Information Retrieval from Semantic Memory: BRDL-Based Knowledge Representation and Maude-Based Computer Emulation

Antonio Cerone(✉) and Diana Murzagaliyeva

Department of Computer Science, Nazarbayev University, Nur-Sultan, Kazakhstan
{antonio.cerone,diana.murzagaliyeva}@nu.edu.kz

Abstract. This paper presents a formal model for the representation of relational information in semantic memory and for its retrieval as a reaction to triggering questions which are normally used in experimental psychology. Information is represented using the Behaviour and Reasoning Description Language (BRDL), while the engine for its retrieval is given by the real-time extension of the Maude rewrite language. Maude's capability of specifying complex data structures as many sorted algebras and the time features of Real-Time Maude are essential in providing a means for formalising alternative human memory models. Furthermore, using Maude's object-oriented modelling style, aspects of such alternative memory models may be implemented in separate, interchangeable modules, thus providing a way for their comparison through in silico experiments. Finally, the results of in silico experiments may be contrasted with the data produced through lab experiments and natural observations to yield, on the one hand, a calibration of the emulation engine underlying BRDL and, on the other hand, important insights into alternative theories of cognition.

Keywords: Cognitive science · Behaviour and Reasoning Description Language (BRDL) · Formal methods · Rewriting logic · Real-Time Maude

1 Introduction

Since the end of the 1960 s, experimental psychology has shown that semantic memory has a complex network-like structure and that information is retrieved by 'navigating' such a network while following relationships between the stored representations of concepts. Collins and Quillian's experiments [10] have shown that the time to retrieve information is proportional to how far we need to

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

navigate the network to find the requested information and support a hierarchically organised memory model. Such a hierarchical model allows us to explain human understanding of simple propositions about class membership and property statement as well as the different retrieval times needed to understand a given proposition or to answer a given question.

After Qullian's hierarchical network model [21], further work was carried out in the 1970 s by replacing the hierarchical structure assumption with a general network representing concept relatedness [9], for dealing with more formal propositions involving universal quantification [11,17] and by focusing on the role of connotative relationships between conceptual components of propositions [22]. The last approach resulted in set-theoretical models, such as the one developed by Smith *et al.* [23]. Although some researchers argued that these two different classes of models, network-based and set-theoretical, are formally isomorphic [13], experiments conducted in the 1980 s on the semantic distance in the retrieval of conceptual relationships showed that both classes of models may be fallacious in some contexts that discriminate between them [3].

The strong emphasis on models for the representation of information in semantic memories continued throughout the 1990 s and 2000 s with little success in producing convincing computational models of the retrieval of the represented relations. Even more recent distributed models such as LISA [12], cognitive architectures such as ATC-R [1] and connectionist models [15] provide only limited retrieval mechanism, and mostly as part of inferential or analogical reasoning engines. Holyoak uses the term 'retrieval gap' to denote this limitation of the current computational models and observes that there is no generally accepted model yet [14].

The purpose of our work is the development of an approach in which different models of semantic memory can be formally described, executed to perform in silico experiments and formally analysed, with the objective of comparing different alternative models of semantic memory. In previous work, we have developed modelling languages for this purpose. The Human Behaviour Description Language (HBDL) [5,6] aims at the modelling of automatic and deliberate human behaviour while interacting with an environment consisting of heterogenous physical components. The Behaviour and Reasoning Description Language (BRDL) [7] originates from and extends HBDL with the linguistic constructs to specify reasoning goals, inference rules and unsolved problems.

Recently we have developed a cognitive engine using the Maude rewrite system [16,19] and its real-time extension, Real-Time Maude [18,20], to execute HBDL models of human behaviour [5,6] and BRDL models that emulate human reasoning [8]. All these implementations are based on direct access to the information in semantic memory. Human behaviour is modelled in terms of basic activities whose representation is stored in semantic memory and which are directly triggered by the content of short-term memory (STM) and the perceptions available in the environment. Human reasoning is modelled in terms of inference rules whose representation is stored in semantic memory and which are directly triggered by the presence of its premises in STM.

