Cognitive Architectures:
Research Issues and Challenges

Pat Langley
Computational Learning Laboratory
Center for the Study of Language and Information
Stanford University, Stanford, CA 94305

John E. Laird
EECS Department
The University of Michigan
1101 Beal Avenue
Ann Arbor, MI 48109

Seth Rogers
Computational Learning Laboratory
Center for the Study of Language and Information
Stanford University, Stanford, CA 94305

Abstract

In this paper, we examine the motivations for research on cognitive architectures and review some candidates that have been explored in the literature. After this, we consider the capabilities that a cognitive architecture should support, some properties that it should exhibit related to representation, organization, performance, and learning, and some criteria for evaluating such architectures at the systems level. In closing, we discuss some open issues that should drive future work in this important area.

Keywords: cognitive architectures, intelligent systems, cognitive processes

Draft: Please do not quote without permission; comments welcome.

1. Background and Motivation

A cognitive architecture specifies the underlying infrastructure for an intelligent system. Briefly, an architecture includes those aspects of a cognitive agent that are constant over time and across different application domains. These typically include:

- the short-term and long-term memories that store content about the agent's beliefs, goals, and knowledge;
- the representation of elements that are contained in these memories and their organization into larger-scale mental structures;
- the functional processes that operate on these structures, including the performance mechanisms that utilize them and the learning mechanisms that alter them.

Because the contents of an agent's memories can change over time, one would not consider the knowledge and beliefs encoded therein to be part of that agent's architecture. Just as different programs can run on the same computer architecture, so different knowledge bases and beliefs can be interpreted by the same cognitive architecture. There is also a direct analogy with a building's architecture, which consists of permanent features like its foundation, roof, and rooms, rather than its furniture and appliances, which one can move or replace.

As we will see, alternative cognitive architectures can differ in the specific assumptions they make about these issues, just as distinct buildings differ in their layouts. In addition to making different commitments about how to represent, use, or acquire knowledge and beliefs, alternative frameworks may claim that more or less is built into the architectural level, just as some buildings embed shelves and closets into their fixed structures, whereas others handle the same functions with replaceable furniture.

Research on cognitive architectures is important because it supports a central goal of artificial intelligence and cognitive science: the creation and understanding of synthetic agents that support the same capabilities as humans. Some work focuses on modeling the invariant aspects of human cognition, whereas other efforts view architectures as an effective path to building intelligent agents. However, these are not antithetical goals; cognitive psychology and AI have a long history of building on the other's ideas (Langley, 2006), and research on cognitive architectures has played a key role in this beneficial interchange.

In some ways, cognitive architectures constitute the antithesis of expert systems, which provide skilled behavior in narrowly defined contexts. In contrast, architectural research aims for breadth of coverage across a diverse set of tasks and domains. More important, it offers accounts of intelligent behavior at the *systems* level, rather than at the level of component methods designed for specialized tasks. Newell (1973a) has argued persuasively for systems-level research in cognitive science and artificial intelligence, claiming "You can't play 20 questions with nature and win". Instead of carrying out micro-studies that address only one issue at a time, we should attempt to unify many findings into a single theoretical framework, then proceed to test and refine that theory.

Since that call to arms, there has been a steady flow of research on cognitive architectures. The movement was associated originally with a specific class of architectures known as *production systems* (Newell, 1973b; Neches et al., 1987) and emphasized explanation of psychological phenomena, with many current candidates still taking this form and showing similar concerns. However, over the past three decades, a variety of other architectural classes have emerged, some less concerned with human behavior, that make quite different assumptions about the representation, organization, utilization, and acquisition of knowledge. At least three invited symposia have brought together researchers in this area (Laird, 1991; VanLehn, 1991; Shapiro & Langley, 2004), and there have been at least two edited volumes (Sun, 2005; VanLehn, 1991). The movement has gone beyond basic research into the commercial sector, with applications to believable agents for simulated training environments (e.g., Tambe et al., 1995), computer tutoring systems (Koedinger, Anderson, Hadley, & Mark, 1997), and interactive computer games (e.g., Magerko et al., 2004).

Despite this progress, there remains a need for additional research in the area of cognitive architectures. As artificial intelligence and cognitive science have matured, they have fragmented into a number of well-defined subdisciplines, each with its own goals and its own criteria for evaluation. Yet commercial and government applications increasingly require *integrated* systems that exhibit intelligent behavior, not just improvements to the components of such systems. This demand can be met by an increased focus on system-level architectures that support complex cognitive behavior across a broad range of relevant tasks.

In this document, we examine some key issues that arise in the design and study of integrated cognitive architectures. Because we cannot hope to survey the entire space of architectural theories, we focus on the ability to generate intelligent behavior, rather than matching the results of psychological experiments.[1] We begin with a brief review of some sample architectures, then discuss the capabilities and functions that such systems should support. After this, we consider a number of design decisions that relate to the properties of cognitive architectures, followed by some dimensions along which one should evaluate them. In closing, we note some open issues in the area and propose some directions that future research should take to address them.

2. Example Cognitive Architectures

Before turning to abstract issues that arise in research on cognitive architectures, we should consider some concrete examples that have been reported in the literature. Here we review four distinct frameworks that fall at different points within the architectural space. We have selected these architectures because each has appeared with reasonable frequency in the literature, and also because they exhibit different degrees of concern with explaining human behavior. We have ordered them along this dimension, with more devoted psychological models coming earlier.

In each case, we discuss the manner in which the architecture represents, organizes, utilizes, and acquires knowledge, along with its accomplishments. Because we review only a small sample of extant architectures, we summarize a variety of other frameworks briefly in the Appendix. Nevertheless, this set should give readers some intuitions about the space of cognitive architectures, which later sections of the paper discuss more explicitly.

One common feature of the architectures we examine is that, although they have some theoretical commitment to parallelism, especially in memory retrieval, they also rely on one or a few decision modules. We have not included connectionist approaches in our treatment because, to our knowl-

[1] Sun (2007) provides another treatment of cognitive architectures that discusses the second topic in greater detail.

edge, they have not demonstrated the broad functionality associated with cognitive architectures in the sense we discuss here. However, they have on occasion served as important components in larger-scale architectures, as in Sun, Merrill, and Peterson's (2001) CLARION framework.

2.1 ACT

ACT-R (Anderson & Lebiere, 1998, Anderson et al., 2004) is the latest in a family of cognitive architectures, concerned primarily with modeling human behavior, that has seen continuous development since the late 1970s. ACT-R 6 is organized into a set of modules, each of which processes a different type of information. These include sensory modules for visual processing, motor modules for action, an intentional module for goals, and a declarative module for long-term declarative knowledge. Each module has an associated buffer that holds a relational declarative structure (often called 'chunks', but different from those in Soar). Taken together, these buffers comprise ACT-R's short-term memory.

A long-term production memory coordinates the processing of the modules. The conditions of each production test chunks in the short-term buffers, whereas its actions alter the buffers upon application. Some changes modify existing structures, whereas others initiate actions in the associated modules, such as executing a motor command or retrieving a chunk from long-term declarative memory. Each declarative chunk has an associated base activation that reflects its past usage and influences its retrieval from long-term memory, whereas each production has an expected cost (in terms of time needed to achieve goals) and probability of success.

On every cycle, ACT determines which productions match against the contents of short-term memory. This retrieval process is influenced by the base activation for each chunk it matches. ACT computes the utility for each matched production as the difference between its expected benefit (the desirability of its goal times its probability of success) and its expected cost. The system selects the production with the highest utility (after adding noise to this score) and executes its actions. The new situation leads new productions to match and fire, so that the cycle continues.

Learning occurs in ACT-R at both the structural and statistical levels. For instance, the base activation for declarative chunks increases with use by productions but decays otherwise, whereas the cost and success probability for productions is updated based on their observed behavior. The architecture can learn entirely new rules from sample solutions through a process of production compilation that analyzes dependencies of multiple rule firings, replaces constants with variables, and combines them into new conditions and actions (Taatgen, 2005).

The ACT-R community has used its architecture to model a variety of phenomena from the experimental psychology literature, including aspects of memory, attention, reasoning, problem solving, and language processing. Most publications have reported accurate fits to quantiative data about human reaction times and error rates. More recently, Anderson (2007) has related ACT-R modules to different areas of the brain and developed models that match results from brain-imaging studies. One the more applied front, the framework has played a central role in tutoring systems that have seen wide use in schools (Koedinger et al., 1997), and it has also been used to control mobile robots that interact with humans (Trafton et al., 2005).

2.2 Soar

Soar (Laird, 2008; Laird, Newell, & Rosenbloom, 1987; Newell, 1990) is a cognitive architecture that has been under continuous development since the early 1980s. Procedural long-term knowledge in Soar takes the form of production rules, which are in turn organized in terms of operators associated with problem spaces. Some operators describe simple, primitive actions that modify the agent's internal state or generate primitive external actions, whereas others describe more abstract activities. For many years, Soar represented all long-term knowledge in this form, but recently separate episodic and semantic memories have been added. The episodic memory (Nuxoll & Laird, 2007) holds a history of previous states, while semantic memory contains previously known facts.

All tasks in Soar are formulated as attempts to achieve goals. Operators perform the basic deliberative acts of the system, with knowledge used to dynamically determine their selection and application. The basic processing cycle repeatedly proposes, selects, and applies operators of the current problem space to a problem state, moving ahead one decision at a time. However, when knowledge about operator selection is insufficient to determine the next operator to apply or when an abstract operator cannot be implemented, an *impasse* occurs; in response, Soar creates a new goal to determine which operator it should select or how it should implement the abstract operator.

This process can lead to the dynamic generation of a goal hierarchy, in that problems are decomposed into subproblems as necessary. The 'state' of a specific goal includes all features of its supergoals, plus any additional cognitive structures necessary to select and apply operators in the subgoal. Processing in a subgoal involves the same basic processing cycle of selecting and applying operators. Subgoals can also deliberately access episodic or semantic memory to retrieve knowledge relevant to resolving the impasse. The top state includes all sensor data obtained from the external environment, so this information is also available to all subgoals. On any cycle, the states at various levels of the goal hierarchy can change, typically due to changes in sensor values or as the result of operator applications in subgoals. When the system resolves the impasse that generated a goal, that goal disappears, along with all the subgoals below it.

Soar has multiple learning mechanisms for different types of knowledge: *chunking* and reinforcement learning acquire procedural knowledge, whereas episodic and semantic learning acquire their corresponding types of declarative knowledge. Chunking occurs when one or more result is produced in a subgoal (Laird, Rosenbloom, & Newell, 1986). When this happens, Soar learns a new *chunk*, represented as a production rule which summarizes the processing that generated the results. A chunk's actions are based on the results, whereas its conditions are based on those aspects of the goals above the subgoal that were relevant to determining the results. Once the agent has learned a chunk, it fires in new situations that are similar along relevant dimensions, often giving the required results directly and thus avoiding the impasse that led to its formation. Reinforcement learning adjusts numeric values associated with rules that help select operators (Nason & Laird, 2004). Episodic learning records the contents of working memory in snapshots, while semantic learning stores individual elements of working memory for later retrieval.

