Abstract. An important component of creating future intelligent systems is a cognitive architecture. In the field of artificial intelligence research of cognitive architectures is important, because, inter alia, cognitive architectures are the contrast to “expert systems”, which means that cognitive architectures have the goal to cover a broad variety of domains, instead of focalizing the acquisition of capabilities in slim domains. Therefore, this seminar paper considers the general components of three current cognitive architectures: SOAR, PRODIGY, and ICARUS. The Soar cognitive architecture is characterized by its competence in utilizing a broad spectrum of knowledge types and knowledge levels to solve (sub)problems. Prodigy’s main components are the general problem solving module and various learning modules. Whereas ICARUS contains modules for conceptual inference, goal choosing, and skill learning. Finally, the architectures’ general components are compared regarding the following properties: representation, organization, utilization, and acquisition and refinement of knowledge.
Outline

1. Introduction .................................................................................................................................................. 3
2. Production System Models of Cognition ............................................................................................................. 3
   2.1. Production System ........................................................................................................................................ 3
   2.2. Cognition .................................................................................................................................................... 4
   2.3. Cognitive Architecture ................................................................................................................................. 5
   2.4. The Cognitive Architecture ACT-R .......................................................................................................... 5
   2.5. The Importance of Cognitive Architectures ............................................................................................. 6
3. Three current Cognitive Architectures ................................................................................................................. 6
   3.1. Principles of SOAR ....................................................................................................................................... 6
   3.2. Principles of PRODIGY ............................................................................................................................... 9
   3.3. Principles of ICARUS ................................................................................................................................. 11
4. Comparison of SOAR, PRODIGY, and ICARUS .......................................................................................... 14
   4.1. Representation of Knowledge ..................................................................................................................... 14
   4.2. Organization of Knowledge ....................................................................................................................... 15
   4.3. Utilization of Knowledge ........................................................................................................................... 15
   4.4. Acquisition and Refinement of Knowledge ............................................................................................... 16
5. Conclusion ..................................................................................................................................................... 18

References
1. Introduction

An important constituent of creating future intelligent systems is a cognitive architecture [Duch et al. 2008]. Langley et al. [2009] indicates that “A cognitive architecture specifies the underlying infrastructure for an intelligent system.” Typically, it contains three main components: Short-term and long-term memories, representation of elements, which are incorporated in these memories, and finally functional procedures that incorporates performance and learning mechanisms. [Langley et al. 2009]

A cognitive architecture consists of a framework, which is essential for encoding knowledge. It does not produce behavior by itself. “Moreover, representations of knowledge without an architecture are like programs without a computer – they do nothing.” [Laird 2012] The Architecture necessitates knowledge representations to call and combine them with other representations. Compared to this, an architecture lacking in content is like a computer without software – it is an empty shell. Generating purposeful behavior needs the combination of knowledge and the architecture. [Laird 2012]

Research of cognitive architectures is important in the field of artificial intelligence, because cognitive architectures are the contrast to “expert systems”, which means that cognitive architectures have the goal to cover a broad variety of domains, instead of focalizing the acquisition of capabilities in slim domains [Sun 2007].

Therefore, this seminar work concentrates on considering three cognitive architectures: SOAR, PRODIGY, and ICARUS. These are selected due to their frequent emergence in the literature and their different approaches in creating human behavior [Langley et al. 2009]. The research question is: How far are the cognitive architectures SOAR, PRODIGY, and ICARUS, different regarding its general components?

The first part introduces to production model systems of cognition with a view to production systems, the definition of cognition, and the importance of research on cognitive architectures. Subsequently, the three cognitive architectures (SOAR, PRODIGY, and ICARUS) are considered regarding its general components. This forms the base for the next chapter, which compares these architectures with respect to four properties: Representation, organization, utilization and acquisition & refinement of knowledge. Finally, the conclusion contains a short summary of each architecture’s main components and depicts a brief future prospect to cognitive architectures.

2. Production System Models of Cognition

2.1. Production System

In the recent decades, computer modeling procedures of mental structures and processes are used to phrase and review psychological theories. The central feature of computer models, which present formal models along the lines of mathematical models, is the qualitative symbolic kind of its expression. Concepts and techniques, whose development originates from the field of artificial intelligence, are the fundament of computer models. The general goal setting when using such models is the modeling of specific cognitive activities. The model of production systems, frequently used in this context is the subject of this chapter.

Production systems can be considered as the most elaborated frame concept, in which structure theories of the memory relate to assumptions about procedures of human information
processing in an integrated way. The starting point of the following explanations create the various types of encoding knowledge contents, as they were established in the context of the representation problem in cognitive psychology. The subsequent types of knowledge representations are differentiated as follows [Laubsch 1985]:

- **Declarative knowledge representation**: (symbolic) Descriptions of terms, objects, facts, or situations, which do not contain information about knowledge procedures (of acquisition, change, or application).
- **Procedural knowledge representation**: Procedure descriptions for knowledge construction, connection and application. For instance, techniques to search in graphs or networks.
- **Control knowledge**: Procedure to control the usage of declarative and procedural knowledge sources.

