Using Maude to Model Motivation in Human Behaviour*

Antonio Cerone[0000−0003−2691−5279] and Olzhas Zhalgendinov

Department of Computer Science, School of Engineering and Digital Sciences,
Nazarbayev University, Astana, Kazakhstan
antonio.cerone@nu.edu.kz olzhas.zhalgendinov@nu.edu.kz

Abstract. Human beings act and think driven by motivation, which can be physiological as well as psychological. Although there is no unified theory of motivation, there are a number of theories in psychology that define conceptual models to explain distinct kinds and aspects of motivation. Such a conceptual fragmentation of the notion of motivation makes it very challenging the attempt to build a formal framework to model motivation. In this paper we address the needs underlying motivation, focusing in particular on physiological needs, such as the ones for food, water and sleep, and on the lowest level of psychological needs, the needs for safety and security. We use BRDL (Behaviour and Reasoning Description Language) to model human behaviour and thinking as well as its psychological motivation and LTS s (Labelled Transition Systems) to model physiological motivation. Finally, we illustrate our translation of this formal framework into the Maude rewrite language, which supports the simulation and analysis of the modelled cognitive systems.

Keywords: Behaviour and Reasoning Description Language (BRDL) · Labelled Transition Systems (LTSs) · Cognitive Modelling · Rewriting Logic · Maude · Theories of Motivation

1 Introduction

Motivation is what causes us to act. It may be a rational reason, an intense desire, an emotional impulse, or a physiological need. Physiological needs, such as the ones for food, water and sleep, keep individuals alive. Desires and impulses are important aspects of the life of each individual and, in an evolutionist sense, they aim at the preservation of the species through social development, which foster collaboration and mutual support, and by intertwining with emotional states, such as love, which is essential for reproduction. Looking for rational reasons to act is a distinctive aspect of the human species and drives cultural and technological development.

This multifaceted nature of motivation makes it difficult to build a general theory of motivation. The first attempt, the *instinct theory* [13], aimed at identifying all possible instincts either physical (e.g., locomotion) or mental (e.g.,

* Work funded by the School of Engineering and Digital Sciences (SEDS), Nazarbayev University, Astana, Kazakhstan.

curiosity). However, the number of proposed instincts became soon countless, thus making the theory too complicated. The focus moved then to the physiology of motivation, with the *drive theory* [11, 12, 18] emphasising the compelling urge (drive) to act in order to reduce physiological needs. But reducing needs is not enough to explain motivation, which made drive theory fall out of favour.

Explaining how to maintain the right balance between deficit and surplus is the objective of *homeostatic-regulation theory* [3]. This theory explains motivation as the tendency of the body to maintain a state of equilibrium (e.g., hunger is balanced by eating). Further theories try to address other aspect of motivation. *Opponent-process theory* [17] links motivation to emotion by explaining the acquisition of motivation as the result of a pattern of emotional experience (e.g., the motivation to use psychoactive drugs). According to *arousal theory* [2, 19], the activity of the central nervous system determines the appropriate level of arousal for a given task in relation to the individual's personality (e.g., in general a low level of arousal would help in a complex task to prevent anxiety, but this is not the case for anxious personalities).

In this paper we aim at building executable models for human behaviour, which incorporate motivational aspects. We use a high-level notation, the Behaviour and Reasoning Description Language (BRDL) that allows psychologists and cognitive scientists to model and analyse human cognition in terms of their required attentional, reasoning and action components. With respect to our previous work on BRDL [6] and its use for modelling motivation generated by physiological needs [7], in this paper we consider the first two levels of Maslow's hierarchy of motivation [14, 15], which include not only *physiological needs* but also *psychological* needs, such as safety and security.

BRDL has been implemented using the Maude rewrite language and toolset [16], thus providing a framework for the in silico simulation of human reasoning [9], some aspects of human learning [8, 10] and the interaction with heterogenous physical components [4, 5]. However, motivation was not considered in such implementation. In this paper we extend the BRDL implementation by addressing the physiological aspects of motivation defined in our previous work [7] as well as psychological needs. Additionally, we also implement the use of variables in the BRDL framework, a feature needed to describe quantitative aspects of motivation, but which also greatly increases the framework expressiveness.

The rest of the paper is organised as follows. Sect. 2 defines our conceptual model for motivation and Sect. 2.1 illustrates it on the example of a user of a vending machine, which will follow us throughout the paper. Sect. 3 introduces BRDL syntax and Sect. 3.1 shows how to model motivation at a cognitive level. Sect. 4 revisits the environmental and physiological models introduced in our previous work [7], with reference to our current example. Sect. 5, after a short overview of Maude (Sect. 5.1), describes the implementation in Maude, including the translation of the BRDL syntax (Sect. 5.2), the infrastructure for modelling the variables (Sect. 5.3) and the rewite rules that define the overall system behaviour (Sect. 5.4). Finally, Sect. 6 shows the results of analysing our example using Maude, draws conclusions and discusses possible future work.

