Enhancing the Learning Capability in SOAR Cognitive Architecture

Shalini

Research Scholar, Department of Computer Science & Engineering,

PDM University, Bahadurgarh, Haryana, India.

shalinisingroha@gmail.com

Dr. S. Srinivasan

Professor, Head of Department, Department of Computer Science & Applications,

PDM University, Bahadurgarh, Haryana, India.

sunderrajan_engg@pdm.ac.in

Dr. Nitin Bansal

Associate Professor, Department of Computer Science & Applications,

PDM University, Bahadurgarh, Haryana, India.

bansal.nitin12@gmail.com

Article Info
Page Number: 9018-9028
Publication Issue:
Vol. 71 No. 4 (2022)

Abstract
Intelligent agents enable the implementation of artificial intelligence by making decisions. The supporting structure for an intelligent system is defined by a cognitive architecture. In this paper, with the help of Soar cognitive architecture, we are trying to empower the advanced agent system by enhancing their intelligence using knowledge representation, environment, and capabilities. Soar's Semantic memory stores the content of agents' information and knowledge in the form of facts and its Episodic memory supports a collection of cognitive capabilities that help an agent observe its environment, think critically, be more capable, and therefore learn.
In our work, we are enhancing the two-digit arithmetic addition agent by increasing its number of digits i.e. from two to three, thereby making it more capable and using rote learning to make it intelligent.

Article History
Article Received:
12 September 2022
Revised: 16 October 2022
Accepted: 20 November 2022
Publication: 25 December 2022

Keywords Cognitive Architecture, Intelligent Agent, Learning, Episodic memory, Semantic Memory

INTRODUCTION

The idea behind cognitive architecture is how the components of the human brain interact to provide intelligent conduct under challenging circumstances [1]. It serves as a manual for intelligent agents. The main goal of Soar Cognitive Architecture is to offer a general mechanism of experience that can hold the full spectrum of an intelligent agent's capabilities [2]. Any intelligent system is made to carry out a number of tasks that, when combined, make up its functional capabilities. In our paper, we talk about the capabilities that our agent support.

To make any intelligent agent perform with all its capabilities, just a small number of skills, such as recognition and decision-making, are strictly necessary [2] [3].

The soar agent has a rote learning method to increase its intelligence. Rote learning concentrates on memorization so that the learner can recall the information exactly as it was read or heard [4]. It is a method of memorizing that relies on repetition. Rote learning saves computed values and recalls them when needed, which results in significant computation time savings. An architecture can access more information sources to guide its behaviour the more such capabilities (learning) it offers [2] [5].