Various application cases require an integrative handling of different knowledge sources. Fundamental for the model architecture of a production system are three system components [Ueckert 1983]:

1. **Working Memory**: Contains declarative knowledge in form of a set of data elements.
2. **Production Memory**: Contains procedural knowledge in form of a set of production rules respectively productions. Simplified one can say a production is a “if-then” formula (implication). The productions left side specifies the conditions, which are compared to the working memory content. These must be satisfied (“true”), so that the action of the production’s right side can be executed.
3. **Interpreter**: Contains the systems control knowledge and controls the information processing operation. Its main tasks are:
   - Evaluation of the productions left side (conditions) considering the working memory contents. The result is a set of applicable productions.
   - Selection respectively decision for the execution of one or more production(s) from the set of applicable productions.
   - Execution of the actions given in the selected production(s).

In the following figure a model architecture of a production system is demonstrated. [Laubsch 1985]

![Figure 1. Model architecture of a production system. (Laubsch 1985)](image)

### 2.2. Cognition

Cognition (lat.: cogiscere; gr.: gignoskein = to know, to percept) has three different connotations according to Funke & Frensch [2006]. First, cognition is related to “a specific
collection of subject areas”. That means it concerns to observable or theoretic approximated phenomena, which are explored and discussed in the field of cognitive psychology. Second, cognition refers to the attempt, to explain and understand intelligent human behavior by means of a cognitive system, which conveys among environmental input and behavior. Both meanings of cognition point out a set of theoretical assumptions, which map the procedures within a cognitive system. Third, cognition is understood as certain methodical approach to explore human behavior.

Moreover, cognition is associated to several issues, such as: memory, perception, attention, pattern recognition, awareness, neurobiology, knowledge representation, cognitive development, language, and reasoning. All these phenomena share that they presuppose the operation of intelligence, especially when intelligence is defined as “the individual’s capability to behave for a specific purpose, to think rational, and to interact with the environment efficiently”. In general, cognitive research follows up with investigating the characteristics of the mental system, which underlies intelligence or rather intelligent behavior. [Funke & Frensch 2006.]

2.3. Cognitive Architecture

Now that we have a rough idea of cognition, cognitive architectures are presented in this chapter. One of the main purposes of cognitive architectures is supporting a wide range of abilities similarly to human capabilities [Duch et al. 2008].

“A cognitive architecture specifies the underlying infrastructure for an intelligent system.” [Langley et al. 2009] Typically it contains the three following main components: Short-term and long-term memories for storing information about the agent’s beliefs, goals, and knowledge. The second main component is the representation of elements, which are incorporated in these memories. The memories’ organization consists of larger-scale mental structures. The third aspect are functional procedures that operate on these mental structures, incorporating the performance mechanism that makes use of these structures and the learning mechanisms that change them.

The knowledge and beliefs encoded in the agent’s memories might not be part of the agent’s architecture, because these contents can alter over time. They behave like different computer programs, which can run on the same computer architecture. In the same way, a single architecture can interpret several distinct knowledge bases. Langley et al. [2009] applies an analogy of building architectures, which are composed of permanent features, such as its foundation, roof and rooms, rather than furniture and devices, which can be rearranged or exchanged. [Langley et al. 2009]

2.4. The Cognitive Architecture ACT-R

The cognitive Architecture ACT-R consists of basic cognitive components which are modeled analogous to the human brain. John R. Anderson can be seen as the ACT-R originator, who is significantly involved into its development until today. ACT-R stands for “Adaptive Control of Thought – Rational”. Rational indicates that this model of thinking is based on normative cost-benefit analyzes. ACT-R’s intention is simulating and furthermore predicting human behavior for a wide scope of cognitive tasks. [Quin et al. 2006]
In this seminar paper the cognitive architecture ACT-R is not considered in more detail. Further information can be found in the seminar paper of Johannes Grünauer, which examines ACT-R.

2.5. The Importance of Cognitive Architectures

The main goal of the scientific area of artificial intelligence and computational intelligence (AI/CI) is creating artificial systems that behave as competent as humans. A substantial assistive device for designing such artificial systems, such as the human mind, are cognitive architectures and therefore important for the scientific area of artificial intelligence. Also, because of the resemblance among human beings and cognitively based intelligent systems, applying cognitive architectures in intelligent system development may facilitate the interaction between artificially intelligent systems and humans.

Moreover, cognitive architectures are the contrast to “expert systems”, which means that cognitive architectures have the goal to cover a broad variety of domains, instead of focalizing the acquisition of capabilities in slim domains. Particularly, intelligent systems within business and industrial applications need a continuous spectrum expansion of intelligent behaviors than separate systems of low functionality. For instance, capabilities for raw image processing, pattern recognition, categorization, reasoning, decision-making, and natural language communications may be needed for such an application. As well as requirements, such as planning, controlling robotic devices, and interacting with other systems and devices. Because of these requirements, it is essential that investigations are done on widely scoped cognitive architectures, able to execute a broad spectrum of cognitive functionalities through multitude task domains. [Sun 2007]

3. Three current Cognitive Architectures

In this chapter, three cognitive architectures, SOAR, PRODIGY, and ICARUS, are presented with the focus on its main components.

