TDT4137 (Cognitive Architectures) Assignment Sheet 3

Danilas Miscenko
Student ID: 536546

October 11, 2020

Acknowledgements

While working on this assignment, some topics were discussed in a group. The group members are Piri Babayev, Danilas Miscenko (me) and Aleksander Simmersholm. Because of the discussion that has taken place, some similarities between answers may arrise, but all of the assignment tasks were completed individually, as the point of the discussions is to get a better understanding of the subject and not copy each others work.

1 Soar

1.1 General questions

Task 1.1: Explain the notion of problem space in Soar.

The problem space is a term used to describe a model representing states the world is in and actions (mental or physical) the agent is able to take from these states. Both the states and the actions (or operators) are somehow related to the problem that the agent has to solve. This means that there may be many problem spaces that relate to different problems. An example of this might be the problem of pitching in baseball with the goal of getting the batter of the opposing team out and the available actions being the different pitches that the agent can throw. Another example of a problem space might be the problem of making food, with the goal of removing the feeling of hunger and the operators being picking out ingredients, preparing the ingredients, choosing what recipe to follow, etc.

The idea of problem states dates back to the earliest days of artificial intelligence research and cognitive modeling using computers.

Task 1.2: What is an *'impasse'*, and how does Soar handle the situation?

An impasse is when the model is not able to decide between several operators. An example of this might be if you are hungry and you can't decide whether you should buy ingredients and cook dinner yourself or just dine out. Both food options would achieve the goal and both are preferred over nothing, so it's difficult to choose between the two options.

To solve this, Soar automatically creates a new substate. In this new substate Soar uses the same approach as solving any other problem, but the goal becomes resolving the impasse. Soar draws on its knowledge of and any past experiences with the operators it's presented before choosing one of them. In our case the model may come to the conclusion that the effort of cooking its own food isn't worth the money saved, so it might order pizza.

Task 1.3: optional Explain two or more different impasse types.

Operator-tie impasse: The type that is described in the previous task. This impasse occurs when two or more operators are suggested that achieve the goal and there doesn't seem to be any preference for one or the other.

Operator no-change impasse: A type of impasse that occurs when the outcome of the operator is unclear. In other words, it is unclear to the model what to do with the suggested operator.

State no-change impasse: This type of impasse occurs when there are no operators suggested.

Conflict impasse: This type of impasse occurs when there is inconsistencies between preferences of operators. In our food example, some rules may state that the model should preference dining out, while other rules state that cooking your own food is preferred.

Task 1.4: Describe the different ways Soar can learn from its problem solving experience.

Chunking: Whenever the model experiences an impasse, the model in creating new substates and solving impasses creates new rules that help solve the impasse. If these new rules that are created were to be saved, then they could also help solve the impasse instantly if it were to occur again. These new rules are called chunks so that they would be easier to differentiate between the rules initially created by the modeller.

Reinforcement learning: This learning method is based around a reward structure, and creates rules that only select which operators to choose when impasses occur. In our food example, a reward might be given based on money saved when acquiring food. Since buying your own ingredients and cooking your own food is traditionally less expensive than dining out, a reward might be given to the model if this operator is chosen instead of the dining out option. Over time, the operator choice rule gets tuned with the accumulation of experience.

Episodic learning: In Soar consequences for chosen operators are recorded in the episodic memory as they are observed. This way, whenever there is an impasse, the model can try and recall if something similar has happened before and base the decision on what operator to choose on past experiences. In the case of the food problem, our model might recall that the last time it dined out, it got food poisoning, so as a preemptive measure of avoiding that it will choose to cook the food itself.

Semantic learning: This type of learning is similar to episodic learning in the way that prior knowledge is retrieved and used for decision making, but the difference is that the knowledge being retrieved is semantic knowledge. Semantic knowledge deals with concepts and things the model perceives as facts. In our example, the model might recall that home-made food tends to be healthier, or that it's less expensive to buy your own ingredients because for dine out food you also pay the cost of labour. It might use these facts to weigh the decision towards cooking your own food.

Task 1.5: Explain the 'symbol structures' that exist in Soar. Explain processes that change these symbol structures over time.

