Declarative Language for Behaviour Description

Imre Piller, Dávid Vincze, and Szilveszter Kovács

Department of Information Technology, University of Miskolc,
Miskolc-Egyetemváros, H-3515, Miskolc, Hungary
{piller,david.vincze,szkovacs}@iit.uni-miskolc.hu

Abstract. The Fuzzy Rule Interpolation (FRI)-based Fuzzy Automaton is an efficient structure for describing complex behaviour models in a relatively simple manner. The goal of this paper is to introduce a novel declarative behaviour description language which is created for supporting special needs of ethologically inspired behaviour model definition. For the sake of simplicity, the grammar is created with as few keywords as possible, keeping the ability to describe complex behavioural patterns as well. The language is a declarative language mainly supporting the behaviour models built upon structures of interpolative fuzzy automata. The paper firstly presents the formal structure of the behaviour description language itself, then gives an overview of the interpreting and processing engine designed for the language. Finally, an application example, a definition of a set of behaviours and a simulated environment is also presented.

1 Introduction

The idea of ethologically inspired behaviour models is originated from the existence of descriptive verbal models based on numerous observations of living creatures' reactions in different situations. Many of them are built around a predefined sequence of environmental situations and events where the reactions are noted and evaluated in detail. The knowledge representation, in this case, is a series of observations of various facts and action-reaction rules. For mathematical modelling of such a system, the rule-based knowledge representation is straightforward. Moreover, the need of following the observed sequences requires a model structure of a state machine or an automaton. Adding the fact that the observable or hidden states are continuous measures, the situation is quite complicated. Summarizing the above requirements, the model has to have a rule-based knowledge representation and the ability of describing event sequences of continuous values and continuous states. The suggested modelling method is the adaptation of the fuzzy automaton where the state is a vector of membership values, the state-transitions are controlled by a fuzzy rule-base and the observations and conclusions are continuous values.

P. Sinčák et al. (eds.), *Emergent Trends in Robotics and Intelligent Systems*,
Advances in Intelligent Systems and Computing 316, DOI: 10.1007/978-3-319-10783-7_11

The goal of this paper is to introduce a declarative behaviour description language supporting the simple definition of ethologically inspired behaviour models, i.e. various structures of fuzzy automata.

2 FRI-Based Fuzzy Automaton

There are numerous versions and understandings of the fuzzy automaton which can be found in literature (a good overview can be found in [1]). In our case, we started from the most common definition of Fuzzy Finite-state Automaton (FFA, summarized in [1]), where the FFA is defined by a tuple (according to [1], [2] and [3]):

$$\tilde{F} = (S, X, \delta, P, O, Y, \sigma, \omega),$$

where

- S is a finite set of fuzzy states, $S = \{\mu_{s1}, \mu_{s2}, \dots, \mu_{sn}\}$.
- X is a finite dimensional input vector, $X = \{x_1, x_2, \dots, x_m\}$.
- $P \in S$ is a fuzzy start state of $\tilde{F}$.
- O is a finite dimensional observation vector, $O = \{o_1, o_2, \dots, o_p\}$.
- Y is a finite dimensional output vector, $Y = \{y_1, y_2, \dots, y_l\}$.
- $\delta: S \times X \to S$ is a fuzzy state-transition function which is used to map current fuzzy state into the next fuzzy state upon an input value.
- $\sigma: O \to X$ is an input function which is used to map the observation to the input value. See, e.g., Figure 1.
- $\omega: S \times X \to Y$ is an output function which is used to map the fuzzy state and input to the output value. See, e.g., Figure 1.

In case of fuzzy rule-based representation of the state-transition function $\delta: S \times X \to X$, the rules have $n + m$ dimensional antecedent space and n dimensional consequent space. Applying the classical fuzzy reasoning methods, the complete state-transition rule-base size can be approximated by the following formula:

$$|R| = n \cdot i^n \cdot j^m, \tag{1}$$

where n is the length of fuzzy state vector S, m is the input dimension, i is the number of the term sets in each dimension of the state vector and j is the number of the term sets in each dimensions of the input vector.

According to (2), the state-transition rule-base size is exponential with the length of the fuzzy state vector and the number of the input dimensions. Applying FRI methods to the state-transition function, the fuzzy model can dramatically reduce the rule-base size (see, e.g., [4]).