3.1. Principles of SOAR

Since the 1980s, the cognitive architecture “Soar” is in continuous development and was developed over time, to implement more abilities. In the first place, Soar included working memory, the decision procedure, procedural memory, and chunking. Although over the years, new modules have been added, many of the important properties of Soar were defined in the early stages in the 1980s. [Laird 2012]

Now Soar involves knowledge-intensive reasoning, reactive execution, hierarchical reasoning, planning, and learning [Laird et al.1987; Laird 2008]. The Soar cognitive architecture is characterized by its competence in utilizing a broad spectrum of knowledge types and knowledge levels to solve (sub)problems, so that behavior in Soar agents occurs through the dynamic combination of available knowledge no matter if the knowledge was coded or captured from experience.

Soars foundation of calculations are goals, problem spaces, states, and operators. Figure 1 shows the Soar Cognitive Architecture, containing primitive memories (represented by square-
edged modules) and processes (represented by round-edged modules). The arrows represent their connections. [Laird 2012]

Figure 2. The diagram of Soar 9 [Laird 2012]

The general components of SOAR are described in the following. [Lehman et al. 2007]

**States and Operators** form the basis structures of the SOAR architecture. All the information about the current situation, involving perception (in the top state), descriptions of the present goals and problem spaces, are included in states. On the other hand, operators are the tools for passing through problem spaces.

**Working Memory (WM)** keeps knowledge, which is relevant to the current situation. It incorporates the state hierarchy and the states’ related operators. From Long-term Memory and motor actions Working Memory retrieves knowledge contents.

**Long-term Memory (LTM)** is the storage unit for domain content. The architecture processes this content to generate behavior. There are three diverse representations of the LTM: Procedural knowledge, encoded as production rules, is about how and when to do things. Basically, it is responsible for controlling behavior. If a rule conditions matches structures in working memory, the rule is fired and it performs its actions. Here SOAR includes a production system as described in chapter 2.1, because of its combination of rules and the working memory.

The second representation of the LTM is semantic knowledge, which is encoded as declarative structures and comprises facts about the world, such as things you “know” and believe that are generally true. The third representation is episodic knowledge, which is encoded as episodes and which are about things you “remember”, such as situations you went through. [Lehman et al. 2007] For example, a simple mobile robot, controlled by Soar, may have episodic memories of its visited places and its perceived objects. Maybe also the programmer implemented semantic knowledge that certain objects offer energy for it. Furthermore, it may have procedural knowledge that, in combination with mental imagery, can navigate in its surroundings to go to a specific place and procedural knowledge that attempts to retrieve related to the current situation and set goals. When the battery level of the robot shrinks, the general procedural knowledge can query the semantic memory by creating a cue in working memory...
that retrieves knowledge for determining that the robot needs to be charged at an electrical loading dock. In a similar way, it can then query the episodic memory for the location of the charge station, which it has recognized earlier. After that it can use its procedural memory and mental imagery to navigate to the electrical loading dock and become charged. Therefore, the knowledge determines the behavior indeed but the architecture allows for it. [Laird 2012]

Creating and adding rules is the approach of programming Soar and these rules are used to select and apply operators [Laird 2015]. During the decision cycle, procedural memory is accessed automatically. In contrast, semantic and episodic memory are accessed consciously through the generation of specific cues in working memory.

**The Perception/Motor Interface** is the method for specifying depictions from the external world to the internal representation in working memory. And then from the internal representation back out to perform an activity in the external world. Perception and action can arise simultaneously with cognition by means of this interface.

**The Decision Cycle** is the basic process in the architecture, which supports cognition. At the kernel of Soar exists an operator selection and application machine. Three phases are determining the decision cycle. The elaboration phase includes parallel access to LTM to elaborate the state, propose new operators, and assess the operators. The end of the elaboration phase and the beginning of the decision phase is defined by the resting state. The decision procedure module evaluates the domain-independent language of operator preferences during the decision phase. Its result is either a modification of the selected operator or a deadlock if the preferences are defective or in contradiction. The application phase follows the decision. In this phase rules fire to change the state.

Impasses indicate a lack of knowledge, nevertheless a possibility for learning. An impasse happens automatically as soon as the selected knowledge by the current state is not satisfactory for the decision procedure to resolve the preferences in working memory to choose an operator. The impasse language definition, as well as the preference language definition, is independent of any domain. The architecture automatically generates a new substate, with the goal resolving the impasse, in case of an impasse occurs. In this manner impasses construct a goal/subgoal hierarchy on the context basis in working memory.

**Four Learning Mechanisms:** Soar includes four architectural learning mechanisms: Chunking, reinforcement learning, episodic memory, and semantic learning.

Chunking, sets up new rules in Long-term Memory automatically, if an impasse formed results. Chunks are rules, which are learned by the model and its procedure is called chunking. Chunking is a compositional learning mechanism, which operates in a deductive manner. A preference for a decision constitutes a deduction from preceding knowledge. The new rule is compound from a “then” part, which consists the deduction, and an “if” part, which contains the knowledge that led to/contributed to the deduction. Furthermore, it accelerates performance and relocates more intensional knowledge, which was fetched in a substate, up to a state where it can be used reactively. To prevent impasses in similar future situations, new rules map the relevant Working Memory elements into Working Memory modifications.