Researchers have used Soar to develop a variety of sophisticated agents that have demonstrated impressive functionality. The most visible has been TAC-Air-Soar (Tambe et al., 1995), which modeled fighter pilots in military training exercises that involved air combat scenarios. More recently, Soar has supported a number of intelligent agents that control synthetic characters in interactive computer games (Margerko et al., 2004). Another thrust has involved using Soar to model the details of human language processing (Lewis, 1993), categorization (Miller & Laird, 1996), and other facets of cognition, but the emphasis has been on demonstrating high-level functionality rather than on fits to quantitative measurements.

2.3 Icarus

Icarus is a more recent architecture (Langley, Cummings, & Shapiro, 2004) that stores two distinct forms of knowledge. Concepts describe classes of environmental situations in terms of other concepts and percepts, whereas skills specify how to achieve goals by decomposing them into ordered subgoals. Both concepts and skills involve relations among objects, and both impose a hierarchical organization on long-term memory, with the former grounded in perceptions and the latter in executable actions. Moreover, skills refer to concepts in their heads, their initiation conditions, and their continuation conditions.

The basic Icarus interpreter operates on a recognize-act cycle. On each step, the architecture deposits descriptions of visible objects into a perceptual bufffer. The system compares primitive concepts to these percepts and adds matched instances to short-memory as beliefs. These in turn trigger matches of higher-level concepts, with the process continuing until Icarus infers all deductively implied beliefs. Next, starting from a top-level goal, it finds a path downward through the skill hierarchy in which each subskill has satisfied conditions but an unsatisfied goal. When a path terminates in a primitive skill with executable actions, the architecture applies these actions to affect the environment. This leads to new percepts, changes in beliefs, and reactive execution of additional skill paths to achieve the agent's goals.

However, when Icarus can find no applicable path through the skill hierarchy that is relevant to a top-level goal, it resorts to problem solving using a variant of means-ends analysis. This module chains backward off either a skill that would achieve the current goal or off the goal concept's definition, and it interleaves problem solving with execution in that it carries out selected skills when they become applicable. Whenever problem solving achieves a goal, Icarus creates a new skill with that goal as its head and with one or more ordered subgoals that are based on the problem solution. If the system encounters similar problems in the future, it executes the learned skills to handle them reactively, without need for deliberative problem solving (Langley & Choi, 2006b).

Researchers have used Icarus to develop agents for a number of domains that involve a combination of inference, execution, problem solving, and learning. These have included tasks like the Tower of Hanoi, multi-column subtraction, FreeCell solitaire, and logistics planning. They have also used the architecture to control synthetic characters in simulated virtual environments, including ones that involve urban driving (Langley & Choi, 2006a) and first-person shooter scenarios (Choi et al., 2007). Ongoing work aims to link Icarus to physical robots that carry out joint activities with humans.

2.4 Prodigy

Prodigy (Carbonell, Knoblock, & Minton, 1990) is another cognitive architecture that saw extensive development from the middle 1980s to the late 1990s. This framework incorporates two main kinds of knowledge. Domain rules encode the conditions under which actions have certain effects, where the latter are described as the addition or deletion of first-order expressions. These refer both to physical actions that affect the environment and to inference rules, which are purely cognitive. In contrast, control rules specify the conditions under which the architecture should select, reject, or prefer a given operator, set of operator bindings, problem state, or goal during the search process.

As in Icarus, Prodigy's basic problem-solving module involves search through a problem space to achieve one or more goals, which it also casts as first-order expressions. This search relies on means-ends analysis, which selects an operator that reduces some difference between the current state and the goal, which in turn can lead to subproblems with their own current states and goals. On each cycle, Prodigy uses its control rules to select an operator, binding set, state, or goal, to reject them out of hand, or to prefer some over others. In the absence of such control knowledge, the architecture makes choices at random and carries out depth-first means-ends search with backtracking.

Prodigy's explanation-based learning module constructs control rules based on its problem-solving experience (Minton, 1990). Successful achievement of a goal after search leads to creation of selection or preference rules related to that goal and to the operators whose application achieved it. Failure to achieve a goal leads to creation of rejection or preference rules for operators, goals, and states that did not produce a solution. To generate these control rules, Prodigy invokes a learning method that analyzes problem-solving traces and proves the reasons for success or failure. The architecture also collects statistics on learned rules and retains only those whose inclusion, over time, leads to more efficient problem solving.

In addition, Prodigy includes separate modules for controlling search by analogy with earlier solutions (Veloso & Carbonell, 1993), learning operator descriptions from observed solutions or experimentation (Wang, 1995), and improving the quality of solutions (Pérez & Carbonell, 1994). Although most research in this framework has dealt exclusively with planning and problem solving, Prodigy also formed the basis for an impressive system that interleaved planning and execution for a mobile robot that accepted asychronous requests from users (Haigh & Veloso, 1996).

3. Capabilities of Cognitive Architectures

Any intelligent system is designed to engage in certain activities that, taken together, constitute its functional capabilities. In this section, we discuss the varied capabilities that a cognitive architecture can support. Although only a few abilities, such as recognition and decision making, are strictly required to support a well-defined architecture, the entire set seems required to cover the full range of human-level intelligent activities.

A central issue that confronts the designer of a cognitive architecture is how to let agents access different sources of knowledge. Many of the capabilities we discuss below give the agent access to such knowledge. For example, knowledge about the environment comes through perception, knowledge about implications of the current situation comes through planning, reasoning, and prediction, knowledge from other agents comes via communication, and knowledge from the past comes through remembering and learning. The more such capabilities an architecture supports, the more sources of knowledge it can access to inform its behavior.

Another key question is whether the cognitive architecture supports a capability directly, using embedded processes, or whether it instead provides ways to implement that capability in terms of knowledge. Design decisions of this sort influence what the agent can learn from experience, what the designers can optimize at the outset, and what functionalities can rely on specialized representations and mechanisms. In this section, we attempt to describe functionality without referring to the underlying mechanisms that implement them, but this is an important issue that deserves more attention in the future.

3.1 Recognition and Categorization

An intelligent agent must make some contact between its environment and its knowledge. This requires the ability to recognize situations or events as instances of known or familiar patterns. For example, a reader must recognize letters and the words they make up, a chess player must identify meaningful board configurations, and an image analyst must detect buildings and vehicles in aerial photographs. However, recognition need not be limited to static situations. A fencing master can identify different types of attacks and a football coach can recognize the execution of particular plays by the opposing team, both of which involve dynamic events.

Recognition is closely related to categorization, which involves the assignment of objects, situations, and events to known concepts or categories. However, research on cognitive architectures typically assumes recognition is a primitive process that occurs on a single cycle and that underlies many higher-level functions, whereas categorization is sometimes viewed as a higher-level function. Recognition and categorization are closely linked to perception, in that they often operate on output from the perceptual system, and some frameworks view them as indistinguishable. However, they can both operate on abstract mental structures, including those generated internally, so we will treat them as distinct.

To support recognition and categorization, a cognitive architecture must provide some way to represent patterns and situations in memory. Because these patterns must apply to similar but distinct situations, they must encode general relations that hold across these situations. An architecture must also include some recognition process that lets it identify when a particular situation matches a stored pattern or category and, possibly, measure the degree to which it matches. In production system architectures, this mechanism determines when the conditions of each production rule match and the particular ways they are instantiated. Finally, a complete architecture should include some means to learn new patterns or categories from instruction or experience, and to refine existing patterns when appropriate.

3.2 Decision Making and Choice

To operate in an environment, an intelligent system also requires the ability to make decisions and select among alternatives. For instance, a student must decide which operation will simplify an integration problem, a speaker must select what word to use next in an utterance, and a baseball player must decide whether or not to swing at a pitch. Such decisions are often associated with the recognition of a situation or pattern, and most cognitive architectures combine the two mechanisms in a recognize-act cycle that underlies all cognitive behavior.

Such one-step decision making has much in common with higher-level choice, but differs in its complexity. For example, consider a consumer deciding which brand of detergent to buy, a driver choosing which route to drive, and a general selecting which target to bomb. Each of these decisions can be quite complex, depending on how much time and energy the person is willing to devote. Thus, we should distinguish between decisions that are made at the architectural level and more complex ones that the architecture enables.

To support decision making, a cognitive architecture must provide some way to represent alternative choices or actions, whether these are internal cognitive operations or external ones. It must also offer some process for selecting among these alternatives, which most architectures separate into two steps. The first determines whether a given choice or action is allowable, typically by associating it with some pattern and considering it only if the pattern is matched. For instance, we can specify the conditions under which a chess move is legal, then consider that move only when the conditions are met. The second step selects among allowable alternatives, often by computing some numeric score and choosing one or more with better scores. Such *conflict resolution* takes quite different forms in different architectures.

Finally, an ideal cognitive architecture should incorporate some way to improve its decisions through learning. Although this can, in principle, involve learning new alternatives, most mechanisms focus on learning or revising either the conditions under which an existing action is considered allowable or altering the numeric functions used during the conflict resolution stage. The resulting improvements in decision making will then be reflected in the agent's overall behavior.

3.3 Perception and Situation Assessment

Cognition does not occur in isolation; an intelligent agent exists in the context of some external environment that it must sense, perceive, and interpret. An agent may sense the world through different modalities, just as a human has access to sight, hearing, and touch. The sensors may range from simple devices like a thermometer, which generates a single continuous value, to more complex mechanisms like stereoscopic vision or sonar that generate a depth map for the local environment within the agent's field of view. Perception can also involve the integration of results from different modalities into a single assessment or description of the environmental situation, which an architecture can represent for utilization by other cognitive processes.

Perception is a broad term that covers many types of processing, from inexpensive ones that an architecture can support automatically to ones that require limited resources and so must be invoked through conscious intentions. For example, the human visual system can detect motion in the periphery without special effort, but the fovea can extract details only from the small region at which it is pointed. A cognitive architecture that includes the second form of sensor must confront the issue of *attention*, that is, deciding how to allocate and direct its limited perceptual resources to detect relevant information in a complex environment.

An architecture that supports perception should also deal with the issue that sensors are often noisy and provide at most an inaccurate and partial picture of the agent's surroundings. Dynamic environments further complicate matters in that the agent must track changes that sometimes occur at a rapid rate. These challenges can be offset with perceptual knowledge about what sensors to invoke, where and when to focus them, and what inferences are plausible. An architecture can also acquire and improve this knowledge by learning from previous perceptual experiences.

