iJournals: International Journal of Software & Hardware Research in Engineering (IJSHRE)
ISSN-2347-4890
Volume 11 Issue 2 February 2023

A General Memory Model of Language and Visual Cognition

Author: Weihan Huang

Master of Computer Science Department, State University of New York, at Buffalo, U.S.A.

Master of Physics Department, National Hsing Hua University, Taiwan

Email :weihanh@yahoo.com.tw

DOI: 10.26821/IJSHRE.11.2.2023.110209

ABSTRACT

The paper "From the Least Effort Point of View" provides the answers to the questions of "What is language?" and "Why do we need language?" However, these philosophical answers do not give us a formally defined language for this philosophy of language. In this paper, firstly I give the formal language of the mechanism of "Least Effort Point of View". We will see that this actually gives the first glance of a General Memory Model (GMM) without type specification. Also in the introduction, I give some examples of GMM in programming language, natural language, association memory and logic. Then an interface and 3 implementations of GMM are proposed. It is especially noted that the Artificial Neural Network(ANN) implementation of GMM shares the same "simple", "small", and "uniform" representation on both sides. And then I compare this model with 3 famous memory models : Structured Query Language(SQL), Object Oriented Programming(OOP), Semantic Networks Processing System(SNePS). Next I build Cognitive Models of Consciousness, Short term memory, and Long Term Memory by GMM. Lastly, I invent a High Level Vision Description Language(HLVDL) and its conversion to the GMM. Thus we will have a general memory model for computer vision with small, simple, and uniform memory elements. Therefore, the GMM can not only do with Language as it is firstly inspired from, but also it can deal with Vision!

Keywords**: **Cognition, Language, Memory, Model , Vision

1. INTRODUCTION TO GENERAL MEMORY MODEL

By the paper of "From least effort point of view, an Investigation on Language"[1], language is using small segments to be associated with large information. Formally speaking, the representation can be a pair of objects

("small segments", large information)

Therefore, let's formally define the pair to be called a Memory Element (name, value). Where the "name" refers to "small segments", and "value" refers to "large information" in the paper of "From the Least Effort Point of View".[1]

If we would like to add type information to the

definition, then the Memory Element will become (name, type, value).

For example of "words" in a dictionary, it can be written as ("words", "word list", new memory). A Memory is simply defined as a collection of Memory Elements. And here the Memory of "words" is just the list of Memory Element "word".

This is the definition of General Memory Model(GMM). There are totally two formal entities Memory Element, and Memory.

Next I give some examples of GMM in programming language, natural language, association memory, and logic.

1.1 Example 1 Function Call Carries The Information of Function Definition([1]1.1, [2])

For example of the function call of "merge sort $array$" [3], it carries the function definition which can be represented by an array of strings. So it will be shown in the following

```
new memory element $function call$ {
    name : "merge sort $array$";
    type : "string array";
    value : ("merge sort $array$ {", "if (length of
            $array$<2), then { return
            $array$; }", ...};
}
```

Please note that the class of the value of Memory Elements is "object", so "string array" which is a subclass[4] of "object" can be the value of the Memory Element $function call$.

1.2 Example 2 Referential Use, Carries information of Properties and Functions ([1]1.2)

```
new memory element $the stone$ {
    name : "the stone";
    type : "object";
    value : new memory;
}
new memory element $color$ {
    name : "color";
    type : "color"
    value : (gray);
}
insert $color$ to (<<memory>>(value of
    $the stone$));
new memory element $shape$ {
        name : "shape";
        type : "shape"
        value : "large and round";
}
insert $shape$ to (<<memory>>(value
    of $the stone$));
new memory element $position$ {
        name : "position";
        type : "position"
        value : "on the first grass land";
}
insert $position$ to (<<memory>>(value of
    $the stone$));
new memory element $functions$ {
```

```
        name : "functions";
        type : "string list"
        value : {"can be sited", "can make the
                house"};
    }
    insert $functions$ to (<<memory>>(value of
        $the stone$));
```

