Proceedings of the 6th WSEAS Int. Conf. on FUZZY SYSTEMS, Lisbon, Portugal, June 16-18, 2005 (pp23-28)

Compiler design for Fuzzy Classifier Systems

Jenny Menolascina, Jose Aguilar-Castro, Francklin Rivas-Echeverría
Universidad de Los Andes
Facultad de Ingeniería
Escuela de Ingeniería de Sistemas
Mérida, Venezuela 5101

Abstract:- A Rule Compiler design for Fuzzy Classifier Systems is described in this paper. The design of the compiler is based on the building of Grammars described like predicates, which represent the system rules. The ANTLR tool (Another Tool for Language Recognition) is used for the implementation of the compiler. We propose also an interface that makes easier to the user the task of writing, compiling and administering the rules stored in the Knowledge Base.
Key Words:- Fuzzy Classifier Systems, Fuzzy Logic, Expert Systems, Compiler.

1 Introduction

In our life we find several complex situations commanded by rules: control systems, safety systems, bank transactions, etc. Rule-based systems are an efficient tool to deal with these specific problems. The Knowledge Base contains the variables and the rules defining the problem, ant the inference engine obtains the conclusions through the application of classic logic to these rules. A rule is defined –in our field of work- as a "If premise, then conclusion" structure, where premise and conclusion are expressions which can be based on fuzzy logic, with one or more affirmative statements, connected via logic operators like "AND", "OR" or "NOT".

Since its appearance in the sixties, Fuzzy Logic applications have earned consolidation [3, 4]. They are found in solutions for industrial control problems, time series prediction, Operative Research, maintenance strategies, search methods in databases, and so on. Probably, the main reasons to such vast array of applications are the conceptual simplicity of the fuzzy systems, their ability to combine in a unified manner the linguistic expressions with numeric data, and their implementation without sophisticated algorithms. Particularly, it is possible to use Fuzzy Classifier Systems (FCS) in situations that imply uncertainty and incomplete or complex information management [2]. The FCS are a type of learning machine that uses rules based on Fuzzy Logic for modeling a problem.

The main objective of this paper is to develop a rule compiler that can be used by Expert Systems and FCS. This is achieved through the utilization of the ANTLR language (Another Tool for Language Recognition) [2], which generates compilers from a grammar specification of the language to be recognized. Hence, one of the main contributions of this paper is the proposal of a grammatical structure that defines how the rules should be. Our system is developed using Java [9], and defines an interactive interface, which allows the user, in a comfortable and practical way, to use the system. This paper is a part of the project named "Computational Platform for the development of Expert Systems and Fuzzy Systems" [8].

This paper presents only the rule compiler's design for the FCS. In order to study the Expert System please refer to [7]. This article is organized as follows: section 2 introduces the Theoretical Framework, section 3 describes the System Design, section 4 presents a Study Case, and finally in section 5, we present the Conclusions and Limitations found, as well as the possible eventual further works.

2 Theoretical Framework

2.1 Fuzzy Logic

Fuzzy Logic is, essentially, the incorporation of the concept of multivalued logic [2, 3, 4]. Human reasoning uses truth values that are not necessarily determining (statements with just true or false values). For instance, when it is said that "The sky

is blue", it could be possible to think how blue the sky is indeed. Likewise, it would be possible to think "if a vehicle moves fast", it would be possible to think how fast it moves, since this last observation does not imply necessarily the quantification of speed with the accuracy required.

The adjective "fuzzy" is due to the fact that the non-determining truth values used in them have, generally, an uncertainty meaning. A half-full glass, notwithstanding the fact that it is also half-empty, is not totally full nor totally empty. This is the type of indeterminate properties that we can manage with fuzzy theory.