WORK FLOW OF SOAR AGENT

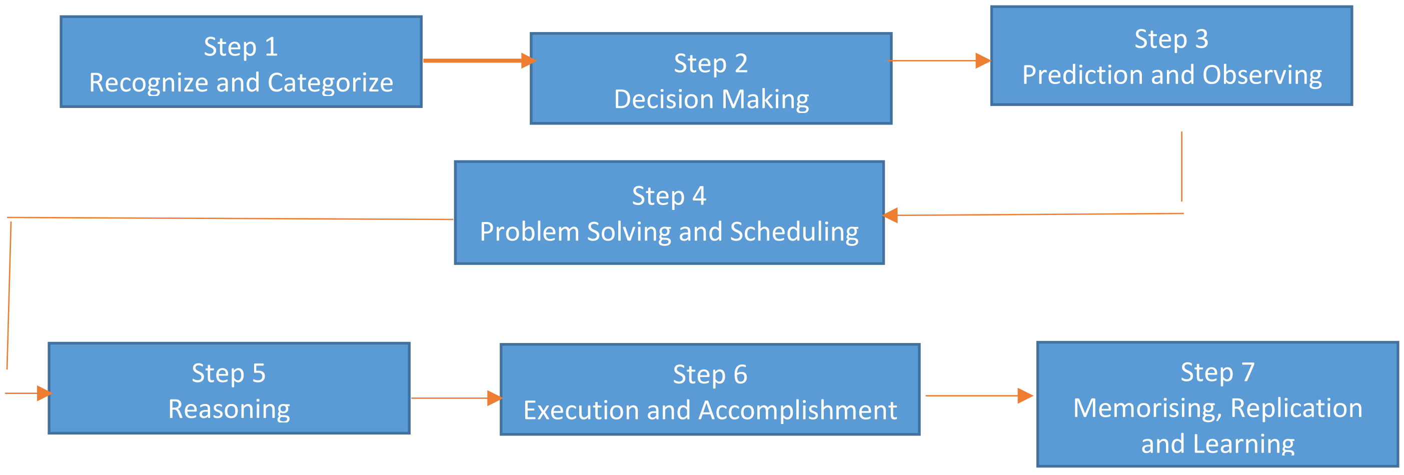


Fig 1 Workflow of Soar Agent

1. Recognize and Categorize

Our agent provides a means to represent facts and situations in memory to aid with recognizing and categorizing [2]. It incorporates a recognition process that let it determine whether a specific situation matches a set of stored facts and, if possible, measures the degree to which it matches. This mechanism determines when each production rule's prerequisites are met and how it will be implemented in various situations [6]. Agents employ rote learning to acquire facts from experience and, when necessary, to improve on already-existing patterns.

2. Decision-Making

To support decision-making, an agent provides a way to represent choices or acts, must offer some facts, and considering it only if the fact is matched [7]. For example, we can specify the conditions in which we get the carrying value (0,1) or (0,1,2) and then consider the operational move (either 2-digit addition or 3-digit addition)) only when the conditions are met. The choices are to choose among the allowable options. The resultant enhancements in decision-making will be reflected in the overall behavior of the agent.

3. Prediction and Observing

One general method entails keeping track of a mapping from a description of the current state and its impact to a description of the outcome [8]. Our agent includes the ability to learn predictive models from experience and refine them over time. When an agent has a method for creating predictions, it can use those predictions to observe the environment.

4. Problem Solving and Scheduling

A cognitive architecture must be able to describe a plan as an (at least partially) ordered sequence of acts, their anticipated effects, and how these effects facilitate subsequent actions. Additionally, the structure could have branches and conditional actions that depend on the results of earlier events as documented by the agent [8]. Our agent constructs a plan from components available in memory.

This means that they do actions based on a collection of facts (states), taking into account any available operators, choosing one or more, and then using those operators to create a new issue state. The system keeps searching until it either finds a workable strategy or decides to give up [9].

5. Reasoning

Reasoning, another essential cognitive function that enables an agent to expand its knowledge state, is closely tied to problem resolution. While reasoning draws conclusions from previous beliefs or assumptions that the agent already has, planning is largely concerned with achieving goals in the real world by taking action [2]. Our agent concludes that whether three columns are generated or that it can the call process-column operator to calculate the sum.

6. Execution and Accomplishment

The ability to perform tasks and activities in the environment is another requirement for cognitive architecture [2]. In certain systems, this occurs entirely reactively, with the agent picking one or more primitive actions on each decision cycle, carrying them out, and then repeating the procedure on the next cycle [8]. Since the agent can detect the environment at every time step, this method is related to closed-loop execution strategies. Open-loop execution, in which an agent invokes a stored process again without referencing the environment, is supported by the use of increasingly sophisticated skills [13].
Our agent performs the addition 20 times. It means the whole procedure of 3-digit addition runs in the form of a loop, every time after the completion of one loop, it repeats the procedure on the next cycle. The process is repeated in the cycle.

7. Memorising, Replication, and Learning

One other capability that our agent enables is Memorising. It describes the capability of encoding and storing agent results in memory for retrieval or access at a later time [10]. Any cognitive action that needs to be remembered must have an architecture that can store the cognitive structures created during it, index them in memory, and retrieve them as needed [11]. Oftentimes, episodic memories are used to describe the resulting content [13].

Replication is a different skill that needs access to cognitive activity traces. One sort of replication activity is the justification of an agent's judgments, plans, decisions, or acts in terms of the cognitive processes that lead to them [2].

Learning is the last crucial trait that our agent possesses. An agent's own problem-solving behavior or the application of previously learned skills may provide the data on which learning functions [9] [12]. But regardless of the source of experience, processing memory structures is always done to enhance the agent's capabilities [13].

Our three-digit Arithmetic agent supports all the above three capabilities.

3-DIGIT ARITHMETIC AGENT IN SOAR

In comparison to the 2-digit Arithmetic agent, the 3-digit Arithmetic agent has a carry of up to two (0, 1 & 2), up from one (0, 1). This agent doesn't employ any math operations. It presents the problem in a three-column format that is meant to be generated.

