The Menso model for human logical mind

Hossain Khass*
Department of Pure Mathematics, Faculty of Mathematical Sciences,
University of Kashan, Kashan 87317-51167, I. R. Iran

August 31, 2019

Abstract

In this paper we try to present a model for the human logical mind. We call this model, the Menso model. The Menso model explains that the human mind recognizes two main types of virtual entities, propositions and objects. Propositions can be either true, false or undetermined. Also, the mind recognizes a number of rules of inference with which it can change undetermined propositions into either true or false propositions. In our model, we have designed a language for writing logical proofs in a way that can be given to The Menso Software. The Menso software is the software we have developed according to the Menso model to show how the Menso model works.

1 Background and motivation and definitions

There are many mathematical theories about logic. But, as far as the author knows, none of them tries to specifically investigate the human mind and derive the logic it uses. Here, we try to extract a logical model based on how the human mind works. The human mind is evolved in thousands of years so that it can help the humans to have kids and prepare their kids to have other kids. This has led to the development of some biological understanding system of the world around. What this understanding system understands may or may not be real, but it works. Here, we try to simulate this biological contraption evolved in the brain, and present it as a model.

To simulate human mind, in the first place, one needs to simulate the logical part of it. The logical system of human mind, is the base for correct thinking. In fact, there must be a system which can determine whether a thought is correct or incorrect, we call this system the *logical system of human mind* or the *LSHM* for short. In this paper, we try to demonstrate how the LSHM works. To this end, we provide a rather simple model, called *Menso* model, inspired by the LSHM. After the Menso model is articulated in details, it will be easier to develop a computer program which simulates the LSHM more completely.

The human mind can consider some mind entities including propositions and objects, and has some rules of inference.

*hosseinkhass@gmail.com

The human sensors, including eye, ear, etc., receive signals which, when processed in mind, lead to mind entities. For example, if a human sees a ball, the ball leads to the creation of some mind entity corresponding to the real ball. In fact, the LSHM, needs internal entities so it can process them. Thus, everything must be in the mind before any logical processes begin.

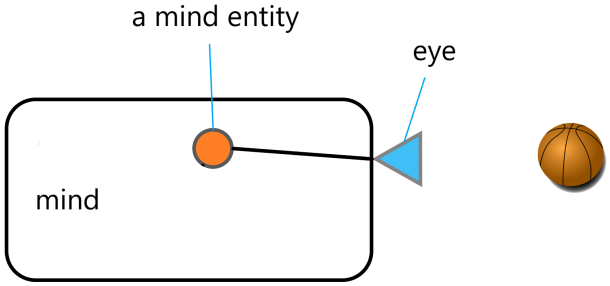

Figure 1: Image 1

We can identify three kinds of mind entities in LSHM:

1) Objects.

2) Propositions.

3) Functions.

Objects are mind entities that usually correspond to an external thing. For instance, watching a ball may cause an object come into existence in the mind; such object represents the ball in the LSHM. However, the LSHM does not necessarily need an external thing to create an object. It can create a completely pure objects with no external equivalents.

Propositions are special mind entities, for which the rules of inference are evolved. Propositions are either true, false or undetermined. The rules of inference are the pathway through which an undetermined proposition, can become a true or a false proposition. This is all the LSHM does: Changing undetermined propositions into determined ones.

Functions are entities which together with one or more other entities produce another entity. A function has a list of input entities, called arguments, and an output entity. For instance, if `f` is a function which takes one argument, and `x` is an appropriate argument, then `f(x)` is an entity which is the output of the function `f` when fed with `x`.

Thus, a mind entity, is either an object, a proposition or a function with certain output and inputs. This is called the *type* of the entity. There are three kinds of types: objects, propositions and functions, and functions include infinitely many types. In fact, two functions have two different types, if the their inputs or output mismatch.