A similar approach to ours was developed by Broccia *et al.* [4], who, driven by the specific objective of modelling human multitasking, used Real-time Maude to extend our initial untimed framework [5]. 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 consider the subset of BRDL that models, on the one hand, propositions that express facts of the real world and questions related to such facts and, on the other hand, the representation of such facts in semantic memory. We develop a modular implementation of such a subset of BRDL using Real-Time Maude, thus providing an emulation of the process of information retrieval from semantic memory to be used to carry out in silico experiments.

The rest of this paper is organised as follows. Section 1.1 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. Section 2 introduces the BRDL syntax for facts and questions and shows how they are modelled in Real-Time Maude. Section 3 presents the implementation of a variant of Quillian's model, which is used to illustrate our approach. Section 4 illustrates how to plan and carry out experiments. Section 5 discusses the timed evolution of the system and the information retrieval process implementation. Section 6 concludes the paper.

1.1 Real-Time Maude

Real-Time Maude [18,20] 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.

The definition of data types is illustrated in Sect. 2.3. The Full Maude syntax is illustrated in Sect. 3.3 for classes and in Sect. 4.1 for messages. Labelled rewrite rules and tick rewrite rules are illustrated in Sect. 4.3 and Sect. 5, respectively.

2 Natural Language Constructs: Facts and Questions

BRDL has a concise, functional-like syntax, which is presented elsewhere [7]. Its conciseness is thought to provide an essential description of the model able to present the bigger picture of the modelled system, and its functional flavour makes it suitable for direct mathematical manipulation without computer support. In this paper we mostly use an alternative, somehow verbose syntax, which is more similar to the English natural language, although ungrammatical in some details, and can be directly manipulated by the Maude system.

2.1 Facts

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.

A *fact* is modelled using the functional-like BRDL syntax [7] as

$$type(category, attribute)$$

where *category* is an object of our knowledge and *attribute* is an attribute of type *type* associated with that category. For example, the fact that ‘a dog is an animal’ is represented using the functional-like BRDL syntax as

$$is\_a(dog, animal).$$

The equivalent English-like syntax manipulated by the Maude system is

```
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 "dog" has "four legs"
a "animal" can "breathe"
a "dog" can "move"
a "dog" can "bark"
a "cat" cannot "bark"
a "cat" is not a "dog"
a "bird" has not "four legs".
```

In such examples `"animal"`, `"dog"`, `"cat"` and `"bird"` are categories, `"four legs"`, `"breathe"`, `"move"`, `"bark"` are attributes and `is a`, `has`, `can`, `is not a`, `has not` 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 `has "four legs"` is called *typed attribute*. The reason why categories and attribute are between double apices will be explained in Sect. 2.3.

Finally, we also consider declarative propositions that describe the absence of knowledge about facts, such as

```
I do not know if a "animal" "bark"
```

We could also model declarative propositions that describe our knowledge about facts, but this would not be more interesting that the fact itself. Actually, if I state that `a "dog" has "four legs"`, it is obvious that `I know that a "dog" has "four legs"`.

2.2 Questions

Questions can be of different kinds. Examples are:

```
what can a "animal" do?
can a "dog" "breathe" ?
is a "dog" a "animal" ?
has a "cat" "four legs"
```

We can note here that the first question may be answered by a set of declarative propositions, whereas the other three questions, which all have the same structure, will be answered by one declarative proposition, either the one stating the fact, negatively or positively, or the one describing the absence of knowledge about the fact. For example, `can a "dog" "breathe" ?` will be answered by one of the following three declarative propositions: `a "dog" can "breathe"`, `a "dog" cannot "breathe"` or `I do not know if a "dog" can "breathe"`.

2.3 Modelling Facts and Question in Real-Time Maude

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.

