*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 1s*
*DOI: https://doi.org/10.17762/ijritcc.v11i1s.5994*
***Article Received**: 20 November 2022 **Revised**: 28 December 2022 **Accepted**: 04 January 2023*

# Developing the Computational Building Blocks for General Intelligent in SOAR

**Shalini[1], Dr. S. Srinivasan[2], Dr. Nitin Bansal[3], Dr. Piyush Prakash[4]**
[1]Research Scholar, Department of Computer Science & Engineering,
PDM University, Bahadurgarh, Haryana, India.
shalinisingroha@gmail.com
[2]Head of Department, Department of Computer Science & Applications,
PDM University, Bahadurgarh, Haryana, India.
sunderrajan_engg@pdm.ac.in
[3]Associate Professor, Department of Computer Science & Applications,
PDM University, Bahadurgarh, Haryana, India.
bansal.nitin12@gmail.com
[4]Associate Professor, Department of Computer Science & Applications,
PDM University, Bahadurgarh, Haryana, India.
piyush19sept@gmail.com

**Abstract**
Cognitive architecture's purpose is to generate artificial agents with capacities similar to the human mind. Soar Cognitive Architecture is to produce the fixed computational building blocks needed for generally intelligent agents— agents that can outright a variety of tasks and encode, use, and learn all types of knowledge to realize the broad cognitive abilities present in humans. This paper introduced an arithmetic agent that does multicolumn, two-digit addition in SOAR. Here, we show the entire calculating procedure, including all of its operators. We are using episodic memory assistance to enhance the set of cognitive abilities that let the agent learn and reason.

**Keywords** Cognitive Architecture, Agents, Episodic Memory, Semantic Memory, Working Memory Element.

## INTRODUCTION

A cognitive architecture is a software implementation of a broad theory of intelligence; it is not a single algorithm or approach to deal with a particular problem. Rather, it is the task-independent infrastructure that learns, encodes, and applies an agent's knowledge to produce behavior [1]. A cognitive architecture is made up of memory systems for storing knowledge. We have several different cognitive architectures, including ACT-R, Theo, Prodigy, SOAR, ATLANTIS, HOMER, and others [3].

An agent is a free-moving item that can exist in any setting. Multiagent systems are systems that assemble various software agents that can speak to one another directly or indirectly. There are three categories of agent architecture.

1. Classic architecture, centered on the physical symbol system theory, uses symbolic representation for reasoning and is referred to as symbol-based architecture.
2. Cognitive architecture is to use of artificial agents with properties similar to those of the human mind.
3. In Semantic web architecture, advocacy agents can better understand preferences and make decisions [9].

Soar Cognitive architecture includes knowledge-reactive implementation, structured thinking, intense reasoning, modeling, and many types of learning[4] Soar is well-known for its capacity to solve issues and the performance of this sector by utilizing a wide range of types and knowledge levels [5][6]. Soar activities include probabilistic reasoning, mathematical ability, computer configuration, algorithms creation, clinical diagnosis, robotics controls, imitating pilots for military experience, and a variety of computerized games [8].

## SOAR ARITHMETIC AGENT

To clarify the fundamental working of Soar for users, we are demonstrating the example of an Arithmetic agent. This soar agent performs two-digit addition with carry and it is not using any math function. Soar is made up of cooperating, task-independent units. There are processing units, learning mechanisms, short- and long-term memories, and interconnections between them. It includes procedural memory (skills and "how-to" knowledge), semantic memory (facts about the world and the agent), and episodic memory (memories of past experiences) [1] [11]. An agent's situational awareness is maintained through working

*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 1s*
*DOI: https://doi.org/10.17762/ijritcc.v11i1s.5994*
***Article Received**: 20 November 2022 **Revised**: 28 December 2022 **Accepted**: 04 January 2023*

memory. In Soar facts are represented in a symbolic representation of words [7] [12].

