A Formal Model for Emulating the Generation of Human Knowledge in Semantic Memory

Antonio Cerone(✉) and Graham Pluck

Department of Computer Science, School of Engineering and Digital Sciences, Nazarbayev University, Nur-Sultan, Kazakhstan
{antonio.cerone,graham.pluck}@nu.edu.kz

Abstract. The transfer of information processed by human beings from their short-term memory (STM) to their semantic memory creates two kinds of knowledge: a semantic network of associations and a structured set of rules to govern human deliberate behaviour under explicit attention. This paper focuses on the memory processes that create the first of these two kinds of knowledge. Human memory storage and processing are modeled using the Real-time Maude rewrite language. Maude's capability of specifying complex data structures as many sorted algebras and the time features of Real-Time Maude are exploited for (1) providing a means for formalising alternative memory models, (2) modelling in silico experiments to compare and validate such models. We aim at using our model for the comparison of alternative cognitive hypothesis and theories and the analysis of interactive systems.

Keywords: Cognitive science · Human memory models · Formal methods · Rewriting logic · Real-Time Maude

1 Introduction

Human *semantic memory* is a core aspect of declarative *long-term memory (LTM)*, comprised of propositional information, specifically word meanings and facts. An example of semantic memory being the fact that a penguin is a bird. Clearly, such information must be acquired from the environment, such as reading, or formal education.

In terms of information flow within the human memory system, sensations of the environment (e.g., sounds heard) are first processed by modality-specific sensory stores, which we globally call *sensory memory*. Items attended in those sensory stores persist for very short time periods and are then passed to a temporary *short-term memory (STM)* limited capacity store (about 7 items [21], usually

Work partly funded by Project SEDS2020004 "Analysis of cognitive properties of interactive systems using model checking", Nazarbayev University, Kazakhstan (Award number: 240919FD3916).

J. Bowles et al. (Eds.): DataMod 2020, LNCS 12611, pp. 104–122, 2021.
https://doi.org/10.1007/978-3-030-70650-0_7

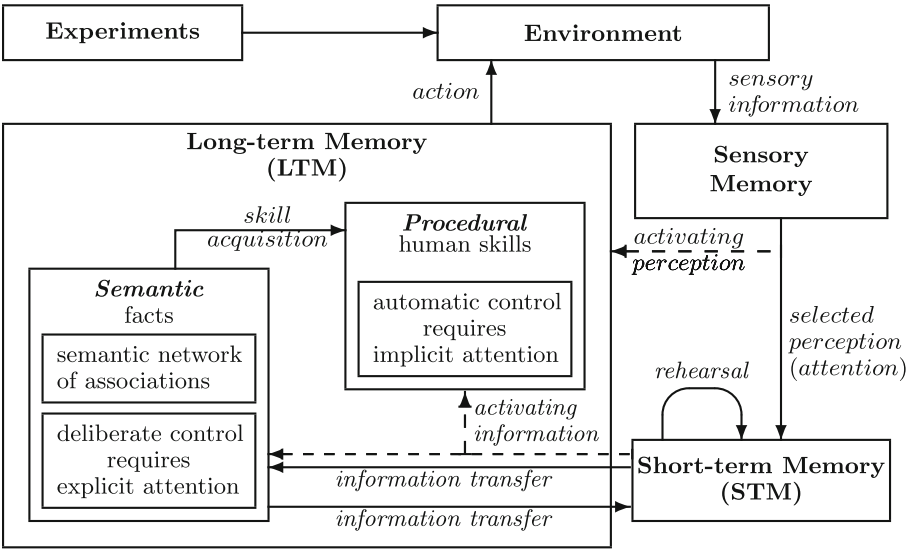

Fig. 1. Human memory architecture

called *chunks* [24], for healthy adults), with rapid access and rapid decay. From STM information passes to LTM, which has a virtually unlimited capacity and where information is organised in structured ways, with slow access but little or no decay. Finally, within LTM, information repeatedly used in practice activities may move from semantic memory to *procedural memory*, thus determining skill acquisition. In fact, the transfer from semantic memory to procedural memory refers to our skills and consists of rules and procedures that we unconsciously use to carry out tasks, particularly at the motor level.

This structure of human memory is depicted in Fig. 1, where continuous arrows show transfer of information between memory components while dashed arrows denote information stored in the source component that activate memory processing in the target component.

In previous work [12] we considered the transfer of information from semantic memory to STM as a means to retrieve knowledge stored in semantic memory, in order to perform in silico experiments in which an emulated human subject answers questions that refer to specific knowledge domains. Such previous work builds up from the definition of the Behaviour and Reasoning Description Language (BRDL) [11] and its implementation for the emulation of human reasoning [13] as well as from earlier work, which was based on the Human Behaviour Description Language (HBDL), a subset of BRDL, and proposed and partly implemented an approach to the modelling of automatic and deliberate human behaviour while interacting with an environment consisting of heterogenous physical components [9,10].

A similar approach to ours was developed by Broccia *et al.* [5], who, driven by the specific objective of modelling human multitasking, used Real-time Maude to extend our initial untimed framework [9]. In their work, however, time is used to model non-cognitive aspects, such as the duration of the task, which is an interface-dependent outcome of the interaction process, and external aspect, such as the delay due to the switching from one task to another. In contrast to Broccia *et al.* we focus on the human component and model the duration of the mental process, which is an important aspect of human cognition.

In this paper we focus on the transfer of factual knowledge from STM to semantic memory and model this memory process using Real-Time Maude [25, 26]. The large time gap between the rapid decay of the information stored in STM (of the order of seconds [22]) and the little or no decay of the information in semantic memory (and in LTM, in general) has pushed research in cognitive psychology to look for something in between. In fact, nowadays, among cognitive scientists there is a tacit acceptance of an intermediate memory stage that somehow bridges the gap between STM and LTM, a form of memory that operates in the range of minutes to a few hours and, possibly, extending even further in time. Alan Dix calls this intermediate level *mezzanine memory* and believes it likely to be carried through long-term potentiation [17,18] while modern neuroscience locates it in the hippocampus [18,19].