The English-like syntax of facts and questions is defined in the Real-Time Maude module `INFORMATION` given in Figure 1. The module imports the predefined modules `NAT` and `STRING`, which support the specification of natural numbers and strings, respectively. Sorts `Fact` and `Question` model facts and questions, respectively, and are subsorts of `BasicItem`, which is in turn subsort of `Item`. Only the `BasicItem` subsort of `Item` is relevant for the subset of the

```
(tomod INFORMATION is
  protecting NAT .
  protecting STRING .

  sorts Fact Question BasicItem Item BasicItemSet ItemSet EmptyItemSet .
  subsorts Fact Question < BasicItem < Item .
  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] .

  sorts Category Attribute TypedAttribute .
  subsort String < Category < Attribute .
  ops can_ is‘a_ has_ : Attribute -> TypedAttribute [ctor] .
  ops cannot_ is‘not‘a_ has‘not_ : Attribute -> TypedAttribute [ctor] .
  op what‘can‘a_do? : Category -> Question [ctor] .
  ops can‘a__? is‘a_a_? has‘a__? : Category Attribute -> Question [ctor] .
  op a__ : Category TypedAttribute -> Fact [ctor] .
  ops I‘dont‘know‘if‘a_can_
      I‘dont‘know‘if‘a_is_ : Category Attribute -> Fact [ctor] .
  op I‘dont‘know‘what‘a_can‘do : Category -> Fact [ctor] .
  op _is‘negative‘of_ : TypedAttribute TypedAttribute -> Bool .

  var A : Attribute .   vars TA1 TA2 : TypedAttribute .
  eq (cannot A) is negative of (can A) = true .
  eq (is not a A) is negative of (is a A) = true .
  eq (has not A) is negative of (has A) = true .
  eq TA1 is negative of TA2 = false [owise] .

  op isItemIn : Item ItemSet -> Bool .
  vars I1 I2 : Item .   var IS : ItemSet .
  eq isItemIn(I1, I2 ; IS) = if I1 == I2 then true
                                       else isItemIn(I1, IS) fi .
  eq isItemIn(I1, none) = false .
endtom)
```

Fig. 1. Module `INFORMATION`.

BRDL implementation considered in this paper. Other subsorts, which are not introduced here, are used in other parts of the BRDL implementation [6,7].

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 defined 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. 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 `TypedAttribute`, `Fact` and `Question` 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.

One of these special relationships is the negation: `cannot`, `is not` and `has not` are the negations of `can`, `is` and `has`, respectively. Negation is expressed as an infix boolean operator `is negative of` characterised by three axioms for the three pairs of attribute types above which return `true`. The last axiom has the `owise` equational attribute, thus it is applied to all other cases and returns `false`.

Finally, the boolean operator `isItemIn` returns `true` if an element of sort `Item` belongs to an element of sort `ItemSet`.

3 Human Memory Model

BRDL is based on Atkinson and Shiffrin's *multistore model* of human memory [2]. This model is characterised by three stores between which various forms of information flow: *sensory memory*, where information perceived through the senses persists for a very short time, *short-term memory (STM)*, which has a limited capacity and where the information that is needed for processing activities is temporarily stored with rapid access and rapid decay, and *long-term memory (LTM)*, which has a virtually unlimited capacity and where information is organised in structured ways, with slow access but little or no decay. We consider a further decomposition of LTM: *semantic memory*, which refers to our *knowledge* of the world and consists of the *facts* that can be *consciously* recalled, and *procedural memory*, which refers to our *skills* and consists of *rules* and *procedures* that we *unconsciously* use to carry out tasks, particularly at the motor level.

This paper focuses on STM and on the part of semantic memory devoted to the storage of *fact representations*, that is, the representations of our knowledge of the facts of the real world in the form of a *hierarchical network*, often called a *semantic network*, as first introduced by Collins and Quillian [10,21]. In Sect. 2.1 we illustrated how to model facts in a natural-language-like fashion.

In Sect. 3.1 we present how to model fact representations and their hierarchical organisation in semantic memory. The hierarchy among categories is expressed using the `is a` type applied to the more general category. For example, the type attribute `is a "animal"` denotes a *generalisation* to category `"animal"`. The more specific category inherits all attributes of the more generic category unless the attribute is redefined at the more specific category level.