Reinforcement learning is a method to learn from the environment’s feedback, often called “reward”, which can be positive or negative. In a more abstract level, succeeding in a task implies positive reward and failing a task implies negative reward. Reinforcement learning, which is based on experience, adapts prognoses of future rewards. These prognoses are utilized to choose actions that maximize expected rewards in the future. Furthermore, reinforcement learning adapts the rule’s preference values, which assess operators.
Episodic memory is responsible for storing a chronology of experiences. An episode can be employed to respond questions about the past, to forecast the result of possible options for action, or to support keeping in view the stride of long-term goals. An episode is recorded automatically as soon as a problem is solved [Nuxoll & Laird 2004]. Episodic memory does not do any generalization, and is therefore a passive learning method. This means there is no differentiation among what should be retrieved and what should not. Everything will be retrieved from episodes, which may be used for the future.

Semantic learning collects more abstract declarative knowledge [such as general facts cf. Laird 2012]. Structures of semantic memory are more general than the structure of episodes. They are static and unconnected from the location and time when they are learned. [Lehman et al. 2007]

3.2. Principles of PRODIGY

PRODIGY is a cognitive architecture which was developed between the middle 1980s to the late 1990s. Its main components are the general problem solving module and several learning modules. Through experience the architectures performance can be improved. [Carbonell et al. 1991]

In Figure 3 the PRODIGY Architecture with its main components is visualized.

![Figure 3. The PRODIGY Architecture [Carbonell et al. 1991]](image)

PRODIGY’s main components are described by literature as follows:

**Domain Knowledge and Control Knowledge:** Domain rules and control rules are the two main types of PRODIGY’s knowledge. Domain rules encode the conditions, under which actions cause certain effects. The effects are described as addition or elimination of first-order
expressions. These expressions can concern physical actions, influencing the environment and reasoning rules, which are mental. [Langley et al. 2009]

On the other hand, there are control rules to lead the search for a solution [Veloso et al. 1995]. The left side of each control rule corresponds to an applicable condition whereas the right side shows whether to select, reject, or prefer a specific candidate. PRODIGY’s steps to make a control decision, with a given default set of candidates (nodes, goals, operators, or bindings, depending on the decision), are as follows.

First it applies the selection rules to choose a candidates’ partial quantity. If no selection rules are holding, all candidates are incorporated. Then this partial quantity is filtered by rejection rules, which eliminate certain residual candidates. Lastly preference rules are responsible for choosing the most preferred candidates. Proposed that backtracking is required, the next preferred candidate is examined, if a general solution is detected, or until all selected and non-rejected candidates are worked off. [Carbonell et al. 1991]

Operators and effects: Prior being able to apply an operator, the operators precondition expression must be fulfilled. Each operator owns a list of effects that outlines the modifications of the world, if this operator will be applied. The precondition expressions are written as well-formed formulas in a form of predicate logic (negation, conjunction, disjunction, and existential and universal quantification). [Carbonell et al. 1991]

Problem-solving-module: One of PRODIGY’s most important modules is the problem-solving module, which comprises search through a problem space to attain one or more goals. The goals are casted as first-order expressions. This search is based on means-ends analysis, which includes selecting an operator that reduces discrepancies among the current state and the goal. This can result in new subproblems, which in turn have their own current states and goals. On each iteration, PRODIGY’s control rules are used to select an operator, binding set, state, or goal, to reject them, or to prefer some over others. [Langley et al. 2009]

PRODIGY is reliant to precise control rules, which may be learned for specific domains. PRODIGY’s problem solver differs from the most domain independent problem solvers. For example, PRODIGY assumes that the decision management in particular at important decisions, is based on the presence of proper control knowledge instead of using least-commitment search strategy. PRODIGY makes a prompt, arbitrary choice, if essential control rules do not exist to a decision. In case the wrong decision is made, PRODIGY attempts to learn the missing control knowledge. The main idea for PRODIGY’s casual commitment strategy is that control rules should exist for every decision with substantial consequences. If they do not exist, the problem solver should not try to be clever without knowledge; Its cleverness should be a result of learning. [Carbonell et al. 1991]

Learning Modules: PRODIGY’s general problem solver relates to different learning modules. You can see the components demonstrated in the PRODIGY architecture of figure 3. An entire search tree is created by the problem solver. This search tree contains all decisions (true ones and false ones) and the final solution. Each learning module utilizes this information differently.

APPRENTICE is a graphic-based user interface, which enables the user to evaluate and control the problem solving and learning process. This interface is connected directly to the central problem solver, which means the interface can receive both domain knowledge and take recommendations how to solve a problem inside the current circumstances.

EBL is an explanation-based learning component [Minton 1988], which extracts control rules from a problem-solving record. Both the domain and important aspects of the problem-solving architecture are described by explanations, which are created from an axiomatized
theory. Afterwards the resulting descriptions are phrased in form of control rules. The control rules that are kept are those, whose utility in search reduction predominate their application effort.

**STATIC** is a method which analyses the PRODIGY domain descriptions (prior solving a problem) to learn control rules. Without any training examples, the STATIC program creates control rules. The core of the STATIC concept is a detailed predictive theory of EBL, which is described in [Etzioni 1993].