We store all the rules of addition (called facts for this agent) in an operator (named generate-facts). Here, facts are stored for single-digit addition in the symbolic form as a production rule. Such as

```
sp {arithmetic*apply*generate-facts*add
  (state <s> ^operator.name generate-facts)
-->
  (<s> ^arithmetic.facts
              <a01> <a19> <a92>)
(<a01> ^digit1 0 ^digit2 1 ^sum 1 ^carry 0)
  (<a19> ^digit1 1 ^digit2 9 ^sum 0 ^carry 1)
  (<a92> ^digit1 9 ^digit2 2 ^sum 1 ^carry 1)
```

These facts can be converted to Semantic memory access (in the application of compute-result).We set the value of ^count to execute the computation, count number of times in initialize-arithmetic. So that it will execute the agent according to the count value. It frames the problem in three-columns that is created in operator generate-problem. This agent checks that all answers are calculated correctly with the help of Soar's math functions that are computed in Verify and Finish problem. If the answer is not computed correctly, the answer will be printed out, and then Soar halts. But this case should never happen. All the key computation rules for this are present in process-column/compute-result Soar [1][6].

We use various key data structures (in symbolic representation) of the Arithmetic agent.

- Add-10 facts: This is used for all facts for adding 10 to 0-9 digits
- Digit 1: It includes 0-9 digits.
- Digit-10: It calculates digit+10.
- Facts: It includes all of the facts about single-digit arithmetic i.e. Digit1 and Digit2.
- Sum: It represents the single-digit result (0-9).
- Carry- It carries 0/1 if the result is 10 or greater.
- Operation: Addition
- Addition-facts: All facts for adding a digit from 0-9 same structure as the facts
- Arithmetic problem: It holds the whole definition of the problem
- One-column: It represents the right-most columns wherever the ones are held linked-list to the rest-of-columns.
- Column t: It is used to check if a column occurs- it makes chunking happy.
- Digit1: It holds the value from 0-9.
- Digit2: It holds the value from 0-9.
- Carry: It carries the value of either 0 or 1. Its value is based on the calculation in the prior column.
- Next-column: It represents the column to the left of the current (It represents nil value if no next-column exit).
- Result: It denotes the result of the digits and carries value.
- Count: It calculates the number of problems to solve.
- Digits: It contains all digits 0-9.

## SELECTING AND USING OPERATORS WITH INTENTION

The *operators* in Soar are a way to categorize knowledge about conditional action and reasoning. An operator could be an internal process like combining numbers or obtaining data for episodic or semantic memory. An operator can also be an external activity, such as starting a mobile robot's forward motion or turning it [1] [10]. Soar divides the understanding of an operator into three tasks: proposing potential operators, evaluating proposed operators, and applying the operator [11].

Whole "knowledge stored in the Soar agent is illustrated as if-then rules. Here rules are stated as "productions". Rules are used to choose and apply things defined as "operators". Soar tests by testing the 'if' part of the rules. These 'if' parts are defined as conditions. If each condition of a rule is true in the present scenario, 'then' or 'action' parts are executed. Executing the actions of a rule is defined as 'firing the rule' [10]. Here, in the next section, we are explaining the various operators as if-then rules.

2-digit Arithmetic Addition Agent includes the following operators:

A. **Initialize-arithmetic:** It includes the name of the problem i.e. ^name arithmetic, which generates the digits 0-9 that utilize in generating problems. Also, initialize the count for the number of problems to resolve. It can also define a specific problem to solve, if a particular problem is defined, it will be resolved 20 times.