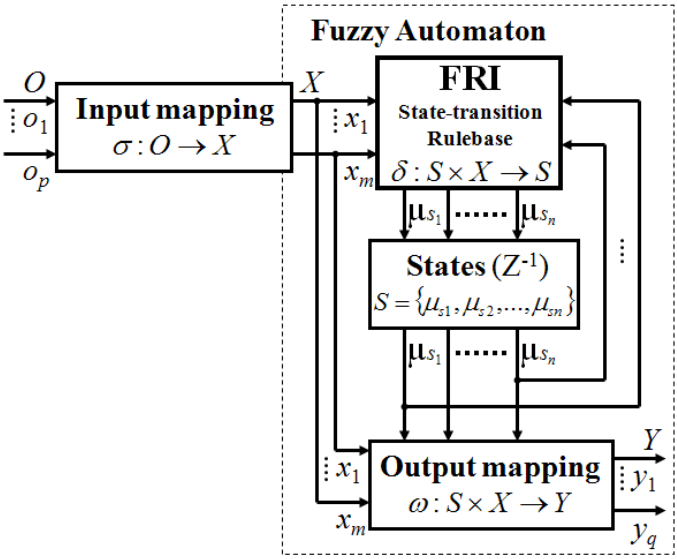


Fig. 1. FRI-based Fuzzy Automaton

3 Behaviour Modelling

In this paper, we consider behaviour modelling as a task of setting up a behaviour-based control system (see, e.g., [5] for an overview). In behaviour-based control the actual behaviour of the system is formed as a fusion of component behaviours appeared to be the most appropriate in the actual situation. According to this concept, the behaviour model is built upon the component behaviours, the behaviour coordination and the behaviour fusion. The behaviour coordination decides which component behaviour is needed, i.e. it calculates the weights for each component behaviours required in solving the actual situation. Then, the behaviour fusion combines the component behaviours to form the actual behaviour according to their corresponding weights. Turning the suggested FRI-based Fuzzy Automaton to be a behaviour model, the output function $\omega: S \times X \to Y$ has to be decomposed to parallel component behaviours and an independent behaviour fusion. In this case, we can get a very similar structure as it is expected in a behaviour-based control. (More detailed description of the suggested Fuzzy Automaton-based behaviour model structure can be found in [6]).

For setting up the structure of the behaviour model by a declarative language, the set of component behaviours and the state-transition rule-base are needed to be defined. In this case, the component behaviours act as symbols and the relation of the state-transition rules forms the model structure.

4 Declarative Language for Behaviour Description

The task of the suggested declarative language is to provide a simple way for defining the state-transition rule-base size in a human readable form close to the original verbal form, as the ethological models are given.

The proposed model is built from various rule-bases. According to the common decomposed FRI models, the rule-bases can have an arbitrary number of input values (antecedents) and a single output value (consequent). The state of the system is determined by the values of the edges. This enables the description of the structure of the system without restrictions to the inner implementation of the rule-bases.

All rule-bases must possess a unique name; also the name of a rule-base needs to be the same as the name of its consequent. Therefore, the connections between the rule-bases can be defined by these unique names of the antecedents and the consequent.

The presented language is a structured language consisting of various blocks. A block can be opened simply by using any valid keyword which, at the same time, defines the type of the block. Blocks should be closed with the 'end' keyword. Some types of blocks have a name between quotation marks after the type of block keyword. All types of blocks can optionally contain a 'description' keyword which is followed by a documentation comment. Depending on the type, the contents of the blocks are slightly different. Considering the block definitions presented hereinafter, our goal was to make them verbose and readable by humans.

In order to provide a formal description of the proposed behaviour description language, we present syntax diagrams. The text element marks a quoted string which can contain arbitrary characters except the quote character itself. Hereinafter, the language will be introduced in a top-down manner.

The most important type of block is the 'rule-base' type (Figure 4).

At the start of the rule-base definition a 'method' block (Figure 4) defines which of the supported consequent calculation methods should be used with its corresponding parameters. Currently, two methods are supported – the FRI method called FIVE (Fuzzy rule Interpolation based on Vague Environment, introduced in [7], [8] and [9]) and direct Shepard interpolation [10]. Of course, other methods can be applied in the future.

The 'term' element (Figure 4) is similar to the function call mechanism found in computer programming languages. After defining the name and the type of the term (term_name, term_type), one can set its parameters with an argument list in parenthesis. The 'term' also defines the name for used measures (see Figure 4), which helps to keep the rules simple but verbose. The form of 'term' is a triplet of term name, term type and value list. For example, the "large" triangle (15, 10, 20) denotes a triangle-shaped fuzzy set definition for the large linguistic term. The available parameter types are integer, logical value and string. The possibilities of the parametrization are determined by the exact method selected previously.