**ANALOGY** is an engine for deriving analogies [Carbonell & Veloso 1988] by investigating related earlier on solved problems to resolve new problems. Justifications for each decision, which are recorded continuously by the problem solver, are utilized to reconstruct solutions for subsequent problem solving situations where justifications of the same value hold true. Just like EBL, ANALOGY is an independent mechanism to purchase domain specific control knowledge.

**ALPINE** is an abstraction learning and planning module [Knobloch 1991]. Based on a domain analysis, the axiomatized domain knowledge is seperated into several abstraction levels. When PRODIGY solves a problem it first finds a solution in an abstract space. Then it utilizes the abstract solution to direct the search for solutions in more precise problem spaces.

**EXPERIMENT** is a learning-by-experimentation module to refine domain knowledge that is insufficient or improper specified [Carbonell 1990]. If a divergence between internal expectations and external observations is identified by the plan execution monitoring, the experimentation is released. Experimentations are employed with the purpose to enhance the factual domain knowledge, rather than the control knowledge. [Carbonell et al. 1991]

**Decision procedure:** The two components of a problem are initial state and goal expression. PRODIGY must find a sequence of certain operators to solve a problem. This means an operator sequence must be found, that if applied to the initial state, creates an end state meeting the goal expression. At the beginning the search tree consists of only one node. This node contains the initial state and the goal expression. Later, each node consists of a state describing the world and a set of goals. A tree expansion happens by iterating the following two steps.

1. Decision phase: PRODIGY decides which goal to process, which node to expand by applying a depth-first expansion, which operator to apply, and which objects to use.

2. Expansion phase: If the preconditions of the instantiated operator are fulfilled, the operator is applied. If not, PRODIGY creates subgoals for the unmatched preconditions. However, in both cases, a new node with updated information about the state or the subgoal is generated. After creating a node whose state meets the goal expression of the highest level, the search closes. These decisions can be influenced by control rules, to rise the efficiency of the problem solver’s search and to enhance the quality of the found solutions. [Carbonell et al. 1991]

### 3.3. Principles of ICARUS

ICARUS is a cognitive architecture for controlling an intelligent agent in a complex physical environment. This architecture was designed to achieve its goals through manipulation of other objects and navigation amongst locations. For example, the agent picks up a cup and puts it in a rubbish can (manipulation), or it goes through the door, down the hall, and into another room.

Manipulation and navigation necessitates several abilities, as identifying physical objects, places, and situations, to produce plans that attains goals, to perform action sequences that implement plans, and to perceive situations that demand an adjustment in plans. ICARUS
provides all these manners as well as gaining and storing new knowledge from experience, and handling learning issues and memory organization. [Langley et al. 1991]

In figure 4 ICARUS’ memories and processes of the ICARUS cognitive architecture are presented and explained in the following.

**Long-term memory**: ICARUS stores all knowledge types in a long-term memory, which consists of a hierarchy of probabilistic concepts. This structure coordinates concepts at different abstraction levels, with specific cases at the final nodes and more generalized concepts at internal nodes. For each experience, the probability of class inclusivity is specified by each node in the hierarchy. Also, a set of attributes or roles are specified by each concept node, as well as the values that fill each attribute or role, and the probability of each value given inclusivity in the concept.

**Composite concepts**: All concepts in ICARUS are based on detectable attributes, such as length is 0.37 feet, which are applied to describe primitive objects. Physical objects are described in this manner. This is one type of ICARUS’ knowledge. ICARUS expects that a composite object consists of several components, which again may have components. For example, a person has got a head, and legs, and arms and the arm consists of a hand, forearm, and an upper arm. A composite concept, such as the person class, is represented by several roles, each defines a set of alternative components, relations between them, and their concerning probabilities. Location concepts are represented in the same way, this means locations are characterized in the form of component objects and relative places.

**Qualitative states**: As well knowledge about events, plans, and actions is represented by a similar approach as composite concepts. However, this type of knowledge includes changes over time. Thus, ICARUS utilizes qualitative states [Forbus 1985] to describe them. Every qualitative state defines a set of objects (potentially with components) as well as a time range during which specific numeric attributes of these objects (for example place, speed, or temperature) alter in a constant direction, and the derivations of these attributes. An example could be the case of an agent picking up a cup, which may include three qualitative states: (1) the arm lowers towards the cup, at the same time the hand opens; (2) the hand clasps the cup; and (3) the arm with the cup moves upwards, away from the cup’s previous position. In ICARUS, a problem is represented in form of an initial qualitative state, a desired qualitative state, and the differences amongst both states.
**Plans:** A plan to solve an issue consists of three constituents – an operator (which usually reduces one or more disparities), an initial subplan (with the aim to convert the state into a state satisfying the operator’s preconditions), and a conclusive subplan (to convert the operator’s postconditions into the desired state). This means that plans in ICARUS contain more than an operational sequence of actions, they incorporate a problem-solving track that generated the operational sequence. In turn, operators can be plans, or they can refer to basic actions (for instance lifting an arm), which the agent can control directly. Finally, the conception of abstract plans consists of a probabilistic summary of specific plans, including pointers to their components with related probabilities, such as abstract states, operators, and subplans. For instance, a general plan for picking up an item (plan of manipulation) can contain three subproblems, along the same lines to the event depicted above. The architecture stores route knowledge (navigation plans) with the same approach, insofar that locations act as states and operators represent actions, such as move and turn. [Langley et al. 1991]

Moreover, the ICARUS architecture consists of further three main components: a perceptual system (ARGUS), a planning system (DAEDALUS), and an execution system (MEANDER). To fetch structured experiences from long-term memory, containing objects, states and plans; ARGUS and DAEDALUS invoice the memory system (LABYRINTH).