Three-Digit Arithmetic Agent, we employ several important data structures**.** The same data structure as a 2-digit arithmetic agent is used by this one, along with some new ones. The following are the additional ones:

- Addn10-facts: This is used for all facts for adding 10 to 0-9 digits.
- Digit3: It includes 0-9 digits.
- Facts: It includes all of the facts about single digit arithmetic (Digit1 and Digit2) and (Digit3 and result).
- C1 & C2: It carries 0/1.
- Result3: It holds the value from 0-2.
- Digit3: It holds the value from 0-9.
- Carry: It carries the value either 0, 1, or 2. Its value is calculated using the results from the previous column.
- Result1: It denotes the result of the digits (Digit3 & result).

The order in which the operators were used during the calculation is depicted in the following figure.

OPERATORS USED FOR PROCESSING

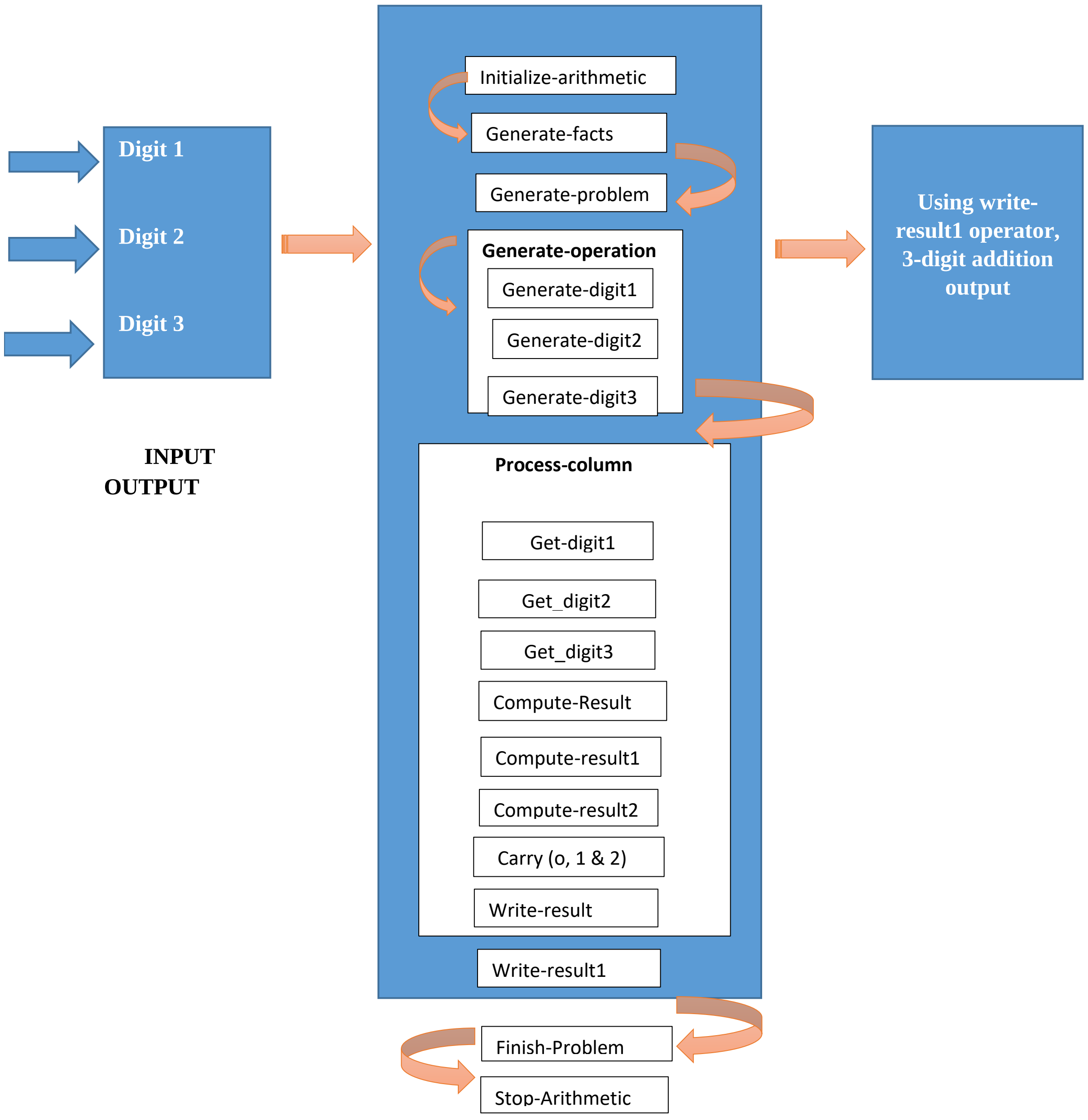


Fig 2 Sequence of Operators

Initialize-arithmetic: It generates the digits 0-9 that are used in producing problems and includes the name of the problem, i.e. ^name arithmetic. Initiate the count for the number of problems to be resolved as well. It can also specify a specific problem to be solved; if a single problem is specified, it will be solved 20 times.

```
If no task is selected
Then propose the initialize –arithmetic operator.

If the initialize-arithmetic operator is selected
Then generate and rectify arithmetic problem 20 times.
```

Generate-facts: This operation preloads whole arithmetic facts into working memory. This fact should not be essentially associated with semantic memory. It generates facts according to the addition of three digits i.e. digit1, digit2 and digit3, where first adding digit1 with digit2 storing their addition in result variable and then again adding the result with digit3 to find the final addition. Hence, performing the addition operation two times where c1 is the carry for first addition and c2 is carry for second addition.

```
If the operator generate-facts selected
    Then it generate-facts for add operator in which it keep all combinations of
(digit 1, digit 2), sum and their corresponding carrying value.

If the operator generate-facts*add1 selected
     Then it generate-facts for add1 operator in which it keeping track of all
possible sums of (digit 3, result), sum and their related carrying values.

If the operator generate-facts*carryf selected
      Then it generate-facts for carryf-facts operator in which it stores the value
of carries (c1 &c2) and result3.
```

Generate-problem: The arithmetic problem " <s> ^arithmetic-problem" is generated by this operator. Individual digits i.e. digit1, digit2, and digit3, the operation (finish-problem-generation and generate-operation), and column by column operator (next-column) are all created. It solves the addition problem.