1.3 Example 3 Association of Memory ([1]1.4)

Association of "I" : social-I, world-I, and picture-I

```
    new memory element $I$ {
        name : "I";
        type : "pronoun";
        value : new memory;
    }
    new memory element $associations$ {
        name : "associations";
        type : "memory";
        value : new memory;
    }
    insert $associations$ to (<<memory>>(value of
        $I$));
    new memory element $social I$ {
        name : "social I";
        type : "memory element";
        value : get element by path {"social
                relations","I"} in (long term memory of
                $$$all memory$$$);
    }
    insert $social I$ to (<<memory>>(value of
        $associations$));
    new memory element $world I$ {
        name : "world I";
        type : "grass land resident";
        value : (self of (current region of
                $$$universe$$$));
    }
    insert $world I$ to (<<memory>>(value of
        $associations$));
    new memory element $picture I$ is (get
        element by path {"all regions
        memory","first grass land","visual
        components","I"} in (long term memory
        of $$$all memory$$$));
    insert $picture I$ to (<<memory>>(value of
        $associations$));
```

1.4 Example 4 Logic ([1]1.5)

```
    new memory $all rules$;
    new memory $all facts$;
    new memory element $rule of father$ {
        name : "rule of father";
        type : "logic rule";
        value : "$x$ is the father of $y$ <==> $x$ is
                the creator of $y$";
    }
    Insert $rule of father$ to $all rules$;

    new memory element $fact of father$ {
```

```
        name : "fact of father";
        type : "logic fact";
        value : "Father is the father of Little Prince.";
}
insert $fact of father$ to $all facts$;
```

This is the end of introduction to GMM. We will formally define the interface and 3 implementations of GMM in the next part.

2. INTERFACE AND IMPLEMENTATIONS OF GENERAL MEMORY MODEL

2.1 Interface of the General Memory Model

```
new memory element $memory element$ {
    name : string : "test";
    type : string : "test type";
    value : object : "test value";
}
new memory $memory$;
insert $memory element$ to $memory$;
get element by name $string$ in $memory$ :
    return $memory element$;
get element by path $string list$ in $memory$ :
    return $memory element$;
```

2.2. Implementation by Linear Search Model

Firstly let's use a Memory Element List (implemented by Java ArrayList [5]) to store the elements in the class of Memory. To get the Memory Element by its name, it linearly searches the Memory Element List from the head to the tail to see if the name of (Memory Element List)[i] matches the given name.

This is the most intuitive model. And the time complexity of this linear search is O(N), where N is the length of Memory Element List.

2.3. Implementation by Binary Search Model[6]

Secondly, let's use the binary search model to get the memory element. Let's maintain the Memory Element List (implemented by Java ArrayList [5]) to be always in sorted order by the given key "name of Memory Element".

Then we can do binary search for the name $input string$ to get the Memory Element. The binary search performs in this way. Firstly, it will get the middle index number of the Memory Element List, and then compare the key names of the $input string$ and the name of the middle Memory Element in the Memory Element List. If the former(input string) is larger than the latter(middle one), it will start another search between the indices of the middle index number and the length of Memory Element List. i.e. Each time we cut in half length of the Memory Element List, till we get the target Memory Element that has the same key name string.

This is the fastest model for searching an element inside a list or array, and the time complexity of this binary search is O(log(N)), where N is the length of Memory Element List.

Note that the insertion which takes the advantage of

the search algorithm will have same complexity O(log(N)) which also maintains the Memory Element List to be always in sorted order.

Is it possible that there is a faster model than this fastest model? The answer is true, however we need to change the architecture of computation. Please see the next section.

2.4. Implementation by Artificial Neural Network Model(ANN)[7]

In this section, we come to the main issue of implementing GMM by ANN. The feature of this issue is that both GMM and ANN have simple, small and uniform representations. So it should be easy to model this by simple algorithm.

2.4.1. Sort in Artificial Neural Network(ANN)?

If we would like to make the Memory Element List to be sorted, then we would like to make an ANN to sort a list of arbitrary length N. For example, the input nodes contain number respectively {3,1,2,4}, and the output nodes contain numbers {1,2,3,4}. This is a hard problem for me, and I search the whole network there is no satisfactory solution especially for the input length is an arbitrary N. So I just felt it would go back to the linear search model that takes the time complexity of O(N). Alas, if we think a parallel computation approach to this problem, then we will get a better time complexity than those of sequential procedures!