2 A Conceptual Model of Motivation

According to the *hierarchical theory of motivation* proposed by Abraham Maslow [14, 15] human needs can be organised into the following hierarchy:

1. *Physiological needs* are at the lowest level and are the basic, essential needs that allow individuals to live, such as the needs for food, water, sleep, which have as *physiological motivators* hunger, thirst, tiredness, respectively.
2. *Safety and security needs* aim at building and maintaining the appropriate living environment in which individuals are protected from environmental danger (safety) and social threats (security).
3. *Belongness and loving needs* aim at being accepted and cared by the society, that is, by the other individuals.
4. *Esteem needs* aim at feeling worthwhile, both in terms of self-appreciation and by comparing themselves to the other individuals.
5. *Self-actualisation needs* aim at fulfilling the human potential for purely hedonistic purposes, independently of external influences.

Level 1 is driven by physiological motivators, such as hunger, thirst and tiredness, while the higher levels are mostly driven by psychological motivators. Only when we have satisfied a specific level of needs, we move to the higher level. Thus, according to Maslow, we consider our safety and security only after having satisfied our physiological needs, such as food, water and sleep. In this paper we develop a model that covers the first two levels of Maslow's hierarchy.

In modelling physiological motivation, we adopt the *homeostatic-regulation theory* [3] which explains motivation as the tendency of the body to maintain a state of equilibrium (e.g., hunger is balanced by eating). However, this theory cannot explain higher-level needs, which are acquired through experience or exposure to a specific cultural environment. In this paper, we assume these needs already acquired.

2.1 Conceptual Model Example: the User of a Vending Machine

Let us consider an example that will follow us throughout the paper. Imagine being a user of a food vending machine in your office. You normally purchase a product from the machine every morning and you consume it later, when you become hungry. Thus, when you purchase the product, you are not driven by hunger, but by the will to be able to safely choose the product you prefer before it runs out and to avoid the lunchtime queue. Therefore, we can say that you have a level-2 need when you purchase and a level-1 need when you eat.

Suppose that one day you are trapped in an important meeting the whole morning and when you get out you are very hungry. You immediately run to the vending machine to purchase some food. This time you do not have a level-2 need when you purchase, but a level-1 need.

The vending machine is operated using a rechargeable card and a pin code, similarly to an automatic teller machine (ATM). You insert the card first and

then you enter the pin code. If you have inserted the correct pin code, the card is returned first and, after you have collected it, the product is delivered. However, if you enter a wrong pin code, you get a warning and the option to either reenter the pin or abort the task. In this case, reentering a wrong pin within one hour will cause the card to be confiscated. If you abort, the card is returned and the product is not delivered. If you enter a wrong pin while you are not hungry, since you do not have a level-1 need, you can satisfy your level-2 needs and delay the purchase one hour in order to be safe from immediate confiscation, in case you entered a wrong pin again. But if you enter a wrong pin while you are hungry, then you are driven by a level-1 need and do not consider any level-2 needs. Thus, in this case, you reattempt the purchase with the risk of an immediate confiscation.

3 Formal Cognitive Model

BRDL models the content of *long-term memory (LTM)* in terms of *cognitive rules* (also called *LTM rules*) that either drive selective attention or represent *factual knowledge* or *procedural knowledge*. Cognitive rules drive the processing of information that has been transferred to *short-term memory (STM)* and may consist of facts retrieved from LTM, perceptions from the environment, action to be carried out on the environment and goals defining what you want to achieve. Thus STM acts as temporary store and is often called *working memory* (WM) when it is considered together with all its information processing functionalities.

Each cognitive rule has a general structure

$$g : info_1 \uparrow perc \implies act \downarrow info_2$$

where

- g is a goal;
- $perc$ is a perception from the environment;
- act is an action performed on the environment;
- $info_1$ is the information to be removed from STM;
- $info_2$ is the information or goal to be stored in STM.

Symbol $\uparrow$ suggests removal from STM whereas symbol $\downarrow$ suggests storage in STM. We call *enabling* the part of the rule on the left of $\implies$ and *performing* the part of the rule on the right of $\implies$. The execution of a cognitive rule is enabled by the presence of goal g and information $info_1$ in STM, and by the perception $perc$ from the environment, and results in the removal of $info_1$ from STM and the human performance of action act on the environment and the storage of new information $info_2$ in STM.

Information $info_1$ consists of a set of *basic items*, which are syntactically listed as sequence with a comma as a separator, but whose order is semantically irrelevant. Each basic item may be a perception, an action or a cognitive state. In addition to basic items, $info_2$ may also contain a goal. In fact, when the goal

g is present in the rule, it is the only goal enabling the rule, while goals that may be in $info_2$ are actually produced in STM by performing the rule.

The syntax of goals is $goal(achievement)$, where $achievement$ is a set of basic items. The goal is achieved when $achievement$ contains the currently performed action or a basic item stored in STM. Once the goal is achieved, it is removed from STM. The absence of goal is denoted by $goal()$. In this case the syntax of a cognitive rule can be shortened as

$$info_1 \uparrow perc \implies act \downarrow info_2$$

When the goal is present and the perception is not, the control of attentional selection and behaviour is *deliberate* and is finalised to accomplish the information that is the arguments of the goal g. When the perception is present and the goal is not, the control of attentional selection and behaviour is *automatic*. When both the goal and the perception are not present, the rule models a mental inferential process consisting in the replacement of $info_1$ with $info_2$. When both the goal and the perception are present, the control of attentional selection and behaviour is *hybrid*, that is, driven by the goal (there is a deliberate decision to act or carry out a mental or attentional process), but reactive to the perception (the modality of acting or processing is automatic).