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

If the initialize-arithmetic operator is selected
   Then create and resolve arithmetic problem 20 times.
```

B. **Generate-facts:** This operator preloads working memory with complete arithmetic facts. This fact should not be essentially connected with semantic memory.

*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 1s*
*DOI: https://doi.org/10.17762/ijritcc.v11i1s.5994*
*Article Received: 20 November 2022 Revised: 28 December 2022 Accepted: 04 January 2023*

If the operator generate-facts selected

Then it generate-facts for add operator in which it stores all combinations of digit 1,

C. **Generate-problem**: This operator generates the arithmetic problem "<s> ^arithmetic-problem". It creates individual digits (digit1 & digit2), the operation (finish-problem-generation and generate-operation), and column by column (next-column). It does add problems.

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

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

If the operator next-column selected
Then it shifts from current column to next-column

If the operator finish-problem generation selected
Then it finish the operation of generate-problem

D. **Process-column**: This operator calculates the result for a column.

If the operator process-column selected
Then it calculate the outcome for a column.

E. **get-digit1 and write-digit1**: This operator retrieve digit1 from column and transfer it on to the state. If there is a carry then it recursively add it to column digit1 to calculate final digit1. Write-digit1 operator return the newly calculated digit1 and possibly carry.

If the operator get-digit1 selected
Then obtain a digit from column and send it to state.

If the operator write-digit1 selected
Then carry 0 and 1 repeatedly adds it to column digit1 to determine the final digit1.

F. **get-digit2**: It retrieves the digit2 value and transfer it on to the state.

If the operator get-digit2 selected
Then obtain a digit from column and send it to state.

G. **compute-result**: It calculate result and carry from digit1 and digit2 by using the facts. This will exchange with semantic memory lookup.

If the operator query is selected