3.1 Fact Representation in Semantic Memory

A fact representation in semantic memory is modelled using the functional-like BRDL [7] syntax as

$$domain : category \mid \xrightarrow{delay} \mid type(attribute)$$

where *delay* is the mental processing time needed to retrieve the association between category *category* and type attribute $type(attribute)$ within the given knowledge domain *domain*. The knowledge domain is used to set a boundary under which the mental processing is retrieving information from the semantic memory and manipulating it. The role of such a boundary is clarified in Sect. 5.

As an example, the fact that ‘a dog is an animal’ is represented within the semantic domain *dogs* as

$$dogs : dog \mid \xrightarrow{1} \mid is\_a(animal)$$

and this generalisation can be retrieved from semantic memory in 1 time unit. The more specific category of a generalisation inherits all typed attributes of the more generic category unless the attribute is redefined at the more specific category level. Therefore,

$$animals : animal \mid \xrightarrow{1} \mid can(move)$$

which is an association of a category with a typed attribute rather than a generalisation, 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*.

The equivalent English-like syntax manipulated by the Maude system of the fact representations above is

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

Such an English-like syntax of fact representations is defined in the Real-Time Maude module `SEMANTIC-MEMORY` shown in Figure 2. This module imports the module `INFORMATION` and the predefined module `NAT-TIME-DOMAIN`, which defines the sort `Time` to model discrete time. Fact representations are defined as element of the sort `FactRepresentetion`. The semantic memory is modelled by the sort `SemanticMemory`, which is defined as a set of fact representations.

```
(tomod SEMANTIC-MEMORY is
  protecting NAT-TIME-DOMAIN .
  including INFORMATION .

  sorts Domain FactRepresentation SemanticMemory .
  subsort String < Domain .
  subsort FactRepresentation < SemanticMemory .
  op emptySemantic : -> SemanticMemory [ctor] .
  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 .

  var M : Time .   var D : Domain .
  var C : Category .   var A : Attribute .
  vars TA1 TA2 : TypedAttribute .   var S : SemanticMemory .
  eq ( a C TA1 ) is negated in ( ( D : C |- M ->| TA2 ) S ) =
                                    TA2 is negative of TA1 .
  eq ( a C TA1 ) is negated in S = false [owise] .
endtom)
```

Fig. 2. Module `SEMANTIC-MEMORY`.

The constructor `__` denotes that sets of fact representations are created by justaposition, with no written operator. The constructor `emptySemantic` denotes an empty semantic memory.

Given the `"dogs" : "dog" |- 1 ->| can "bark"` fact representation, let us consider the following downward extension of the animal–dog hierarchy:

```
"dogs" : "hund" |- 1 ->| is a "dog"
"dogs" : "basenji" |- 1 ->| is a "hund"
"dogs" : "basenji" |- 1 ->| cannot "bark"
```

The category `"basenji"` has the `cannot "bark"` typed attribute, which redefines the `can "bark"` typed attribute of the `"dog"` category. In fact, a basenji is an exceptional dog breed that cannot bark.

In order to make sure that the category `"basenji"` does not inherit the `can "bark"` typed attribute from the more general `"dog"` category, we introduce the `is negated in` user-defined infix operator, which returns `true` only if the fact passed as the left argument is negated by some fact representation in the semantic memory passed as the right argument. As shown in Sect. 5, this operator is used by the Maude engine to prevent the retrieval of general information that is negated for the more specific category we are considering. In this way, given the fact representations above, `(a "basenji" can "bark") is negated in ("dog" : "basenji" |- 1 ->| cannot "bark")` equals `true`, thus the question `can a "basenji" "bark" ?` does not trigger the retrieval of the typed attribute `can "bark"`.

```
(tomod TIMED-INFORMATION is
   protecting INFORMATION .
   protecting NAT-TIME-DOMAIN-WITH-INF .

   sorts TimedItem TimedItemSet TimedBasicItem FutureBasicItem .
   subsort TimedItem < TimedItemSet .

   op _<‘decay_> : Item TimeInf -> TimedItem [ctor] .
   op _for_ : BasicItem TimeInf -> TimedBasicItem [ctor] .
   op _in_ : TimedBasicItem Time -> FutureBasicItem [ctor] .
   op emptyTIS : -> TimedItemSet [ctor] .
   op _;_ : TimedItemSet TimedItemSet -> TimedItemSet
                [ctor assoc comm id: emptyTIS format (b o n b)] .
   eq ITEM:Item < decay 0 > = emptyTIS .

   op removeTime : TimedItemSet -> ItemSet .
   var I : Item .
   var TIS : TimedItemSet .
   var T : TimeInf .
   eq removeTime(emptyTIS) = none .
   eq removeTime((I < decay T >) ; TIS) =  I ; removeTime(TIS) .

   ops DECAY-TIME MAX-RETRIEVAL-TIME : -> TimeInf .
endtom)
```