3.1 Cognitive Model Example: the User of a Vending Machine

The cognitive model of the vending machine user described in Sect. 2.1 is formalised in BRDL as follows:

$$goal(collect\ food, abort\ collect\ food) : \uparrow requested\ card \implies insert\ card \downarrow expect\ requested\ pin \quad (1)$$

$$goal(eat) : \uparrow requested\ card \implies insert\ card \downarrow expect\ requested\ pin \quad (2)$$

$$expect\ requested\ pin \uparrow requested\ pin \implies enter\ pin \downarrow expect\ returned\ card \quad (3)$$

$$expect\ returned\ card \uparrow returned\ card \implies collect\ card \downarrow expect\ delivered\ food \quad (4)$$

$$expect\ returned\ card \uparrow wrong\ pin\ warning \implies \downarrow wrong\ pin\ warning \quad (5)$$

$$goal(collect\ food, abort\ collect\ food) :\ wrong\ pin\ warning \uparrow \implies abort \downarrow abort\ collect\ food \quad (6)$$

$$goal(eat) :\ wrong\ pin\ warning \uparrow requested\ pin \implies enter\ pin \downarrow expect\ returned\ card \quad (7)$$

$$abort\ collect\ food \uparrow returned\ card \implies collect\ card \downarrow \quad (8)$$

$$expect\ delivered\ food \uparrow delivered\ food \implies collect\ food \downarrow food\ available \quad (9)$$

$$goal(eat) :\ food\ available \uparrow \implies eat \downarrow \quad (10)$$

The interaction is initiated by either rule 1 or rule 2. Rule 1 addresses level-2 needs by having the goal to either collect the food, to consume later, or abort the

interaction, as a protection from card confiscation in case of wrong pin. Rule 2 addresses level-1 needs by having the goal to eat the food immediately. Both rules also require the perception that the vending machine requests the user to insert the card in order to be enabled and determine the action of inserting the card and the storage in STM of the expectation that the machine will request the pin next.

Rules 3 and 4 are driven by the matching expectations and perception that the machine requests the pin and returns the card, respectively, and determine the actions of entering the pin and collecting the card, respectively. Rules 5 models the implicit attention to the perception of the warning that a wrong pin has been entered, thus failing to meet the expectation.

Rules 6 and 7 model the reaction of the user to the warning. Rule 6 is driven by a goal expressing level-2 needs and aborts the interaction as a consequence of the warning thus avoiding card confiscation. Since the rule stores item *abort collect food* in STM and such an item is in the goal achievement, the goal is achieved and removed from STM, but card collection occurs through rule 8, which does not have goal and models an automatic behaviour. The goal removal prevents rule 1 from being enabled upon the request by the system to reenter the pin. However, a hungry user has goal $goal(eat)$, which expresses a level-1 need, thus enabling rule 7 once the system requests the user to reenter the pin. In this way, a user with level-1 ignores any level-2 needs.

Item *food available* is stored in STM by rule 9. Finally, rule 10 is enabled by the goal $goal(eat)$ and the food availability, and determines the action of eating the food. A user who is hungry while purchasing the food already has the goal $goal(eat)$ and eats the food immediately. Instead, a user who has purchased the food in advance will have the goal $goal(eat)$ at a later stage, when becoming hungry.

4 Formal Environmental Model and Physiological Model

With reference to the example in Sec 3.1, we illustrate in Sect. 4.2 how the user interacts with the external environment and in Sect. 4.3 how goals (such as $goal(eat)$) are stored in STM as the result of physiological processes (such as becoming hungry). But first, in Sect. 4.1 we introduce *Labelled Transition Systems* (LTS) to model the *external environment* with which the user interacts as well as the *internal physiology* of the user.

4.1 Labelled Transition Systems(LTS)

We define an LTS by

- a set of perceptions;
- a set of invisible atomic states;
- an initial state consisting of a set of perceptions and a set of invisible atomic states;

– a set of transition rules $visible_1\ [invisible_1] \xrightarrow{act} visible_2\ [invisible_2]$, where sets of perceptions $visible_1$ and $visible_2$ and set of invisible atomic states $invisible_1$ and $invisible_2$ are represented by element separated by commas.

The system evolves starting from the initial state. Each transition rule models the transition from a source state consisting of a visible components $visible_1$ and an invisible component $invisible_1$ to a target state consisting of a visible components $visible_2$ and and invisible component $invisible_2$. The transition is triggered by action act.

4.2 External Environment: Interaction and Use of Variables

The interaction between the human component and the external environment is given through the synchronisation between a cognitive rule

$$g : info_1 \uparrow perc \Longrightarrow\ act \downarrow info_2$$

and a transition rule

$$perc, visible_1\ [invisible_1] \xrightarrow{act} visible_2\ [invisible_2]$$

which share the same action act. The transition is enabled if the current state of the LTS includes $perc, visible_1$ as a subset of its visible component and $invisible_1$ as a subset of its invisible component. The transition changes the state of the LTS by replacing $perc, visible_1$ by $visible_2$ in its visible component and $invisible_1$ by $invisible_2$ in its invisible component. Note that $visible_2$ may contain $perc$.