Then we get a value of digit1 and digit2 through query.

If the semantic Memory Retrieval operator is selected, get a value of digit1, digit2, and sum and carry using arithmetic facts
Then we obtain a value of carry and sum value in query.

If the operator use is selected
Then it come by with the final value of result and carrv

H. **Carry**: It transfer the carry to next-column.

If the operator carry is selected
Then it send the carry value 1 to next-

I. **new-column**: It generates a new column if there is a carry at the left most column for an additional problem.

If the operator new-column is selected
Then it make a new column if there is a carry at the leftmost column for supplementary problem.

J. **write-result**: This operator transfer the result to the current-column

If the operator write-result is selected
Then it send the result to the current-column

K. **Next-column**: This operator is use when a result has been calculated for a column then it moves to the upcoming column.

If the operator next-column is selected a column's result has been determined

L. **Finish-problem**: It executes when there is a result for a column with no next-column.

If the operator finish-problem is selected when a column's result has no next-column

*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 1s*
*DOI: https://doi.org/10.17762/ijritcc.v11i1s.5994*
***Article Received**: 20 November 2022 **Revised**: 28 December 2022 **Accepted**: 04 January 2023*

M. **Stop-arithmetic**: It stops the agent when the count value becomes zero then it halts.

If the operator stop arithmetic is selected and count value drops to zero
Then it comes to a stop.

## HIERARCHY OF OPERATORS

Figure 1 illustrates a sequential hierarchy of operators that demonstrate how the 2-digit addition goes on

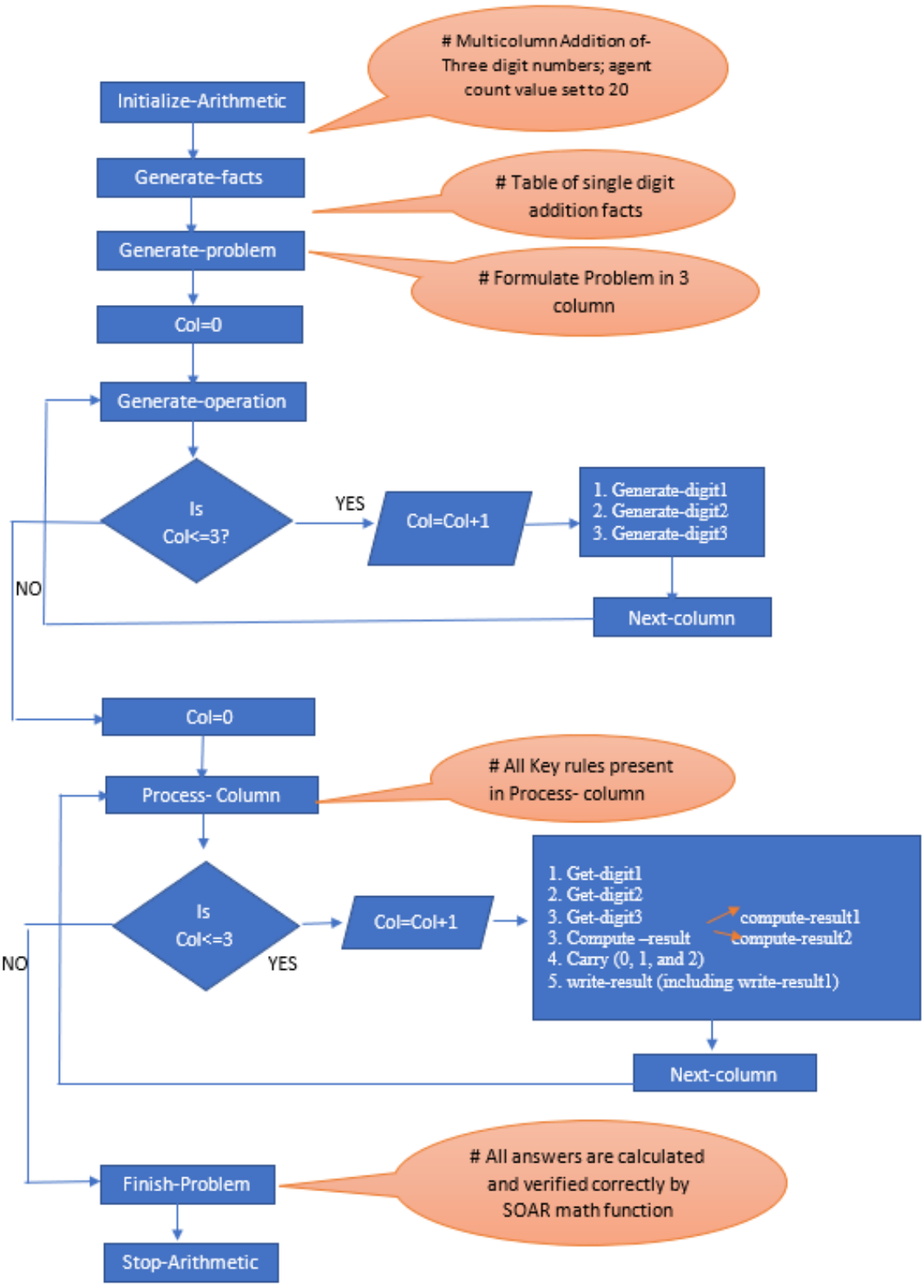


Figure 1 Graphical abstract of Arithmetic Agent

*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 1s*
*DOI: https://doi.org/10.17762/ijritcc.v11i1s.5994*