An intelligent agent should also be able to move beyond perception of isolated objects and events to understand and interpret the broader environmental situation. For example, a fire control officer on a ship must understand the location, severity, and trajectory of fires in order to respond effectively, whereas a general must be aware of an enemy's encampments, numbers, and resources to defend against them successfully. Thus, situation assessment requires an intelligent agent to combine perceptual information about many entities and events, possibly obtained from many sources, to compose a large-scale model of the current environment. As such, it relies both on the recognition and categorization of familiar patterns in the environment, which we discussed earlier, and on inferential mechanisms, which we will consider shortly.

3.4 Prediction and Monitoring

Cognitive architectures exist over time, which means they can benefit from an ability to predict future situations and events accurately. For example, a good driver knows approximately when his car will run out of gas, a successful student can predict how much he must study to ace a final, and a skilled pilot can judge how close he can fly to the ground without crashing. Perfect prediction may not be possible in many situations, but perfection is seldom necessary to make predictions that are useful to an intelligent system.

Prediction requires some model of the environment and the effect actions have on it, and the architecture must represent this model in memory. One general approach involves storing some mapping from a description of the current situation and an action onto a description of the resulting situation. Another approach encodes the effects of actions or events in terms of changes to the environment. In either case, the architecture also requires some mechanism that uses these knowledge structures to predict future situations, say by recognizing a class of situations in which an action will have certain effects. An ideal architecture should also include the ability to learn predictive models from experience and to refine them over time.

Once an architecture has a mechanism for making predictions, it can also utilize them to monitor the environment. For example, a pilot may suspect that his tank has a leak if the fuel gauge goes down more rapidly than usual, and a commander may suspect enemy action if a reconnaissance team fails to report on time. Because monitoring relates sensing to prediction, it raises issues of attentional focus when an architecture has limited perceptual resources. Monitoring also provides natural support for learning, since errors can help an agent improve its model of the environment.

3.5 Problem Solving and Planning

Because intelligent systems must achieve their goals in novel situations, the cognitive architectures that support them must be able to generate plans and solve problems. For example, an unmanned air vehicle benefits from having a sensible flight plan, a project manager desires a schedule that allocates tasks to specific people at specific times, and a general seldom moves into enemy territory without at least an abstract course of action. When executed, plans often go awry, but that does not make them any less useful to an intelligent agent's thinking about the future.

Planning is only possible when the agent has an environmental model that predicts the effects of its actions. To support planning, a cognitive architecture must be able to represent a plan as an (at least partially) ordered set of actions, their expected effects, and the manner in which these effects enable later actions. The plan need not be complete to guide behavior, in that it may extend only a short time into the future or refer to abstract actions that can be expanded in different ways. The structure may also include conditional actions and branches that depend on the outcome of earlier events as noted by the agent.

An intelligent agent should also be able to construct a plan from components available in memory. These components may refer to low-level motor and sensory actions but, often, they will be more abstract structures, including prestored subplans. There exist many mechanisms for generating plans from components, as well as ones for adapting plans that have been retrieved from memory. What these methods have in common is that they involve problem solving or search. That is, they carry out steps through a space of problem states, on each step considering applicable operators, selecting one or more operator, and applying it to produce a new problem state. This search process continues until the system has found an acceptable plan or decides to give up.

The notion of problem solving is somewhat more general than planning, though they are typically viewed as closely related. In particular, planning usually refers to cognitive activities within the agent's head, whereas problem solving can also occur in the world. Especially when a situation is complex and the architecture has memory limitations, an agent may carry out search by applying operators or actions in the environment, rather than trying to construct a plan internally. Problem solving can also rely on a mixture of internal planning and external behavior, but it generally involves the multi-step construction of a problem solution. Like planning, problem solving is often characterized in terms of search through a problem space that applies operators to generate new states, selects promising candidates, and continues until reaching a recognized goal.

Planning and problem solving can also benefit from learning. Naturally, improved predictive models for actions can lead to more effective plans, but learning can also occur at the level of problem space search, whether this activity takes place in the agent's head or in the physical world. Such learning can rely on a variety of information sources. In addition to learning from direct instruction, an architecture can learn from the results of problem-space search (Sleeman et al., 1982), by observing another agent's behavior or *behavioral cloning* (Sammut, 1996), and from delayed rewards via *reinforcement learning* (Sutton & Barto, 1998). Learning can aim to improve problem solving behavior in two ways (Langley, 1995a). One focuses on reducing the branching factor of search, either through adding heuristic conditions to problem space operators or refining a numeric evaluation function to guide choice. Another focuses on forming macro-operators or stored plans that reduce the effective depth of search by taking larger steps in the problem space.

Intelligent agents that operate in and monitor dynamic environments must often modify existing plans in response to unanticipated changes. This can occur in several contexts. For instance, an agent should update its plan when it detects a changed situation that makes some planned activities inapplicable, and thus requires other actions. Another context occurs when a new situation suggests some more desirable way of accomplishing the agent's goal; such opportunistic planning can take advantage of these unexpected changes. Monitoring a plan's execution can also lead to revised estimates about the plan's effectiveness, and, ultimately, to a decision to pursue some other course of action with greater potential. Replanning can draw on the same mechanisms as generating a plan from scratch, but requires additional operators for removing actions or replacing them with other steps. Similar methods can also adapt to the current situation a known plan the agent has retrieved from memory.

3.6 Reasoning and Belief Maintenance

Problem solving is closely related to *reasoning*, another central cognitive activity that lets an agent augment its knowledge state. Whereas planning is concerned primarily with achieving objectives in the world by taking actions, reasoning draws mental conclusions from other beliefs or assumptions that the agent already holds. For example, a pilot might conclude that, if another plane changes its course to intersect his own, it is probably an enemy fighter. Similarly, a geometry student might deduce that two triangles are congruent because they share certain sides and vertices, and a general might infer that, since he has received no recent reports of enemy movement, a nearby opposing force is still camped where it was the day before.

To support such reasoning, a cognitive architecture must first be able to represent relationships among beliefs. A common formalism for encoding such relationships is first-order logic, but many other notations have also been used, ranging from production rules to neural networks to Bayesian networks. The relations represented in this manner may be logically or probabilistically sound, but this is not required; knowledge about reasoning can also be heuristic or approximate and still prove quite useful to an intelligent agent. Equally important, the formalism may be more or less expressive (e.g., limited to propositional logic) or computationally efficient.

Naturally, a cognitive architecture also requires mechanisms that draw inferences using these knowledge structures. Deductive reasoning is an important and widely studied form of inference that lets one combine general and specific beliefs to conclude others that they entail logically. However, an agent can also engage in inductive reasoning, which moves from specific beliefs to more general ones and which can be viewed as a form of learning. An architecture may also support abductive inference, which combines general knowledge and specific beliefs to hypothesize other specific beliefs, as occurs in medical diagnosis. In constrained situations, an agent can simply draw all conclusions that follow from its knowledge base, but more often it must select which inferential knowledge to apply. This raises issues of search closely akin to those in planning tasks, along with issues of learning to make that search more effective.

Reasoning plays an important role not only when inferring new beliefs but when deciding whether to maintain existing ones. To the extent that certain beliefs depend on others, an agent should track the latter to determine whether it should continue to believe the former, abandon it, or otherwise alter its confidence. Such *belief maintenance* is especially important for dynamic environments in which situations may change in unexpected ways, with implications for the agent's behavior. One general response to this issue involves maintaining dependency structures in memory that connect beliefs, which the architecture can use to propagate changes as they occur.

3.7 Execution and Action

Cognition occurs to support and drive activity in the environment. To this end, a cognitive architecture must be able to represent and store motor skills that enable such activity. For example, a mobile ground robot or unmanned air vehicle should have skills or policies for navigating from one place to another, for manipulating its surroundings with effectors, and for coordinating its behavior with other agents on its team. These may be encoded solely in terms of primitive or component actions, but they may also specify more complex multi-step skills or procedures. The latter may take the form of plans that the agent has generated or retrieved from memory, especially in architectures that have grown out of work on problem solving and planning. However, other formulations of motor skill execution, such as closed-loop controllers, have also been explored.

A cognitive architecture must also be able to execute skills and actions in the environment. In some frameworks, this happens in a completely reactive manner, with the agent selecting one or more primitive actions on each decision cycle, executing them, and repeating the process on the next cycle. This approach is associated with closed-loop strategies for execution, since the agent can also sense the environment on each time step. The utilization of more complex skills supports open-loop execution, in which the agent calls upon a stored procedure across many cycles without checking the environment. However, a flexible architecture should support the entire continuum from fully reactive, closed-loop behavior to automatized, open-loop behavior, as can humans.

Ideally, a cognitive architecture should also be able learn about skills and execution policies from instruction and experience. Such learning can take different forms, many of which parallel those that arise in planning and problem solving. For example, an agent can learn by observing another agent's behavior, by successfully achieving its goals, and from delayed rewards. Similarly, it can learn or refine its knowledge for selecting primitive actions, either in terms of heuristic conditions on their application or as a numeric evaluation function that reflects their utility. Alternatively, an agent can acquire or revise more complex skills in terms of known skills or actions.

3.8 Interaction and Communication

Sometimes the most effective way for an agent to obtain knowledge is from another agent, making communication another important ability that an architecture should support. For example, a commander may give orders to, and receive reports from, her subordinates, while a shopper in a flea market may dicker about an item's price with its owner. Similarly, a traveler may ask and receive directions on a street corner, while an attorney may query a defendant about where he was on a particular night. Agents exist in environments with other agents, and there are many occasions in which they must transfer knowledge from one to another.

Whatever the modality through which this occurs, a communicating agent must represent the knowledge that it aims to convey or that it believes another agent intends for it. The content so transferred can involve any of the cognitive activities we have discussed so far. Thus, two agents can communicate about categories recognized and decisions made, about perceptions and actions, about predictions and anomalies, and about plans and inferences. One natural approach is to draw on the representations that result from these activities as the input to, and the output from, interagent communication.

A cognitive architecture should also support mechanisms for transforming knowledge into the form and medium through which it will be communicated. The most common form is spoken or written language, which follows established conventions for semantics, syntax, and pragmatics onto which an agent must map the content it wants to convey. Even when entities communicate with purely artificial languages, they do not have exactly the same mental structures and they must translate content into some external format. One can view language generation as a form of planning and execution, whereas language understanding involves inference and reasoning. However, the specialized nature of language processing makes these views misleading, since the task raises many additional issues.

An important form of communication occurs in conversational dialogues, which require both generation and understanding of natural language, as well as coordination with the other agent in the form of turn taking. Learning is also an important issue in language and other forms of communication, since an architecture should be able to acquire syntactic and semantic knowledge for use at both the sentence and dialogue levels. Moreover, some communicative tasks, like question answering, require access to memory for past events and cognitive activities, which in turn benefits from episodic storage.