For example, the interaction between a user and a vending machine modelled in Sect. 3.1 occurs through actions *insert card* (rules 1 and 2), *enter pin* (rule 3 and 7), *collect card* (rules 4 and 8), *abort* (rule 6) and *collect food* (rule 9). If we consider rule 4, the machine detection of the user's action of collecting the card is formalised by transition rule

$$returned\ card\ [\ ] \xrightarrow{collect\ card} delivered\ food\ [\ ] \tag{11}$$

Rule 4 and transition 11 share action *collect card*. If the machine is in state *returned card*, the transition, which has only visible part, is enabled and synchronises with rule 4, which has *returned card* as the perception. The transition execution changes the machine state to *delivered food* thus enabling rule 9.

As an example of use of variables in defining the environment, consider the situation in which the hungry user of our vending machine enters a wrong pin for the second time, thus causing a card confiscation. The LTS that models the vending machine must use a variable *pin attempts* to count the attempts to enter. Thus the transition that perform the card confiscation is modelled as follows:

$$\begin{aligned} &requested\ pin\ [\ pin\ attempts\ = 1\ ] \xrightarrow{enter\ pin} \\ &\qquad confiscated\ card\ \ [\ pin\ attempts\ += 1\ ] \end{aligned} \tag{12}$$

Note that on the left-side of the transition variables are used with conditions, whereas in its right-side they are used within assignments.

4.3 Interaction with the Internal Physiology

Interaction with the internal physiology is modelled through the direct effect of a transition rule on STM or through the transition being triggered by the content of STM. There are three kinds of transition rules that directly effect STM. Given information *info*, which may include goals,

- $visible_1\ [invisible_1] \stackrel{\downarrow info}{\longrightarrow} visible_2\ [invisible_2]$ stores *info* in STM;
- $visible_1\ [invisible_1] \stackrel{info\uparrow}{\longrightarrow} visible_2\ [invisible_2]$ removes *info* from STM;
- $visible_1\ [invisible_1] \stackrel{\downarrow info\uparrow}{\longrightarrow} visible_2\ [invisible_2]$ is triggered by the presence of *info* in STM but does not change the content of STM.

As discussed in Sect. 2, we adopt the homeostatic-regulation theory [3], which explains motivation as the tendency of the body to maintain a state of equilibrium (e.g., hunger is balanced by eating). To this purpose, we identify the need with the motivator (the need of food is identified with its motivator 'hunger') and associate a numerical value with it. We consider two thresholds for the need, an activation threshold α and a saturation threshold σ such that $0 < \sigma < \alpha$.

We can suppose that initially the value of the need, which in our example is *hunger*, is below the α threshold. In this situation the motivator is inactive. The passing of time makes the need increase as a function of the human activity. When the need reaches the α threshold, the motivator becomes active. This means that we must carry out the appropriate activity, driven by a goal, to satisfy the need and, as a result, decrease its numerical value. Therefore, an iterative activity is carried out until the need has dropped down to the saturation threshold σ. Each step of the iterative cycle is driven by the goal and continues while the need is greater than the saturation threshold σ. In our example, the goal established in STM by an hungry person is eating ($goal(eat)$) and it drives the deliberate behaviour of eating, which is modelled by cognitive rule 10 defined in Sect. 3.1. Once the need is as low as the saturation threshold σ, the motivator goes back to the inactive state.

Rule 10 models the cognitive aspects of hunger, that is, our deliberate eating activity. However, there are several physiological aspects that control the feeling of hunger and motivate us to eat and to stop eating. We use LTSs to model such physiological aspects.

With reference to our example, the *physiological motivation process* can be modelled using three transition rules:

activation $[hunger > \alpha,\ inactive] \longrightarrow [active]$
This transition rule is enabled when condition $hunger > \alpha$ holds and the motivator state is *inactive*. The transition changes the state to *active*.

iteration $[hunger > \sigma,\ active] \stackrel{goal(eat)\downarrow}{\longrightarrow} [active]$
While condition $hunger > \sigma$, the motivator state is *active* and there is no goal $goal(eat)$ in STM, goal $goal(eat)$ keeps being stored in STM.

saturation $[0 \leq hunger \leq \sigma,\ active] \longrightarrow [inactive]$
This transition rule is enabled when condition $0 \leq hunger \leq \sigma$ holds and the motivator state is *active*. The transition changes the state to *inactive*.

At the end of each iteration step of the physiological motivation process, the execution of cognitive rule 10 causes action *eat* to be performed. Since action *eat* is an argument of $goal(eat)$, the goal is achieved and thus removed from STM.

The *physiological satisfaction process* is determined by the feedback of the eating activity, which decreases the feeling of hunger. By denoting such a decrease by δ, we can model the satisfaction process as follows:

$$[hunger > \sigma] \xrightarrow{eat} [hunger - = \delta] \tag{13}$$

This transition rule is enabled when condition $hunger > \sigma$ holds and the transition occurrence decreases *hunger* by a quantity δ.

We note that physiological states, such as needs, are modelled as invisible states since they are not directly visible from outside the LTS that models them. What is visible is the resultant behaviour, for example the fact that we eat.

5 Translation into Maude

5.1 A Short Overview of Maude

Maude [16] is a formal modelling language and high-performance simulation and model-checking tool for distributed systems. It makes use of (1) algebraic equational specifications in a functional programming style to define data types, and (2) rewriting logic specifications, expressed using rewrite rules, to define the system evolution. 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. It is possible to declare the same operator on various subsorts (*subsort overloading*). In this case the `ditto` keyword may be used to specify the same equational attributes used in the previous declaration of that operator. Equational attributes `pred` and `gather` may be used to enforce precedence among operators. 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 attribute in an equation denotes that the axiom is used for all cases that are not matched by the previous equations.