A Fuzzy System can be developed based on a set of heuristic rules, in which the inputs and outputs linguistic variables are represented by fuzzy sets. The following figure shows the main components [3]:

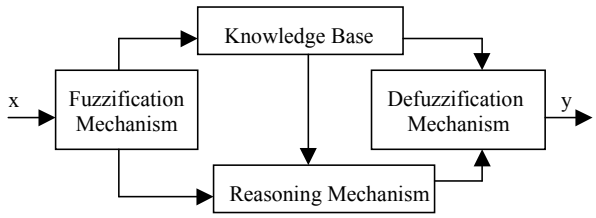

Figure 1. Fuzzy Logic System Scheme

A Fuzzy System is composed of a mechanism that transforms discreet data into Fuzzy data (fuzzification mechanism), another mechanism that makes the inverse process based on one of the classic techniques of defuzzification, such as the centroid method, a knowledge base that stores the Fuzzy rules, and the mechanism of Fuzzy reasoning.

2.2. Fuzzy Classifier System

One of the most important challenges in the Intelligent Computing area consists of modelling intelligent behavior through the use of Intelligent Techniques (Artificial Neural Networks, Genetic Algorithms, etc.). The systems that attempt to model intelligent behaviors similar to the humans' belong to the area known as Learning Machines [2, 8]. The FCSs are a type of Learning Machine based on Fuzzy Logic.

The FCSs try to imitate the way in which human beings make decisions. These systems are generally robust and tolerant to imprecision and noises in the input data. The FCSs apply the Fuzzy Logic with the aim of imitating human reasoning in computers. In order to achieve this goal, mathematic theory based on fuzzy set is used to map subjective notions, such as hot, warm, cold, to concrete values that can be manipulated by computers. A FCS is composed of the following elements (see Figure 2):

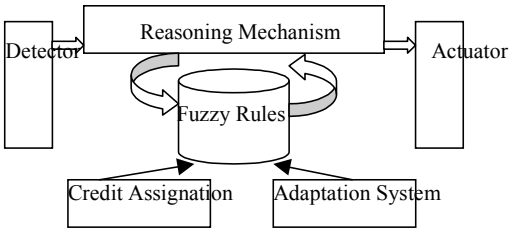

Figure 2. Fuzzy Classifier System Scheme

A message detector system is the responsible of the information fuzzification. An actuator system generates the commands derived from the reasoning process of the system. It also performs defuzzification tasks, in case of being required. The Fuzzy Rules System has a Fuzzy reasoning mechanism that takes such rules and the messages provenient from the exterior to perform the inference process. The Adaptative System allows the generation or removal of rules in the rules system, in accordance with their quality. Those rules that are not suitable for the environment where the FCS is operating must be discarded and the combination of the best rules generates new rules. In order to determine the non-suitable rules and the best ones, the Credit Assignment System is used. This system gives points to each rule, taking into account if it is activated or activates others rules when a requirement (message) arrives to the FCS.

2.3. Compiler

A translator is any program that takes as input a text written in a language –called source- and gives as output a different text in a language called object. The translator is called a compiler if the language is of programming high level, and the object is a low level language (assembler or machine code) [5, 6].

The compiler, besides translating, performs other series of operations that, mostly, are focused on the errors detection in the source program. A compilation is constituted by the following phases [5, 6].

Lexical Analyzer: The Lexical Analyzer, also called scanner, detects basic units of information in the source program that belong to the language. These units are called tokens or lexic units. A token is an element of the source language that has its own meaning. It can be the reserved words of a language, identifiers, operators, etc. Some examples of lexical errors can be the reserved words spelled in a wrong way, not allowed identifiers, etc.

Syntax Analyzer: The Syntax Analyzer or parser takes the tokens received from the scanner and searches in it the possible syntax errors that could appear.

Semantic Analyzer: The Semantic Analyzer completes the two previous phases, incorporating certain proofs that can not be assimilated to the simple recognition of a chain. For example, not declared variables or an operator applied to a non-compatible operating agent.