3.9 Remembering, Reflection, and Learning

A cognitive architecture can also benefit from capabilities that cut across those described in the previous sections, in that they operate on mental structures produced or utilized by them. Such abilities, which Sloman (2001) refers to as *metamanagement mechanisms*, are not strictly required for an intelligent agent, but their inclusion can extend considerably the flexibility and robustness of an architecture.

One capacity of this sort involves *remembering* – the ability to encode and store the results of cognitive processing in memory and to retrieve or access them later. An agent cannot directly remember external situations or its own physical actions; it can only recall cognitive structures that describe those events or inferences about them. This idea extends naturally to memories of problem solving, reasoning, and communication. To remember any cognitive activity, the architecture must store the cognitive structures generated during that activity, index them in memory, and retrieve them when needed. The resulting content is often referred to as *episodic memories*.

Another capability that requires access to traces of cognitive activity is *reflection*. This may involve processing of either recent mental structures that are still available or older structures that the agent must retrieve from its episodic store. One type of reflective activity concerns the justification or *explanation* of an agent's inferences, plans, decisions, or actions in terms of cognitive steps that led to them. Another revolves around *meta-reasoning* about other cognitive activities, which an architecture can apply to the same areas as explanation, but which emphasizes their generation (e.g., forming inferences or making plans) rather than their justification. To the extent that reflective processes lay down their own cognitive traces, they may themselves be subject to reflection. However, an architecture can also support reflection through less transparent mechanisms, such as statistical analyses, that are not themselves inspectable by the agent.

A final important ability that applies to many cognitive activities is *learning*. We have discussed previously the various forms this can take, in the context of different architectural capacities, but we should also consider broader issues. Learning usually involves generalization beyond specific beliefs and events. Although most architectures carry out this generalization at storage time and enter generalized knowledge structures in memory, some learning mechanisms store specific situations and generalization occurs at retrieval time through analogical or case-based reasoning. Either approach can lead to different degrees of generalization or transfer, ranging from very similar tasks, to other tasks within the same domain, and even to tasks within related but distinct domains. Many architectures treat learning as an automatic process that is not subject to inspection or conscious control, but they can also use meta-reasoning to support learning in a more deliberate manner. The data on which learning operates may come from many sources, including observation of another agent, an agent's own problem solving behavior, or practice of known skills. But whatever the source of experience, all involve processing of memory structures to improve the agent's capabilities.

4. Properties of Cognitive Architectures

We can also characterize cognitive architectures in terms of the internal properties that produce the capabilities described in the previous section. These divide naturally into the architecture's representation of knowledge, the organization it places on that knowledge, the manner in which the system utilizes its knowledge, and the mechanisms that support acquisition and revision of knowledge through learning. Below we consider a number of design decisions that arise within each of these facets of an intelligent system, casting them in terms of the data structures and algorithms that are supported at the architectural level. Although we present most issues in terms of oppositions, many of the alternatives we discuss are complementary and can exist within the same framework.

4.1 Representation of Knowledge

One important class of architectural properties revolves around the representation of knowledge. Recall that knowledge itself is not built into an architecture, in that it can change across domains and over time. However, the representational formalism in which an agent encodes its knowledge constitutes a central aspect of a cognitive architecture.

Perhaps the most basic representational choice involves whether an architecture commits to a single, uniform notation for encoding its knowledge or whether it employs a mixture of formalisms. Selecting a single formalism has advantages of simplicity and elegance, and it may support more easily abilities like learning and reflection, since they must operate on only one type of structure. However, as we discuss below, different representational options have advantages and disadvantages, so that focusing on one framework can force an architecture into awkward approaches to certain problems. On the other hand, even mixed architectures are typically limited to a few types of knowledge structures to avoid complexity.

One common tradition distinguishes *declarative* from *procedural* representations. Declarative encodings of knowledge can be manipulated by cognitive mechanisms independent of their content. For instance, a notation for describing devices might support design, diagnosis, and control. First-order logic (Genesereth & Nilsson, 1987) is a classic example of such a representation. Generally speaking, declarative representations support very flexible use, but they may lead to inefficient processing. In contrast, procedural formalisms encode knowledge about how to accomplish some task. For instance, an agent might have a procedure that lets it solve an algebra problem or drive a vehicle, but not recognize such an activity when done by others. Production rules (Neches et al., 1987) are a common means of representing procedural knowledge. In general, procedural representations let an agent apply knowledge efficiently, but typically in an inflexible manner.

We should clarify that a cognitive architecture can support both declarative and procedural representations, so they are not mutually exclusive. Also, all architectures have some declarative and procedural aspects, in that they require some data structures to recognize and some interpreter to control behavior. However, we typically reserve the term *knowledge* to refer to structures that are fairly stable (not changing on every cycle) and that are not built into the architecture. Moreover, whether knowledge is viewed as declarative or procedural depends less on its format than on what architectural mechanisms can access it. For example, production rules can be viewed as declarative if other production rules can inspect them.

Although much of an agent's knowledge must consist of skills, concepts, and facts about the world it inhabits, an architecture may also support *meta-knowledge* about the agent's own capabilities. Such higher-level knowledge can support meta-reasoning, let the agent "know what it knows", and provide a natural way to achieve cognitive penetrability, that is, an understanding of the cognitive steps taken during the agent's activities and the reasons for them. Encoding knowledge in a declarative manner is one way to achieve meta-knowledge, but an emphasis on procedural representations does not mean an architecture cannot achieve these ends through other means.

Another contrast parallels the common distinction between activities and the entities on which they operate. Most cognitive architectures, because they evolved from theories of problem solving and planning, focus on *skill knowledge* about how to generate or execute sequences of actions, whether in the agent's head or in the environment. However, an equally important facet of cognition is *conceptual knowledge*, which deals with categories of objects, situations, and other less action-oriented concepts. All cognitive architectures refer to such categories, but they often relegate them to opaque symbols, rather than representing their meaning explicitly. There has been considerable work on formalisms and methods for conceptual memory, but seldom in the context of cognitive architectures.

Yet another distinction (Tulving, 1972) involves whether stored knowledge supports a *semantic memory* of generic concepts, procedures, and the like, or whether it encodes an *episodic memory* of specific entities and events the agent has encountered in the environment. Most cognitive architectures focus on semantic memory, partly because this is a natural approach to obtaining the generalized behavior needed by an intelligent agent, whereas an episodic memory seems well suited for retrieval of specific facts and occurrences. However, methods for analogical and case-based reasoning can produce the effect of generalized behavior at retrieval time, so an architecture's commitment to semantic or episodic memory does not, by itself, limit its capabilities. Neither must memory be restricted to one framework or the other.

Researchers in artificial intelligence and cognitive science have explored these design decisions through a variety of specific representational formalisms. An early notation, known as *semantic networks* (Ali & Sowa, 1993; Sowa, 1991), encodes both generic and specific knowledge in a declarative format that consists of nodes (for concepts or entities) and links (for relations between them). *First-order logic* was another early representational framework that still sees considerable use; this encodes knowledge as logical expressions, each cast in terms of predicates and arguments, along with statements that relate these expressions in terms of logical operators like conjunction, disjunction, implication, and negation. *Production systems* (Neches, Langley, & Klahr, 1987) provide a more procedural notation, retaining the modularity of logic, which represent knowledge as a set of condition-action rules that describe plausible responses to different situations. *Frames* (Minsky, 1975) and *schemas* offer structured declarative formats that specify concepts in terms of attributes (slots) and their values (fillers), whereas *plans* (Hendler et al., 1990) provide a structured framework for encoding courses of action. In addition, some approaches augment symbolic structures with strengths (as in neural networks) or probabilities (as in Bayesian networks), although, as typically implemented, these have limited expressiveness.

4.2 Organization of Knowledge

Another important set of properties concerns the manner in which a cognitive architecture organizes knowledge in its memory. One choice that arises here is whether the underlying knowledge representation scheme directly supports 'flat' or hierarchical structures. Production systems and propositional logic are two examples of flat frameworks, in that the stored memory elements make no direct reference to each other. This does not mean they cannot influence one another; clearly, application of one production rule can lead to another one's selection on the next cycle, but this happens indirectly through operation of the architecture's interpreter.

In contrast, stored elements in structured frameworks make direct reference to other elements. One such approach involves a *task* hierarchy, in which one plan or skill calls directly on component tasks, much as in subroutine calls. Similarly, a *part-of* hierarchy describes a complex object or situation in terms of its components and relations among them. A somewhat different organization occurs with an *is-a* hierarchy, in which a category refers to more general concepts (its parents) and more specialized ones (its children). Most architectures commit to either a flat or structured scheme, but task, part-of, and is-a hierarchies are complementary rather than mutually exclusive.

A second organizational property involves the granularity of the knowledge stored in memory. For example, both production systems and first-order logic constitute fairly fine-grained forms of knowledge. An architecture that encodes knowledge in this manner must use its interpreter to compose them in order to achieve complex behavior. Another option is to store more coarse-grained structures, such as plans and macro-operators, that effectively describe multi-step behavior in single knowledge structures. This approach places fewer burdens on the interpreter, but also provides less flexibility and generality in the application of knowledge. A structured framework offers one compromise by describing coarse memory elements in terms of fine-grained ones, thus giving the agent access to both.

Another organizational issue concerns the number of distinct memories that an architecture supports and their relations to each other. An intelligent agent requires some form of *long-term memory* to store its generic skills and concepts; this should be relatively stable over time, though it can change with instruction and learning. An agent also requires some short-term memory that contains more dynamic and short-lived beliefs and goals. In most production system architectures, these two memories are structurally distinct but related through the matching process, which compares the conditions of long-term production rules with short-term structures. Other frameworks treat short-term memory as the active portion of the long-term store, whereas others replace a single short-term memory with a number of modality-specific perceptual buffers. A cognitive architecture may also allocate its stable knowledge to distinct long-term memories, say for procedural, conceptual, and episodic structures, as appears to occur in humans.

4.3 Utilization of Knowledge

A third class of properties concerns the utilization of knowledge stored in long-term memories. As we have seen, this can range from low-level activities like recognition and decision making to high-level ones like communication and reflection. We cannot hope to cover all the design choices that arise in knowledge utilization, so we focus here on issues which deal with cognitive behavior that occurs across cycles, which is typically a central concern of architectural developers.

One such design decision involves whether problem solving relies primarily on heuristic search through problem spaces or on retrieval of solutions or plans from long-term memory. As usual, this issue should not be viewed as a strict dichotomy, in that problem space search itself requires retrieval of relevant operators, but a cognitive architecture may emphasize one approach over the other. For instance, production system architectures typically construct solutions through heuristic search, whereas case-based systems retrieve solutions from memory, though the latter must often adapt the retrieved structure, which itself can require search.