Maude *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'.

Core Maude supports the definition of *functional modules*, which start with keyword `fmod` and end with keyword `endfm`, for algebraic equational specifications, and *system modules*, which start with keyword `mod` and end with keyword `endm`, for rewriting logic specifications. All Core Maude statements, apart from module definitions are ended by a dot. Modules can be imported using the keyword `protecting` followed by the name of the module ended by a dot.

Core Maude also enables module reusability with *parametrised modules*, which allow the use of such sorts as `List`, `Map`, and `Maybe`. These modules do not exist on their own, but are generated dynamically depending on the value domain for the elements of these sorts. The importation of the modules is performed by defining a view from the Core Maude's `TRIV` theory to a user-defined module and mapping sort `Elt` to a sort that defines the value domain for the elements of parametrized sorts.

One of the ways to check formal models in Maude is the `search` command: `search` t `=>*` t'. This command finds all terms that satisfy term pattern t' and can be reached by applying rewrite rules any arbitrary number of times starting from term t.

Full Maude is the object-oriented extension of Core Maude. It supports the definition of classes and objects within *object modules*, which start with keyword `omod` and end with keyword `endom`, and are enclosed between parentheses '(' and ')'. In fact, all commands and modules must be entered enclosed in parentheses when using Full Maude. 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.

5.2 BRDL and LTS Syntax and Manipulation Operators

The Maude Syntax for the basic items introduced in Sect. 3, the BRDL cognitive rules and the transitions that make up an LTS are defined in functional module `ENTITIES`, whose contents are described in this section.

Basic items are defined by sort `InfoItem`. They are the building blocks for cognitive rules, whose components are defined by the following sorts:

```
sorts InfoItem Goal Information ContentSTM ContentSTMList .
subsort String < InfoItem < Information .
```

```
subsorts InfoItem Goal < ContentSTM .
subsorts ContentSTM  Information < ContentSTMList .
```

Sort `InfoItem` includes strings (it has predefined sort `String` as a subsort), and models the basic items introduced in Sect. 3. In fact, we have been modeling items of information as strings (possibly of multiple words) throughout the paper.

We have seen in Sect. 3 that in the general form of cognitive rule

$$g : info_1 \uparrow perc \implies act \downarrow info_2$$

perc and *act* are basic items while $info_1$ and $info_2$ are sets of basic items, with $info_2$ possibly containing goals. Therefore, we define sorts `Information`, to model $info_1$ and `ContentSTMList`, to model $info_2$, as follows:

```
op noInfo : -> Information [ctor] .
op _,_ : Information Information ->
                Information [ctor comm assoc id: noInfo prec 10] .
op _,_ : ContentSTMList ContentSTMList -> ContentSTMList [ditto] .
```

with `ContentSTMList` further characterised by the subsort declarations above stating that it has as subsort both `ContentSTM` (it may contain goals) and `Information` (it may contain basic items). A goal is defined as

```
op goalFromAnyOf : Information -> Goal [ctor] .
```

and the general form of cognitive rule is modelled as follows:

```
sort CognitiveRule .
op _:_|>_==_=>_>|_ :
        Goal Information InfoItem Parameter InfoItem ContentSTMList ->
                CognitiveRule [ctor] .
```

Similar definitions of constructors, e.g., `_:_|> ==_=>_>|_`, `_:_|>_==_=> >|_` and `_:_|> ==_=> >|_`, model the cases in which one of the components of the cognitive rule is missing.

Object module `COGNITION` defines operators `established` and `enabling` to manipulate short-term memory (STM), whose content is modelled by the `ContentSTMList` sort, and sort `LTM` to model long-term memory (LTM) as a set of cognitive rules:

```
op established : Goal ContentSTMList -> Bool .
enabling : Information ContentSTMList -> Bool .
sort LTM .   subsort CognitiveRule < LTM .
op emptyLTM : -> LTM [ctor] .
op _;_ : LTM LTM -> LTM [ctor comm assoc id: emptyLTM)] .
```

It also declares the class `Cognition` as follows

```
class Cognition | cognitiveLoad : Nat, stmCapacity : Nat,
                  shortTermMem : ContentSTMList, longTermMem :  LTM .
```

where attribute `stmCapacity` is the maximum number of basic items that can be in STM and `cognitiveLoad` models a number of additional items that are assumed to be in STM but are not modelled explicitly.