Error-Handle: its mission is to try to correct the errors found in the different phases of the compilation. The types of errors that a program can have are the following: lexical, syntax, semantic and logic errors (those due to the performance of something wrong for the problem to be solved), execution errors (Examples of this type of errors are: division by zero (0), reading from a not open file, or without any information, etc.).

3 System Design

The Computational Platform for the Fuzzy Classifier is shown below (See figure 3).

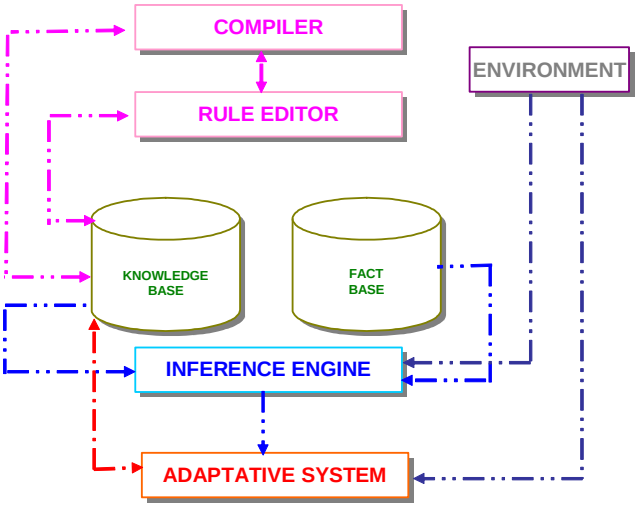

Figure 3 Fuzzy Classifier System Modular Design

The system is composed of various sub-systems [8]: the Rule Edition Sub-system, composed by the Compiler and the Rule Editor; the Rule Performance and Adaptation Sub-systems, which contain, respectively, the Inference Engine and the Adaptative System. Finally, the Information Storage contains the Knowledge Base and the Fact Base.

The Inference Engine starts the Fuzzy reasoning process taking the system input variables and verifying the rules that are activated (Knowledge Base). The Fuzzy reasoning mechanism used is the classic "Modus Ponens" [8]. The Adaptative System updates automatically the set of rules, in accordance with the usage they have during the functioning of the system.

The Edition Sub-system is composed by the following components [7]:

1. A compiler, it performs the Lexic, Syntax and Semantic Analyses of the rules.
2. A Rule Editor, the interface used to write the rules.

The Knowledge and Rules Bases are used either by the compiler and the Rule Editor.

3.1 Compiler

It has the following architecture:

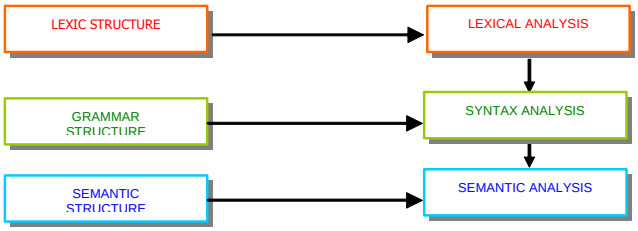

Figure 4. Compiler Architecture

This paper explains the Fuzzy Classifier System compiler design. The compiler of the Expert Systems is simpler (See [7]).

a) LEXIC STRUCTURE

It is composed by lexic components. An example is shown below:

id	→	letter(letter|digit)*
letter	→	['A'-'Z"a'-'z"_']
digit	→	['0'-'9']

b) GRAMMAR STRUCTURE

The language used for specifying the Fuzzy Classifier System follows the Grammar shown next (the grammar is described by a set of production rules, whose initial production is sd_rule.

sd_rule	→	IF fuzzy_prop_list THEN fuzzy_prop_list EOF
fuzzy_prop_list	→	fuzzy_prop((OP_Y|OP_O|OP_Ym| OP_Om) fuzzy_prop)*
Fuzzy_Proa	→	(OP_NO|OP_NOm)? frase
Frase	→	Fuzzy_atribute IS value
Fuzzy_atribute	→	ID
Value	→	ID

c) SEMANTIC STRUCTURE

In this system, variables are Fuzzy and each one of them has a linguistic value set associated. Our system has input variables (that can be used only in the premises side), or input/output variables (they can be used in both sides). The semantic verifications performed allow the compiler to be sure about the following aspects:

- Variables should exist in the Knowledge Base.
- The linguistic values correspondent to the Fuzzy Sets used.
- The premise of a rule should not be equivalent to the consequent of it.
- The input variables should not appear in the consequent nor the output variables in the premises side.