2 Menso language

The Menso model reads a code, written in Menso language, and checks if there is anything wrong with it. Such code represents a thought in the LSHM. The LSHM checks the

thoughts of human mind to see if they are logically correct and so the human mind can jettison the incorrect thoughts. The rest of this section describes the Menso language. For testing the codes you will need the *Edukadoj Menso Mind Logic Simulation Software*, for short the *Menso Software*. We have developed the Menso software to show how the Menso model works.[1]

To add a new object, named `a`, to the list of original entities of the Menso model, we write:

```
new object a;
```

Using this code, the object `a` is added to the list of original entities in the Menso model's mind. A similar process can happen in the LSHM, but the LSHM may not have a clear ordered list of entities, instead, it has a neurological memory. To add a new undetermined proposition, named `p`, to the list of original entities, we write:

```
new prop p;
```

Functions can be added in a similar way: To add a function, named `R`, with one argument of type **`object`** and an output of type **`prop`**, we can write:

```
new prop*(object) R;
```

Or equivalently we can write:

```
new prop R(object);
```

Then `R` will be a function that takes one argument of type **`object`** and produces a proposition. The type of `R` is **`prop*(object)`** which is a function kind. Thus, if `a` is an object then, `R(a)` is a proposition. We can define functions with more than one argument:

```
new object f(object, object);
```

Entities added by **`new`** are called *original entities*. Other entities are called *constructed entities*. By an *entity*, we mean either an original entity or a constructed one.[2]

In Menso language, functions can have function kind arguments or outputs. Also we can define unary or binary operators in Menso language:

```
new operator object −(object);
new operator object +(object, object);
```

With the above code, `−a` and `a + b` will be meaningful, provided `a` and `b` are objects. Operators in Menso language can have either one or two arguments. There are four predefined operators in Menso language:

1) "`=`" for comparing entities.

2) "`&`" for logical "and".

3) "`|`" for logical "or".

[1] To get a copy of the Menso software contact the author. The Software work in a Windows OS.

[2] In Menso software, "original entities" are called "entities" and "entities" are called "mensos".

4) "!" for logical negation.

In Menso model, some propositions are already regarded trivially true or false. For instance, for any entity `x`, the proposition `x = x` is considered a true proposition. We can check it with **`check`** command:

```
new object x;
check x = x;
```

Such propositions will be called *trivial*. In Menso model, a proposition is true, if and only if, it is trivially true or it is listed in the *Truths* list (strictly true). Checking a trivially true proposition will add it to the Truths list. We can add an undetermined proposition `p` to the Truths list using **`add`** command. For example, the code:

```
new prop p;
add p;
```

will add `p` to the Truths list, and so, `p` will no longer be undetermined. We use **`add`** to add what are called axioms in mathematics; axioms are the primary propositions considered true, from which other propositions, called theorems in mathematics, are deduced.

If two entities are equal, in Menso model, we can replace one with another in a proposition using replace command. For instance in the code:

```
new object x;
new object f(object);
add x = f(x);
replace x with f(x) in x = f(x);
check f(x) = f(f(x));
```

`x` is replaced with `f(x)` in the true proposition `x = f(x)` and this has led to the true proposition `f(x) = f(f(x))`. In Menso model, two propositions are equal, if and only if, they are equivalent.

In Menso language, we can have quantifiers:

```
new prop p(object);
add (all object x)p(x);
check (some object x)p(x);
```

`(all object` `x)p(x)` means "for any object `x`, the proposition `p(x)` is true". Also, **`(some object`** `x)p(x)` means "for some object `x`, the proposition `p(x)` is true".

There are a few commands about quantifiers in Menso language: The **`pick`** command, can pick a specific value for the variable of a universal quantifier:

```
new prop p(object);
add (all object x)p(x);
new object a;
pick a in (all object x)p(x);
check p(a);
```

Here, **`pick`** `a` **`in (all object`** `x)p(x);` means "since for every object `x`, `p(x)` is true and `p(x)` is an object, the proposition `p(a)` is true". The pick command, can also be used to for existential quantifiers:

```
new prop p(object);
new object a;
add p(a);
pick a in (some object x)p(x);
check (some object x)p(x);
```

Here, `pick a in (some object x)p(x);` means "since `p(a)` is true, so there is some object `x` such that `p(x)` is true". In Menso language, conditional propositions are written with the form `(if p)q` where `p` and `q` are propositions.

There are three spaces in Menso language with which we can prove quantifiers or conditional propositions:

1) Suppose space.

2) Set space.

3) Select space.