Fig. 3. Module TIMED-INFORMATION.

3.2 Short-Term Memory (STM) Model

STM is normally used as a buffer where the information that is needed for processing activities is temporarily stored. In our model the kind of information stored in the STM belongs to sort Item. However, the limited capacity of STM requires the presence of a mechanism to empty it when the stored information is no longer needed. In fact, information in STM decays very quickly, normally in less than one minute. To implement STM decay, we need to associate a time with the elements of sort Item.

Module TIMED-INFORMATION in Figure 3 imports the module INFORMATION and the predefined module NAT-TIME-DOMAIN-WITH-INF, which defines a new sort TimeInf by extending the sort Time with value INF to model an infinite time. Sort TimedItem associates a time with the elements of sort Item using the constructor _<‘decay_> and sort TimedItemSet define sets of elements of TimedItem using the constructor ;. The equation on the constructor _<‘decay_> ensures that if the time to decay has reached zero, the term is removed. Therefore, STM is modelled as an element of sort TimedItem. The operator removeTime removes the time from an element of sort TimedItem and returns the corresponding element of sort TimedItem. Finally, undefined operation DECAY-TIME is declared to be used as a constant holding the user-defined STM decay time as we will see in Sect. 4.3.

3.3 Human Memory as a Maude Class

Full Maude supports the definition of classes. Class objects are elements of the pre-defined sort `Configuration`, which defines the state of the overall system.

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

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

STM is modelled by the field `shortTermMem`. Semantic memory is modelled by the field `SemanticMem`.

4 Experimental Environment and Its Evolution

We model an environment consisting of perceptions in the form of questions to which a human subject has to answer. Once they appear in the environment, questions persist for a certain time before disappearing. In a typical experiment, questions will be shown to the human subject for a few seconds.

4.1 Modelling Perceptions

We use Full Maude *messages* to model perceptions. 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 persistence of a perception is modelled by the user-defined infix operator `for` in module `TIMED-INFORMATION` given in Fig. 3. Note that the time may be infinite to denote that the perception persists forever.

Therefore, a perception is modelled by operation `perc` as follows.

```
sorts Perception .
subsort Perception < Msg .
op perc : TimedBasicItem -> Perception [ctor] .
var BI : BasicItem .
eq perc(BI for 0) = none .
```

The equation ensures that if the persistence time has reached zero, the perception is removed from the configuration (`none` is the empty configuration).

4.2 Planning Experiments

We call *planned experiment* any single question presented to a human subject together with the time that must pass before the question is actually presented. Such a time is modelled by the user-defined infix operator `in` in module `TIMED-INFORMATION` given in Fig. 3.

Therefore, a planned experiment is modelled by operation `exp` as follows.

```
sorts Experiment .
subsorts Experiment < Msg .
op exp : FutureBasicItem -> Experiment [ctor] .
```

For example, we can plan an experiment similar to the one by Collins and Quillian [10] to find how *response time* (RT) may depend on the hierarchical structure of semantic memory. The subjects are presented with questions on a screen, each question for 30 s. Within these 30 s the subject has to answer the question. The experimental setting is as follows.

```
op init : -> Configuration .
op human : -> Oid .
op initSemanticMem : -> SemanticMemory .
op aHuman : -> Object .
eq aHuman = < human : Human | shortTermMem : emptyTIS,
                              semanticMem : initSemanticMem > .

eq init = (exp(((can a "dog" "breathe" ?) for 30000) in 0))
          (exp(((can a "animal" "move" ?) for 30000) in 30000))
          (exp(((has a "dog" "four legs" ?) for 30000) in 60000))
          (exp(((can a "hound" "track" ?) for 30000) in 90000))
          (exp(((can a "basenji" "bark" ?) for 30000) in 120000))
          (exp(((is a "armadillo" a "mammal" ?) for 30000) in 150000))
          (exp(((can a "giraffe" "bark" ?) for 30000) in 180000))
          (exp(((is a "swallow" a "bird" ?) for 30000) in 210000))
          aHuman .
```

The time is expressed in milliseconds.