When a cognitive architecture supports multi-step problem solving and inference, it can accomplish this in different ways. One approach, known as *forward chaining*, applies relevant operators and inference rules to the current problem state and current beliefs to produce new states and beliefs. We can view forward chaining as progressing from a known mental state toward some goal state or description. In contrast, *backward chaining* applies relevant operators and inference rules to current goals in order to generate new subgoals, which involves progression from some goal state or description toward current states or beliefs. A third alternative, *means-ends analysis* (e.g., Carbonell et al., 1990; Ernst & Newell, 1969), combines these two approaches by selecting operators through backward chaining but executing them whenever their preconditions are satisfied.

To clarify this dimension, production system architectures typically operate in a forward chaining fashion, while Prolog (Clocksin & Mellish, 1981) provides a good example of backward chaining. However, it is important to distinguish between problem solving techniques that are supported directly by an architecture and ones that are implemented by knowledge stated within that architecture. For instance, backward-chaining behavior can arise within a forward-chaining production system through rules that match against goals and, upon firing, add subgoals to short-term memory (e.g., Anderson & Lebiere, 1998). Such knowledge-driven behavior does not make the architecture itself any less committed to one position or another.

Computer scientists often make a strong distinction between *sequential* and *parallel* processing, but this dichotomy, as typically stated, is misleading in the context of cognitive architectures. Because an intelligent agent exists over time, it cannot avoid some sequential processing, in that it must take some cognitive and physical steps before others are possible. On the other hand, most research on cognitive architectures assumes that retrieval of structures from long-term memory occurs in parallel or at least that it happens so rapidly it has the same effect. However, frameworks can genuinely differ in the number of cognitive structures they select and apply on each cycle. For example, early production system architectures (Newell, 1973b) found all matching instantiations of rules on each cycle, but then selected only one for application; in contrast, some more recent architectures like Soar (Newell, 1990) apply all matching rules, but introduce constraints elsewhere, as in the number of goals an agent can simultaneously pursue. Thus, architectures differ not so much in whether they support sequential or parallel processing, but in where they place sequential bottlenecks and the details of those constraints. Some architectures, like ACT-R (Anderson et al., 2004) model cognitive bottlenecks in order to simulate limitations on human performance.

Given that a cognitive architecture has some resource limitations which require selection among alternative goals, rules, or other knowledge structures, it needs some way to make this selection. Early production system architectures handled this through a process known as *conflict resolution*, which selected one or more matched rules to apply based on criteria like the recency of their matched elements, the rules' specificities, or their strength. Computer programs for game playing instead select moves with some numeric evaluation function that combines features of predicted states, whereas systems that incorporate analogical or case-based reasoning typically select structures that are most similar to some target. Again, it is important to distinguish the general mechanism an architecture uses to select among alternative decisions or actions from the knowledge it uses to implement that strategy, which may differ across tasks or change with learning.

Another central issue for the utilization of knowledge concerns the relation between cognition and action. A *deliberative* architecture is one that plans or reasons out a course of action before it begins execution, whereas a *reactive* architecture simply selects its actions on each decision cycle based on its understanding of the current situation. Deliberation has advantages in predictable environments, but it requires an accurate model of actions' effects and forces the agent to construct a plan for each new problem it encounters. Reaction has advantages in dynamic and unpredictable environments, but requires the presence of control knowledge for many different situations. Some architectures (e.g., Carbonell et al., 1990) lean toward deliberation because they grew out of research on problem solving and planning, whereas other frameworks (e.g., Brooks, 1986) emphasize reactive execution to the exclusion of deliberation. Both positions constitute extremes along a continuum that, in principle, should be controlled by agent knowledge rather than built into the architecture.[2]

A similar issue arises with respect to the relation between perception and action (Schmidt, 1975). A *closed-loop* control system senses the environment on every cycle, thus giving an agent the opportunity to respond to recent changes. In contrast, an *open-loop* system carries out an extended action sequence over multiple cycles, without bothering to sense the environment. Closed-loop approaches are often associated with reactive systems and open-loop methods with deliberative ones, but they really involve distinct issues. Closed-loop control has the advantage of rapid response in dynamic domains, but requires constant monitoring that may exceed an agent's perceptual resources. Open-loop behavior requires no sensing and supports efficient execution, but it seems most appropriate only for complex skills that necessitate little interaction with the environment.

[2] Another response is to support deliberation and reactive control in separate modules, as done in Bonasso et al.'s (1997) 3T framework.

Again, these two extremes define a continuum, and an architecture can utilize domain knowledge to determine where its behavior falls, rather than committing to one or the other.

4.4 Acquisition and Refinement of Knowledge

A final important class of properties concerns the acquisition of knowledge from instruction or experience. Although such learning mechanisms can be called intentionally by the agent and carried out in a deliberative fashion, both their invocation and execution are typically handled at the architectural level, though the details vary greatly. One important issue is whether a cognitive architecture supports many such mechanisms or whether it relies on a single learning process that (ideally) interacts with knowledge and experience to achieve many different effects. For instance, early versions of ACT included five distinct learning processes, whereas early versions of Soar included only one such mechanism.

The literature on cognitive architectures commonly distinguishes between processes that learn entirely new knowledge structures, such as production rules or plans, and ones that fine tune existing structures, say through adjusting weights or numeric functions. For example, Soar learns new selection, rejection, or preference rules when it creates results in a subgoal, whereas ACT-R updates the utilities associated with production rules based on their outcomes. An architectural learning mechanism may also revise existing structures by adding or removing components. For instance, early versions of ACT included a discrimination method that added conditions to production rules and a generalization method that removed them.

Another common distinction involves whether a given learning process is analytical or empirical in nature (Schlimmer & Langley, 1992). Analytical methods rely on some form of form of reasoning about the learning experience in terms of knowledge available to the agent. In contrast, empirical methods rely on inductive operations that transform experience into usable knowledge based on detected regularities. In general, analytical methods are more explanatory in flavor and empirical methods are more descriptive. This is actually a continuum rather than a dichotomy, in which the critical variable is the amount of knowledge-based processing the learner carries out. Architectures can certainly utilize hybrid methods that incorporate ideas from both frameworks, and they can also combine them through different learning mechanisms. For example, Prodigy utilizes an analytic method to construct new rules and an empirical method to estimate their utility after gaining experience with them.

A fourth issue concerns whether an architecture's learning mechanisms operate in an *eager* or a *lazy* fashion. Most frameworks take an eager approach that forms generalized knowledge structures from experience at the time the latter enter memory. The interpreter can then process the resulting generalized rules, plans, or other structures without further transformation. Methods for rule induction and macro-operator construction are good examples of this approach. However, some architectures take a lazy approach (Aha, 1997) that stores experiences in memory untransformed, then carry out implicit generalization at the time of retrieval and utilization. Analogical and case-based methods (e.g., Veloso, & Carbonell, 1993) are important examples of this approach.

A final property revolves around whether learning occurs in an incremental or nonincremental manner. Incremental methods incorporate training cases one at a time, with limited memory for previous cases, and update their knowledge bases after processing each experience. In contrast, nonincremental methods process all training cases in a single step that operates in a batch procedure. Because agents exist over time, they accumulate experience in an online fashion, and their learning mechanisms must deal with this constraint. Incremental methods provide a natural response, but the order of presentation can influence their behavior (Langley, 1995b). Nonincremental approaches avoid this drawback, but only at the expense of retaining and reprocessing all experiences. Most architectural research takes an incremental approach to learning, though room remains for hybrid methods that operate over limited subsets of experience.

5. Evaluation Criteria for Cognitive Architectures

As with any scientific theory or engineered artifact, cognitive architectures require evaluation. However, because architectural research occurs at the systems level, it poses more challenges than does the evaluation of component knowledge structures and methods. In this section, we consider some dimensions along which one can evaluate cognitive architectures. In general, these involve matters of degree, which suggests the use of quantitative measures rather than all-or-none tests. Langley and Messina (2004) discuss additional issues that arise in the evaluation of integrated intelligent systems.

Recall that ability to explain psychological phenomena is an important dimension along which to evaluate architectures. For example, in recent years, research within a number of architectural frameworks (Anderson et al., 2004; Sun et al., 2001) has emphasized fitting timing and error data from detailed psychological experiments, but that is not our focus here. However, it is equally important to demonstrate that an architecture supports the same qualitative robustness that humans exhibit. The criteria we discuss in this section are based directly on such qualitative aspects of human behavior, even when a system may produce them through entirely different means.

Cognitive architectures also provide a distinctive approach to constructing integrated intelligent systems. The convential wisdom of software engineering is that one should develop independent modules that have minimal interaction. In contrast, a cognitive architecture offers a *unified* theory of cognition (Newell, 1990) with tightly interleaved modules that support synergistic effects. However, claims about synergy in cognitive systems are difficult to test empirically,[3] so here we focus on other criteria that are linked directly to functionality.

5.1 Generality, Versatility, and Taskability

Recall that cognitive architectures are intended to support general intelligent behavior. Thus, *generality* is a key dimension along which to evaluate a candidate framework. We can measure an architecture's generality by using it to construct intelligent systems that are designed for a diverse set of tasks and environments, then testing its behavior in those domains. The more environments in which the architecture supports intelligent behavior, and the broader the range of those environments, the greater its generality.

[3] Langley and Choi (2006) provide qualitative arguments that their Icarus framework benefits from interactions among its modules, but even evidence of this sort is rare.

However, demonstrating the generality of an architecture may require more or less effort on the part of the system developer. For each domain, we might implement a new system in low-level assembly code, which makes few theoretical commitments or high-level mechanisms, but this approach would take much too long. We can define the *versatility* of a cognitive architecture in terms of the difficulty encountered in constructing intelligent systems across a given set of tasks and environments. The less effort it takes to get an architecture to produce intelligent behavior in those environments, the greater its versatility.

Generality and versatility are related to a third notion, the *taskability* of an architecture, which acknowledges that long-term knowledge is not the only determinant of an agent's behavior in a domain. Briefly, this concerns an architecture's ability to carry out different tasks in response to goals or other external commands from a human or from some other agent. The more tasks an architecture can perform in response to such commands, and the greater their diversity, the greater its taskability. This in turn can influence generality and versatility, since it can let the framework cover a wider range of tasks with less effort on the developer's part.

5.2 Rationality and Optimality

We usually consider an agent to be intelligent when it pursues a behavior for some reason, which makes the *rationality* of an architecture another important dimension for its evaluation. We can measure a framework's rationality by examining the relationship among its goals, its knowledge, and its actions. For instance, Newell (1982) states "If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action". Since an architecture makes many decisions about action over time, we can estimate this sense of rationality by noting the percentage of times that its behavior satisfies the criterion.