The minutes-hours magnitude of decay time makes it difficult to study mezzanine memory in an experimental setting: too long to observe decay within a single experimental session and too short to measure the effect of the decay between two consecutive experimental sessions. For this reason there is little mention of such a kind of intermediate memory in the literature. In *reading comprehension* it is sometime called *long-term short-term memory*, while other time it is identified with *working memory* and is thus seen as a different level of memory that overlaps with both STM and LTM. For example in the context of reading a book, there is an interplay between what we store in STM from what we are currently reading, the recall of what we have read in the same chapter several minutes before, and the more long-term memory of the book plot, which can date back to several days, when we read previous chapters.

A more extreme position is the 'Levels of Processing' framework of Craik and Lockhart [16]. They argue that deeper (semantic) processing causes slower decay, but they are talking of a unitary memory store which encompasses both STM and LTM processes. Nevertheless, as they consider only one memory store, their reduced decay rate argument applies to short-term storage. This idea of 'deeper processing' is equivalent to elaborative rehearsal.

In this work we aim at proposing an in silico approach to filling the experimental gap between STM and LTM. Although it is hard to carry out experiments on human subjects to validate hypotheses about the mezzanine memory or, more in general, about the mechanism underlying the transfer of information between STM and LTM, we expect that the in silico emulation of such experiments would produce important insights into this matter. Moreover, this approach would, in some sense, blur the difference between experimental investi-

gation and case study-based investigation. In fact, the in silico emulation would allow the researcher to consider a large amount of data and a specific human memory model for a single individual, and perform an intensive analysis within a much shorter time than in a real-life case study.

The rest of this paper is organised as follows. The Real-Time Maude code illustrated in this paper can be downloaded from a GitHub repository[1]. Section 2 briefly introduces informal cognitive models of the information transfer from STM to LTM from the cognitive psychology literature, with reference to maintenance rehearsal and elaborative rehearsal. Section 3 first provides a brief highlight of Real-Time Maude and refers to the sections of the paper where the different aspects of the language are illustrated. Then it extensively presents the Real-time Maude formal models for the information to be processed and for its STM and semantic memory stores. Section 4 is devoted to the formal models of memory processes: perception in Sect. 4.1, maintenance rehearsal in Sect. 4.2, elaborative rehearsal in Sect. 4.3, and the actual learning process that consolidates knowledge in semantic memory in Sect. 4.4. Section 5 illustrates our formal models using the two cases of *rote learning*, in Sect. 5.1, and *effective learning*, in Sect. 5.2. Section 6 concludes the paper.

2 Cognitive Models for Information Transfer

The sequential processing from STM to LTM is captured by several cognitive models, most commonly, the Multistore Model [1] and Working Memory Model [3], however, both are equivalent in proposing structural distinctions between phonologically-coded STM storage for verbal content and a separate LTM. The STM and LTM distinction is known partially through cognitive neuroscience, as it is observed that brain lesions can selectively impair either phonological STM capacity or LTM contents (either semantic or episodic). Furthermore, severe reductions in STM capacity caused by brain lesions, such as the inability to hold more than two items in phonological STM simultaneously, also prevent the acquisition of new semantic memory entries in LTM [4]. Thus, indicating that items for storage in semantic memory must first be processed within phonological STM.

The mechanism by which information transfers from STM to semantic memory is *elaborative rehearsal* [2]. This involves using the items within STM to access existing entries within semantic memory. This deep processing, based on semantics, increases the chance that the items will become stored in LTM, probably by strengthening their appropriate connection within the nexus of semantic entries. This elaborative rehearsal within STM, which induces transfer to LTM, can be contrasted with *maintenance rehearsal*. This latter form of processing can be seen as phonological looping of the items to renew their representations within STM, thereby delaying signal decay.

[1] http://github.com/AntonioCerone/Publications/tree/master/2020/DataMod/Cognition.

As shown in Fig. 1, among the sensory information briefly stored in sensory memory, attention selects some and transfers it to STM. Information in STM can then be used

- to activate the deliberate control in semantic memory or the automatic control in procedural memory, or,
- after elaborative rehearsal, to create associations as well as deliberate control rules in semantic memory.

Moreover, information can be retrieved from semantic memory and transferred to STM. Furthermore, the repeated use of rules for deliberate control in semantic memory produces skill acquisition and the resultant creation of rules for automatic control in procedural memory. In previous work we have formally modeled human behaviour under deliberate and automatic control [13] as well as information retrieval from semantic memory [12].

3 Real-Time Maude Models of STM and Semantic Memory

Real-Time Maude [25,26] is a formal modeling language and high-performance simulation and model-checking tool for distributed real-time systems. It is based on Full Maude, the object-oriented extension of Core Maude, which is the basic version of Maude. Real-Time Maude makes use of

- algebraic equational specifications in a functional programming style to define data types;
- labeled rewrite rules to define local transitions;
- tick rewrite rules to advance time in the entire system state.