**LABYRINTH:** At first, the LABYRINTH sorts each component of an experience trough memory. It begins at the root node of the memory hierarchy. The memory system utilizes at each layer an evaluation feature, called category utility [Gluck & Corter 1985]. This evaluation feature is applied to choose the best child node, in which the experience shall be integrated, or to decide whether to store the experience as a separate disjunct, if the experience is new enough. In the first case, the process is repeated in the next layer. In the second case the sorting stops. After the components of an experience are sorted, the LABYRINTH categorizes the composite based on both the categorizations of the components and relations between them. Then it searches the best bindings among components in the concept and those in the experience are searched by the LABYRINTH. If the experience has various layers in its componential structure, this recursive operation proceeds until the top level of the experience has been categorized. For example, to classify a table, LABYRINTH would first categorize the table’s legs and top. After that it would use the concepts, build by these components, and the spatial relations between them, to categorize the table instance itself. Equally LABYRINTH categorizes novel problems. It would first categorize the objects, which are incorporated in the initial states and in the final states. Then it classifies these two states and in conclusion it classifies the general problem description in form of its states and the discrepancies amongst them.

**ARGUS:** The ARGUS module satisfies the ability to percept and identifies objects and events with the purpose of being capable to interact with the environment. This module receives sequences of qualitative states, which are produced by a ‘parser’ subprogram. This subprogram constantly observes objects inside the agent’s sensory range, identifies qualitative breaks, and permanently generates a qualitative description of events and its included objects. ARGUS accepts these descriptions of both the objects and states, applies an attention mechanism similar to Gennari’s CLASSIT [1990], to concentrate on a partial quantity of objects and attributes, and delivers the minimized description to the LABYRINTH module for categorization. A call to LABYRINTH in certain cases can recover not only an abstract state A, but as well a problem with a high priority assigned for which A is the initial state. For example, if the agent is eating...
supper, while it recognizes a tiger enters the room unexpectedly, the current task (eating) priority drops and ARGUS sends a new issue to the active part of memory.

**DAEDALUS:** To generate plans, the DAEDALUS module [Allen & Langley, 1990] applies a variant of means-end analysis. This indicates separating the initial problem (initial state and desired state) into subproblems. Recursively on these subproblems the system invokes itself. To fetch a proper operator or stored plan, based on the problem's preconditions, postconditions, or the distinctions it lowers, DAEDALUS calls LABYRINTH.

However, this results in two subproblems: One subproblem is, transforming the initial state into another state, which fulfills the operators’ precondition. The other subproblem is transforming the state that arises from applying the operator into a state that meets the goal conditions. A solid way to solve a problem of the terminating case of recursion is applying an operator whose preconditions are met. DAEDALUS’ method to react on loops or impasses, is backtracking and demanding another operator, and then triggering a heuristic depth-first search through the means-end scope. In case of ICARUS collected previous experience of an equivalent problem class, the LABYRINTH module will call the whole plan which is related to that class. At this point, DAEDALUS applies a form of derivational analogy, this means it checks each subplan to specify its relevance to the problem or subplan at hand. When appropriate it uses this as a guide and is turning to means-end analysis to handle new facets. [Lehman et al. 2007]

**MEANDER:** Finally, the third component of ICARUS is MEANDER, whose main tasks are executing plans and representing its domain knowledge in terms of motor schemas [Langley et al. 1990]. MEANDER executes a plan, generated from DAEDALUS, as follows: MEANDER passes through the recursive plan structure and conducts each step sequentially. If MEANDER comes across a primitive action in the plan, it performs the related effectors (regulates activity) for the assigned time-period. In case of a blockading complication, MEANDER ultimately finishes execution and directs ICARUS to the desired state the plan was designed for. Indeed, at any given time mistaken expectations can appear, which then may require a rescheduling from the current (unexpected) situation. [Lehman et al. 2007] On this occasion, ARGUS, the perceptual system, has control over the level of observing mistaken expectations during plan execution [Langley et al. 1991].

4. **Comparison of SOAR, PRODIGY, and ICARUS**

Langley et al. [2009] describes that cognitive architectures can be characterized in respect of internal properties that generate capabilities of cognitive architectures. Concerning this matter this chapter compares the three considered cognitive architectures SOAR, PRODIGY, and ICARUS, by means of the following four outlined properties of Langley’s et al. [2009] research: Representation, organization, utilization, and acquisition and refinement of knowledge. Each property in turn is separated into several properties from [Langley et al. 2009]. I will only go into the most relevant properties, to avoid going beyond the scope of this seminar paper.