The operator `initSemanticMem` must be defined with an equation whose right part comprises the content of the semantic memory. The purpose of the experiment above could be the validation of the following research hypotheses:

1. Higher RTs may be due to the fact that the typed attribute is associated with a more general category than the one mentioned in the question;
2. In some cases, the RT for a negative answer may be considerably smaller than the average RT for a positive answer to a similar question;
3. The retrieval of less known or seldom used categories or associations may result in a higher RT.

Hypothesis 1 can be validated using questions like `can a "dog" "breathe" ?` and `can a "animal" "move" ?`: the RT for the former question is higher if the typed attribute `can "breathe"` is associated with the more general category `"animal"`. Hypothesis 2 can be validated using questions like `can a "basenji" "bark" ?` and `can a "giraffe" "bark" ?`: the RT for the former question is lower if the typed attribute `cannot "bark"` is directly associated with category `"basenji"`. Hypothesis 3 can be validated using questions like `is a "armadillo" a "mammal" ?` and `is a "swallow" a "bird" ?`: the RT for the former question is higher if the fact representation stating that the less known category `"armadillo"` is a `"mammal"` has a higher mental processing time than the fact representation stating that the well-known category `"swallow"` is a `"bird"`.

Initially, in silico experiments can be conducted with no aim at validating research hypotheses and contrasted to experiments with human subjects to calibrate the Maude emulation engine. Once the calibration process is concluded, in silico experiments and experiments with human subjects aiming at the validation of research hypothesis can be conducted and contrasted. These kinds of experiment may be used to compare alternative models corresponding to different models of semantic memory.

As a final remark, we would like to clarify that the experiment above has been defined for a purely illustrative purpose. In a real experimental context, a separate set of planned experiments should be considered for each of the research questions.

4.3 Environment Evolution

Maude models system evolution using rewrite logic. Labeled rewrite rules

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

define local transitions from state t to state t'.

The following labelled rewrite rule transforms a planned experiments into a perception when the scheduled time has been reached.

```
rl [activate-perception] :
  (exp((BI for T) in 0))
   REST
=>
  (perc (BI for T))
   REST .
```

The following labelled rewrite rule make a ‘copy’ of the perception and stores it in the STM at any time when the perception is available in the environment and associates the value of the constant operator `DECAY-TIME` with such a copy, provided that the untimed version of the STM (`IS := removeTime(TIS)`) does not contain the perception yet (`not isItemIn(BI, IS)`).

```
crl [perceive] :
  (perc(BI for T))
   < H : Human | shortTermMem : TIS >
   REST
=>
  < H : Human | shortTermMem : (BI < decay DECAY-TIME >) ; TIS >
   (perc(BI for T))
   REST
 if IS := removeTime(TIS) /\ not isItemIn(BI, IS) .
```

5 Tick Rewrite Rules for Information Retrieval

Information retrieval from semantic memory and time passing are modelled using tick rewrite rules. Tick rewrite rules

$$\texttt{rl [}l\texttt{]: \{}t\texttt{\} => \{}t'\texttt{\} in time } \varDelta$$
$$\texttt{crl [}l\texttt{]: \{}t\texttt{\} => \{}t'\texttt{\} in time } \varDelta \texttt{ if } \mathit{cond}$$

advance time in the *global* state t by $\varDelta$ time units.

We use tick rewrite rules to model the navigation through semantic memory to retrieve the information needed to answer the question in STM. The RT is added to the current time and the retrieved information is used to build the answer and store it in STM together with the associated decay time DECAY-TIME.