Maude *equational logic* supports declaration of *sorts*, with keyword `sort` for one sort, or `sorts` for many. A sort `A` may be specified as a subsort of a sort `B` by `subsort A < B`. Operators are introduced with the `op` (for a single definition) and `ops` (for multiple definitions) keywords:

$$\texttt{op}\ f\ \texttt{:}\ s_1 \ldots s_n\ \texttt{->}\ s.$$
$$\texttt{ops}\ f_1\ f_2\texttt{:}\ s_1 \ldots s_n\ \texttt{->}\ s.$$

Operators can have user-defined syntax, with underbars '`_`' marking the argument positions and '`‘’`' to denote a space. Some operators can have *equational attributes*, such as `assoc`, `comm`, and `id`, stating that the operator is associative, commutative and has a certain identity element, respectively. Such attributes are used by the Maude engine to match terms *modulo* the declared axioms. An operator can also be declared to be a constructor (`ctor`) that defines the carrier of a sort. Axioms are introduced as equations using the `eq` keyword or, if they can be applied only under a certain condition, using the `ceq` keyword, with the condition introduced by the `if` keyword. Variables used in equations are placeholders in a mathematical sense and cannot be assigned values. They must be

declared with the keyword `var` for one variable, or `vars` for many. The use of the `owise` (or `otherwise`) equational attributes in an equation denotes that the axiom is used for all cases that are not matched by the previous equations. All Maude statements are ended by a dot.

In the rest of this section we define the formal infrastructure that we use to model STM, semantic memory and the information items stored in them. Some additional details about Core Maude data types are illustrated in Sect. 3.2. The Full Maude syntax for classes is illustrated in Sect. 4 while the syntax for messages as well as labelled rewrite rules and Real-Time Maude tick rewrite rules are illustrated in Sect. 4.1.

3.1 Facts, Questions and Goals

Humans, throughout their lives, acquire knowledge of the *facts of the real world* and are able to refer to them and reason about them using *declarative propositions*. Since a declarative proposition is just a natural language description of a fact, we will often use the word 'fact' also to denote the declarative proposition that describes it. Moreover, human beings reason about facts, and organise such facts in their semantic memory triggered by *questions* that are put to them, or they put to themselves.

We model a *fact* in Real-Time Maude as follows:

```
a "dog" is a "animal".
```

The article 'a' is used for any noun, although this is ungrammatical when the noun starts with a vowel as in the case of 'animal'. Other examples of facts are:

```
a "animal" can "breathe"
a "dog" can "move"
a "dog" can "bark"
a "cat" cannot "bark"
a "cat" is not a "dog"
```

In such examples `"animal"`, `"dog"` and `"cat"` are categories, `"breathe"`, `"move"` and `"bark"` are attributes and `is a`, `can`, `is not a` and `cannot` are types to be applied to attributes. Categories may also be used as attributes as in `a "dog" is a "animal"`. The application of a type to an attribute, such as `is a "animal"` or `can "breathe"`, is called *typed attribute*.

A *question* may be of several kinds [12]. In this paper we consider only `can` questions and `is a` questions, such as:

```
can a "dog" "breathe" ?
is a "dog" a "animal" ?
```

The two questions, which have the same structure, will be answered by one declarative proposition, by stating the fact either negatively or positively.

A *goal* specifies what a human being aims at achieving as the result of an activity. Goals drive deliberate behaviour, which exploits the knowledge in

semantic memory, but do not affect automatic behaviour, which exploits the knowledge in procedural memory [9,10,13]. In deliberate behaviour, goals activate *attention*, a selective processing activity that aims to focus on one aspect of the environment while ignoring others, thus allowing the human mind to focus on goal-relevant stimuli in the environment (*explicit attention*). Another form of attention, called *implicit attention*, is grabbed by sudden stimuli that are associated with the current mental state or carry emotional significance, thus determining automatic behaviour. For the purpose of this paper we only consider explicit attention.

We model a goal by considering two aspects: a *domain of knowledge* to which we refer and what we gain once the goal is achieved. For example, rehearsing facts of our knowledge about dogs may be our goal. Then `"dogs"` is the domain and `"rehearsed"` is what we gain as the achievement of our goal. Moreover, the goal of rehearsing a fact will activate our explicit attention to focus on the presence of fact descriptions in the environment, such as a written statement describing a fact.

3.2 Modelling Basic Information Items and Goals

We model *basic information items* (facts and questions) and *goals* in Real-Time Maude as follows:

```
sorts Fact Question Domain BasicItem Item Goal .
subsorts Fact Question < BasicItem < Item .
subsort Goal < Item .

sorts BasicItemSet ItemSet EmptyItemSet .
subsort BasicItem < BasicItemSet .
subsorts EmptyItemSet < BasicItemSet < ItemSet .
subsort Item < ItemSet .
op none : -> EmptyItemSet [ctor] .
op _;_ : BasicItemSet BasicItemSet ->
             BasicItemSet [ctor assoc comm id: none format (b o n b)] .
op _;_ : ItemSet ItemSet -> ItemSet [ctor ditto] .
op _;_ : EmptyItemSet EmptyItemSet -> EmptyItemSet [ctor ditto] .
op goal : Domain BasicItemSet Nat Nat -> Goal [ctor] .
```

Sorts `Facts`, `Question` and `Goal` model facts, questions and goals, respectively. The first two are subsorts of `BasicItem`. Sorts `BasicItem` and `Goal` are subsorts of `Item`.

Both `BasicItem` and `Item` are organised into sets by defining the two sorts `BasicItemSet` and `ItemSet` using the `;` user-defined infix operator, which is given the appropriate equational attributes for the properties that characterise sets. The `ditto` equational attribute is a short form for all attributes of the previous sort declaration. The `format` equational attribute is used to format the output with spaces, colours and newlines in order to make it more readable. By declaring `BasicItem` as a subsort of `BasicItemSet` and `Item` as a subsort of `ItemSet` we implicitly define singletons of sorts `BasicItemSet` and `ItemSet`.

However, the `none` empty set needs to be explicitly introduced as the only element of sort `EmptyItemSet`, which is subsort of `BasicItemSet`, in turn subsort of `ItemSet`.