4.1. **Representation of Knowledge**

The research of Langley et al. [2009] outlines the following several properties of knowledge representation:
- Is the notation for encoding knowledge uniform, or does the architecture utilize a combination of formalisms?
- Is the architecture’s knowledge representation declarative or procedural?
- Does the architecture support skill knowledge or conceptual knowledge?
- Does the architecture contain semantic memory and/or episodic memory?

SOARs knowledge is categorized into three knowledge types: Procedural knowledge encoded as production rules, semantic knowledge encoded as declarative knowledge, and episodic knowledge encoded as episodes. On the other hand, in PRODIGY domain and control rules build the architecture’s knowledge types. In turn in ICARUS, knowledge is not encoded by rules as in SOAR or PRODIGY, the ICARUS’ knowledge types are organized in probabilistic concepts. These concepts can be physical objects or location concepts or qualitative states.

In my view, SOAR and PRODIGY support skill knowledge, because they hold knowledge about how to generate or execute sequences of actions in their “heads” and in the environment. Whereas ICARUS focuses on conceptual knowledge, due to the fact it is concerned with objects, situations, locations etc., as described by Langley et al. [1991].

4.2. Organization of Knowledge

The research of Langley et al. [2009] outlines several properties of knowledge organization:
- Knowledge organization in the memory
- Number of distinct memories

SOAR organizes its knowledge in three distinct long-term memories: procedural, semantic, and episodic memories. Thereby, SOAR’s representation scheme supports “flat” instead of “hierarchical” structures. Langley et al. [2009] describes, if the stored memory elements do not refer to each other (e.g. in production systems) it implies “flat” structures. This is the case in SOAR. However, SOAR contains an elaboration method, within the decision cycle, similar to ICARUS’ conceptual inference, but an explicit conceptual hierarchy does not exist in SOAR [Langley 2006]. PRODIGY stores its knowledge such as SOAR in distinct memories, but only in two memories: The memory for control knowledge and the one for domain knowledge as you can see in figure 3. The knowledge in ICARUS is stored in the long-term memory, which is not separated in distinct ones, in contrast to SOAR and PRODIGY. I assume, that the knowledge representation in ICARUS supports hierarchical structures, in contrast to SOAR, which supports flat structures. I assume that ICARUS supports hierarchical structures because of its hierarchical organization of probabilistic concepts. Langley et al. [2009] describes the case, if a category relates to more general concepts (its parents) and more specialized ones (its children) you can classify this structure to the class of the “is-a” hierarchy. Therefore, I estimate that the knowledge in ICARUS is organized in an “is-a” hierarchy, because the concepts are coordinated at different abstraction levels, more precisely the specific cases are located at the final nodes and the more generic concepts are at inner nodes, as described in Langley et al. [1991].

4.3. Utilization of Knowledge

The research of Langley et al. [2009] outlines the following several properties of knowledge utilization:
- Does problem solving relies on heuristic search through problem spaces or on retrieval of solutions or plans from long-term memory?
- When the architecture supports multi-step problem solving and inference. Does it apply forward-chaining, or backward-chaining, or means-end analysis?

Forward-chaining applies relevant operators and inference rules to the current problem state and current beliefs to produce new states and beliefs. Backward-chaining applies relevant operators and inference rules to current goals in order to produce new subgoals. Means-end analysis is a combination of these approaches. Selecting operators through backward chaining but executing them, if their preconditions match. [Langley et al. 2009]

- Is the architecture a deliberative (planning or reasoning out actions before starting the execution) or a reactive one (selection of an action on each decision cycle based on its understanding of the current situation)?

SOAR uses a heuristic search through problem spaces, which is applied in SOAR’s decision cycle as described in [Lehman et al. 2007]. As well PRODIGY utilizes a heuristic approach in searching through a problem space to reach one or more goals. Whereas ICARUS retrieves plans, which were self-generated from the DAEDALUS module, from the memory system LABYRINTH as pictured in [Langley et al. 1991]. SOAR uses a backward chaining approach, because it applies relevant operators to the current goals in order to generate new subgoals. PRODIGY utilizes knowledge to search through a problem space to reach one or more goals. This search applies means-end analysis, which contains selecting an operator that reduces differences among the current state and the goal. The control rules are utilized to select, reject or prefer an operator, state, or goal each iteration. In ICARUS, a variation of means-end analysis is applied to generate plans with the DAEDALUS module. You can see the combination of the approaches of selecting operators through backward chaining and executing them whenever their preconditions are satisfied, if DAEDALUS is situated in a loop or an impasse. Because it eventually applies backtracking and demands another operator followed by a heuristic depth-first search through the means-end scope as described in [Langley et al. 1991].

Regarding the third property, I assume that SOAR is a reactive cognitive architecture, because it selects its actions on each decision cycle based on its comprehension of the current situation. More concrete, the Decision Cycle consists of three stages: The elaboration phase, which studies the current state, proposes new operators and assesses the operators; the decision procedure, which evaluates the language of the operator preferences and results either in a modification of the selected operator or a deadlock if the preferences are in contradiction; and the application phase, where the selected rules are fired. Such as SOAR, I assume PRODIGY is a reactive cognitive architecture, because it selects its actions on each step during searching through the problem space based on its understanding of the current situation. To solve a problem, PRODIGY must find an operator sequence that if applied to the initial state, generates an end state meeting the goal expression. ICARUS generates plans with the DAEDALUS module before beginning the execution. That is why I assume that this architecture is a deliberative one.