The syntax diagram of the antecedent definition can be seen in Figure 4 and the diagram of the consequent definition in Figure 4. The cardinal difference between the

two block types is that all antecedents have the 'name' part. The accessible term types depend on the previously selected method ('method' block described above).

The antecedent and consequent definitions attach symbols, which are real values, to the inputs and outputs, the connection between them is defined with the help of the 'rule' keyword. The general form of 'rule' is shown in Figure 4.

The first 'text' defines the name of the output term in the consequent block. The predicates (see Figure 4 for the form of predicates) are given in a form of a list which contains the antecedent name and the term name pairs. A rule explains that the consequent value is the value after the 'rule' keyword when the predicates are true.

5 The Interpreter and the Behaviour Engine

The processing of the rule-base files follows the classical method with two stages. The first stage is the tokenization which has the task of extracting the tokens from the raw text input. In this implementation five different token types are considered.

The tokenizer distinguishes two types of string tokens: the keyword and the previously introduced text type. The keywords of the proposed language and also the term types are read as keyword tokens. Other strings are quoted and handled as text types. For parametrization the tokenizer provides numerical and parenthesis token types. There is also an empty type which marks the end of the file.

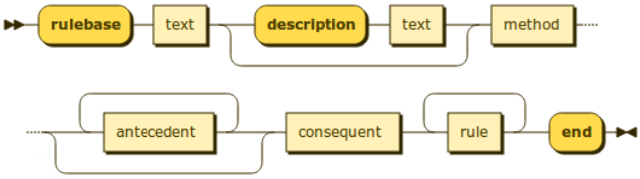
The language employs a line comment feature which is denoted by a hashmark symbol. In the first step, the comments are eliminated. The tokenizer ignores special or incorrect characters, in fact, those are treated as simple whitespace characters which are semantically important only for the purpose of separating words.

An unterminated text token causes the whole definition file to be invalidated. Error handling is an important part in the processing of the text input.

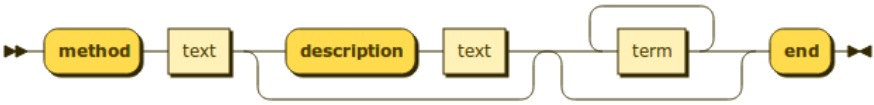
The second stage of the processing is the parsing or grammar analysis. Parsing is based on the output received from the tokenizer. The language interpreter checks the grammar of the rule-base definition with a recursive parser. All significant structural blocks have a corresponding method for parsing, for example, the whole definition file is read with the read RuleBase method of the Parser class. For the inner elements, these can be called recursively.

The grammar analysis part is more prone to errors than the tokenization steps. In the case when an error occurs, an information from the tokenizer is required for the parser for the location of the error. This is necessary because the parser itself does not have any knowledge about the position of the error in the definition file.

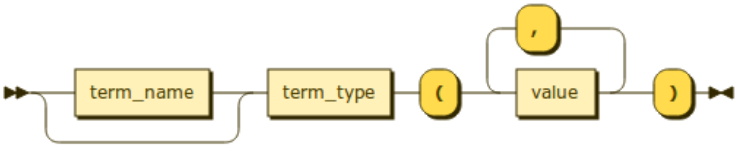
The rule-base definition file is represented by the rule-base class. This class stores the method name and its parameters, the antecedents, the consequent and the rules themselves. It also includes methods for creating the described rule-base object. After the syntax analysis was successfully completed, it is necessary to check the integrity of the rule-base object.



(a) rulebase definition

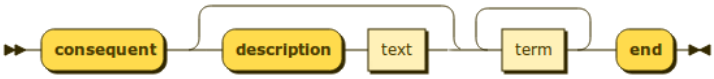


(b) method block definition

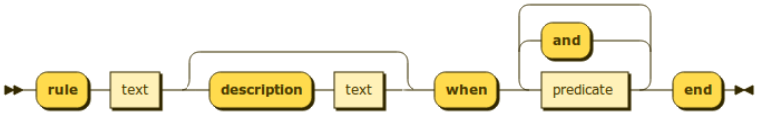


(c) term block definition

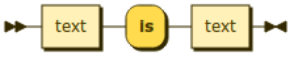
(d) antecedent block definition



(e) consequent block definition



(f) rule block definition



(g) predicate block definition

Fig. 2. Syntax diagrams of the description language

The following checks are made:

- All the terms in the antecedent are unique.
- All the antecedent names are unique.
- In consequent part, all names are unique.
- The output term in the rule is defined.
- When the predicate is valid, the antecedent and the term are existing.