The sorts `Category` and `Attribute` include Maude-predefined sort `String` as a subsort:

```
sorts Category Attribute TypedAttribute .
subsort String < Category < Attribute .
subsort String < Domain .
```

This allows us to freely use any string, which is enclosed by double quotes in Maude syntax, as a category or attribute, while leaving open the option to use other representations in possible extensions of the module. The elements of sorts `TypedAttributes`, `Facts` and `Questions` are instead defined using constructors, since they have special relationships between each other and need to be manipulated in special, distinct ways by the Maude engine. Attribute types `can` and `is a` as well as facts and questions constructed using them are modeled as follow:

```
ops can_ is‘a_ : Attribute -> TypedAttribute [ctor] .
ops cannot_ is‘not‘a_ : Attribute -> TypedAttribute [ctor] .
op a__ : Category TypedAttribute -> Fact [ctor] .
ops can‘a__? is‘a__? : Category Attribute -> Question [ctor] .
op _is‘negative‘of_ : TypedAttribute TypedAttribute -> Bool .
```

One of these special relationships is the negation: `cannot` and `is not` are the negations of `can` and `is`, respectively. Negation is expressed as an infix boolean operator `is negative of`.

Goals are defined using the constructor `goal`. In addition to the two aspects mentioned in Sect. 3.1, the knowledge domain, of sort `Domain`, and the achievement, of sort `BasicItemSet`, the constructor `goal` has two additional arguments of sort `Nat`. The first `Nat` argument models the number of times the goal is planned to be achieved, for example, the number of time we want to rehearse a given fact. The second argument is the *goal determination*, namely how determinate we are in achieving the goal. However, the usage of this argument is beyond the purpose of our paper.

3.3 Modelling Explicit Attention and Goal Achievements

In order to model the explicit attention we need to extract achievements from the goals. We use the operators `isAchievement` and `explicitAttention`, which are defined as follows:

```
sort Achievement .
subsort Achievement < BasicItem .
ops foundAnswer rehearsed : -> Achievement [ctor] .
ops isAchievement explicitAttention : BasicItem ItemSet -> Bool .

vars BI1 BI2 : BasicItem .   var BIS : BasicItemSet .
var IS : ItemSet .    var D : Domain .    vars DET REP : Nat .
eq isAchievement(BI1, goal(D, (BI2 ; BIS), DET, REP) ; IS) =
```

```
          BI1 == BI2 or isAchievement(BI1, IS) .
eq isAchievement(BI1, IS) = false [owise] .

var Q : Question .   var F : Fact .
eq explicitAttention(Q,
               goal(D, (foundAnswer ; BIS), DET, REP) ; IS) = true .
eq explicitAttention(F,
               goal(D, (rehearsed ; BIS), DET, REP) ; IS) = true .
eq explicitAttention(BI1, IS) = false [owise] .
```

For the purpose of our paper we only consider two achievements: `foundAnswer`, which drives attentions to questions to be answered, and `rehearsed`, which drives attentions to facts to be rehearsed.

3.4 STM—Short-Term Memory

STM is normally used as a buffer where the information that is needed for processing activities is temporarily stored. In our previous work [12] we modeled the mechanism for emptying STM when the stored information is no longer needed by associating a *decay time* with each information item stored in STM. A fixed value of decay time was associated with the information item at the moment this was first stored in STM, then decreased according to the passing of time and, when it was down to 0, the item was removed from STM. We now extend this model by separating the actual *lifetime* of an information item from its decay time at the current time of its lifetime. These two time aspects of an STM-stored information item are the two time arguments of the constructor `chunk_decay_of_`, which is defined as follows:

```
sorts Chunk ShortTermMemory .
subsort Chunk < ShortTermMemory .

op chunk_decay_of_ : ItemSet TimeInf TimeInf -> Chunk [ctor] .
op emptySTM : -> ShortTermMemory [ctor] .
op _;_ : ShortTermMemory ShortTermMemory ->
     ShortTermMemory [ctor assoc comm id: emptySTM format (b o n b)] .

eq chunk ITEM:Item decay 0 of T:TimeInf = emptySTM .
```

Another extension with respect to our previous model is the use of the notion of *chunk*. Information items can be aggregated to form chunks whereby STM capacity refers to the number of chunks rather than the number of information items. The exact mechanism of chucking in human memory is poorly understood. However, it is thought that conjoining items by their associations within LTM (e.g. common word collocations), such that a person's past experience would influence the chunk selection, is part of the mechanism, as it is a form of data compression where pieces of information are renamed under a new label, such as '6' and '1' being labeled as '61' [24].

As shown in the Maude code above, operator `chunk_decay_of_`, whose syntax is user-defined, associates two possibly infinite times with the aggregate set

of information item that forms the chunk (an element of sort `Chunk`). The sort `TimeInf` extend the sort `Time` with value `INF` to model an infinite time. Obviously, an infinite decay time means no decay at all; although this is unrealistic for human STM, it may be useful in testing and calibrating newly defined models. The sort `ShortTermMemory` defines sets of chunks (of sort `Chunk`, which is a subsort of sort `ShortTermMemory`) using the constructor `;`. The nullary operator `emptySTM` defines an STM that does not contain any information, that is, an empty set of chunks. The equation on the constructor `chunk_decay_of_` ensures that if the decay time has reached zero, the chunk is removed from STM.

Both times specified in the chunk are initialised with a standard decay time, whose value is fixed *a priori* when the chunk is first stored in STM. Then the first time argument (decay time) decreases according to the passing of time whereas the second argument (lifetime) does not change as long as the chunk is passively kept in STM without being used. However, if the chunk is used to carry out maintenance rehearsal or to access information in semantic memory, then also the lifetime may increase, thus consolidating the information chunk and paving the way for its transfer to semantic memory. This modelling approach is consistent with the cognitive neuroscience finding that the phonological looping of items renews their representations within STM, thereby delaying signal decay. In Sect. 4.2 and Sect. 4.3 we present formal models of this mechanism for maintenance rehearsal and elaborative rehearsal, respectively.