In order to model LTSs that can interact with the human cognition, we define a sort `Event` that characterises all events on which LTS and human cognition can synchronise and a sort `SystemState` that comprises a visible component consisting of the set of basic items and an invisible component consisting of a set of basic items 'hidden' within a sort `Invisible`:

```
sorts Event Invisible SystemState SystemRule .
subsort InfoItem < Event .
op _|> : ContentSTM -> Event [ctor prec 11 gather (e)] .
op |>_>| : ContentSTM -> Event [ctor prec 12] .
op >|_ : ContentSTM -> Event [ctor prec 11 gather (e)] .
op [_] : Information -> Invisible [ctor] .
op __ : Information Invisible -> SystemState [ctor] .
op __--_->__ : Information Invisible Event Information Invisible
                          -> SystemRule [ctor] .
```

Operators `_|>`, `|>_>|` and `>|_` define the mutual effects between transition rules and STM, which are expressed in BRDL by *info* $\uparrow$, $\uparrow$ *info* $\downarrow$ and $\downarrow$ *info* respectively. Thus an event is either defined by one of these three operators or is an element of sort `InfoItem`, which is subsort of `Event`. The transition rules of the LTS are modelled by sort `SystemRule`. Their Maude syntax closely reflects the BRDL syntax presented in Sect. 3. The operator `extSynch`, which is defined as follows

```
op extSynch : InfoItem InfoItem -> Bool .
vars S1 S2 : String .
eq extSynch(S1,S2) = S1 == S2 .
```

checks whether two basic items are made up by identical strings. We show in Sect. 5.4 that this operator is used in rewrite rules to check whether the event `S2` of a transition is the same as the action `S1` in a cognitive rule thus enabling their synchronisation.

Object module `LTS`

```
(omod LTS is protecting ENTITIES .
    op emptyLTS : -> LTS [ctor] .
    op _;_ : LTS LTS -> LTS [ctor comm assoc id: emptyLTS] .
    sort LTS .   subsort SystemRule < LTS .
    class System | currentState : SystemState, transitions :  LTS .
endom)
```

defines the sort `LTS` as a set of transition rules by including `SystemRule` as a subsort and defining the '`;`' operator to construct the set. Class `System` defines the LTS as the current state, which will be initialised by the initial state when creating the object, and a set of transitions.

Finally, object module `HUMAN`

```
(omod HUMAN is protecting COGNITION + LTS + CONFIGURATION .
    class Human | cognition : Oid, physiology : Oid .
endom)
```

defines the human component as a combination of cognition and physiology.

5.3 Introducing Variables

Our previous BRDL translation into Maude [5, 8–10] does not make use of variables to define cognition and environment. This results in models with duplicated cognitive rules and transitions, since each possible value of the same entity requires a dedicated rule/transition. This problem also involves physiological entities, such as *hunger*, whose value changes continuously controlled by the activation and saturation thresholds.

Therefore, we extend the standard BRDL logic to match and apply LTS transitions on information containing items with variable arguments. Variables allow the use of placeholders in a transition to generalise integer values in the current state of a system. This section describes how we map values from a system state to variables in the left-hand side of a transition and how we generate a new state from the right-hand side of the transition.

First, transitions with variables have a different matching logic. Therefore, we introduce the `GENERIC-ENTITIES` module, which extends the `ENTITIES` module to construct transitions from information containing variables. We declare a sort `GenericItem` as an extension of the `InfoItem` sort. `GenericItem` defines information items that may or may not contain variables. Similarly, the `GenericInfo` sort is a supersort of `Information` containing both `GenericItem` and `InfoItem`.

Then we define a logic for matching variables. The variables operate by replacing actual values in the system state. Therefore, we first introduce a functional module `MATCHING` to define basic concepts by using sorts `Variable` and `Value` to represent placeholders and actual contained values, respectively. These sorts are not populated in the module to support the use of any data types of values and any format of variables. Then, a new sort `Matching` is introduced to reflect the substitutions made when values are matched with variables:

```
sorts MatchingItem Matching Variable Value .
subsort MatchingItem < Matching .
op _:=_ : Variable Value -> MatchingItem [ctor prec 50] .
op noMatch : -> Matching [ctor] .
op _;_ : Matching Matching ->
                        Matching [ctor comm assoc id: noMatch prec 51] .
```

Finally, we define how the matching is constructed and how it is applied to the given system state, left-hand side and right-hand side of a transition to generate a new system state. Functional module `GENERICS` contains the definition of the `genericMatch` operator to generate matching for transition application:

```
protecting (MAYBE * (op maybe to failMatch)){Matching} .
op genericMatch : Matching Information GenericInfo  -> Maybe{Matching} .
```

This operator takes initial matching, current system information and the left-hand side of a transition as arguments. The resulting sort `Maybe{Matching}` means that the operator returns either a valid `Matching` of variables in a transition onto values in the system information or a term `failMatch`. This is defined

by the importation of parametrised module `MAYBE` with mapping of the `maybe` operator onto `failMatch`. The operator is reduced according to the equation:

```
ceq genericMatch(MATCH, (INFOITEM, INFO), (ITEM, GENERIC))
= matchItem(MATCH, INFOITEM, ITEM);
  genericMatch((MATCH); matchItem(MATCH, INFOITEM, ITEM), INFO, GENERIC)
if matchItem(...) :: Matching /\ genericMatch(...) :: Matching .
```

The equation defines the information matching by applying `matchItem` on pairs of generic items `ITEM` on the left-hand side of a transition and information items `INFOITEM` in the system state. The condition ensures that this equation is applied only if `matchItem` of all pairs of items produce valid matchings. Otherwise, a different order of items will be considered because the information is a set and any item can match `ITEM` and `INFOITEM`.

The `resolveGeneric` operator transforms the system state depending on the generated matching and right-hand side of the transition:

```
protecting (MAYBE * (op maybe to failResolve)){Information} .
op resolveGeneric : Matching GenericInfo -> Maybe{Information} .
```

The operator constructs the values of the information resulting from transition application by using `resolveItem` to substitute variables in each generic item and by appending it to the rest of the substitutions.