Note that this notion of rationality takes no position about how to select among multiple actions that are relevant to the agent's goals. One response to this issue comes from Anderson (1991), who states "The cognitive system optimizes the adaptation of the behavior of the organism". The notion of *optimality* assumes some numeric function over the space of behaviors, with the optimal behavior being the one that produces the best value on this function. Although optimality is an all-or-none criterion, we can measure the degree to which an architecture approaches optimality by noting the percentage of times its behavior is optimal across many decision cycles or the ratio of actual to optimal value it achives averaged over time.

However, Simon (1957) has argued that, because intelligent agents have limited cognitive resources, the notion of *bounded rationality* is more appropriate than optimality for characterizing their behavior. In his view, an agent has bounded rationality if it behaves in a manner that is as nearly optimal with respect to its goals as its resources will allow. We can measure the degree to which a cognitive architecture exhibits bounded rationality in the same manner as for optimality, provided we can incorporate some measure of the resources it has available for each decision.

5.3 Efficiency and Scalability

Because cognitive architectures must be used in practice, they must be able to perform tasks within certain time and space constraints. Thus, *efficiency* is another important metric to utilize when evaluating an architecture. We can measure efficiency in quantitative terms, as the time and space required by the system, or in all-or-none terms, based on whether the system satisfies hard constraints on time and space, as in work on real-time systems. We can also measure efficiency either at the level of the architecture's recognize-act cycle or at the level of complete tasks, which may give very different results.

However, because architectures must handle tasks and situations of different difficulty, we also want to know their *scalability*. This metric is closely related to the notion of complexity as used in the formal analysis of algorithms. Thus, we can measure an architecture's space and time efficiency in terms of how they are influenced by task difficulty, environmental uncertainty, length of operation, and other complicating factors. We can examine an architecture's complexity profile across a range of problems and amounts of knowledge. The less an architecture's efficiency is affected by these factors, the greater its scalability.

A special case of scalability that has received considerable attention arises with cognitive architectures that learn over time. As learning mechanisms add knowledge to their long-term memory, many such systems become slower in their problem-solving behavior, since they have more alternatives from which to choose. This *utility problem* (Minton, 1990) has arisen in different architectures that employ a variety of representational formalisms and retrieval mechanisms. Making architectures more scalable with respect to such increased knowledge remains an open research issue.

5.4 Reactivity and Persistence

Many cognitive architectures aim to support agents that operate in external environments that can change in unpredictable ways. Thus, the ability to react to such changes is another dimension on which to evaluate candidate frameworks. We can measure an architecture's *reactivity* in terms of the speed with which it responds to unexpected situations or events, or in terms of the probability that it will respond on a given recognize-act cycle. The more rapidly an architecture responds, or the greater its chances of responding, the greater its reactivity.[4]

Of course, this definition must take into account the relation between the environment and the agent's model of that environment. If the model predicts accurately what transpires, then reactivity becomes less of an issue. But if the environment is an uncertain one or if the agent has a weak model, then reactivity becomes crucial to achievement of the agent's goals. Alternative cognitive architectures can take different positions along this spectrum, and we must understand that position when evaluating their reactivities.

An issue related to reactivity that has received substantial attention is known as the *frame problem* (McCarthy, 1963). This arises in any dynamic environment where an agent must keep its model of the world aligned with the world itself, despite the inability of the agent to sense the world in its entirety. Even when it is not hard to detect environmental changes themselves, propagating the effect of these changes on knowledge, goals, and actions can be difficult. Many research efforts have addressed the frame problem, but making architectures more robust on this front remains an open area for research.

[4] The notion of interruptability is closely related to reactivity, but is associated primarily with architectures that deliberate or pursue explicit plans, which can be interrupted when unexpected events occur.

Despite the importance of reactivity, we should note that, in many contexts, *persistence* is equally crucial. An architecture that always responds immediately to small environmental changes may lose sight of its longer-term objectives and oscillate from one activity to another, with no higher purpose. We can measure persistence as the degree to which an architecture continues to pursue its goals despite changes in the environment. Reactivity and persistence are not opposites, although they may appear so at first glance. An agent can react to short-term changes while still continuing to pursue its long-term objectives.

5.5 Improvability

We expect intelligent agents to improve their behavior over time. One means to this end involves direct addition of knowledge by the system's programmer or user. The key question here is not whether such additions are possible, but how effective they are at improving the agent's behavior. Thus, we can measure *improvability* of this type in terms of the agent's ability to perform tasks that it could not handle before the addition of knowledge. More specifically, we can measure the rate at which performance improves as a function of programmer time, since some architectures may require less effort to improve than others.

Another path to improvement involves the agent learning from its experience with the environment or with its own internal processes. We can measure an architecture's capacity for learning in the same way that we can measure its capacity for adding knowledge – in terms of its ability to perform new tasks. Since cognitive agents exist over time, this means measuring their improvement in performance as a function of experience. Thus, the method commonly used in machine learning of separating training from test cases makes little sense here, and we must instead collect learning curves that plot performance against experience in an online setting.

We should note that different forms of learning focus on different types of knowledge, so we should not expect a given mechanism to improve behavior on all fronts. For example, some learning processes are designed to improve an agent's ability to recognize objects or situations accurately, others focus on acquisition of new skills, and still others aim to make those skills more efficient. We should use different tests to evaluate an architecture's ability to learn different types of knowledge, although we would expect a well-rounded architecture to exhibit them all.

Because learning is based on experience with specific objects or events, evaluating the generality, transfer, and reusability of learned knowledge is also crucial. We want learning to involve more than memorizing specific experiences, though such episodic memory also has its uses. We can determine the degree of generalization and transfer by exposing the agent to situations and tasks that differ from its previous experience in various ways and measuring its performance on them. Again, a key issue concerns the rate of learning or the amount of acquired knowledge that the architecture needs to support the desired behavior.

5.6 Autonomy and Extended Operation

Although we want intelligent agents that can follow instructions, sometimes we also expect them to operate on their own over extended periods. To this end, the architectures that support them must be able to create their own tasks and goals. Moreover, they must be robust enough to keep from failing when they encounter unexpected situations and to keep from slowing down as they accumulate experience over long periods of time. In other words, a robust architecture should provide both *autonomy* and *extended operation*.

We can measure an architecture's support for autonomy by presenting agents with high-level tasks that require autonomous decision making for success and that benefit from knowledge about the domain. For example, we can provide an agent with the ability to ask for instructions when it does not know how to proceed, then measure the frequency with which it requests assistance as a function of its knowledge. We can measure the related ability for extended operation by placing an agent in open-ended environments, such as a simulated planetary expedition, and noting how long, on average, it continues before failing or falling into inaction. We can also measure an agent's efficiency as a function of its time in the field, to determine whether it scales well along this dimension.

6. Open Issues in Cognitive Architectures

Despite the many conceptual advances that have occurred during three decades of research on cognitive architectures, and despite the practical use that some architectures have seen on real-world problems, there remains considerable need for additional work on this important topic. In this section, we note some open issues that deserve attention from researchers in the area.

The most obvious arena for improvement concerns the introduction of new capabilities. Existing architectures exhibit many of the capacities described in Section 3, but few support all of them, and even those achieve certain functionalities only with substantial programmer effort. Some progress has been made on architectures that combine deliberative problem solving with reactive control, but we need increased efforts at unification along a number of other fronts:

- Most architectures emphasize the generation of solutions to problems or the execution of actions, but categorization and understanding are also crucial aspects of cognition, and we need increased attention to these abilities.
- The focus on problem solving and procedural skills has drawn attention away from episodic knowledge. We need more research on architectures that directly support both episodic memory and reflective processes that operate on the structures it contains.
- Most architectures emphasize logic or closely related formalisms for representing knowledge, whereas humans also appear to utilize visual, auditory, diagrammatic, and other specialized representational schemes. We need extended frameworks that can encode knowledge in a variety of formalisms, relate them to each other, and use them to support intelligent behavior more flexibly and effectively.
- Although natural language processing has been demonstrated within some architectures, few intelligent systems have combined this with the ability to communicate about their own decisions, plans, and other cognitive activities in a general manner.
- Physical agents have limited resources for perceiving the world and affecting it, yet few architectures address this issue. We need expanded frameworks that manage an agent's resources to selectively focus its perceptual attention, its effectors, and the tasks it pursues.

- Although many architectures interface with complex environments, they rarely confront the interactions between body and mind that arise with real embodiment. For instance, we should examine the manner in which physical embodiment impacts thinking and consider the origin of agents' primary goals in terms of internal drives.
- Emotions play a central role in human behavior, yet few systems offer any account of their purposes or mechanisms. We need new architectures that exhibit emotion in ways that link directly to other cognitive processes and that modulate intelligent behavior.
- From an engineering standpoint, architectures are interesting if they ease development of intelligent agents through reuse, but we need research on whether this is best accomplished through specialized functional capabilities that are utilized repeatedly or through reusable knowledge that supports multiple tasks.
- From an engineering standpoint, architectures are primarily interesting if they can ease development of intelligent agents. To that end, reusability is key, but it is not clear if the architectures themselves need to support specialized capabilities that can be reused, or if it is possible to develop reusable knowledge that supports multiple tasks.

Architectures that demonstrate these new capabilities will support a broader class of intelligent systems than the field has yet been able to develop.

We also need additional research on the structures and processes that support such capabilities. Existing cognitive architectures incorporate many of the underlying properties that we described in Section 4, but a number of issues remain unaddressed.

- Certain representational frameworks – production systems and plans – have dominated architectural research. To explore the space of architectures more fully, we should also examine designs that draw on other representational frameworks like frames (Minsky, 1975), case bases (Aamodt & Plaza, 1994), description logics (Nardi & Brachman, 2002), and probabilistic formalisms (Richardson & Domingos, 2006).
- Many architectures commit to a single position on properties related to knowledge utilization, but this is not the only alternative. We should also explore frameworks that change their location on a given spectrum (e.g., deliberative vs. reactive behavior) dynamically based on their situation.
- Most architectures incorporate some form of learning, but none have shown the richness of improvement that humans demonstrate. We need more robust and flexible learning mechanisms that are designed for extended operation in complex, unfamiliar domains and that build in a cumulative manner on the results of previous learning over long periods of time.

These additional structures and processes should both increase our understanding of the space of cognitive architectures and provide capabilities that are not currently available.

The research community should also devote more serious attention to methods for the thoughtful evaluation of cognitive architectures. Metrics like those we proposed in Section 5 are necessary but not sufficient to understand scientifically the mapping from architectural properties to the capabilities they support. In addition, we must identify or create complex environments, both physical and simulated, that exercise these capabilities and provide realistic opportunities for measurement.

We will also need an experimental method that recognizes the fact that cognitive architectures involve integration of many components which may have synergistic effects, rather than consisting of independent but unrelated modules (Langley & Messina, 2004). Experimental comparisons among architectures have an important role to play, but these must control carefully for the task being handled and the amount of knowledge encoded, and they must measure dependent variables in unbiased and informative ways. Systematic experiments that are designed to identify sources of power will tell us far more about the nature of cognitive architectures than simplistic competitions.