Using a suppose space we can prove conditional propositions. In a suppose space, a proposition is supposed true temporarily. In a set space, a new original entity is added to the list of original entities temporarily:

```
new prop q(object);
new prop p;
add (all object x)(if p)q(x);
suppose p
{
    check p;
    set object t
    {
        pick t in (all object x)(if p)q(x);
        check (if p)q(t);
        check q(t);
    }
    check (all object t)q(t);
}
check (if p)(all object t)q(t);
```

Here, we prove that if `(all object x)(if p)q(x)` is true then `(if p)(all object t)q(t)` is true. For the proof, we temporarily suppose that `p` is true in a suppose space. In this space, we prove that `(all object t)q(t)` is true. This proposition, automatically turns into `(if p)(all object t)q(t)` after the suppose space. Here, we also use a set space to prove `(all object t)q(t)`. The object `t`, is added temporarily in the set space. Then `q(t)` is proved, which turns into `(all object t)q(t)` after the set space.

A suppose space can also be used for proof-by-contradiction: A proposition `p` is supposed true temporarily and inside the suppose space, the proposition `false` is proved. Then, after the suppose space, `!p` will be in the Truths list.

A select space is used for proving existential quantifiers using other existential quantifiers:

```
new prop p(object);
new prop q(object);
add (all object x)(if p(x))q(x);
```

```
add (some object x)p(x);
select x in (some object x)p(x)
{
    check p(x);
    pick x in (all object t)(if p(t))q(t);
    check (if p(x))q(x);
    check q(x);
}
check (some object x)q(x);
```

Here, we prove that if the quantifiers `(all object x)(if p(x))q(x)` and `(some object x)p(x)` are true, then, the quantifier `(some object x)q(x)` is true too. To this end, we use a select space; we select an x for which `p(x)` is true, temporarily. Then we prove `q(x)` inside the select space. `q(x)` turns into `(some object x)q(x)` after the space. A select space can be the general space only if the existential quantifier is a unique existential quantifier:

```
new prop p(object);
add (some object x)p(x);
add (all object x)(all object y)(if p(x) & p(y))x=y;
select a in (some object x)p(x);
check p(a);
```

In Menso model, a proposition is false, if and only if, its negation is true. Thus, to prove the falsity of a proposition, we need to prove the truth of its negation.

3 Menso rules of inference

We say that a proposition is strictly true (false) if it (its negation) is in the Truth list. We say that a proposition is trivially true (false) if it (its negation) can be checked using **check** command. The following rules of inference are defined for the Menso model, and implemented in Menso software.

1) For any propositions `p` and `q`, the propositions `p & q` and `q & p` are regarded the same in any situations.

2) If both `p` and `q` are strictly true, then, `p & q` is trivially true.

3) If at least one of `p` or `q` is strictly false, then, `p & q` is trivially false.

4) If `!p | !q` is true, then `p & q` is trivially false.

5) If `(if p)!q` or `(if q)!p` is strictly true, then, `p & q` is trivially false.

6) For any propositions `p` and `q`, the propositions `p | q` and `q | p` are regarded the same in any situations.

7) If at least one of `p` or `q` is strictly true, then, `p | q` is trivially true.

8) If at least one of `(if !p)q` or `(if !q)p` is strictly true, then, `p | q` is trivially true.

9) If both `p` and `q` are strictly false, then, `p | q` is trivially false.

10) If `!p & !q` is strictly true, then, `p | q` is trivially false.

11) If at least on of `(if !p)q` or `(if !q)p` is strictly false, then, `p | q` is trivially false.

12) If `p` is strictly false, then, `(if p)q` is trivially true.

13) If `q` is strictly true, then, `(if p)q` is trivially true.

14) If `p & !q` is strictly false, then, `(if p)q` is trivially true.

15) If `!p | q` is strictly true, then, `(if p)q` is trivially true.

16) If `(if !q)!p` is strictly true, then, `(if p)q` is trivially true.

17) If `(if p)q1` and `(if p)q2` are strictly true, then, `(if p)(q1 & q2)` is trivially true.

18) If `(if p)(q1)& (if p)(q2)` is strictly true, then, `(if p)(q1 & q2)` is trivially true.

19) If `!p | q` is strictly false, then, `(if p)q` is trivially false.

20) If `p & !q` is strictly true, then, `(if p)q` is trivially false.

21) If `p` and `!q` are strictly true, then, `(if p)q` is trivially false.

