-organizing neural pattern recognition machine. Computer Vision, Graphics, and Image Processing **37**, 54-115 (1987)

[**12**.6] P. Langley, J.E. Laird, S. Rogers: Cognitive architectures: Research issues and challenges. Cognitive Systems Research **10**, 141-160 (2009)

[**12**.7] R. Sun, E. Merrill, T. Peterson: From implicit skills to explicit knowledge: A bottom-up model of skill learning. Cognitive Science **25**, 203-244 (2001)

[**12**.8] J.R. Anderson, D. Bothell, M.D. Byrne, S. Douglass, C. Lebiere, Y. Qin: An integrated theory of the mind. Psychological Review **111**, 1036-1060 (2004)

[**12**.9] N.A. Taatgen, J.R. Anderson: Constraints in cognitive architectures. In: The Cambridge Handbook of Computational Psychology, ed. by R. Sun (Cambridge University Press, New York 2008), pp. 170-185

[**12**.10] J.R. Anderson: *The Adaptive Character of Thought* (Erlbaum, Hillsdale 1990)

[**12**.11] N.A. Taatgen, C. Lebiere, J.R. Anderson: Modeling paradigms in ACT-R. In: Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation, ed. by R. Sun (Cambridge University Press, New York 2006), pp. 29-52

[**12**.12] N.A. Taatgen, J.R. Anderson: Why do children learn to say "broke"? A model of learning the past tense without feedback. Cognition **86**, 123-155 (2002)

[**12**.13] G.F. Marcus, S. Pinker, M. Ullman, M. Hollander, T.J. Rosen, F. Xu: Overregularization in language acquisition. Monographs of the Society for Research in Child Development **57**, 1–182 (1992)

[**12**.14] J.E. Laird, A. Newell, P.S. Rosenbloom: Soar: An architecture for general intelligence. Artificial Intelligence **33**, 1-64 (1987)

[**12**.15] J.F. Lehman, J. Laird, P. Rosenbloom: *A Gentle Introduction to Soar, an Architecture for Human Cognition* (University of Michigan 2006)

[**12**.16] R.E. Wray, R.M. Jones: Considering Soar as an agent architecture. In: Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation, ed. by R. Sun (Cambridge University Press, New York 2006), pp. 53-78

[**12**.17] J.E. Laird: Extending the Soar cognitive architecture. Proceedings of the First Conference on Artificial General Intelligence (IOS Press, Amsterdam 2008) pp. 224-235

[**12**.18] D.L. Schacter, A.D. Wagner, R.L. Buckner: Memory systems of 1999. In: The Oxford Handbook of Memory, ed. by E. Tulving, F.I.M. Craik (Oxford University Press, New York 2000), pp. 627-643

[**12**.19] S. Nason, J.E. Laird: Soar-RL: Integrating reinforcement learning with Soar. Cognitive Systems Research **6**, 51-59 (2005)

[**12**.20] K.R. Scherer: Appraisal considered as a process of multi-level sequential checking. In: Appraisal Processes in Emotion: Theory, Methods, Research, ed. by

K.R. Scherer, A. Schor, T. Johnstone (Oxford University Press, New York 2001), pp. 92-120

[**12**.21] C. Watkins: *Learning From Delayed Rewards* (Cambridge University, Cambridge 1990)

[**12**.22] R. Sun: *Duality of the Mind: A Bottom-up Approach Toward Cognition* (Lawrence Erlbaum Associates, Mahwah 2002)

[**12**.23] R. Sun, T. Peterson: Multi-agent reinforcement learning: Weighting and partitioning. Neural Networks **12**, 127-153 (1999)

[**12**.24] L. Hirschfield, S. Gelman (Eds.): *Mapping the Mind: Domain Specificity in Cognition and Culture*. (Cambridge University Press, Cambridge 1994)

[**12**.25] R. Michalski: A theory and methodology of inductive learning. Artificial Intelligence **20**, 111-161 (1983)

[**12**.26] R. Sun, P. Slusarz, C. Terry: The interaction of the explicit and the implicit in skill learning: A dual-process approach. Psychological Review **112**, 159-192 (2005)

[**12**.27] S. Helie, R. Sun: Incubation, insight, and creative problem solving: A unified theory and a connectionist model. Psychological Review **117**, 994-1024 (2010)

[**12**.28] R. Sun: *Integrating Rules and Connectionism for Robust Commonsense Reasoning* (John Wiley and Sons, New York 1994)

[**12**.29] F. Toates: *Motivational Systems* (Cambridge University Press, Cambridge 1986)

[**12**.30] R. Sun: Motivational representations within a computational cognitive architecture. Cognitive Computation **1**, 91-103 (2009)

[**12**.31] T. Nelson (Ed.): *Metacognition: Core Readings* (Allyn and Bacon, Boston 1993)

[**12**.32] J.D. Smith, W.E. Shields, D.A. Washburn: The comparative psychology of uncertainty monitoring and metacognition. Behavioral and Brain Sciences **26**, 317-373 (2003)

[**12**.33] R. Sun, I. Naveh: Simulating organizational decision-making using a cognitively realistic agent model. Journal of Artificial Societies and Social Simulation **7** (2004)

[**12**.34] K.M. Carley, M.J. Prietula, Z. Lin: Design versus cognition: The interaction of agent cognition and organizational design on organizational performance. Journal of Artificial Societies and Social Simulation **1** (1998)

[**12**.35] S. Helie, J.G. Waldschmidt, F.G. Ashby: Automaticity in rule-based and information-integration categorization. Attention, Perception, & Psychophysics **72**, 1013-1031 (2010)

[**12**.36] S. Helie, J.L. Roeder, F.G. Ashby: Evidence for cortical automaticity in rule-based categorization. Journal of Neuroscience **30**, 14225-14234 (2010)

[**12**.37] J.R. Anderson: *The Architecture of Cognition* (Harvard University Press, Cambridge 1983)

[**12**.38] D. Rumelhart, J. McClelland, The PDP Research Group (Eds.): *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations* (MIT Press, Cambridge 1986)

[**12**.39] S. Russell, P. Norvig: *Artificial Intelligence: A Modern Approach* (Prentice Hall, Upper Saddle River 1995)

[**12**.40] C. Lebiere, J.R. Anderson: A connectionist implementation of the ACT-R production system, Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society (Lawrence Erlbaum Associates, Hillsdale 1993) pp. 635-640

[**12**.41] B. Cho, P.S. Rosenbloom, C.P. Dolan: Neuro-Soar: A neural-network architecture for goal-oriented behavior. *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society* (Lawrence Erlbaum Associates, Hillsdale 1991) pp. 673-677

[**12**.42] R. Sun, S. Helie: Psychologically realistic cognitive agents: Taking human cognition seriously. Journal of Experimental & Theoretical Artificial Intelligence (in press)

[**12**.43] N. Wilson, R. Sun, R. Mathews: A motivationally-based simulation of performance degradation under pressure. Neural Networks **22**, 502-508 (2009)