***Article Received**: 20 November 2022 **Revised**: 28 December 2022 **Accepted**: 04 January 2023*

## SEMANTIC AND EPISODIC MEMORY

In contrast to episodic memory, which stores what an agent "remembers," semantic memory stores information that an agent "knows" about itself and the world [13][14]. As a result, semantic memory acts as a knowledge base that encodes both generic, context-free knowledge about the universe and specific knowledge about an agent's surroundings, skills, and long-term objectives [15].

In our Arithmetic agent as discussed above semantic memory gathers facts about the agent and depicts it in the form of symbolic representation. Every time when the agent needs the facts for addition it retrieves the facts from semantic memory. Our agent with episodic memory is able to recall the circumstances surrounding previous experiences as well as the order in which they occurred. All the calculated results are saved in episodic memory. So, when in the future a particular number whose result is already there in episodic memory again comes for addition, it doesn't go for the whole procedure. It automatically gets the output from Episodic Memory. It's called learning. Therefore, performance increases in terms of decision-making and processing time.

## THE OUTPUT OF ARITHMETIC AGENT

We execute the agent in soar Debugger. Before we execute the agent in the Debugger, we will:

1. Excise all previous productions by pressing the "Excise all" button that is present at the bottom bar of the Debugger.
2. Freshly initialize your agent using the "Init-soar" button and using Soar button.
3. Load your agent on the Debugger using the "Source" button
4. Press the "Run" button.

Here, a total of 75 productions are sourced from the agent. This whole agent executes 20 times. The output of the agent is shown below figure:

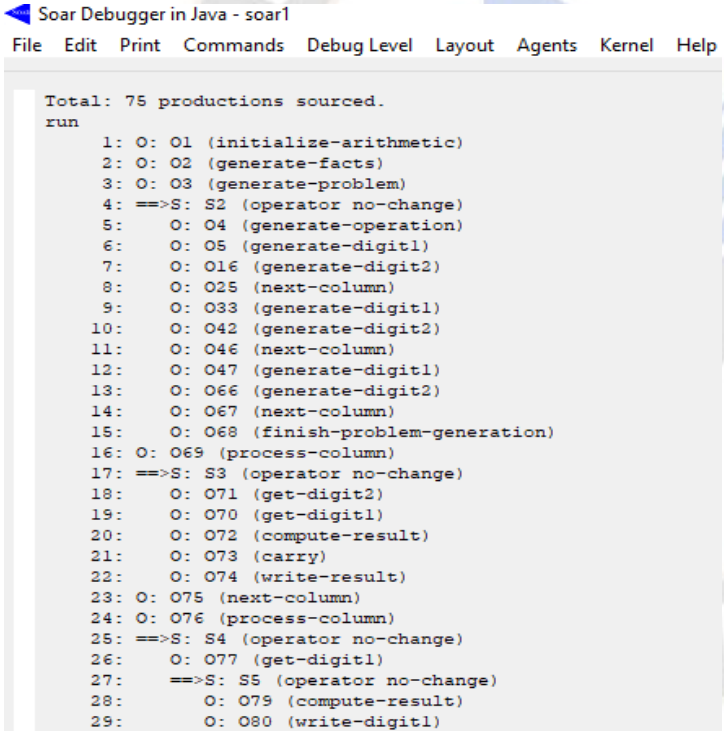


Fig. 2 Running process in Arithmetic Agent

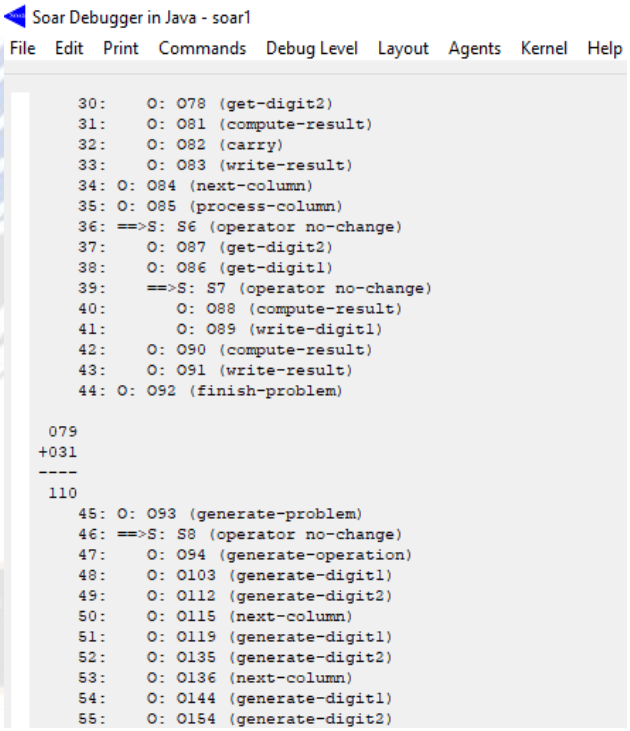


Fig. 3 First-time Output of 2-digit addition

Here, we are showing four screenshots regarding the output of the 2-digit addition arithmetic agent. Figures 2 and 3 show the running stage of 2-digit addition. Figure 4 shows the long-run trace (75 productions) of 2-digit addition.

*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 1s*
*DOI: https://doi.org/10.17762/ijritcc.v11i1s.5994*
***Article Received**: 20 November 2022 **Revised**: 28 December 2022 **Accepted**: 04 January 2023*