4.4. Acquisition and Refinement of Knowledge

The research of Langley et al. [2009] outlines the following several properties of knowledge acquisition and knowledge refinement:
- Does the architecture gain knowledge from instruction or experience?
- Does the architecture use processes for learning new knowledge structures (such as production rules or plans) or does it use processes for refining existing structures (through adjusting weights or numeric functions)?
- Is the architecture’s learning process analytical or empirical in nature?

SOAR has multiple learning mechanisms [Laird 2012], which acquires and refine its knowledge. Chunking creates new rules in long term-memory automatically, if an impasse generated results. If SOAR comes to an impasse in the decision procedure, which indicates a lack of knowledge, it is a possibility for learning. A new substate is generated by the architecture, with the goal solving the impasse. In this manner, a goal/subgoal hierarchy is created by impasses in working memory. Moreover, reinforcement learning adapts the rule’s preferences values, which assess operators. Episodic memory collects a chronology of experiences and on the other hand, semantic learning collects more abstract declarative knowledge. [Lehman et al. 2007]

As SOAR, the PRODIGY cognitive architecture applies multiple learning mechanisms to acquire and refine its knowledge, as Carbonell et al. describes [1991]. PRODIGY employs learning strategies as explanation-based learning, analogy, abstraction, experimentation, and static analysis. ICARUS acquires knowledge through expanding its decision trees as an incremental reflex procedure, what is comparable to SOAR’s chunking strategy [Carbonell et al. 1991]. The ARGUS module in the ICARUS architecture is responsible for acquiring probabilistic concept knowledge by perceiving and identifying objects in the environment. It receives sequences of qualitative states, which are created by a “parser” subprogram. This subprogram observes objects, sensed by the agent, identifies qualitative breaks, and permanently draws up a qualitative description of events and its incorporated objects. [Lehman et al. 2007]

According to the property if the architecture’s learning process is analytical or empirical in nature, I assume that SOAR includes analytical and empirical methods for learning. I assume SOAR includes an analytical approach, because chunking operates in a deductive manner to set up new rules. The new rule is compound from a “then” part, which consists the deduction, and an “if” part, which contains the knowledge that led to/contributed to the deduction. On the other hand I assume SOAR applies empirical methods to learn knowledge by means of episodic memory, because episodic memory retrieves everything from episodes and does not make any generalization or analysis. PRODIGY utilizes both, analytic and empirical methods. Analytic methods (e.g. the analyzing module STATIC or the derivation module ANALOGY) are used to create new rules. An empirical method (e.g. with the learning-by-experimentation module EXPERIMENT) is used to transforms experience into beneficial knowledge based on detected regularities. SOAR and PRODIGY apply empirical learning in the way of learning from impasses. Regarding the ICARUS architecture, I would assess it uses an analytical method to learn things, because within the perception module ARGUS, a “parser” subprogram is located, which constantly monitors objects inside the agent’s perception range and generates a qualitative description of events and its included objects.

Another difference between ICARUS and SOAR that I found in the literature, independent from the mentioned above properties, is that ICARUS has the capability of purely inductive learning [Lehman et al. 2007], which means that it learns from solved problems. Concerning this matter, ICARUS only learns from success, when it reaches goals, but it is not able to learn from failure [Langley 2006]. Whereas SOAR has the capability to learn from both via
reinforcement learning, which is a strategy to learn from positive and negative reward [Lehman et al. 2007].

5. Conclusion

This seminar paper considers the general components of three current cognitive architectures. This may be the groundwork for further research on these architectures. For example a general analysis of the components’ advantages and disadvantages of each architecture could be realized.

To summarize, SOAR involves knowledge-intensive reasoning, reactive execution, hierarchical reasoning, planning, and learning [Laird et al. 1987; Laird 2008]. The Soar cognitive architecture is characterized by its competence in utilizing a broad spectrum of knowledge types (procedural, episodic, semantic, and declarative knowledge) and knowledge levels to solve (sub)problems. With Soar, various advanced agents were developed with the capability of using a large diversity of methods to work on a broad field of tasks, including mental arithmetic, reasoning tasks, configuring computers, algorithm design, medical diagnosis, natural-language processing, robotic control, simulating pilots for military training, and controlling non-player characters in computer games. [Laird 2012]

PRODIGY supports different learning methods to solve interesting problems in complex task domains. The general problem solving module and various learning modules are its main components. As SOAR, PRODIGY applies multiple learning mechanisms to acquire and refine its knowledge, such as explanation-based learning, analogy, abstraction, experimentation and static analysis.

The ICARUS cognitive architecture is designed for physical agents that supports prominent strategies in new modes. It contains modules for conceptual inference, goal choosing, and skill learning. However, the capabilities storing and accessing episodic memory for gaining and refining concepts is missing [Langley 2006].

Despite a lot of conceptual progress and despite cognitive architectures are used to solve real-world problems, the existing architectures, share a difficulty. The most functionalities which are implemented in cognitive architectures, are achieved only with a significant programmer effort. Therefore, a continuous research on cognitive architectures is very important. [Langley 2009]

References