The following idle function is used to update all time-related components of the system:

```
op idle : Configuration Time -> Configuration [frozen (1)] .
op idle : TimedItemSet Time -> TimedItemSet .
op idle : TimedBasicItem Time -> TimedBasicItem .
op idle : FutureBasicItem Time -> FutureBasicItem .

eq idle(none, T) = none .
eq idle(< H : Human | >  REST, T) = < H : Human | >  idle(REST, T) .
eq idle(perc(TBI) REST, T) = perc(idle(TBI, T))  idle(REST, T) .
eq idle(exp(FBI) REST, T) = exp(idle(FBI, T))  idle(REST, T) .

eq idle(emptyTIS, T) = emptyTIS .
eq idle((I < decay T1 >) ; TIS , T) =
                        (I < decay (T1 monus T) >) ; idle(TIS, T) .
eq idle(BI for T1, T) = BI for (T1 monus T) .
eq idle(TBI in FT , T) = TBI in (FT monus T) .
```

The module TIMED-EVOLUTION contains the tick rewrite rules that implement a variant of Qullian's hierarchical network model [21]. As an illustrative example, we present the conditional tick rule for answering a can-question, such as can a "dog" "breathe" ?.

```
crl [can] :
  < H : Human | cognitiveLoad : N,
     shortTermMem : TIS ; ((can a C A ?) < decay T1 >),
     semanticMem : S >
   REST
=>
  < H : Human | cognitiveLoad : N,
     shortTermMem : ((a C can A) < decay DECAY-TIME >) ; idle(TIS, T),
     semanticMem : S >
  idle(REST,T)
in time T
if IS := removeTime(TIS) /\
   not isItemIn(a C can A, IS) /\
   T := canRetrievalTime(C, A, S) /\ T <= MAX-RETRIEVAL-TIME /\
   not ( ( a C can A ) is negated in S ) .
```

The retrieval time is calculated using the canRetrievalTime operator, which searches in the semantic memory S for a fact representation with category C1 and typed attribute can A, where either C1 = C or C1 is a generalisation of C.

However, an additional condition for the application of the rule is that the RT `T` is less than or equal to the `MAX-RETRIEVAL-TIME` constant value. When the RT is greater than the `MAX-RETRIEVAL-TIME`, instead, the application of another conditional tick rule with complementary condition results in the answer `I dont know if C can A`. The use of such a constant as an RT upper bound supports the modelling of situations in which, although the information is in semantic memory, the semantic distance is actually too high for retrieval. Two more rules model the two possible situations in which the answer is `C cannot A`. The most explicit situation is when there is a fact representation with category `C` and typed attribute `cannot A` in the semantic memory. An implicit situation occurs when `C` is the generalisation of another category `C1`, such that the fact representation with category `C1` and typed attribute `can A` is in the semantic memory. As an example of such an implicit situation, the question `can a "animal" "bark" ?` results in the fact `a "animal" cannot "bark" ?`.

Further rules are used for covering the situations in which we answer `is a` and `has a` questions. The `what can` question results in the retrieval of a number of facts. This is achieved by repeated applications of the rule. Moreover, the answer to this question depends on the knowledge domain on which the human subject focuses. For example, the question `what can a "dog" do?` returns only `a "dog" can "bark"` if the focus is on the knowledge domain `dogs`, whereas it returns also `a "dog" can "breathe"` and `a "dog" can "move"` if the focus is on the knowledge domain `animals`. This requires the use of two distinct rules, one for the presence and one for the absence of focus. However, the technical details concerning these rules are beyond the illustrative purpose of this paper.

Finally, the following conditional tick rule models the time passing.

```
crl [time-passing] :
    < H : Human | shortTermMem : TIS >
     REST
  =>
    < H : Human | shortTermMem : idle(TIS,1) >
     idle(REST,1)
  in time 1
  if not questionFound(TIS) /\ noExperimentStart(REST)
                            /\ noPerceptionAvailable(REST) .
```

The condition ensure that time is increased by this rule only when there is no question in STM (`not questionFound(TIS)`), there is no planned experiment at the current time (`noExperimentStart(REST)`) and there is no perception available in the environment (`noPerceptionAvailable(REST)`). In this way the `time-passing` rule may be applied only if no other rule can be applied.