The Short term memory (STM) of Soar is encoded as a symbolic graph structure used to represent objects with properties and relations. This rep-

resentation of objects is used to model the current situation the agent is in. The way this changes is through production rules which either detect impasses or suggest an operator to use. If an operator is used, then it is applied to the current situation and the consequences of the operator are recorded as the new current situation. If an impasse occurs, then the architecture uses its prior knowledge to create a new rule that suggests an operator to the current situation, thereby either changing or adding new rules to the rule-set and applying the operator chosen for the current situation. By encountering impasses the model effectively learns how to deal with a situation it hasn't encountered before, making the decision process faster in the future.

1.2 Memory

Task 1.6: Describe the three types of long term memory (LTM) structures in Soar and their roles in problem solving

Soar distinguishes between three types of long term memory:

- **Procedural:** This LTM type is used for facts about how and when to do things. How to solve a math problem, how to drive a car, when to cross the street, etc.
- **Semantic:** Semantic knowledge is used for facts about objects or concepts that the agent has internalised as true. Examples of these might be that a bicycle has 2 wheels, a canine has four legs, the sky is blue, etc.
- **Episodic:** This LTM type is used for storing experiences from the past. Be it action outcomes, or other things such as falling off a bicycle and scraping a knee.

Task 1.7: optional Describe the working memory (WM) in Soar and its three components

Working memory is used for knowledge that is specific to the current situation and is relevant in solving current problems. The three components of WV are:

- **The context stack**, which specifies the hierarchy of active goals, problem spaces, states and operators
- **Objects** such as goals and states (and their subobjects)
- **Preferences** that encode the procedural search-control knowledge

Task 1.8: How does the WM and LTM communicate / cooperate?

WM, as stated in the previous task, is used for knowledge that is specific for the current situation, while LTM is used for knowledge that is useful for many situations. WM can therefore use the knowledge it has to retrieve other knowledge the model deems relevant from LTM.

1.3 Recent extensions of Soar

Task 1.9: optional Mention a recent extension to the Soar architecture. In what way does this improve the overall system?

One of many recent extensions to Soar is reinforcement learning. The way it works is that the model is given rewards and punishments based on how well it selects the operators. This is then used to weigh the rules used for suggesting these operators. The better the operator did in a specific situation, the higher the reward, the more weight is assigned to that rule and the more likely is it that that rule is going to be used again in a similar situation.

This improves Soar by making it more efficient over time. The model learns through its experiences, resulting in less time for deliberation on which operators to suggest.

1.4 Practical

Task 1.10: Write the following rule (presented in plain language) as a Soar production rule.

```
if      we are in the ball-in-net problem space            AND
        we desire to get the ball in the net               AND
        the current state says the ball is not in the net

then    propose an operator to apply to the current state  AND
        call this operator 'kick ball'
```

The rule in Soar syntax:

```
sp {ball-in-net*propose*kick-ball
    (state <s> ^name ball-in-net
               ^net <n>)
    (<n> ^empty)
-->
    (<s> ^operator <o> +)
    (<o> ^name kick-ball)}
```

2 ICARUS

Task 1.11: Describe the *conceptual inference* process in ICARUS.

When ICARUS receives the set of objects it perceives, it stores them in a perceptual buffer. In this buffer the objects are being matched against long-term conceptual definitions. When an object is matched with a conceptual definition of such an object, the object together with its conceptual definition are stored in the belief memory of ICARUS. Effectively, ICARUS infers a conceptual definition of the object it perceives from prior knowledge.

Task 1.12: optional What is the relation between a goal and a skill?

A goal is a want of something to happen, and for that something to happen the state of the world has to change or be changed somehow. A skill is a mechanism to change the state of the world in some way, meaning that a skill can be used to achieve some goal.

Task 1.13: How can ICARUS learn from its problem solving behavior?