3.5 Semantic Memory

Semantic memory has been modeled in our previous work to represent knowledge about rules that govern human behaviour [9,10,13], acquired inference rules [13] and facts of the real world [12,13]. In this paper we focus on fact representation and explain our approach using a case study taken from our previous work [12].

A *fact representation* in semantic memory is modeled as follows:

```
sorts FactRepresentation SemanticMemory .
subsort FactRepresentation < SemanticMemory .

op emptySemantic : -> SemanticMemory .
op _:_|-_->|_ : Domain Category Time TypedAttribute ->
     FactRepresentation [ctor format (!r o b o r o b o)] .
op __ : SemanticMemory SemanticMemory ->
     SemanticMemory [ctor assoc comm id: emptySemantic format (o n o)] .
op _is‘negated‘in_ : Fact SemanticMemory -> Bool .
```

The finite time that appears as one of the arguments of the fact representation constructor is the retrieval time (RT) of the fact from semantic memory. As an example, the fact that ‘a dog is an animal’ is represented within the semantic domain `"dogs"` as

```
"dogs" : "dog" |- 1 ->| is a "animal"
```

and this form of generalisation can be retrieved from semantic memory in 1 time unit. The more specific category of a generalisation (e.g., `"dog"`) inherits

all typed attributes of the more generic category (e.g., `"animal"`) unless the attribute is redefined at the more specific category level. Therefore,

```
"animals" : "animal" |- 1 ->| can "move"
```

which is an association of a category with a typed attribute, specifies that 'an animal can move' and, since an animal is a generalisation of a dog, such a typed attribute is inherited by the category `"dog"` ('a dog can move').

Fact representations are defined as elements of the sort `FactRepresentetion`. The semantic memory is modelled by the sort `SemanticMemory`, which is defined as a set of fact representations. The constructor `__` denotes that sets of fact representations are created by justapposition, with no written operator. The constructor `emptySemantic` denotes an empty semantic memory.

4 Modelling Memory Processes

In this section we show how the different memory components are put together as a Full Maude class object and, in Sect. 4.1, 4.2, 4.3 and 4.4 how tick rewrite rules are used to model memory processes.

A declaration `class` C `|` $att_1 : s_1, \ldots, att_n : s_n$ declares a class C with attributes att_1 to att_n of sorts s_1 to s_n. An *object* of class C is represented as a term `<` $O : C \mid att_1 : val_1, ..., att_n : val_n$ `>` of sort `Object`, where O, of sort `Oid`, is the object's *identifier*, and where val_1 to val_n are the current values of the attributes att_1 to att_n.

We model the structure of human memory using the following Real-Time Maude class:

```
class HumanMemory | shortTermMem : ShortTermMemory,
                    semanticMem : SemanticMemory .
```

STM is modelled by the field `shortTermMem`. Semantic memory is modelled by the field `SemanticMem`. Note that this model has been simplified for the purpose of this paper and does not include some other memory components, such as procedural memory, and memory attributes, such as the cognitive load.

4.1 Perception

Although sensations of the environment are initially processed by *sensory memory* before being passed to STM, in our model we assume that *perceptions* available in the environment are selected using attention and directly transferred to STM. In order to model perceptions we use Full Maude messages. A message is an element of the pre-defined sort `Msg` and has the same syntax as an operation but, in addition, is also an element of the pre-defined sort `Configuration`. The system state is a term of sort `Configuration`, and is a *multiset* of objects and messages. Multiset union is denoted by an associative and commutative juxtaposition operator, so that rewriting is *multiset rewriting*.

Therefore, a perception is modelled as a message using the constructor `perc` as follows:

```
sorts Perception TimedBasicItem FutureBasicItem .
subsorts Perception < Msg .

op _for_ : BasicItem TimeInf -> TimedBasicItem [ctor] .
op perc : TimedBasicItem -> Perception [ctor] .
var BI : BasicItem .
eq perc(BI for 0) = none .
```

The persistence of a perception in the environment is modeled by the constructor `for`. Note that the time may be infinite to denote that the perception persists forever. The equation ensures that if the persistence time has reached zero, the perception is removed from the configuration (`none` is the empty configuration).

The processing of information within memory components and the transfer of information between components are modeled using Real-Time Maude tick rewrite rules. Labeled rewrite rules

$$\texttt{rl}\ [l] : t \texttt{ => } t' \quad \text{or} \quad \texttt{crl}\ [l] : t \texttt{ => } t'\ \texttt{if}\ cond$$

define local transitions from state t to state t'. Tick rewrite rules

$$\texttt{rl}\ [l] : \{t\} \texttt{ => } \{t'\}\ \texttt{in time}\ \Delta \quad \text{or} \quad \texttt{crl}\ [l] : \{t\} \texttt{ => } \{t'\}\ \texttt{in time}\ \Delta\ \texttt{if}\ cond$$

advance time in the *entire* state t by Δ time units.

The following rewrite rule models the storage in STM of information perceived from the environment:

```
crl [perception-explicit-storage] :
   (perc (BI for T))
   < H : Human | shortTermMem : STM >
   REST
=>
   < H : Human | shortTermMem :
      (chunk BI decay DECAY-TIME of DECAY-TIME) ; idle(STM,TP) >
   (perc (BI for (T monus TP)))
   REST
in time TP
if TP := tp(perc (BI for T)) /\ IS := removeTime(STM) /\
    explicitAttention(BI, IS) /\  not isItemIn(BI, IS) .
```

The basic information `BI`, which is available in the environment for the time `T`, is stored in STM if it is not already there (`not isItemIn(BI, IS)`) and the untimed content `IS` of STM drives explicit attention on it (`explicitAttention(BI, IS)`). As we have seen in Sect. 3.3, the `explicitAttention` operator checks whether the structure of `BI` has a matching information in a goal stored in STM. Operator `tp` gives the time for storing the information in STM. As suggested by Kolers [20] we assign 100 milliseconds (ms) as the time to move orthographic information to a phonological storage, such as the one represented by STM, by defining operator `tp` as a constant.