```
If the operator generate-problem selected
   Then generate digit 1, digit 2 & digit 3

If the operator generate-operation selected
Then generate operation addition, operation-symbol and column c1, c2 and
c3

If the next-column operator is chosen
Then it switches from the current column to the following column.

If the operator selected finish-problem generation
Then it completes the generate-problem operation.
```

Process-column: This operator determines a column's outcome.

```
If the operator process-column is chosen
  Then result for a column is calculated.
```

get-digit1 & write-digit1: This operator gets the first digit from the column and passes it to the state. If there is a carry, the final digit1 is calculated by recursively adding it to column digit1. The write-digit1 operator returns the newly calculated digit1 and possible carry. When compared to the two-digit arithmetic agent, which only has 0 and 1 carry, the values of carries in this case can be 0, 1, or 2.

```
If the operator get-digit1 is chosen
    Then take a digit from the column and send it to state.

If the operator write-digit1 is chosen
    Then carrying 0, 1, and 2 repeatedly adds it to column digit1 to determine the
final digit1.
```

get-digit2: It obtains the value of digit 2 and sends it to the state.

```
If the get-digit2 operation is chosen
  Then get a digit from the column and send it to state.
```

get-digit3 & write-digit3: It obtains and communicates the value of digit 3 to the state.

```
If the get-digit3 operation is selected
   Then get a digit from the column and transmit it to state.

If the operator write-digit1 operator is chosen
    Then the final digit3 is calculated by constantly adding it to column digit3 while
carrying the numbers 0, 1 and 2.
```

```
If the operator compute-result is choose
       Then we get a value of digit1, digit2 & operation (op) through query.
If the operator compute-result1 is select
       Then we get a value of digit3, result & operation (op) through query.
If the operator compute-result2 is use
       Then we get a value of c1, c2 & operation (op) through query.
```

compute-result: It uses facts to calculate the outcome and carry it from digit1 to digit2 as well as digit3 to result. This will be exchanged with a lookup of semantic memory. Unlike the two-digit arithmetic agent, which only uses compute-result once, we are utilising it three times as compute-result, compute-result1 and compute-result2 in this case.

If the Semantic Memory Retrieval (compute-result) operator is chosen, use arithmetic facts to obtain a value of digit1, digit2, sum and carry
Then we get a value of carry and sum in query.