In order to make the compiler, the ANTLR tool was used. This tool uses the LL(k) algorithm for the lexic, syntax and semantic analyses [1]. Hence, this tool integrates the generation of lexic, syntax and semantic analyses. The LL(k) algorithm works with grammar. That is, the structures previously shown should be given to the tool. This algorithm examines the input from left to right. The LL(k) algorithm is implemented through the definition of a function for each rules of production.

The ANTLR receives files with the .g extension, which describe the grammar of the language to be compiled. This tool uses these files to generate new ones, written in Java language, which contain classes to perform each one of the compiler's phases [1]. Hence, the compiler is composed by classes generated through the use of the ANTLR, which are: RuleParserSD, RuleLexerSD, RecontreeSD, RuleParserSD and RecontreeSD. They are stored in the GraGeneralSD.g and GraDSem.g files. It is also done similarly for the compiler of Expert Systems (See [7]).

3.2. Rules Editor

The Rules Editor is part of the general system interface. It contains the rules stored to be modified, delete, allows building the new one for the Fuzzy Classifier System. We access to the Rules Editor through the Designer's panel, which, at the same time, can be accessed from the System's main panel.

Figure 5 shows the Main Panel. The left side presents all the Knowledge Bases, either for the Expert Systems or Fuzzy Classifier Systems.

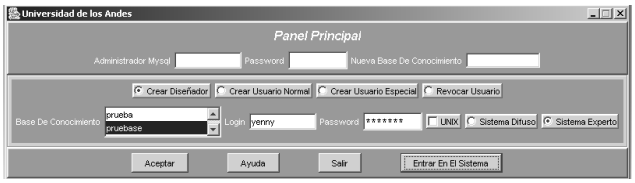

Figure 5. Main Panel.

In order to open the Designer's panel a Knowledge Base is selected from the main panel, then, the login and the password of a Designer user is required. At the moment of pressing the "Enter into the system" key, the designer's panel appears, as it is shown in the following figure.

Figure 6. Designer´s Panel

The Designer's panel in the lower- right side, shows a key called "Rule Editor". This key opens the Rules Editor window for the construction of the rules, and the report of the possible mistakes that can appear or the successful compilation of them.

3.3. Description of the Computational Platform

The Java language was used for developing the system. For our rule editor, we have built the following packages: Expert-Master Package, Antlr Package, Compiler Package and Interface Package.

The Knowledge Base is implemented using Mysql. The compiler and the Rule Editor access it by the class mysql-connector-Java. It is used through the standard API of Java. In order to obtain

the connection, it is used the Drive Manager class of the Standard API, which locates the class com.mysql.jdbc. This connection is used through an object that implements the connection interface.

4 Study Case for a Fuzzy System

4.1. Problem definition:

We like the construction of a Fuzzy system to determine the percentage of opening or closure of a control valve destined to maintain the pressure inside a tank under normal levels (see Figure 7). At the inside of the tank is generated a chemical reaction fed through pipes that transport components. These pipes do not have valves.