4.2 Maintenance Rehearsal

As mentioned in Sect. 1 there is no agreement on what fills in the decay time gap between STM and LTM. However, in order to explain how learning occurs through maintenance rehearsal, Burgess and Hitch [6,7] claim the existence of two learning mechanisms operating in parallel during STM storage. This is part of a model of working memory [4]. That model includes a 'fast' short-term learning process that is the basis of STM and is associated with trace decay, and a 'slow' learning process that gradually leads to LTM (in the same cells), but enhances within STM. These dual learning processes are said to operate in parallel, and are biologically plausible. They are used to explain the Hebb Repetition effect, which is that if the same list is repeated several times, recall from STM is improved, suggesting a longer-range learning mechanism (longer than rehearsal resetting the decay level). As that model emphasises rehearsal as being the basis of the fast learning mechanism, it would be reasonable to assume that the parallel slow learning mechanism would be delaying decay.

The following rewrite rule models the effect of the learning mechanism proposed by Burgess and Hitch:

```
crl [perception-explicit-maintenance] :
   (perc (BI for T))
   < H : Human | shortTermMem : (chunk BI decay T1 of T2) ; STM >
   REST
=>
   < H : Human | shortTermMem :
      (chunk BI decay T2 of maintenance-effect(T2)) ; idle(STM,TP) >
   (perc (BI for (T monus TP)))
   REST
in time TP
if TP := tp(perc (BI for T)) /\ T2 < STM-TO-LTM-THRESHOLD /\
   IS := removeTime(STM)  /\  (rehearsed, IS) .
```

The operator `isAchievement`, which was defined in Sect. 3.3, is used to activate the rehearsal loop when the untimed content `IS` of STM includes a goal having `rehearsed` as its achievement.

Burgess and Hitch's 'fast' short-term learning process is controlled by the `T1` decay time, while the parallel slow learning mechanism is represented by an increase in the `T2` information lifetime. Such an increase can be set in a way that can accommodate a specific hypothesis or theory by using appropriate equations to define operator `maintenance-effect`. For example, we can model a small, constant increase at each rehearsal loop or we may implement Naveh-Benjamin and Jonides' suggestion [23], that the first rehearsal is the most important, because it involves producing the rehearsal plan, with subsequent loops of that plan adding little to the transfer to LTM.

Finally, the decay time `T1` is reset to the current lifetime `T2` (before increase) and the condition on the `STM-TO-LTM-THRESHOLD` keeps the rehearsal process alive until the appropriate threshold for transferring to LTM is reached.

4.3 Elaborative Rehearsal

Suppose that we know that `a "animal" can "move"` but we do not know that `a "dog" is a "animal"`. Once we read this new fact, which thus enters our STM, elaborative rehearsal could be activated by questions about dogs that require the retrieval of animal's attributes. The question `can a "dog" "move" ?` would allow us to use the just read new fact (`a "dog" is a "animal"`) to retrieve the answer as an attribute of category `"animal"`. As explained in Sect. 2, the usage of information within STM to access existing entries within semantic memory would increase its chance to become stored in semantic memory.

In our previous work [12], we introduced a tick rule for answering a `can` question that explored the knowledge in semantic memory to answer a question stored in STM, but without using any fact possibly stored in STM in the retrieval process. The following tick rewrite rule extends our previous tick rule by (1) using information stored in STM in combination with the knowledge in semantic memory in order to answer the question; (2) modifying the lifetime of the facts stored in STM that are used in the retrieval process.

```
crl [retrieval-can-elaborative-is-a] :
   < H : Human |
      shortTermMem : (chunk goal(D, foundAnswer, N1, N2) decay T1 of T2) ;
                     (chunk (can a C A ?) decay T3 of T4) ; STM,
      semanticMem : S >
   REST
=>
   < H : Human |
      shortTermMem : NEW-GOAL-CHUNK ;
                     (chunk F decay DECAY-TIME of DECAY-TIME) ;
                     idle(NEW-STM, T),
       semanticMem : S >
   idle(REST,T)
in time T
if F := a C can A /\ IS := removeTime(STM) /\ not isItemIn(F, IS) /\
   T := canRetrievalTime(C, A, S, IS) /\ T <= MAX-RETRIEVAL-TIME /\
   NEW-STM := elaborativeRetrieval(C, A, S, STM) /\
   not ( F is negated in S ) /\ not isItemIn(a C cannot A, IS) /\
   NEW-GOAL-CHUNK := if N1 > 0
         then (chunk goal(D, foundAnswer, N1 monus 1, N2)
                     decay DECAY-TIME of DECAY-TIME)
         else emptySTM fi .
```

The retrieval time `T` is calculated using the `canRetrievalTime` operator, which searches in the semantic memory `S` for a fact representation with category `C`n and typed attribute `can A`, where either `C`n = `C` or `C`n is a generalisation of `C` through a chain of facts

`a C is a C1`, `a C1 is a C2`, ... `a C`$(n-1)$ `is a C`n

which either have representations in semantic memory or are in STM.

The operator `elaborativeRetrieval` performs a similar search in semantic memory but with the purpose of modifying the lifetime of all facts stored in

STM that are used in the process. As in the case of maintenance rehearsal, it is unknown to which extent to modify the lifetime. Again, we can then set such a modification in a way that can accommodate a specific hypothesis or theory. This is achieved by appropriately defining an operator `elaborative-effect`, similar to `mantainence-effect`, and use it within the definition of the operator `elaborativeRetrieval`.