22) If `(if !q)!p` is strictly false, then, `(if p)q` is trivially false.

23) For any proposition `p`, the propositions `p` and `!!p` are regarded the same in any situations.

24) If `p & q` is strictly true, then p and q are trivially true.

25) If `p` and `(if p)q` are strictly true, then, `q` is trivially true.

26) If `p & (if p)q`, is strictly true, then, `q` is trivially true.

27) If `p` is strictly false and `p | q` is strictly true, then, `q` is trivially true.

28) If `!p & (p | q)` is strictly true, then, `q` is trivially true.

29) If `p = true` is strictly true, then, `p` is trivially true.

30) If `p` is strictly true, then, `p = true` is trivially true.

31) If `p = false` is strictly true, then, `p` is trivially false.

32) If `p` is strictly false, then, `p = false` is trivially true.

33) If two entities strictly equal a third one, then, they are trivially equal.

34) If `(if p)q` and `(if q)p` are strictly true, then, `p = q` is trivially true.

35) If `(if p)(q)& (if q)p` is strictly true, then, `p = q` is trivially true.

36) If `p = q` is strictly true, then, `(if p)q` is trivially true.

37) If `p = q` is strictly true, then, `p | !q` is trivially true.

38) If `p = q` is strictly true, then, `p & !q` is trivially false.

39) If assuming `p` is true, `q` is trivially true, then, `(if p)q` is trivially true.

40) If for some proposition `p`, both `p` and `!p` are strictly true, then, false is trivially true.

41) If the proposition `p` in `(all/some type x)p` is independent of `x`, then, `(all/some type x)p` will be trivially equal to `p`.

42) If `(all type x)p(x)` and `(all type x)q(x)` are strictly true, then, `(all type x)p(x)& q(x)` is trivially true.

43) If `!(all type x)!p(x)` is strictly true, then, `(some type x)p(x)` is trivially true.

44) If `!(some type x)!p(x)` is strictly true, then, `(all type x)p(x)` is trivially true.

45) If `(all type1 x)(all type2 y)p(x,y)` is strictly true, then, `(all type2 y)(all type1 x)p(x,y)` is trivially true.

46) If `(some type1 x)(some type2 y)p(x,y)` is strictly true, then, `(some type2 y)(some type1 x)p(x,y)` is trivially true.

47) If `(all type1 x)(some type2 y)p(x,y)` is strictly true, then, `(some type2 f(type1))(all type1 x)p(x,f(x))` is trivially true.

48) If assuming `x` is a new original entity, `p(x)` is trivially true, then `(all type x)p(x)` and `(some type x)p(x)` are trivially true.

49) If `(all type x)p(x)` is strictly true, then, `(some type x)p(x)` is trivially true.

50) If assuming a trivially true proposition `p` is strictly true, the other rules will say that the proposition `q` is trivially true, then the Menso model is allowed to assume `q` is trivially true but it may fail to do so.

The preceding rules are implemented in Menso software and users can write Menso codes and check them based on these rules.

In the following exemplary code, we assume that objects are the elements of a group, as defined in group theory:

```
new operator object .(object,object);
add (all object x)(all object y)(all object z)
(
    (x.y).z = x.(y.z)
);
add (some object 1)
(
    (all object x)((x.1 = x) & (1.x = x)) &
    (all object x)(some object y)((x.y = 1) & (y.x = 1))
```

```
);

noadds;

set object 1
{
    set object I
    {
        suppose
        ((all object x)((x.1 = x) & (1.x = x)) &
         (all object x)(some object y)((x.y = 1) & (y.x = 1))) &
        ((all object x)((x.I = x) & (I.x = x)) &
         (all object x)(some object y)((x.y = I) & (y.x = I)))
         {
             check (all object x)((x.1 = x) & (1.x = x)) &
                   (all object x)(some object y)((x.y = 1) & (y.x = 1));

             pick I in (all object x)((x.1 = x) & (1.x = x));
             check I.1 = I;

             check (all object x)((x.I = x) & (I.x = x)) &
                   (all object x)(some object y)((x.y = I) & (y.x = I));

             pick 1 in (all object x)((x.I = x) & (I.x = x));
             check I.1 = 1;
             check 1 = I;
         }
    }
}

select 1 in (some object 1)
(
    (all object x)((x.1 = x) & (1.x = x)) &
    (all object x)(some object y)((x.y = 1) & (y.x = 1))
);

check (all object x)((x.1 = x) & (1.x = x));
check (all object x)(some object y)((x.y = 1) & (y.x = 1));

define e = 1;
```