2.4.2 Parallel Processing[8]

Let's try parallel algorithm for searching a Memory Element.

2.4.2.1. Search Memory by Key O(1)

```
for every element[i] in memory, run in parallel
    if((key value of element[i] - input key)=0) {
        return element[i];
    }
}
```

2.4.2.2 Insert Memory Element to Memory O(1)

```
add $memory element$ to (memory element list of
    $memory$);
```

It is marvelous that we get only a constant time complexity O(1) to finish the tasks of search and insertion. And what is more is that the Memory Element List being sorted is not required at all.

2.4.3. Artificial Neural Network(ANN) Model

Recall one feature of ANN model is that each node runs in parallel. So we just need to arrange the topology so that they can reach the goal by parallelism.

2.4.3.1. Search Memory by Key O(1)

The following is the network for each element i and node i. The node computes the Boolean value of (key value of element[i] = input key), and connects Output to element[i] if it is true.

element[i] ----> node ^ if(key value of element[i]=input key), Output connects to

```
                    |       element[i]
                   /
input key     -- /

        element[i] <--> Output
```

2.4.3.2. insert Memory Element to Memory O(1)

Since sorted order is not required by ANN modeling, we can simply insert the Memory Element to the Memory Element List.

add $memory element$ to (memory element list of $memory$);.

3. COMPARING GMM WITH SQL[9], OOP[10] AND SNePS[11]

In this part, I compare GMM (General Memory Model) with 3 famous memory system SQL(Structured Query Language), OOP(Object Oriented Programming), and SNePS(Semantic Network Processing System).

For SQL and OOP, every single record in SQL and every single object in OOP can be modeled by GMM Memory Element. However, GMM Memory Element is just as simple as possible for the spirit of "From the Least Effort point of View" while keeping the abilities to model SQL and OOP behaviors. The small, simple, and uniform representation of memory element (name,type,value) makes it easier to be implemented by Artificial Neural Network which also have small, simple and uniform representation of all neurons. Also it is noted that new data member of objects can be added dynamically by inserting a new memory element in GMM. This dynamic feature is more like Lisp[12], while SQL and OOP cannot do it.

SNePS is similar to GMM for that SNePS can dynamically add the data member of objects. However, SNePS cannot have Memory which has a Memory Element Array. For example of exploring the world, we must build an array of countries, and arrays of cities. But SNePS cannot have arrays inside the network. Moreover, for the example of associating a function call with its function definition, an example of "From the Least Effort Point of View", the function definition can be represented by a string array. GMM can make it because the value of Memory Element is of class Object, it can carry the subclass "string array". However, SNePS cannot have "string array" inside the network. So these are the major differences between GMM and SNePS

4. COGNITIVE MODELS OF CONSCIOUSNESS[13], SHORT TERM MEMORY[14], AND LONG TERM MEMORY[15]

In my modeling, Consciousness means current focused memory element, it changes after attention changes. Short Term Memory is the cache of most recently used memory, it disappears after a sleep. And Long Term Memory is just the memory that stores all memory elements, it continues after sleeping. Functionally speaking, Short Term Memory is to cache the memory that was retrieved from Long Term Memory. The length of Short Term Memory is set to be 10 by me. So to find an Memory Element

inside Short Term Memory it only requires O(log(10)), this is much faster than searching in the Long Term Memory, which requires O(log(N)) in sequential computation. So the procedure of looking for a Memory Element is firstly to find it in the Short Term Memory, if not found, then find it in the Long Term Memory.

Let's take Consciousness as the current focused Memory Element, then we can find the input Memory Element by time complexity O(1). Therefore, in a sequential modeling, we should firstly compare the input with Consciousness, if not found then find the input in Short Term Memory, if not found, then find it in the Long Term Memory.

These are the basic cognitive models of Consciousness, Short Term Memory, and Long Term Memory by GMM, where Consciousness is modeled by GMM Memory with Memory Element List of length 1, and Short Term Memory is modeled by GMM Memory with Memory Element List of length 10. And Long Term Memory is modeled by GMM Memory with Memory Element List of arbitrary length N.