As we noticed in Sect. 4.2, variables may be uses within either conditions in the left-hand side of a transition or assignments in the right-hand side of a transition. This is implemented by the following sort and operator declarations:

```
sorts Condition ValueExpr GenericExpr .
subsort ValueExpr Variable < GenericExpr < GenericArgument .
subsort Value < ValueExpr < ExactArgument .
op _given_ : GenericExpr Condition -> GenericArgument .
op _'-_ : GenericExpr GenericExpr -> GenericExpr [prec 33 gather (E e)] .
op _'+_ : GenericExpr GenericExpr -> GenericExpr [prec 33] .
```

The operators `'+` and `'-` contain a quote symbol to avoid conflicts with predefined operators for numbers, a common approach used in Maude [1].

As examples of use of variables, transition 12 introduced in Sect. 4.2 is translated to Maude code as

```
"requested pin" [ pinAttempts(a given a '== 1) ] -- "enter pin" ->
                                "confiscated card" [ pinAttempts(a '+ 1) ]
```

and transition 13 introduced in Sect. 4.3 is translated to Maude code as

```
noInfo [ hunger(a given a '> 0) ] -- "eat" -> noInfo [ hunger(a '- 1) ]
```

5.4 Rewrite Rules

The evolution of the Maude model is defined by the `GENERIC-EVOLUTION` system module, which consists of the following rewrite rules:

```
crl [GENERIC_SYSTEM_EVOLUTION] :
  < TS : System | currentState : (INFO1, INFO3 [ INV1 , INV3 ]),
    transitions : (INFO2 [ INV2 ] -- auto -> INFO4 [ INV4 ]) ; TRANS >
=>
  < TS : System | currentState :
      (resolveState( INFO1, INFO2, INFO4 ), INFO3
        [ resolveState( INV1, INV2, INV4 ) , INV3 ]),
    transitions : (INFO2 [ INV2 ] -- auto -> INFO4 [ INV4 ]) ; TRANS >
if checkState( INFO1, INFO2, INFO4 ) /\ checkState( INV1, INV2, INV4 )
```

Fig. 1. Rewrite Rule for autonomous system evolution.

`GENERIC_SYSTEM_EVOLUTION` which is shown in Fig. 1, defines the autonomous evolution of a system modelled by an LTS that does not interact with a human component. The full definition of the `System` class of module `LTS` is given in Sect. 5.2. The rewrite rule checks whether in the LTS `TS` there is a transition whose source state `INFO2 [ INV2 ]` matches the current state (with `TRANS` being the rest of the transitions) using the `checkState` operator and, if the matching is found, performs the transition using the `resolveState` operator to change the current state. Each of these two operators is applied separately to the invisible and visible parts of the current state and exploits the operators `genericMatch` and `resolveGeneric` discussed in Section 5.3 to respectively generate a valid `Matching` from mapping values `INFO1` and `INV1` onto generic `INFO2` and `INV1`, respectively, and successfully resolve all variables in `INFO4` and `INV4`.

`GENERIC_PHYSIOLOGY_REMOVES_INFO` which is shown in Fig. 2, is one of the three rewrite rules that define the interaction of cognition with physiology. This rewrite rule defines the effect of operator *info* `|>`, which implements *info* ↑. The definitions of `Human`, `Cognition` are given in Section 5.2. If item `CONTENTSTM1` is in STM, then the rule is applied to a transition with event `CONTENTSTM1 |>` by removing `CONTENTSTM1` from STM and performing the transition as in rewrite rule `GENERIC_SYSTEM_EVOLUTION`.

`GENERIC_PHYSIOLOGY_ADDS_INFO` defines the effect of operator `>|` *info*, which implements ↓ *info*, by adding item *info* to STM and performing the matching transition as in rewrite rule `GENERIC_SYSTEM_EVOLUTION`.

`GENERIC_PHYSIOLOGY_READS_STM` defines the effect of operator `|>` *info* `>|`, which implements ↑ *info* ↓, by checking whether item *info* is in STM without changing the contents of STM and performing the matching transition as in rewrite rule `GENERIC_SYSTEM_EVOLUTION`.

`GENERIC_EXTERNAL_SYNCHRONIZATION` defines the interaction between a human component and an external environment, such as user interfaces. It implements the synchronisation process described in Sect. 4.2.

`GENERIC_ACTION` defines the interaction between a human component and an external environment without any human perception of the state of the environment.

```
crl [GENERIC_PHYSIOLOGY_REMOVES_INFO] :
  < H : Human | cognition : CO, physiology : PHY >
  < CO : Cognition | cognitiveLoad : CL, stmCapacity : CAP,
    shortTermMem : (STMEM); CONTENTSTM1,
    longTermMem : LTMEM >
  < PHY : System | currentState : INFO1, INFO3 [ INV1 , INV3 ],
    transitions :
      (INFO2 [ INV2 ] -- (CONTENTSTM1 |>) -> INFO4 [ INV4 ]) ; TRANS >
=>
  < H : Human | cognition : CO, physiology : PHY >
  < CO : Cognition | cognitiveLoad : CL, stmCapacity : CAP,
    shortTermMem : (STMEM),
    longTermMem : LTMEM >
  < PHY : System | currentState :
    (resolveState( INFO1, INFO2, INFO4 ), INFO3
      [ resolveState( INV1, INV2, INV4 ) , INV3 ]),
  transitions :
    (INFO2 [ INV2 ] -- (CONTENTSTM1 |>) -> INFO4 [ INV4 ]) ; TRANS >
if CL + load(STMEM) <= CAP
  /\ checkState( INFO1, INFO2, INFO4 ) /\ checkState( INV1, INV2, INV4 )
```

Fig. 2. Rewrite Rule for a physiology removing information from STM.