Here, we prove the uniqueness of the identity element of the group. The command **noadds** has no effect on the code. It just bans using **add** command "hereafter". The code:

```
define e = 1;
```

is equivalent to:

```
new object e;
add e = 1;
```

Note that **noadds** does not ban using **define** command. The above code shows how Menso language and Menso software can be used to check mathematical proofs of propositions. However, Menso language is **not** designed specifically for writing mathematical proofs.

4 A mathematical theory

Although mathematical texts can be represented in Menso model and can be written in Menso language, the Menso model can also be represented as a mathematical model.

Let the set N be the list of all possible original entities and Y be the list of all possible types. For any subset N' of N, and any function $t : N' \to Y$, we let $P(t)$ be the set of all possible propositions, made by original entities $n' \in N'$ such that n' has type $t(n')$.

The **new** command can be represented as function which takes N' and f and adds one element to N' and one new domain element to f. So new$(N_1^{'}, f_1) = (N_2^{'}, f_2)$.

The **add** command can be represented as a function add : $P(t) \times \mathcal{P}(P(t)) \to \mathcal{P}(P(t))$ where $\mathcal{P}(P(t))$ is the power set of $P(t)$ and add$(p, A) = A \cup \{p\}$.

The set of all rules of inference can be represented by a function $d : \mathcal{P}(P(t)) \to \mathcal{P}(P(t))$.

Let $E(t, y)$ be the set all possible entities with type y that can be formed using original entities in N'. The **replace** command can be represented by the function:

$$\text{replace} : \left(\bigcup_{y \in Y} E(t, y) \times E(t, y) \times P(t) \right) \times \mathcal{P}(P(t)) \to \mathcal{P}(P(t))$$

replace$((a, b, p), A) = A$ if a is not strictly equal to b.

For a type y, Let $O(t, y)$ be the set of all universal quantifiers in $P(t)$ where their dummy variables are of type y. The **pick** command for universal quantifiers can be represented by the function:

$$\text{pick}_1 : \left(\bigcup_{y \in Y} E(t, y) \times O(t, y) \right) \times \mathcal{P}(P(t)) \to \mathcal{P}(P(t))$$

For a type y, Let $S(t, y)$ be the set of all existential quantifiers in $P(t)$, where their dummy variables are of type y. The **pick** command for existential quantifiers can be represented by the function:

$$\text{pick}_2 : \left(\bigcup_{y \in Y} E(t, y) \times S(t, y) \right) \times \mathcal{P}(P(t)) \to \mathcal{P}(P(t))$$

Inside a space, we start with a subset $N_0^{'}$ of N, a function $t_0 : N_0^{'} \to Y$ and a subset A_0 of $P(t_0)$. Through a sequence of the aforementioned functions, or suppose/set/select functions to defined later, we get a sequence $(N_1, t_1, A_1), (N_2, t_2, A_2), , (N_n, t_n, A_n)$. Here, N_k is the set of all original entities, $t_k : N_k^{'} \to Y$ determines the types of original entities and A_k is the list of all strictly true propositions. Also we get $(N_{k+1}, t_{k+1}, A_{k+1})$ by applying one of the aforementioned functions to (N_k, t_k, A_k). A space function is any function that maps (N_0, t_0, A_0) to (N_n, t_n, A_n).

A *suppose-function* is a function that gets $(N_k^{'}, t_k, A_k)$, adds some proposition `p` to A_k to get $A_k^{'}$, and applies a space function to $(N_k^{'}, t_k, A_k^{'})$ and gets $(N_{k+1}^{'}, t_{k+1}, A_{k+1}^{'})$. Then it defines $A_{k+1} = A_k \cup \text{if}(A_{k+1}^{'} \setminus A_k)$ where if is a function that adds **(if p)** before the propositions in a set of propositions. Suppose-functions are the mathematical equivalent

of suppose-spaces. Similarly, we can define *set-functions* to emulate set spaces and *select-functions* to emulate select spaces. We call any of the aforementioned functions a *Menso function*. Define $M = \{(N', t, A) \mid N' \subseteq N, t : N' \to Y, A \subseteq P(t)\}$. Then a Menso function can be of the form $m : M \to M$.