If the Semantic Memory Retrieval (compute-result1) operator is chosen, use add1-facts to get a value of digit3, result, sum and carry
Then we obtain a value of carry and sum in query.

If the Semantic Memory Retrieval (compute-result2) operator is chosen, use carryf-facts to retrieve a value of c1, c2, result3
Then we get a value of result3 and carry in query.

If the operator use with compute-result is selected
Then it will get the value of result, carry and c1.

If the operator use with compute-result1 is selected
Then it will retrieve the value of result1, carry and c2.

If the operator use with compute-result2 is selected
Then it will obtain the value of result3, and carry.

Carry: It transfer the carry to next-column. The Carry value either be 0, 1, or 2.

If the operator carry is selected
Then whatever the carry value either 1 or 2 it send the carry value to next-column.

new-column: If there is a carry in the leftmost column, it creates a new column for an additional problem.

If the operator new-column is used, it will create a new column if the leftmost column has a carry for a supplemental problem.

write-result: Our ultimate outcome is kept in the result1 operator in this case, whereas the two-digit arithmetic operator stores the final result in the result operator. The final outcome is sent to the current-column by this operation.

If the operator write-result is chosen
Then result is sent to the currently selected column.

If write-result1 is selected as the operator
Then result1 is delivered to the column that is currently chosen.

Next-column: When a result has been calculated for a column, this operator is used to move to the next column.

If the operator next-column is chosen, the result of a column has been established
Then The following column is then calculated after that.

Finish-problem: When there is a result for a column with no next-column, it executes.

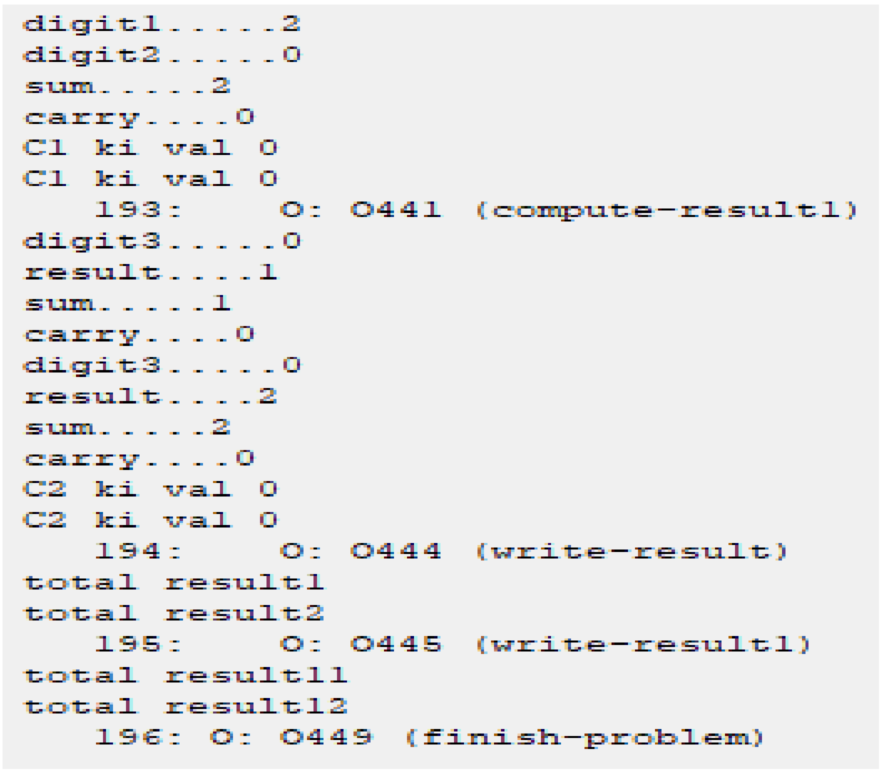
If the operator finish-problem is chosen and there is no next-column in the result of a column
Then it executes.

Stop-arithmetic: When the count value reaches zero, the agent will come to a complete stop

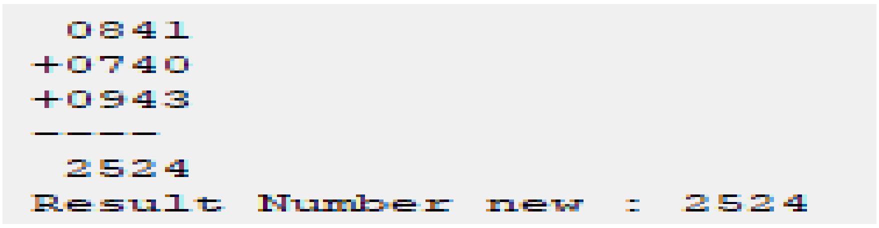
If the operator stop is chosen and the count value reaches zero
Then the calculation stops.