From the paper *"A unified cognitive architecture for physical agents"*:

> *When the agent achieves a goal G by chaining off a skill clause with head B, which in turn required achieving its start condition A, ICARUS constructs a new skill clause with the head G, the ordered subgoals A and B, and the same start condition as that for the first subskill. When the agent achieves a goal G through concept chaining, the system creates a skill clause with head G, with subgoals based on G's subconcepts in the order it achieved them, and with start conditions based on those subconcepts satisfied at the outset.*

This means that whenever an impasse occurs, a new skill clause is created based on the problem solving behavior the system exhibits.

Task 1.14: optional What key features separates ICARUS from other architectures?

The paper *"A unified cognitive architecture for physical agents"* discusses some key features that differentiate ICARUS from other architectures like SOAR or ACT-R. The key features they note are:

- Cognition occurs in a physical context, with mental structures being grounded in perception and action.
- Concepts and skills encode different aspects of knowledge that are stored as distinct cognitive structures.

- Each element in a short-term memory has a corresponding generalized structure in long-term memory.
- Long-term memories have hierarchical organizations that define complex structures based on simpler ones.
- Conceptual inference and skill execution are more basic than problem solving, which uses these processes.

Task 1.15: In ICARUS, the LISP programming language is used to create concepts, rules, and beliefs. LISP uses something that is known as prefix notation, explain what this is.

Prefix notation, or Polish notation, is a mathematical notation which specifies the operation that is to be preformed before the operands the operation is to be preformed on.

An example of this would be **+ 1 2**. Here, the operator + is to be used on the operands **1** and **2**, meaning that **1** and **2** will be summed. Another way to look at this is like a sentence in the spoken english language;

Sum together the numbers 1 and 2.

Task 1.16: Provide at least one example of a syntactically valid clause in LISP which expresses a reasonable statement related to the real world which is *not* appearing in the lecture slides or the recommended papers.

A conceptual rule for pitching in baseball

```
((pitching ?self ?batter)
 :percepts  (self ?batter-left-handed ?batter-right-handed)
            (wind-speed wind-direction)
 :tests     (< batter-left-handed batter-right-handed))
```

3 ACT-R

Task 1.17: How does ACT-R relate to the concept of symbolic vs. sub-symbolic cognitive architectures?

The production rules are comprised of patterns of activation in different buffers in the architecture which get matched to the actual patterns of activation which are happening in the buffers, making ACT-R a sub-symbolic cognitive architecture.

Task 1.18: optional Describe the ACT-R architecture and how it is structured. How is this related to the brain?

Quote from the "An Integrated Theory of the Mind" paper:

Thus, a production rule in ACT-R corresponds to a specification of a cycle from the cortex, to the basal ganglia, and back again. The conditions of the production rule specify a pattern of activity in the buffers that the rule will match, and the action specifies changes to be made to buffers.

The names of the different regions of the architecture of ACT-R are borrowed from neuro-science and research of the brain, and are given to the regions based on what the different regions of the brain are thought to be responsible for. Also, the way the production rules work in ACT-R is reminiscent of the way different patterns of neurons firing correspond to brain activity.

Task 1.19: How does learning work in ACT-R?

Learning in ACT-R strongly resembles the Chunking mechanism from Soar combined with reinforcement learning.

When two production rules are suggested, they get concatenated into one single production rule that achieves both results from the two rules it has been created from. This rule then gets suggested along with the rules it has been created from. Since the new rule is completely fresh, its probabilities and costs will be determined by Bayesian priors, which will always be lower than the two rules the new one was created from. However, the next time this same rule is generated, it's probabilities of being picked increase. Over time, this new rule will be picked over its parent rule, and if the new rule is better at achieving the goal, its cost will be lower that the parent rule. Over time, this rule will phase out the parent rule.

Task 1.20: Describe how ACT-R retrieves relevant information in its cognition process.

The retrieval is based on the activation equation

$$A_i = B_i + \sum_j W_i S_j i \tag{1}$$

where B_i is the base-level activation of the chunk i, the W_j's reflects the attentional weighting of the elements that are part of the current goall, and the $S_j i$'s are the strengths of association from the elements j to chunk i.

This basically means that information is retrieved based on how relevant it is to the current goal combined with how often the particular chunk of information is retrieved. This mimmics how we as humans recall information when trying to solve some task.