4.4 Transfer from STM to Semantic Memory

The following rule models the transfer of information from STM to Semantic Memory.

```
crl [from-STM-to-LTM-fact] :
   < H : Human |
      shortTermMem : (chunk goal(D, rehearsed, N1, N2) decay T1 of T2) ;
                     (chunk (a C TA) decay T3 of T4) ; STM,
      semanticMem : S >
   REST
=>
   < H : Human |
      shortTermMem : STM,
      semanticMem : (D : C |- 1 ->| TA) S >
   REST
if T4 >= STM-TO-LTM-THRESHOLD .
```

Since the cognitive psychology literature does not provide any information on possible values of the `STM-TO-LTM-THRESHOLD` threshold, we should give an estimation depending on the specific hypothesis or theory we consider in defining the lifetime increments in the cases of maintenance rehearsal and elaborative rehearsal. For instance, if we follow Naveh-Benjamin and Jonides' hypothesis [23], that the first rehearsal loop is the most important, then a reasonable decay time cut-off could be the time taken to get the phonological code into STM. In Sect. 4.1 we considered Kolers' suggestion [20] that the time to move orthographic information to a phonological storage in STM should require approximately 100 ms. However, the conversion of the ortographical format into the phonological format suitable for the storage is achieved by subvocalization which, according to Mueller and Krawitz [22], requires between 1.5 and 2 s. This is in accordance with the original Collins and Quillian experiments [15], which used a two second presentation to their human participants. Therefore, a reasonable threshold should be at least 3 or 4 s, by considering 1.5–2 s for the initial transfer and another 1.5–2 s for the first rehearsal loop.

5 In Silico Experiments

In order to perform in silico experiments we need to define an infrastructure to plan experiments and make them actual at the specified time. We call *planned*

experiment the presentation of a single piece of orthographic information (orthographic representation of a fact or a question) to a human subject, together with the time that must pass before it is actually presented.

The experimental infrastructure is modeled as follows:

```
sorts FutureBasicItem SingleExperiment Experiment .
subsort SingleExperiment < Experiment < Msg .
op _in_ : TimedBasicItem Time -> FutureBasicItem [ctor] .
op exp : FutureBasicItem -> SingleExperiment [ctor] .
op noExp : -> Experiment [ctor].
op repeat_times‘starting‘in_:_ : Nat Time SingleExperiment -> Experiment .

var T : Time .   var E : SingleExperiment .
eq repeat 0 times starting in T : E = none .
eq repeat 1 times starting in T : E = E .
```

A single experiment is modeled as a message `exp((BI for PT) in FT)`, where `FT` denotes in how many time units the experiment is scheduled and `PT` denotes the number of time units the perception of the basic information `BI` persists in the environment. A sequence of experiments is modeled as a message `repeat N times starting in TI : E`, where experiment `E` is repeated for `N` times and the sequence starts in `TI` time units. Two simple rewrite rules generate single experiments from a sequence defined using the `repeat_times‘starting‘in_:_` operator and a perception from a single experiment.

5.1 Rote Learning

In rote learning, maintenance rehearsal is used for achieving the transfer of the information from STM to semantic memory. For example, if the time is given in milliseconds,

```
repeat 10 times starting in 5000 :
          ((a "dog" is a "animal") for 2000) in 3000
```

models an experimental session that starts in 5 s and in which a human subject is presented the sentence ‘a dog is an animal’ 10 times, every 3 s, each time for 2 s. Every rehearsal loop increases the fact lifetime in STM until the threshold for transferring to semantic memory is reached so that a representation of the fact is created in semantic memory.

5.2 Effective Learning

In order to model effective learning, we have to make sure that the fact to be learned is first stored in STM and then used to retrieve information from semantic memory. For example

```
((a "dog" is a "animal") for 2000) in 5000
repeat 10 times starting in 3000 :
          ((can a "dog" "move" ?) for 2000) in 3000
```

models an experimental session that starts in 5 s and in which a human subject is first presented the fact 'a dog is an animal' once and is then presented the question 'can a dog move?' 10 times, every 3 s, each time for 2 s. The retrieval process needed to answer the question starts by using the fact in STM and ends by using the representation of the fact `a "animal" can "move"` in semantic memory. The repeated use of fact `a "dog" is a "animal"` increases its lifetime in STM until the threshold for transferring to semantic memory is reached so that a representation of the fact is created in semantic memory.

6 Conclusion and Future Work

In this paper we presented an approach to the formal modelling of memory processes underlying the transfer of information from STM to LTM, with focus on the consolidation of factual knowledge in semantic memory. We have tested our approach on a simple experimental setting. In our future work we aim at investigating how our approach can cope with more complex experimental settings.

A number of hypotheses and theories from cognitive psychology have been considered as good candidates to be investigated within our approach. Such hypotheses and theories are normally conceptual in nature, with only vague, controversial quantitative characterisations. For example, STM decay time has been traditionally proposed to be 2 s, but some argue that it is much longer, between 4 and 10 s [22]. Other suggest less than 3 s and Campoy proposes an average estimate of 2,700 ms [8]. Although there are no specific theories that describe mezzanine memory, the manipulation of the chunk lifetime supports the emulation of processes that fill the time gap between STM and semantic memory.

In fact, our in silico experiments can be used to compare such alternative hypotheses and theories, or even contribute to the formulation of new theories, as we aim in the case of mezzanine memory, as part of our future work. One way to carry out such a comparison is to determine and test alternative quantitative implementations of conceptual hypotheses or theories, as we proposed for Burgess and Hitch's parallel learning mechanisms [6,7], in Sect. 4.2, and for Naveh-Benjamin and Jonides' hypothesis [23], in Sect. 4.4. Another way is the direct comparison of alternative estimates form cognitive psychology or neuroscience. This is the case for STM decay time and for Mueller and Krawitz's conversion of the ortographical format into the phonological format.

As part of our future work, the results of in silico experiments may also be compared with real datasets to evince which model best mimics reality. In addition to a manual comparison, we aim at the generalisation of an approach from our previous work [14] in which 'formal validation' is achieved by converting a dataset into a formal representation that can be composed in parallel with the system model. In the context of this paper, the system model is actually the human memory model. Model-checking would then be used to verify properties that may only hold when the dataset matches the in silico experiment.