Our field still has far to travel before we understand fully the space of cognitive architectures and the principles that underlie their successful design and utilization. However, we now have over two decades' experience with constructing and using a variety such architectures for a wide range of problems, along with a number of challenges that have arisen in this pursuit. If the scenery revealed by these initial steps are any indication, the journey ahead promises even more interesting and intriguing sites and attractions.

Acknowledgments

Production of this paper was supported, in part, by Grant HR0011-04-1-0008 from DARPA IPTO and by Grant IIS-0335353 from NSF. The document borrows greatly, in both organization and content, from the University of Michigan Web site at `http://ai.eecs.umich.edu/cogarch0/`, which was authored by R. Wray, R. Chong, J. Phillips, S. Rogers, B. Walsh, and J. E. Laird. We thank Ron Brachman for convincing us that the paper was needed and encouraging us to write it.

References

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, *7*, 39–59.

Aha, D. W. (1997). *Lazy learning*. Dordrecht, Germany: Kluwer.

Albus, J. S., Pape, C. L., Robinson, I. N., Chiueh, T.-C., McAulay, A. D., Pao, Y.-H., & Takefuji, Y. (1992). RCS: A reference model for intelligent control. *IEEE Computer*, *25*, 56–79.

Ali, S. S. & Shapiro, S. C. (1993). Natural language processing using a propositional semantic network with structured variables. *Minds and Machines*, *3*, 421–451.

Anderson, J. R. (1991). Cognitive architectures in a rational analysis. In K. VanLehn (Ed.), *Architectures for intelligence*. Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R. (2007). *How can the human mind exist in the physical universe?*. New York: Oxford University Press.

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Lawrence Erlbaum.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, (4). 1036–1060.

Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D., & Slack, M. (1997). Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, *9*, 237–256.

Bratman, M. E. (1987). *Intentions, plans, and practical reason.* Cambridge, MA: Harvard University Press.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, *RA-2*, 14–23.

Carbonell, J. G., Knoblock, C. A., & Minton, S. (1990). Prodigy: An integrated architecture for planning and learning. In K. Van Lehn (Ed.), *Architectures for intelligence*. Hillsdale, NJ: Lawrence Erlbaum.

Choi, D., Konik, T., Nejati, N., Park, C., & Langley, P. (2007). A believable agent for first-person shooter games. *Proceedings of the Third Annual Artificial Intelligence and Interactive Digital Entertainment Conference* (pp. 71–73). Stanford, CA: AAAI Press.

Clocksin, W. F., & Mellish, C. S. (1981). *Programming in* Prolog. Berlin: Springer-Verlag.

Drummond, M., Bresina, J., & Kedar, S. (1991). The Entropy Reduction Engine: Integrating planning, scheduling, and control. *SIGART Bulletin*, *2*, 61–65.

Ernst, G., & Newell, A. (1969). *GPS: A case study in generality and problem solving*. New York: Academic Press.

Fikes, R., Hart, P. E., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, *3*, 251–288.

Firby, R. J. (1994). Task networks for controlling continuous processes. *Proceedings of the Second International Conference on AI Planning Systems* (pp. 49–54). Chicago: AAAI Press.

Freed, M. (1998). Managing multiple tasks in complex, dynamic environments. *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (pp. 921–927). Madison, WI: AAAI Press.

Gat, E. (1991). Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots. *SIGART Bulletin*, *2*, 17–74.

Genesereth, M. R., & Nilsson, N. J. (1987). *Logical foundations of artificial intelligence*. Los Altos, CA: Morgan Kaufmann.

Gratch, J. (2000). Emile: Marshalling passions in training and education. *Proceedings of the Fourth International Conference on Autonomous Agents* (pp. 325–332). Barcelona, Spain.

Haigh, K., & Veloso, M. (1996). Interleaving planning and robot execution for asynchronous user requests. *Proceedings of the International Conference on Intelligent Robots and Systems* (pp. 148–155). Osaka, Japan: IEEE Press.

Hayes-Roth, B., Pfleger, K., Lalanda, P., Morignot, P., & Balabanovic, M. (1995). A domain-specific software architecture for adaptive intelligent systems. *IEEE Transactions on Software Engineering*, *21*, 288–301.

Hendler, J., Tate, A., & Drummond, M. (1990). AI planning: Systems and techniques. *AI Magazine*, *11*, 61–77.

Ingrand, F. F., Georgeff, M. P., & Rao, A. S. (1992). An architecture for real-time reasoning and system control. *IEEE Expert*, *7*, 34–44.

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, *8*, 30–43.

Konolige, K., Myers, K. L., Ruspini, E. H., & Saffiotti, A. (1997). The Safira architecture: A design for autonomy. *Journal of Experimental & Theoretical Artificial Intelligence*, *9*, 215–235.

Laird, J. E. (1991). Preface for special section on integrated cognitive architectures. *SIGART Bulletin*, *2*, 12–123.

Laird, J. E. (2008). Extending the Soar cognitive architecture. *Proceedings of the Artificial General Intelligence Conference*. Memphis, TN: IOS Press.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Langley, P. (1995a). *Elements of machine learning*. San Francisco: Morgan Kaufmann.

Langley, P. (1995b). Order effects in incremental learning. In P. Reimann & H. Spada (Eds.), *Learning in humans and machines: Towards and interdisciplinary learning science*. Oxford: Elsevier.

Langley, P. (2006). *Intelligent behavior in humans and machines* (Technical Report). Computational Learning Laboratory, CSLI, Stanford University, CA.

Langley, P., & Choi, D. (2006a). A unified cognitive architecture for physical agents. *Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence*. Boston: AAAI Press.

Langley, P., & Choi, D. (2006b). Learning recursive control programs from problem solving. *Journal of Machine Learning Research*, *7*, 493–518.

Langley, P., Cummings, K., & Shapiro, D. (2004). Hierarchical skills and cognitive architectures. *Proceedings of the Twenty-Sixth Annual Conference of the Cognitive Science Society* (pp. 779–784). Chicago, IL.

Langley, P., & Messina, E. (2004). Experimental studies of integrated cognitive systems. *Proceedings of the Performance Metrics for Intelligent Systems Workshop*. Gaithersburg, MD.

Lewis, R. L. (1993). An architecturally-based theory of sentence comprehension. *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society* (pp. 108–113). Boulder, CO: Lawrence Erlbaum.

Magerko, B., Laird, J. E., Assanie, M., Kerfoot, A., & Stokes, D. (2004). AI characters and directors for interactive computer games. *Proceedings of the Sixteenth Innovative Applications of Artificial Intelligence Conference* (pp. 877-884). San Jose, CA: AAAI Press.

McCarthy, J. (1963). *Situations, actions and causal laws* (Memo 2). Artificial Intelligence Project, Stanford University, Stanford, CA.

Meyer, M., & Kieras, D. (1997). A computational theory of executive control processes and human multiple-task performance: Part 1. Basic mechanisms. *Psychological Review*, *104*, 3–65.

Miller, C. S., & Laird, J. E. (1996). Accounting for graded performance within a discrete search framework. *Cognitive Science*, *20*, 499–537.

Minsky, M. (1975). A framework for representing knowledge. In P. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill.

Minton, S. N. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, *42*, 363–391.

Muscettola, N., Nayak, P. P., Pell, B., & Williams, B. (1998). Remote Agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, *103*, 5–48.

Musliner, D. J., Goldman, R. P., & Pelican, M. J. (2001). Planning with increasingly complex models. *Proceedings of the International Conference on Intelligent Robots and Systems.*

Nardi, D., & Brachman, R. J. (2002). An introduction to description logics. In F. Baader et al. (Eds.), *Description logic handbook*. Cambridge: Cambridge University Press.

Nason, S., & Laird, J. E. (2004). Soar-RL: Integrating reinforcement learning with Soar. *Proceedings of the Sixth International Conference on Cognitive Modeling* (pp. 220–225).

Neches, R., Langley, P., & Klahr, D. (1987). Learning, development, and production systems. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development.* Cambridge, MA: MIT Press.

Newell, A. (1973a). You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. In W. G. Chase (Ed.), *Visual information processing* New York: Academic Press.

Newell, A. (1973b). Production systems: Models of control structures. In W. G. Chase (Ed.), *Visual information processing*. New York: Academic Press.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, *18*, 87–127.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Nuxoll, A. M. & Laird, J. E. (2007). Extending cognitive architecture with episodic memory. *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*. Vancouver, BC: AAAI Press.

Pell, B., Bernard, D. E., Chien, S. A., Gat, E., Muscettola, N., Nayak, P. P., Wagner, M. D., & Williams, B. C. (1997). An autonomous spacecraft agent prototype. *Proceedings of the First International Conference on Autonomous Agents* (pp. 253–261). Marina del Rey, CA: ACM Press.

Pérez, M. A. & Carbonell, J. G. (1994). Control knowledge to improve plan quality. *Proceedings of the Second International Conference on AI Planning Systems* (pp. 323–328). Chicago: AAAI Press.

Remington, R., Matessa, M., Freed, M., & Lee, S. (2003). Using Apex / CPM-GOMS to develop human-like software agents. *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems.* Melbourne: ACM Press.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning*, *62*, 107–136.

Sammut, C. (1996). Automatic construction of reactive control systems using symbolic machine learning. *Knowledge Engineering Review*, *11*, 27–42.

Schlimmer, J. C., & Langley, P. (1992). Machine learning. In S. Shapiro (Ed.), *Encyclopedia of artificial intelligence* (2nd ed.). New York: John Wiley & Sons.

Schmidt, R. A. (1975). A schema theory of discrete motor skill learning. *Psychological Review*, *82*, 225–260.

Shapiro, D., & Langley, P. (2004). *Symposium on learning and motivation in cognitive architectures: Final report*. Institute for the Study of Learning and Expertise, Palo Alto, CA.
http://www.isle.org/symposia/cogarch/arch.final.pdf

Simon, H. A. (1957). *Models of man*. New York: John Wiley.

Sleeman, D., Langley, P., & Mitchell, T. (1982). Learning from solution paths: An approach to the credit assignment problem. *AI Magazine*, *3*, 48–52.

Sloman, A. (2001). Varieties of affect and the CogAff architecture schema. *Proceedings of the AISB'01 Symposium on Emotion, Cognition, and Affective Computing*. York, UK.

Sowa, J. F. (Ed.). (1991). *Principles of semantic networks: Explorations in the representation of knowledge*. San Mateo, CA: Morgan Kaufmann.

Sun, R. (Ed.). (2005). *Cognition and multi-agent interaction: Extending cognitive mdoeling to social simulation*. Cambridge University Press.

Sun, R. (2007). The importance of cognitive architectures: An analysis based on CLARION. *Journal of Experimental and Theoretical Artificial Intelligence*, *19*, 159–193.