6 Conclusion and Future Work

We have developed an approach for formally modelling fact representations in semantic memory and carrying out the in silico emulation of experiments aiming at the comparison of different models of semantic memory. This comparison is an essential part of our future work. The code illustrated in this paper can be downloaded at

http://github.com/AntonioCerone/Pubblications/tree/master/2020/CIFMA

Although the code refers to the Quillian's hierarchical network model [21] and Collins and Quillian's experiments [10], the module `TIMED-EVOLUTION` can be replaced with a module that implements another semantic memory model.

This code is also simplified for illustrative purposes. It does not include the mechanisms that allow to output the outcome of the in silico experiments in a form suitable to carry out the comparison of alternative semantic memory models.

Acknowledgments. We would like to thank Graham Pluck for reading the manuscript and providing important comments. The first author would also like to thank Peter Csaba Ölveczky for his kind hospitality in Oslo in May 2019 and for many insightful discussions about Maude, which strongly influenced the way Maude was used in this paper.

References

1. Anderson, J.R., Lebiere, C.J.: The Atomic Components of Thought. Lawrence Erlbaum, Hillsdale (1998)
2. 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)
3. Berry, C., Grove, C.: Semantic distance in memory structure: the retrieval of conceptual relationships. Q. J. Exp. Psychol. **35A**, 553–570 (1983)
4. 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)
5. 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
6. 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
7. 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
8. Cerone, A., Ölveczky, P.C.: Modelling human reasoning in practical behavioural contexts using real-time maude. In: Sekerinski, E. (ed.) FM 2019. LNCS, vol. 12232, pp. 424–442. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-54994-7_32
9. Collins, A.M., Loftus, E.F.: A spreading-activation theory of semantic processing. Psychol. Rev. **82**, 407–428 (1975)
10. Collins, A.M., Quillian, M.R.: Retrieval time from semantic memory. J. Verbal Learn. Verbal Behav. **8**, 240–247 (1969)

11. Glass, A.L., Holyoak, K.T.: Alternative onceptions of semantic memory. J. Verbal Learn. Verbal Behav. **5**, 598–606 (1975)
12. Hammel, J.E., Hollok, K.J.: Distributed representations of structures: a theory of analogical access and mapping. Psychol. Rev. **104**, 427–466 (1997)
13. Hollan, J.D.: Feature and semantic model: seth theoretic or network model? Psychol. Rev. **82**, 154–155 (1975)
14. Holyoak, K.T.: Analogy and relational reasoning. In: Holyoak, K.T., Morrison, R.G., (eds.) The Oxford handbook of thinking and reasoning, pp. 234–259. Oxford University Press (2012)
15. Leech, R., Mareschal, D., Cooper, R.P.: Analogy as relational priming: a developmental and computational perspective on the origin of a complex cognitive skill. Behav. Brain Sci. **31**, 357–378 (2008)
16. Martí-Oliet, N., Meseguer, J.: Rewriting logic: roadmap and bibliography. Theoret. Comput. Sci. **285**(2), 121–154 (2002)
17. Meyer, D.E.: On the representation and retrieval of stored sematic information. Cogn. Psychol. **1**, 242–300 (1970)
18. 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
19. Ölveczky, P.C.: Designing Reliable Distributed Systems. Undergraduate Topics in Computer Science. Springer (2017)
20. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of real-time-Maude. Higher-order symbolic Comput. **20**(1–2), 161–196 (2007)
21. Quillian, M.R.: The teachable language comprehender: a simulation program and theory of language. Commun. ACM **12**, 459–476 (1969)
22. Rips, L.J., Shoben, E.J., Smith, E.E.: Semantic distance and the verification of semantic relations. J. Verbal Learn. Verbal Behav. **12**, 1–20 (1973)
23. Smith, E.E., Shoben, E.J., Rips, L.J.: Comparison processes in semantic memory. Psychol. Rev. **81**, 214–241 (1974)