6 Conclusion and Future Work

We have extended our framework for modelling and analysing human reasoning and behaviour [6, 7] with a generalised notion of goal, which allows the choice between alternative achievements. For example, this allows a user interacting with an interface to consider the task completed either by succeeding or aborting. This means to address level-2 psychological needs by safely aborting the task when a danger or threat is perceived.

We also translated into Maude the formal framework for emotions and motivators defined in previous work [6, 7] by incorporating it in the previous BRDL translation [5, 8–10] and further generalising it with the introduction of variables.

The BRDL translation into Maude defined in this paper has been used to model and analyse the user task defined in Sect. 3.1. We used Maude model-checking capabilities to verify that the absence of level-1 needs results in a safer choices that meet level-2 needs. Starting from the initial cognitive state $goal(collect\ food, abort\ collect\ food)$, which addresses level-2 needs, we can verify that the card is confiscated only when the user is hungry. We use Maude `search` command to find a counterexample to this property. The searched term models a vending machine state that contains the *confiscated card* basic item and a physiology state that contains *inactive* hunger item. The result of the

search given by Maude is `No solution`, thus verifying property. The Maude code of the example can be downloaded from GitHub[1].

Although it was not described in this paper for space reasons, the extension of the Maude translation supports the generation of emotion defined using BRDL and LTSs in our previous work [7]. In our future work we are planning to model the effect of generated emotion on motivation and decision making.

References

1. Alpuente, M., Ballis, D., Romero, D.: A rewriting logic approach to the formal specification and verification of web applications. Science of Computer Programming **81**, 79–107 (2014)
2. Anderson, K.L.: Arousal and the inverted-uy hypothesis: Aq critique of nessiss's "reconceptualizing arousal". Psychological Bullettin **17**, 96–100 (1990)
3. Cannon, W.B.: The Wisdom of the Body. Norton (1932)
4. Cerone, A.: A cognitive framework based on rewriting logic for the analysis of interactive systems. In: Software Engineering and Formal Methods (SEFM 2016), pp. 287–303. No. 9763 in Lecture Notes in Computer Science, Springer (2016)
5. Cerone, A.: Towards a cognitive architecture for the formal analysis of human behaviour and learning. In: STAF collocated Workshops 2018 (FMIS), pp. 216–232. No. 11176 in Lecture Notes in Computer Science, Springer (2018)
6. Cerone, A.: Behaviour and reasoning description language (BRDL). In: SEFM 2019 Collocated Workshops (CIFMA), Lecture Notes in Computer Science, vol. 12226, pp. 137–153. Springer (2020)
7. Cerone, A.: A BRDL-based framework for motivators and emotions. In: SEFM 2023 Collocated Workshops (CIFMA), Lecture Notes in Computer Science, vol. 13765, pp. 351–365. Springer (2023)
8. Cerone, A., Murzagaliyeva, D.: Information retrieval from semantic memory: BRDL-based knowledge representation and Maude-based computer emulation. In: SEFM 2020 Collocated Workshops (CIFMA), Lecture Notes in Computer Science, vol. 12524, pp. 150–165. Springer (2021)
9. Cerone, A., Ölveczky, P.C.: Modelling human reasoning in practical behavioural contexts using Real-Time Maude. In: FM'19 Collocated Workshops - Part I (FMIS), Lecture Notes in Computer Science, vol. 12232, pp. 424–442. Springer (2020)
10. Cerone, A., Pluck, G.: A formal model for emulating the generation of human knowledge in semantic memory. In: Proc. of DataMod 2020, Lecture Notes in Computer Science, vol. 12611, pp. 104–122. Springer (2021)
11. Hull, C.L.: Principles of Behaviour. Appleton-Century-Crofts (1943)
12. Hull, C.L.: A behaviour system: An introduction to behaviour theory concerning the individual organism. Yale University Press (1952)
13. James, W.: Psychology. Holt (1890)
14. Maslow, A.H.: A theory of human motivation. Psychological Review **50**, 370–396 (1943)
15. Maslow, A.H.: Motivation and Personality. Harper, 2nd edn. (1970)
16. Ölveczky, P.C.: Designing Reliable Distributed Systems. Undergraduate Topics in Computer Science, Springer (2017)

[1] https://github.com/AntonioCerone/Publications/tree/master/2023/CIFMA

17. Solomon, R.L.: The opponent-process theory of motication: The costs of pleasuree and the banefits of pain. American Psychologist **35**, 681–712 (1980)
18. Woodworth, R.S.: Dynamic Psychology. Columbia University Press (1918)
19. Yerkes, R.M., Dodson, J.B.: The relation of strength of stimulus to rapidity of habit formation. Journal of Comparative Neurology and Psychology **18**, 459–482 (1908)