5. MODEL VISUAL MEMORY BY HIGH LEVEL VISION DESCRIPTION LANGUAGE

For computer vision [16], I invent a high level vision description language (HLVDL) for image understanding. Then I will convert HLVDL language to GMM, so that the vision memory model is formed by GMM. Because GMM can be easily implemented by ANN, so it would be easy for us to create ANN model for computer vision.

5.1. HLVDL Language

```
class HLVDL {
    name : string;
     type : string; // such as : building, person, object
    position words : string list : {};
    rectangle : rectangle;
    visual components : HLVDL list : {};
      background : string; // for region e.g. "pattern :
                                green grass"
}
```

The "position words" like "up", "center", "down", "left" and "right" can be used to get a rough position of the visual object VO1 inside the container visual object VO2. The rectangle (row, column, height, width) gives exact position of the visual object VO1 inside the container visual object VO2. For "visual components", it contains the parts of the image. For example, the image Little Prince contains the parts : crown, face and body. And the parts are also described by HLVDL recursively.

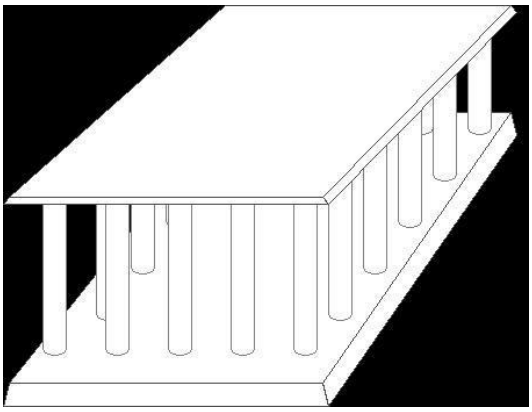
5.1.1 Example of image of Little Prince

```
new HLVDL $crown$ {
    name : "crown";
```

```
    type : "object";
    position words : {"upper"};
    rectangle : new rectangle position (1,1) size
                (20,60);
    visual components : {};
}
new HLVDL $face$ {
    name : "face";
    type : "object";
    position words : {"up"};
    rectangle : new rectangle position (24,12) size
                (16,32);
    visual components : {};
}
new HLVDL $body$ {
    name : "body";
    type : "object";
    position words : {‘middle”,"down"};
    rectangle : new rectangle position (45,1) size
                (60,60);
    visual components : {};
}
new HLVDL $Little Prince$ {
    name : "Little Prince";
    type : "person";
    position words : {"right", "down"};
    rectangle : new rectangle position (450,700) size
                (105,60);
    visual components : { $crown$, $face$,
                $body$};
}
```

We can see that $crown$ is on the top of image, so its position word is {"upper"}. While face is at the up part of image, so its position words are {"up"}. More exact positions in the images are given by the data member "rectangle". And the visual components of $Little Prince$ contains the parts { $crown$, $face$, $body$}.

5.1.2 Example of Image of Academy Building



```
new HLVDL $roof$ {
    name : "roof";
    type : "object";
    position words : {"up"};
    rectangle : new rectangle position (1,1) size
                (168,428);
    visual components : {};
}
new HLVDL $pillars$ {
    name : "pillars";
    type : "object";
    position words : {“middle"};
    rectangle : new rectangle position (24,60) size
                (300,420);
    visual components : {};
}
new HLVDL $floor$ {
```

```
    name : "floor";
    type : "object";
    position words : {"down"};
     rectangle : new rectangle position (92,1) size
                 (338,444);
    visual components : {};
}
new HLVDL $academy$ {
    name : "Academy";
    type : "building";
    position words : {"right", "up"};
     rectangle : new rectangle position (5,600) size
                 (340,448);
      visual components : { $roof$, $pillars$,
                          $floor$};
}
```

Thus HLVDL is a high level vision description language for image understanding because it provides the parts with both rough and exact positions in the container image. For example, $academy$ is the container image of $roof$. $Little Prince$ is the container image of $crown$. And they provide rough position "position words" and exact position "rectangle".