The rule-base class contains only information which is available from the description. For calculations, the node object should be instantiated according to the properties of the method. It also requires parsing because the meaning and the usage of the method and term function parameters depend on the node type.

When all the rule-base definition files have been processed, the connections between rule-bases are automatically discovered based on the unique names of rule-bases and inputs. The antecedent names which are not used as rule-base names will be the inputs of the system, and the outputs will be the consequences of the rule-bases.

For the calculations to succeed, it is necessary to set the initial values. Omitting initialization can cause problems because inappropriate values possibly render the system unstable.

6 Application Example

As a demonstration, a simple behaviour set consisting of three rule-bases is presented. The inputs and outputs of the system are simulated by a small application developed for this purpose. The simulation environment employs an agent which is able to explore its environment (move around). Also, there are rewards distributed in the environment which can be collected by the agent. Furthermore, there are objects in the environment which are considered to be a source of danger for the agent, therefore, these should be avoided by the agent. Based on the described behaviours (rule-bases), the agent is controlled by the behaviour engine. After a certain time of operation, it is necessary for the agent to have a rest for a short while because the tiredness inner state of the agent is increasing while it is moving. The main action type of the agent is the collection of reward objects (the number of collected rewards is counted).

The inputs supplied for the behaviour engine are the distances measured from the closest reward object, distances from the place where the agent can have a rest, distances from the dangerous object, measure of tiredness and the number of collected rewards. These values are registered and provided by the simulation environment.

The above described behaviour is defined by the following rule-bases. The interest rule-base shows the measure of interest for the exploration action. Its value depends on the number of rewards, tiredness of the agent and the distance from the nearest reward object. The second rule-base is the fear, which depends on the distance from the dangerous object and on the value of tiredness. The third rule-base is the activity. Its output determines where the agent should head towards: go to the reward object, go to the resting place, or avoid the dangerous object. The heading direction is determined based on the derived interest and fear values.

According to the dependencies, the interest rule-base has three antecedent definitions. For example, the distance of the reward object looks as follows:

```
antecedent "reward distance"
description "The distance from the closest reward."
"small"   stem(0)
"average"   stem(50)
"large"   stem(100)
end
```

The *stem* in the definition is a control point on a $(n + 1)$-dimensional surface of the rule-base. The output has three possible values, these are defined in the consequent block.

```
consequent description "The measure of the interest."
"low"   stem(0)
"average"   stem(10)
"high"  stem(20)
end
```

The connection between the input and output values can be defined by the following rule blocks:

```
rule "low" when
"reward distance" is "large" and "danger di tance" is
"low"
end
rule "high" when "tiredness" is "low" end
```

For the simulation of the above presented behaviour, an application with a graphical user interface was created (a screenshot of the simulated environment is shown in Figure 3). The demonstration application is available in [11].

In the first frame of Figure 3, the agent is in the centre of the operating area. The small yellow circles denote the rewards to be collected and the larger dark yellow circle near the agent is the place for resting. The darker purple points represent the dangerous objects. During the first four steps, the agent collects the rewards in its vicinity. When it arrives at the position in the fifth frame, it gets tired, therefore it goes to its resting place. In the last three steps, the agent collects four rewards at the bottom of the area, then goes back to rest again. This will be the final position because the dangerous objects are too close, hence the remaining rewards won't be collected.

The antecedent definitions for distance-like values are similar to the already presented values in the case of 'reward distance', also the other definitions are easy to read. The output of the activity rule-base determines the current behaviour and the rules are based on the previously calculated internal properties, namely on the interest and the fear. This complex behaviour can be defined with simple and verbose rules. Examples of some rules from the activity rule-base follow:

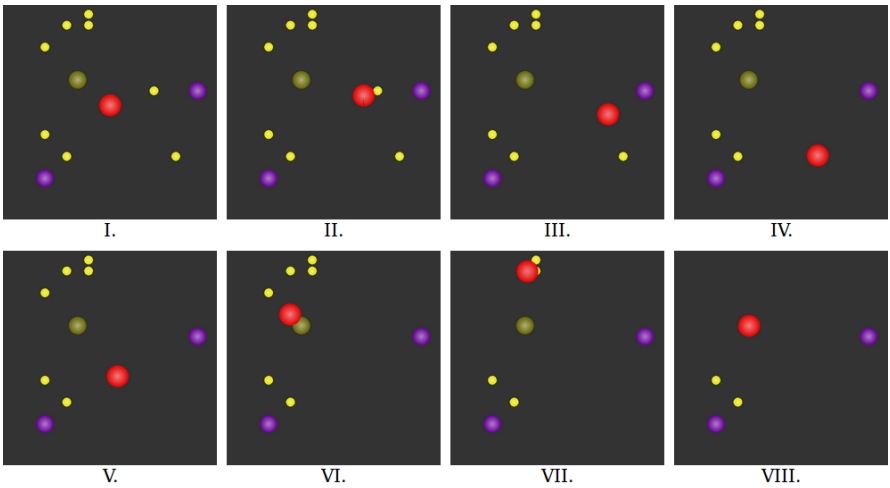


Fig. 3. The simulation environment for testing the behaviour descriptions. The path of the agent can be followed on the above stated frame sequence. The agent takes a rest, which can be seen on Frame V. The last frame (VIII) shows the stabilized state.

```
rule "collect reward" when

      "interest" is "high" and "fear" is "low"
  end

  rule "collect reward" when
      "interest" is "normal" and "count of collected
rewards" is "few"
  end

  rule "go to rest" when
      "count of collected rewards" is "a lot of"
  end
```

7 Conclusion

The presented declarative behaviour description language is mainly intended to be used by non-technical users, especially ethologists. An interpreter for the proposed behaviour description language was implemented as a complete behaviour engine with the ability of handling inputs, outputs and states. Currently, for evaluating the behaviour descriptions, a fuzzy rule interpolation-based fuzzy automaton calculation method and direct Shepard interpolation are implemented, but there are no restrictions to the usable calculation methods within the behaviour engine, other methods can be implemented, too. Furthermore, the engine enables the construction of hybrid systems with different types of rule-bases, which allows for the construction of complex

behaviour-based systems. Overall, the main goal of the paper is to introduce a tool for non-technical users to describe observed behaviour sets in a way which is directly interpretable by using automated processing.

Acknowledgments. This research was supported by the Hungarian National Scientific Research Fund grant no: OTKA K77809. This research was carried out as part of the TAMOP-4.2.2.C-11/1/KONV-2012-0002 project with support of the European Union, co-financed by the European Social Fund.

References

1. Doostfatemeh, M., Kremer, S.C.: New directions in fuzzy automata. International Journal of Approximate Reasoning 38, 175–214 (2005)
2. Omlin, W., Giles, C.L., Thornber, K.K.: Equivalence in knowledge representation: Automata, rnns, and dynamical fuzzy systems. Proc. IEEE 87(9), 1623–1640 (1999)
3. Belohlavek, R.: Determinism and fuzzy automata. Inf. Sci. 143, 205–209 (2002)
4. Kovács, S.Z.: Interpolative Fuzzy Reasoning in Behaviour-based Control. In: Reusch, B. (ed.) Computational Intelligence, Theory and Applications. Advances in Soft Computing, vol. 2, pp. 159–170. Springer, Germany (2005) ISBN 3-540-22807-1
5. Pirjanian, P.: Behaviour Coordination Mechanisms - State-of-the-art, Tech-report IRIS-99-375, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California (October 1999)
6. Kovács, S.: Fuzzy Rule Interpolation in Embedded Behaviour-based Control. In: 2011 IEEE International Conference On Fuzzy Systems (FUZZ-IEEE 2011), Taipei, Taiwan, June 27-30, pp. 436–441 (2011)
7. Kovács, S.: New Aspects of Interpolative Reasoning. In: Proceedings of the 6th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Granada, Spain, pp. 477–482 (1996)
8. Kovács, S., Kóczy, L.T.: Approximate Fuzzy Reasoning Based on Interpolation in the Vague Environment of the Fuzzy Rule base as a Practical Alternative of the Classical CRI. In: Proceedings of the 7th International Fuzzy Systems Association World Congress, Prague, Czech Republic, pp. 144–149 (1997)
9. Kovács, S., Kóczy, L.T.: The use of the concept of vague environment in approximate fuzzy reasoning. In: Fuzzy Set Theory and Applications, vol. 12, pp. 169–181. Tatra Mountains Mathematical Publications, Mathematical Institute Slovak Academy of Sciences, Bratislava, Slovak Republic (1997)
10. Shepard, D.: A two dimensional interpolation function for irregularly spaced data. In: Proc. 23rd ACM Internat. Conf., pp. 517–524 (1968)
11. The simulation application is, `http://users.iit.uni-miskolc.hu/~piller/`