```
75 productions (0 default, 75 user, 0 chunks)
   + 0 justifications
                                                          |   Computed
Phases:      Input   Propose   Decide   Apply    Output  |    Totals
=============================================================|==========
Kernel:      0.000    0.028    0.015    0.057    0.068   |      0.167
=============================================================|==========
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    0.028    0.015    0.057    0.068   |      0.167

Values from single timers:
 Kernel CPU Time:       0.167 sec.
 Total  CPU Time:       0.170 sec.

836 decisions (0.200 msec/decision)
2174 elaboration cycles (2.600 ec's per dc, 0.077 msec/ec)
2303 inner elaboration cycles
694 p-elaboration cycles (0.830 pe's per dc, 0.241 msec/pe)
3474 production firings (1.598 pf's per ec, 0.048 msec/pf)
19857 wme changes (10204 additions, 9653 removals)
WM size: 551 current, 595.696 mean, 639 maximum
   837: O: O1802 (stop-arithmetic)
Finished
Interrupt received.
This Agent halted.

An agent halted during the run.
```

Fig. 4 Long run Trace (75 productions) of 2-digit addition

## PERFORMANCE ENHANCEMENT

Every time, the entire agent uses a distinct set of numbers. Every time an agent takes a number of decisions after calculation. Each decision takes kernel CPU time in microseconds, respectively. The graph in figure 5 demonstrates that the time required to execute a specific number of decisions is at its highest, followed by a reduction in the time required to execute the same agent repeatedly.

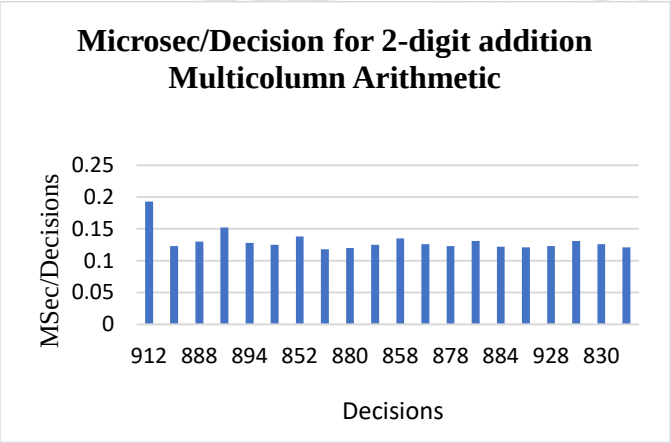


Fig. 5 Graph for Microsec/Decision for 2-digit addition Multicolumn Arithmetic

## CONCLUSION

In this paper, we build an Arithmetic agent that has computational capability like humans. It performs the addition 20 times with a different set of numbers where two numbers of 3 digits are generated randomly and adds these two numbers with carry forward to the next column.

All calculations are kept in the episodic memory. As a result, when adding a certain number in the future whose result is already stored in episodic memory, it does not go through the entire process. The output from episodic memory is automatically obtained. Learning is what it is. As a result, efficiency in terms of decision-making and processing time improves.

We can extend this agent by increasing the number of digits of both numbers (i.e., columns). Also, we can increase the numbers (rows) from two to three or more.

## REFERENCES

[1] Laird, J. E. , Introduction to Soar. https://arxiv.org/abs/2205.03854,2022.

[2] Andrew M. Nuxoll, John E.Laird,"Enhancing Intelligent agents with Episodic Memory", ScienceDirect, Cognitive Systems Research,34-48,2012.

[3] Langley, P., Laird, J. E., and Rogers, S., "Cognitive architectures: Research issues and challenges", Cognitive Systems Research, 10**(**2), pp 141-160**,**2019.

[4] Newell, A. :" Unified Theories of Cognition." Harvard University Press**,** 1990**.**

[5] Laird, J. E., Newell, A., Rosenbloom, P. S. "Soar: An Architecture for General Intelligence" Artificial Intelligence, **33**(3), 1-64,1987.

[6] Rosenbloom, P. S., Laird, J. E., & Newell, A " The Soar papers: Research on Integrated Intelligence" , MIT Press, Cambridge, MA,1993.

*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 1s*
*DOI: https://doi.org/10.17762/ijritcc.v11i1s.5994*
***Article Received**: 20 November 2022 **Revised**: 28 December 2022 **Accepted**: 04 January 2023*

[7] Laird, J. E., & Rosenbloom, P.S "The evolution of the Soar cognitive architecture" In T. Mitchell (ed.) Mind Matters, 1-50, 1993.

[8] Laird, J.E. , " The Soar Cognitive Architecture" , AISB Quarterly, #134 ,2012.

[9] Shalini, Dr. S. Srinivasan & Nitin , "A Review on SOAR Cognitive Architecture. International Conference on Information, Technology and Management (ICITM-2019)" , Laxmi Devi Institute of Engg. & Technology , Alwar, Rajasthan, India,2019.

[10] John E. Laird , "Extending the Soar Cognitive Architecture, Artificial General Intelligence", 2008, Proceedings of the First AGI Conference, AGI 2008, March 1-3, 2008, University of Memphis, Memphis, TN, USA, 2008.

[11] Nason, S., and Laird, J. E., " Soar-RL: Integrating reinforcement learning with Soar. Cognitive Systems Research" , **6**(1), 51-59,2005..

[12] Derbinsky, N., Laird, J. E., and Smith, B., "Towards Efficiently Supporting Large Symbolic Declarative Memories", Proceedings of the Tenth International Conference on Cognitive Modeling, Philadelphia, PA, 2010.

[13] Tulving, E. , " Elements of Episodic Memory", Oxford: Clarendon Press, 1983.

[14] . Lindes, P. Intelligence and Agency, Journal of Artificial General Intelligence 11(2), 47-49 doi:10.2478/jagi-2020-0003, 2020.

[15] Laird, J. E. (2020). Intelligence, Knowledge & Human-like Intelligence, Journal of Artificial General Intelligence 11(2), 41-44. doi:10.2478/jagi-2020-0003.