5.2 Convert HLVDL Language to General Memory Model(GMM)

Lastly I convert HLVDL to Memory Element, and convert HLVDL List to Memory Element List.

```
convert $HLVDL list$ to memory
                : return $memory element list$ {
  new memory element list $element list$ is {};
  for each $HLVDL$ in $HLVDL list$, do {
       insert (convert $HLVDL$ to memory) to
          $element list$;
  }
  return $element list$;
}

convert $HLVDL$ to memory
                : return $memory element$ {
    new memory $memory$;

    new memory element $name1$ {
        name : "name";
        type : "string";
        value : (name of $HLVDL$);
    }
    insert $name1$ to $memory$;

    new memory element $type$ {
         name : "type";
         type : "string";
        value : (type of $HLVDL$);
    }
    insert $type$ to $memory$;

    new memory element $position words$ {
        name : "position words";
        type : "string list";
        value : (position words of $HLVDL$);
    }
    insert $position words$ to $memory$;
```

```
        new memory element $rectangle$ {
            name : "rectangle";
            type : "rectangle";
            value : (rectangle of $HLVDL$);
        }
        insert $rectangle$ to $memory$;

        new memory element $visual components$ {
            name : "visual components";
            type : "HLVDL list memory";
            value : convert (visual components of
                    $HLVDL$) to memory;
        }
        insert $visual components$ to $memory$;
      new memory element $background$ {
        name : "background";
        type : "string";
        value : (background of $HLVDL$);
    }
    insert $background$ to $memory$;

  new memory element $element$ {
        name : (name of $HLVDL$);
        type : "HLVDL memory";
        value : $memory$;
    }

    return $element$;
}
```

Therefore, HLVDL can be converted to GMM, through implementing GMM by ANN, we can easily build visual memory by neural networks.

6. CONCLUSION

From the paper of "From the least effort point of view, an Investigation on Language"[1], language is using small segments to be associated with large information. This inspires me to develop the GMM. There are two formal objects in GMM, Memory Element and Memory. The simple, small and uniform representation of memory elements makes it easy to be implemented by ANN. In part 3, I compare the GMM with SQL, OOP, and SNePS, in that I show the differences and advantages of GMM. Next I model the Consciousness, Short Term Memory and Long Term Memory by GMM. And lastly, I invent the HLVDL for understanding images, i.e. by a high level vision description language, and convert the HLVDL to GMM. This is a striking step for we can use ANN to implement High Level Vision Memory through GMM. Therefore, the GMM can not only do with Language as it is firstly inspired from, but also it can deal with Computer Vision!.

7. REFERENCES

[1] Weihan Huang, (2022), "From the Least Effort Point of View, an Investigation on Language", International Journal of Social Relevance and Concern(IJSRC), Volume 10 Issue 9, (September 2022), pp6-10.
Link to the paper :
http://ijournals.in/wp-content/uploads/2022/09/2.IJSRC-100902-Weihan.pdf

[2] Function call , Function definition, Actual

argument

In the function definition f(x,y)=x+y. The f is called the function name, and x and y are called the formal parameters. A function call f(1,2) contains its function name f and the actual arguments 1 and 2.

[3] Merge sort
https://www.educative.io/edpresso/how-to-implement-a-merge-sort-in-java

[4] Subclass
https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)#Subclasses_and_superclasses

[5] ArrayList
https://en.wikipedia.org/wiki/Dynamic_array

[6] Binary Search Model
https://en.wikipedia.org/wiki/Binary_search_algorithm

[7] Artificial Neural Network (ANN)
https://en.wikipedia.org/wiki/Artificial_neural_network

[8] Parallel Processing
https://en.wikipedia.org/wiki/Parallel_computing

[9] SQL (Structured Query Language)
https://en.wikipedia.org/wiki/SQL

[10] OOP (Object Oriented Programming)
https://en.wikipedia.org/wiki/Object-oriented_programming

[11] SNePS
The knowledge frontier, pp 262, 11. SNePS considered as a fully Intensional Propositional Semantic Network. Link to the paper :
https://cse.buffalo.edu/~rapaport/676/F01/shapiro.rapaport.87.pdf

[12] Lisp
https://en.wikipedia.org/wiki/Lisp_(programming_language)

[13] Consciousness
https://en.wikipedia.org/wiki/Consciousness

[14] Short Term Memory
https://en.wikipedia.org/wiki/Short-term_memory

[15] Long Term Memory
https://en.wikipedia.org/wiki/Long-term_memory

[16] Computer Vision
https://en.wikipedia.org/wiki/Computer_vision