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

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Taatgen, N. A. (2005). Modeling parallelization and speed improvement in skill acquisition: From dual tasks to complex dynamic skills. *Cognitive Science*, *29*, 421–455.

Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P. S., & Schwamb, K. B. (1995). Intelligent agents for interactive simulation environments. *AI Magazine*, *16*, 15–39.

Trafton, J. G., Cassimatis, N. L., Bugajska, M., Brock, D., Mintz, F., & Schultz, A. (2005). Enabling effective human-robot interaction using perspective-taking in robots. *IEEE Transactions on Systems, Man and Cybernetics*, *25*, 460–470.

Tulving, E. (1972). Episodic and semantic memory. In E. Tulving & W. Donaldson (Eds.), *Organization of memory*. New York: Academic Press.

VanLehn, K. (Ed.) (1991). *Architectures for intelligence*. Hillsdale, NJ: Lawrence Erlbaum.

Veloso, M. M., & Carbonell, J. G. (1993). Derivational analogy in Prodigy: Automating case acquisition, storage, and utilization. *Machine Learning*, *10*, 249–278.

Wang, X. (1995). Learning by observation and practice: An incremental approach for planning operator acquisition. *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 549–557). Lake Tahoe, CA: Morgan Kaufmann.

Appendix. Representative Cognitive Architectures

Many researchers have proposed and studied cognitive architectures over the past three decades. Some have been only thought experiments, while others have been implemented and utilized as tools by people at many institutions. Here we review briefly a number of architectures that have appeared in the literature. We have not attempted to be exhaustive, but this set should give readers an idea of the great diversity of research in this area.

- ACT-R (Anderson, 2007; Anderson et al., 2004), the most recent instantiation of the ACT family, includes a declarative memory for facts and a procedural memory consisting of production rules. The architecture operates by matching productions on perceptions and facts, mediated by the real-valued activation levels of objects, and executing them to affect the environment or alter declarative memory. Learning in ACT-R involves creating new facts and productions, as well as updating base activations and utilities associated with these structures.
- The AIS architecture (Hayes-Roth et al., 1995) stores procedural knowledge as a set of behaviors, each with associated triggering conditions, and control plans, which specify temporal patterns of plan steps. These match against, modify, and interact through a declarative memory that stores factual knowledge, intended activities, and traces of the agent's experience. On each cycle, a meta-controller selects among enabled behaviors and selects which ones to execute. The architecture includes a deliberative cognitive layer, which is responsible for situation assessment and planning, and a more rapid physical layer, which handles perception and action in the environment.
- APEX (Freed, 1998) organizes knowledge in hierarchical procedures, with higher-level elements indexed by the task they address and referring to subtasks they invoke. These match against the contents of a perceptual memory, with an agenda selecting tasks that it adds to an agenda. The architecture associates cognitive, perceptual, and motor resources; this can lead to conflicts among tasks on the agenda, which the system resolves by selecting those with highest priority. This can lead to interruption of tasks and later return to them when resources become available.
- CIRCA (Musliner et al., 2001) incorporates a stable memory for possible action, temporal, and event transitions, along with a dynamic memory for specific plans and events. The cognitive subsystem generate a planned course of action, encoded as a nondeterministic finite state graph, starting first with an abstract plan and refining it as appropriate. The architecture passes this structure to a real-time subsystem that operates in parallel with the cognitive subsystem, letting the former execute the plan while the latter attempts to improve it.
- CLARION (Sun et al., 2001) stores both action-centered and non-action knowledge in implicit form, using multi-layer neural networks, and in explicit form, using symbolic production rules. Corresponding short-term memories contain activations on nodes and symbolic elements that the architecture matches against long-term structures. Performance involves passing sensory information to the implicit layer, which generates alternative high-value actions, and to the explicit layer, which uses rules to propose actions; the architecture then selects the candidate with the highest expected value. Learning involves weight revision in the implicit system, using a combination of reinforcement learning and backpropagation to estimate value functions, and

construction of production rules by extraction from the implicit layer, error-driven revision, and instantiation of rule templates.

- CogAff (Sloman, 2001) is an architectural schema or framework designed to support interaction between cognition and affect. Although it does not commit to specific representations, it does posit three distinct levels of processing. A reactive level uses condition-action associations that respond to immediate environmental situations. A deliberative layer operates over mental goals, states, and plans to reason about future scenarios. Finally, metamanagement mechanisms let an agent think about its own thoughts and experiences. Affective experience is linked to interruption of some layers by others, with more sophisticated emotions occurring at higher levels.
- Emile (Gratch, 2000) provides an architectural account of emotions and their effect on behavior. Long-term knowledge includes Strips operators for use in plan generation and construal frames that specify conditions (relating events, expectations, goals, and standards) for eliciting different emotions. As the agent acquires new information about expected events, an appraisal module generates emotions in response, with initial intensity being a function of their probability and importance, but decaying over time. The agent's own emotions focuses efforts of the planning module and biases action selection, while inferences about other agents' emotions guide its dialogue choices.
- The Entropy Reduction Engine (Drummond et al., 1991) includes long-term memories for domain operators that describe the effects of actions, domain and behavioral constraints, situated control rules that propose actions to achieve goals, and reduction rules that decompose complex problems into simpler ones. The architecture uses its operators and constraints to produce temporal projections, which it then compiles into control rules that a recognize-act cycles uses to determine which actions to execute. The projection process is supplemented by a problem reduction module, which uses the decomposition rules to constrain its search. Successful projections lead the system to learn new control rules, whereas prediction failures lead to revision of operators and domain constraints.
- EPIC (Meyer & Kieras, 1997) encodes long-term knowledge as production rules, organized as methods for accomplishing goals, that match against short-term elements in a variety of memories, including visual, auditory, and tactile buffers. Performance involves selecting matched rules and applying them in parallel to move eyes, control hands, or alter the contents of memory. Research on EPIC has included a strong emphasis on achieving quantitative fits to human behavior, especially on tasks that involve interacting with complex devices.
- FORR (Epstein, 1992) includes a declarative memory for facts and a procedural memory represented as a hierarchy of weighted heuristics. The architecture matches perceptions and facts against the conditions of heuristics, with matched structures proposing and rating candidate actions. Execution affects the environment or changes the contents of declarative memory. Learning involves creating new facts and heuristics, adjusting weights, and restructuring the hierarchy based on facts and metaheuristics for accuracy, utility, risk, and speed.
- GLAIR (Shapiro & Ismail, 2003) stores content at a knowledge or cognitive level, a perceptual-motor level, and a sensori-actuator level. The highest layer includes generalized structures that define predicates in logical terms, with abstract concepts and procedures ultimately being

grounded in perceptual features and behavioral routines at the middle layer. The system supports inference, belief revision, planning, execution, and natural language processing, with high-level beliefs being inferred from perceptions and with commands at the sensori-actuator level being derived from the agent's goals and plans.

- Icarus (Langley & Choi, 2006a; Langley et al., 2004) represents long-term knowledge in separate memories for hierarchical skills and concepts, with short-term beliefs, goals, and intentions cast as instances of these general structures. The performance element first infers all beliefs implied by its concepts and its perceptions of the environment, then selects an applicable path through the skill hierarchy to execute. Means-ends problem solving occurs when no skills relevant to the current goal are applicable, whereas learning creates new skills based on traces of successful problem solving.
- PolyScheme (Cassimatis et al., 2004) is a cognitive architecture designed to achieve human-level intelligence by integrating multiple representations, reasoning methods, and problem-solving techniques. Each representation has an associated specialist module that supports forward inference, subgoaling, and other basic operations, which match against a shared dynamic memory with elements that are grounded in perception and action. PolyScheme make a stronger semantic commitment than most architectures, encoding all structures with a basic set of relations about time, space, events, identity, causality, and belief.
- Prodigy (Carbonell et al., 1990) encodes two kinds of long-term structures – domain operators that describe the effects of actions and control rules that specify when the system should select, reject, or prefer a given operator, binding, state, or goal. Short-term structures include descriptions of states and contents of a goal stack. Problem solving involves means-ends analysis, which repeatedly selects an operator to reduce differences between the current goal and state until it finds a sequence that achieves the top-level goal. An explanation-based learning module analyzes problem-solving traces and creates new selection, rejection, and preference rules to reduce search on future tasks. Other modules control search by analogy with earlier solutions, learn operator descriptions from experimentation, and learn to improve the quality of solutions.
- PRS (Ingrand et al., 1992), which stands for Procedural Reasoning System, was an early architecture in the Beliefs-Desires-Intentions paradigm. The framework stores hierarchical procedures with conditions, effects, and ordered steps that invoke subprocedures. Dynamic structures include beliefs about the environment, desirs the agent wants to achieve, and intentions the agent plans to carry out. On each cycle, PRS decides whether to continue executing its current intention or to select a new intention to pursue.
- The Remote Agent architecture (Pell et al., 1998) was developed to control autonomous, mission-oriented spacecraft. Long-term structures include mission goals, possible activities and constraints on their execution, and qualitative models of the spacecraft's components, whereas dynamic structures include plans about which activities to pursue, schedules about when to carry them out, and inferences about the operating or failure modes. The architecture incorporates processes which retrieve high-level goals, generate plans and schedules that should achieve them, execute these schedules by calling low-level commands, monitor the modes of each spacecraft component, and recover in case of failures.

- RCS (Albus et al., 1992) is an architectural framework for developing intelligent physical agents. Expertise resides in a hierachical set of knowledge modules, each with its own long-term and short-term memories. Knowledge representation is heterogeneous, including frames, rules, images, and maps. Modules operate in parallel, with a sensory interpreter examining the current state, a world model predicting future states, value judgement selecting among alternatives, and behavior generation carrying out tasks. Higher-level modules influence their children in a top-down manner, whereas children pass information back up to their parent modules.
- Soar (Laird et al. 1987, Newell, 1990) encodes procedural long-term memory as production rules, whereas working memory memory contains a set of elements with attributes and values. The performance system matches productions against elements in working memory, and generates subgoals automatically when it cannot continue. When processing in the subgoal lets the agent overcome this impasse, the architecture adds a new chunk to long-term memory that summarizes the subgoal processing. In recent versions, episodic and semantic learning store working memory elements as structures in long-term memory, while reinforcement learning alters weights associated with rules that select operators.
- 3T (Bonasso et al., 1997) stores long-term knowledge in three layers or tiers. The lowest level consists of sensori-motor behaviors, which the architecture executes reactively, whereas the middle layer stores reactive action packages (Firby, 1994) that sequence these behaviors. The highest layer contains abstract operators, which a deliberative planner uses to generate a partial-order plan that the middle layer serializes and executes. In addition to this high-level plan, each skill and reactive action package has its own short-term memory. A predecessor of 3T, the Atlantis architecture (Gat, 1991), organized its knowledge and behavior in a very similar manner.