EXPERIMENTAL RESULTS

We use Soar Debugger to run the agent. A total of 109 productions are sourced from the agent. The entire agent runs 20 times. The figure below shows the output of the agent in Soar Debugger.



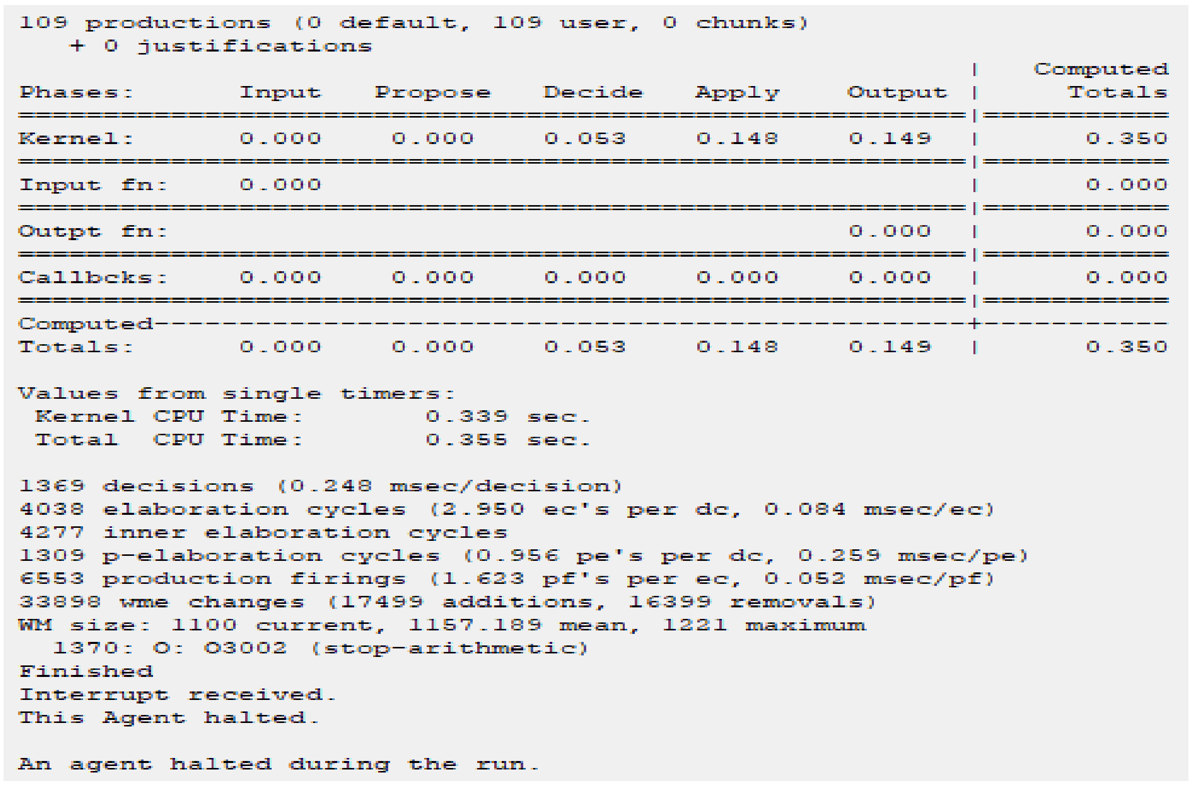
```
digit1.....2
digit2.....0
sum.....2
carry....0
C1 ki val 0
C1 ki val 0
   193:    O: O441 (compute-result1)
digit3.....0
result....1
sum.....1
carry....0
digit3.....0
result....2
sum.....2
carry....0
C2 ki val 0
C2 ki val 0
   194:    O: O444 (write-result)
total result1
total result2
   195:    O: O445 (write-result1)
total result11
total result12
   196: O: O449 (finish-problem)
```



```
 0841
+0740
+0943
----
 2524
Result Number new : 2524
```