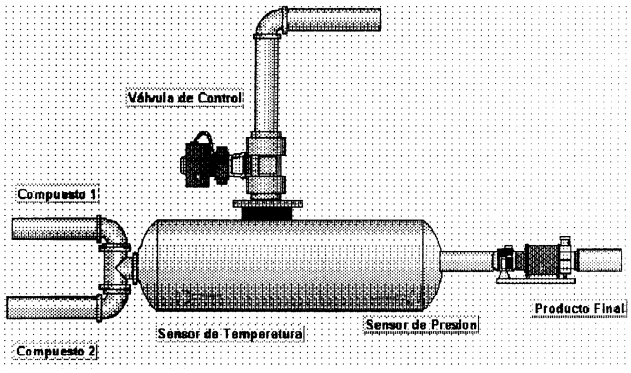

Figure 7. Physical system to be modeled.

The only way of maintaining the levels of pressure around an operation point is allowing the escape of gasses generated by the reaction, if pressure is high; or retaining these gasses inside the tank, if pressure is low, using the control valve for this purpose. There are two sensors in the tank: one that measures the temperature and the other that measures the pressure.

4.2. Modelling the problem using FCSs:

The process was previously modelled based on the temperature and the pressure inside the tank (input variables to the system) and the percentage of opening and closure of the control valve (output variable of the system). Some of the Rules that characterize this system are:

1. If Temperature is High AND Pressure is High Then Valve_aperture is Open
2. If Temperature is High AND Pressure is Low Then Valve_aperture is Close
3. If Temperature is High AND Pressure is Medium Then Valve_aperture is Maintained

We will take only one of them to perform the tests to the Rules Editor Sub-system. The rule chosen is the number 3.

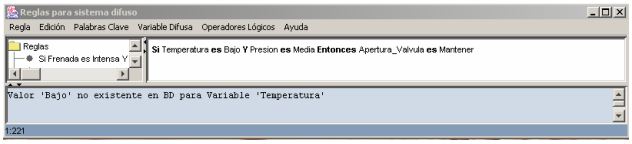

Figure 8. Compilation errors Example detected by the compiler

As it is shown in Figure 8, it was given a linguistic value to the temperature variable that it does not have (it is not in the Database). It is detected by the compiler, which sends an “error” message.

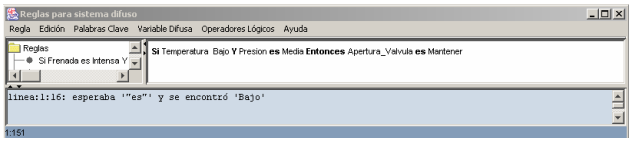

Figure 9. Compilation errors Example detected by the compiler

Figure 9 shows that there is no “IS”, the word that separates the Fuzzy variable from its associated linguistic value (fuzzy set). That is why the error message indicates that was expected an “IS” before the linguistic value.

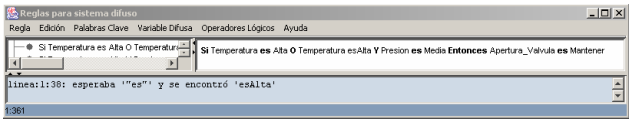

Figure 10. Compilation errors Example detected by the compiler

Figure 10 shows the expectation of a blank between the “IS” and the linguistic value. As a consequence, it indicates an error, since “ishigh” is not a linguistic value.

Figure 11. Compilation errors Example detected by the compiler

Figure 11 attempts to save a rule without finishing it (only the premises side has been described). At this moment, it asks if we want to save the rule. When saying “yes” the compiler displays a message that says: Unexpected end of rule.

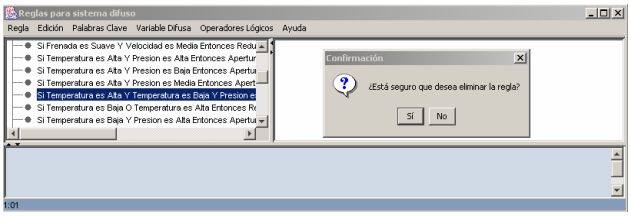

Figure 12. Example of a rule deletion

The rule highlighted with blue is the one that is expected to be deleted, as it is shown in Figure 12, the system demands: Are you sure you want to eliminate the rule?

