Cognitive Architectures – Assignment 3

Soar General Questions

Task 1.1

A problem space is the space in which a set of states and operators transforms on state to another. In SOAR, all goal-oriented behavior can be seen as a search through a space of possible problem spaces, while attempting to achieve a specific goal.

Task 1.2

Impasse is the event in which Soar cannot resolve its operator preferences and fails to choose a specific action. An impasse is what happens when the decision cycle cannot decide. If such an event arises, soar may use different strategies to solve the impasse. Soar will create a substate in response to such an Impasse, a state that only contains the information relevant to solving the impasse.

Task 1.3

There are several different types of Impasses:

- *Tie*, the preferences do not distinguish between two or more operators with acceptable preferences.
- *Conflict,* at least two values have conflicting better or worse preference
- *Constraint-failure,* more than one required for an operator, or if a value has both a require and a prohibit preference
- *No-change:* State no-change or Operator no-change. No operator selected during the decision procedure.

Task 1.4

One way that Soar can learn from its problem-solving experience is through reinforcement learning. Based on experience, reinforcement learning adjusts predictions of future rewards, which are then used to select actions that maximize future expected rewards.

When a solution is found, Soar uses a learning technique called chunking to transform the actions taken into a new rule. This rule can be applied when Soar encounters the situation again.

Another way soar can learn from its problem-solving experience is through something called episodic memory, or the memory of experiences.

Task 1.5

The Soar theory of cognitive behavior requires use of symbols and abstractions. In addition to the soar being a general symbolic system, the symbol structures in Soar contains associated statistical metadata, such as information on recency and frequency of use. This metadata influences retrieval, maintenance and learning of the symbolic structures.

Soar Memory

Task 1.6

Soar distinguishes three types of long-term memory structures. Procedural, semantic and episodic. Procedural knowledge is related to when and how to do things. Semantic knowledge is pure facts about the problem space/world. Episodic memory consists of knowledge from previous situations, things you “remember”.

Task 1.7

The working memory in SOAR stores a description of the current state of the problem space in a reasoning process. This description is a pattern. After a selected production rule is fired (and thus changed the contents of the working memory) the control cycle repeats with the modified working memory. The working memory has three components:

- The context stack, which contains the hierarchy of active goals, problem spaces, states, and operators.
- Objects, such as goals and states.
- Preferences, that encode the procedural search-control knowledge.

Task 1.8

In Soar the working memory and the long-term memory communication in a manner in which the contents of working memory trigger retrieval from long term memory.

Soar Practical

Task 1.10

```
;; Propose kick.

sp {b-i-n*propose-op*kick

        (state <s> ^problem-space >p> ^desired <d>)

        (<p> ^name ball-in-net)

        (<d> ^is-in-net yes)

        (<s> ^is-in-net no)

        -->

        (<s> ^operator <o>)

        (<o> ^name kick-ball)}
```

Icarus

Task 1.11

Icarus relies on three types of symbol structures to support its interpretation of the environment: *Concepts*, *Rules/Clauses* and *Beliefs*. These three types of structures support *conceptual inference*. ICARUS implements conceptual inference by repeating on every cycle:

1. Updating low-level sensory data
2. The inference module recognizes concepts by matching them against perceptual elements or instances of concepts.

Conceptual inference occurs from the bottom up. In other words, you start with feeding data about observed percepts into the bottom nodes, and then you go upwards until you meet the top nodes.

Task 1.13

ICARUS retains traces of successful decompositions, where each trace includes details about the goal, skill, and the initially satisfied and unsatisfied subgoals. As the system solves each subgoal, it generalizes the associated trace and stores it as a new skill. The skill hierarchy is expanded with new learned skills for later use when ICARUS learns to accomplish new goals or to accomplish goals in another way.

Task 1.15

A prefix notation is a mathematical notation in which to write down expressions. LISP uses a variant of the Polish prefix notation, in which the operators precede the operands.

Task 1.16

```
((colder ?object1 ?object2)

        :percepts((object ?object1 temperature ?temp1)

                   (object ?object2 temperature ?temp2))

        :tests((< ?temp1 ?temp2))

)
```

ACT-R

Task 1.17

The production system is symbolic. The sub symbolic structure is a set of parallel processes that can be summarized by several mathematical equations. The sub symbolic equations control many of the symbolic processes. If several productions match the state of the buffers, a sub symbolic utility equation estimates the relative cost and benefit associated with each production and selects the production with the highest utility.

Task 1.19

The ACT-R architecture has two memory systems, a declarative memory, and a procedural memory. Associated to each of these memory systems is several learning mechanisms that add and maintain the knowledge in them.

Here is an example of implicit learning in procedural memory: For each rule, the expected outcome is calculated using the equation:

$$U_i = P_i G - C_i$$

Saying the that expected utility is dependent on Pi, which is the expected probability of success, based on previous experiences. And Ci which is the expected cost, also based on previous experiences. This is just one example of how ACT-R uses previous experience to learn and improve.

Task 1.20

The Retrieval Buffer holds information retrieved from declarative memory. Whether and how fast a chunk can be retrieved from declarative memory depends on the sub symbolic retrieval equations, these equations consider the context and the history of usage of that fact.