According to the Definitions of [1], the model above, is a *formal theory*, The set M is the set of all propositions in this formal theory and the Menso functions are the set all rules of inference in it. However, one should start the formal theory after the **noadds** command, so that we have a set of axioms. Now, all we know about formal theories can be applied to the Menso model.

5 Application

The Menso model can be used for checking mathematical proofs. But it is not designed specifically for writing mathematical texts. For mathematical texts, many more facilitating commands are needed.

The Menso model is a simulation of the LSHM. The LSHM is used in many circumstances in a human lifetime. For example, the LSHM is used when a human plays chess. Since Menso model is simulating the LSHM, it must have something to do with how to play chess. To think about how to move in a chess game, the LSHM has to know if a move is good or bad. Here, the LSHM, checks the move. In fact, a move is a new chessboard arrangement, which is legal after the previous one. The LSHM, thinks if the new arrangement is good or bad, however, it is not necessarily a sophisticated chess engine. Instead it may be an amateur with a few simple propositions with which it can check if the new arrangement is good. "good" means "better than other possible new arrangements".

Experience may have **add**ed the proposition "Losing the queen is bad" to the LSHM Truth list. So the LSHM, checks if in the new arrangement, the queen is lost. There may be a lot of other propositions added as axioms to the LSHM. Considering all these propositions and combining them using the rules of inference, the LSHM decides if a move is good. The propositions based on experience can easily be added to the Menso model using **add** command. Then Menso codes about whether the new chessboard arrangement (after a move,) is good, can be checked by the Menso model. However, writing and designing such code is not what the Menso model does. It just checks them.

To see if a chessboard arrangement is good, there must be a general Menso code part, which can be applied to any arrangement. And there must be a specific part for a specific game:

```
1  // all games:
2  new prop IsTheKingChecked(object);
3  new prop WasQueenAvailable(object);
4  new prop IsQueenAvailble(object);
5  new prop result1;
6  new prop result2;
7
8  add (all object game)(if !IsQueenAvailble(game))(if
       WasQueenAvailable(game))result1 = false;
9  add (all object game)(if !IsTheKingChecked(game))result2 = false;
10
```

```
// specific game:
new object game;
add IsQueenAvailble(game);
add IsTheKingChecked(game);
```

Note that the above code is just a demonstration and a complete usable code has to be much longer. In the general part of the code, the general rules about whether an arrangement is good are added. For example the proposition:

```
add (all object game)(if !IsQueenAvailble(game))(if
    WasQueenAvailable(game))result1 = false;
```

is a general rule. Here, `game` is a game arrangement and `result1` is an initial result, one of factors leading to the final decision. Initially, there will be a few results based on different assessment rules. The final results will be based on the initial results. The same thing happens in a human mind (the LSHM).

The Menso model cannot be used to understand a dynamically changing situation. It just can be used to assess a current situation more deeply. For example, using the Menso model, we can check if a current static chessboard arrangement is good or bad. A player has to prove that a move (that is, the new arrangement) is good before the move takes place.

Now, it is clear how the LSHM, and so the Menso model, works in playing a chess game. The LSHM works in the same way in real life. When the mind thinks about how to react to the circumstances, it encounters in real life, it uses the LSHM to see if the next step will be a good arrangement or a bad one. Based on this, the mind will react properly. This is how the Menso model can be used in simulating the human mind.

Menso model cannot thinks. It just can check thoughts. It is a very important first step in simulation how the LSHM thinks. In fact, the Menso model can be developed into a thinking model. Thinking softwares have been developed for games and others things, however, if a thinking software is developed based on a developed version of Menso model, it will be a general-purpose thinker just like the human mind, but it may have faults and imperfections just like the human mind.

The Menso language can be represented as a visual language and can be developed to a logic game in which serious logical and mathematical problems can be solved through a computer game.

References

[1] Karlis Podnieks, Vilnis Detlovs. Introduction to mathematical logic. DOI: 10.13140/RG.2.2.30866.66245