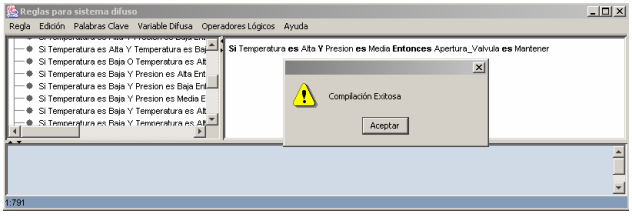

Figure 13. Successful Compilation

Figure 13 shows a message that says: "Successful Compilation". This happens when the rules are well built. At the moment of pressing this choice, the rule is immediately saved in the Knowledge Base and added to the left panel located in the Rule Editor.

5 Conclusions

This paper has developed a Fuzzy Classifier Systems Compiler, which would make easier its use by those who are interested in them. This compiler works with rules such as "**If** a **Then** b", where "a" represents the condition and "b" the action. For instance, a Fuzzy Classifier System rule could be:

"**If** level **is** very high **and** pressure **is** medium, **then** level **is** medium".

The only operators used are "AND", "OR" and the unary operator "NOT".

An advantage offered by the implementation of this system is that it uses an interface done in Java; based on a Rules Editor that presents to the user the possibility of editing any rule. This editor uses a menu bar where is possible to find all the components and Fuzzy variables needed to construct rules.

The interface is helpful for the user at the moment of building rules. When editing rule through the menus, the possibility of mistakes is reduced, due to the fact that the interface adjusts continually the status of the menus. It is based on a lexical and syntax analysis of the rule that is being written. On the contrary, if the rules are written without using the menus, it is possible to commit mistakes. These are detected by the compiler when the rule is intended to save.

Two types of grammars were built for the construction of rules, one for the Expert Systems and the other one the Fuzzy Classifier Systems. The ANTLR language was used for the programming of these grammars. This language is written in Java and generates Java, C++ and C#.

The developed compiler only works with a very simplified format of rules. In the further works of the system, it could be proposed the enrichment of the grammar. Among these improvements could be included the following: the variables used should take diverse values in determined times (it is to say, introducing the temporal concept). In addition, it should be permitted that rules should not only use Fuzzy proportions of the type: "Fuzzy variable **is** linguistic value", but also the use of the form: "**The** Fuzzy variable **from** the object **is** linguistic value". For example, they could have the following shape: "**The** humidity **of** soil **is** high".

6 References

[1] T. Parr, *ANTLR: Parser Generator and Translator Generator*. http://www.antlr.org.

[2] J. Aguilar, F. Rivas, *Introducción a las Técnicas de Computación Inteligente*. MERITEC. Venezuela. 2001.

[3] D. Sáez, *Fundamentos de Lógica Difusa*. Univ. Quilmes, Argentina, 2002.

[4] S. Kartalopoulos, *Understanding Neural Networks and Fuzzy Logic: Based concepts*, IEEE Press, USA, 1996.

[5] A. Aho, V. Sethi, R. Ullman, *Compiladores, Principios, técnicas y herramientas*. Addison-Wesley, Iberoamericana, 1990.

[6] J. Tremblay, P. Sorenson. *The theory and practice of Compiler Writing.* Mc-Graw-Hill, USA, 1985.

[7] Y. Menolascina, J. Aguilar, F. Rivas, *Diseño de un Compilador en java para Sistemas Expertos,* to be Publicated, *Conferencia Iberoamericana en Sistemas, Cibernética e Informática*, Orlando, USA, July 2005.

[8] J. Sanchez, J. Aguilar, F Rivas, *Knowledge Base and Inference Motor for an Automated Management System for Developing Expert Systems and Fuzzy Classifiers, WSEAS Transactions on Systems Journal*, No. 2, Vol, 3, pp. 682-687, 2004.

[9] Bruce, Eckel. *Thinking in Java Second.* Edition. Pearson Education. 2000