Fig 3 Output of 3-digit Arithmetic Adiition agent

When the final carry value is 1, our agent will perform the 2-digit addition. And if it is 2, then it will perform the 3-digit addition according to the facts. Figure 4 illustrates the long run trace of 3-digit arithmetic addition agent.



```
109 productions (0 default, 109 user, 0 chunks)
   + 0 justifications
                                                                    |    Computed
Phases:          Input    Propose    Decide    Apply     Output     |      Totals
=================================================================== |============
Kernel:          0.000     0.000     0.053     0.148     0.149      |       0.350
=================================================================== |============
Input fn:        0.000                                              |       0.000
=================================================================== |============
Outpt fn:                                                0.000      |       0.000
=================================================================== |============
Callbcks:        0.000     0.000     0.000     0.000     0.000      |       0.000
=================================================================== |============
Computed------------------------------------------------------------+------------
Totals:          0.000     0.000     0.053     0.148     0.149      |       0.350

Values from single timers:
 Kernel CPU Time:        0.339 sec.
 Total  CPU Time:        0.355 sec.

1369 decisions (0.248 msec/decision)
4038 elaboration cycles (2.950 ec's per dc, 0.084 msec/ec)
4277 inner elaboration cycles
1309 p-elaboration cycles (0.956 pe's per dc, 0.259 msec/pe)
6553 production firings (1.623 pf's per ec, 0.052 msec/pf)
33898 wme changes (17499 additions, 16399 removals)
WM size: 1100 current, 1157.189 mean, 1221 maximum
  1370: O: O3002 (stop-arithmetic)
Finished
Interrupt received.
This Agent halted.

An agent halted during the run.
```

Fig 4 Long run Trace (109 productions) of 3-digit addition

CONCLUSION

The main objective of Soar is to serve as the framework for the imitation of human cognitive ability.

In our work, we extend a two-digit arithmetic agent, making it more capable and intelligent by increasing the number of digits i.e. by increasing the numbers (rows) from two to three. We express our agent according to the workflow of the soar cognitive architecture. Because we want to give our agent more control, it provides a set of cognitive capabilities that enable an agent to notice its environment, think critically, be more competent, and subsequently learn.

REFERENCES

1. Laird, J. and Rosenbloom, P., "The Evolution of the Soar Cognitive Architecture", Mind Matters, T. Mitchell (Ed.), (1996), 1-50.
2. Pat Langley, John E.Laird, Seth Rogers, "Cognitive Architectures: Research Issues and Challenges", Volume 10, Issue 2, June (2009), Pages 141-160
3. Laird, J. E., Newell, A., & Rosenbloom, P. S. Soar: An architecture for general intelligence. Artificial Intelligence, 33, (1987),1–64.
4. Laird, J. E.. "Extending the Soar cognitive architecture", Proceedings of the Artificial General Intelligence Conference. Memphis, TN: IOS Press (2008).

5. Laird, J. E., "Introduction to Soar". https://arxiv.org/abs/2205.03854,(2022).
6. Lindes, P. "Intelligence and Agency", Journal of Artificial General Intelligence 11(2), 47-49. doi:10.2478/jagi-2020-0003 (2020).
7. Laird, J. E., "Intelligence, Knowledge & Human-like Intelligence", Journal of Artificial General Intelligence 11(2), 41-44. doi:10.2478/jagi-2020-0003 (2020).
8. James R. Kirk, John E. Laird, "Interactive Task Learning for Simple Games", Advances in Cognitive Systems 3 (2014) 13-30.
9. Gorski, N. A., Laird, J. E. Learning to use episodic memory Cognitive Systems Research, 12,(2013) 144-153.
10. Neha Rajan and Sunderrajan Srinivasan, "Exploring learning Capability of an agent in SOAR: Using 8-Queens Problem", Journal of Computer Science, volume 16, issue 5, pages 642-650, (2020).
11. Nitin Bansal, S. Srinivasan, "How Soar Agent Learns: Episodic Memory International Journal of Computer Applications (0975 – 8887) Volume 181 – No. 36, January(2019)
12. . Derbinsky. N., " Efficiently Implementing Episodic Memory in Soar",September 8, (2008).
13. Andrew M. Nuxoll , John E. Laird, "Enhancing Intelligent agent with Episodic Memory", Cognitive Systems Research 17-18 (2012), 34-48.