Finally, the human memory model of a user can be combined with the model of the used computer system. Such an overall model can be formally verified

using Real-time Maude model-checking features. This is also part of our future work.

References

1. Atkinson, R.C., Shiffrin, R.M.: Human memory: a proposed system and its control processes. In: Spense, K.W. (ed.) The Psychology of Learning and Motivation: Advances in Research and Theory II, pp. 89–195. Academic Press (1968)
2. Atkinson, R.C., Shiffrin, R.M.: The control of short-term memory. Sci. Am. **225**(2), 82–90 (1971)
3. Baddeley, A.: The episodic buffer: a new component of working memory? Trends Cogn. Sci. **4**(11), 417–423 (2000)
4. Baddeley, A., Papagno, C., Vallar, G.: When long-term learning depends on short-term storage. J. Mem. Lang. **27**(5), 586–595 (1988)
5. Broccia, G., Milazzo, P., Ölveczky, P.C.: Formal modeling and analysis of safety-critical human multitasking. Innovations Syst. Softw. Eng. **15**(3–4), 169–190 (2019). https://doi.org/10.1007/s11334-019-00333-7
6. Burgess, N., Hitch, G.J.: Memory for serial order: a network model of the phonological loop and its timing. Psychol. Rev. **106**(3), 551–581 (1999)
7. Burgess, N., Hitch, G.J.: A revised model of short-term memory and long-term learning of verbal sequences. J. Mem. Lang. **55**(4), 627–652 (2006)
8. Campoy, G.: Evidence for decay in verbal short-term memory: a commentary on Berman, Jonides, and Lewis (2009). J. Exp. Psychol. Learn. Mem. Cogn. **38**(4), 1129–1136 (2012)
9. Cerone, A.: A cognitive framework based on rewriting logic for the analysis of interactive systems. In: De Nicola, R., Kühn, E. (eds.) SEFM 2016. LNCS, vol. 9763, pp. 287–303. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41591-8_20
10. Cerone, A.: Towards a cognitive architecture for the formal analysis of human behaviour and learning. In: Mazzara, M., Ober, I., Salaün, G. (eds.) STAF 2018. LNCS, vol. 11176, pp. 216–232. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04771-9_17
11. Cerone, A.: Behaviour and reasoning description language (BRDL). In: Camara, J., Steffen, M. (eds.) SEFM 2019. LNCS, vol. 12226, pp. 137–153. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57506-9_11
12. Cerone, A., Murzagaliyeva, D.: Information retrieval from semantic memory: BRDL-based knowledge representation and Maude-based computer emulation. In: Cleophas, L., Massink, M. (eds.) SEFM 2020. LNCS, vol. 12524, pp. 159–175. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67220-1_13
13. Cerone, A., Ölveczky, P.C.: Modelling human reasoning in practical behavioural contexts using Real-Time Maude. In: Sekerinski, E., et al. (eds.) FM 2019. LNCS, vol. 12232, pp. 424–442. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-54994-7_32
14. Cerone, A., Zhexenbayeva, A.: Using formal methods to validate research hypotheses: the Duolingo case study. In: Mazzara, M., Ober, I., Salaün, G. (eds.) STAF 2018. LNCS, vol. 11176, pp. 163–170. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04771-9_13
15. Collins, A.M., Quillian, M.R.: Retrieval time from semantic memory. J. Verbal Learn. Verbal Behav. **8**, 240–247 (1969)

16. Craik, F.I., Lockhart, R.S.: Levels of processing: a framework for memory research. J. Verbal Learn. Verbal Behav. **11**(6), 671–684 (1972)
17. Dix, A.: Personal communication (2019)
18. Fiebig, F., Lansner, A.: Memory consolidation from seconds to weeks through autonomous reinstatement dynamics in a three-stage neural network model. In: Liljenström, H. (ed.) Advances in Cognitive Neurodynamics (IV). ACN, pp. 47–53. Springer, Dordrecht (2015). https://doi.org/10.1007/978-94-017-9548-7_7
19. Kesner, R.: Parallel processing of spatial and temporal information in rodents and humans: role of the hippocampus. In: Call, J., Burghardt, G.M., Pepperberg, I.M., Snowdon, C.T., Zentall, T. (eds.) APA Handbooks in Psychology®. APA Handbook of Comparative Psychology: Basic Concepts, Methods, Neural Substrate, and Behavior, pp. 517–538. American Psychological Association (2017)
20. Kolers, A.P.: A pattern-analyzing basis of recognition. In: Cermak, L.S., Craik, F.I. (eds.) Levels of Processing in Human Memory, pp. 363–384. Psychology Press, Hove (2014)
21. Miller, G.A.: The magical number seven, plus or minus two: some limits on our capacity to process information. Psychol. Rev. **63**(2), 81–97 (1956)
22. Mueller, S.T., Krawitz, A.: Reconsidering the two-second decay hypothesis in verbal working memory. J. Math. Psychol. **53**(1), 14–25 (2009)
23. Naveh-Benjamin, M., Jonides, J.: Maintenance rehearsal: a two-component analysis. J. Exp. Psychol. Learn. Mem. Cogn. **10**(3), 369 (1984)
24. Norris, D., Kalm, K., Hall, J.: Chunking and redintegration in verbal short-term memory. J. Exp. Psychol. Learn. Mem. Cogn. **46**(5), 872–893 (2019)
25. Ölveczky, P.C.: Real-Time Maude and its applications. In: Escobar, S. (ed.) WRLA 2014. LNCS, vol. 8663, pp. 42–79. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12904-4_3
26. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time-Maude. Higher-Order Symb. Comput. **20**(1–2), 161–196 (2007)