D I P L O M A R B E I T

Learning Fuzzy Rules With Genetic Algorithm And Neural Nets

ausgeführt am Institut für Informationssysteme
Abteilung für Datenbanken und Expertensysteme
der Technischen Universität Wien

unter Anleitung von
o.Univ.Prof. Dr. Georg Gottlob
und
Univ.Ass. Dr. Wolfgang Slany
als verantwortlich mitwirkendem Universitätsassistenten

durch
Markus Bonner
Stöberplatz 4/28
A-1160 Wien

Wien, im September 1996

Abstract

Various artificial intelligence methods have different advantages. Combining such methods allows to combine these advantages and widen the fields of application for these methods. Fuzzy logic can be used to handle vague and uncertain knowledge and imprecise information. Genetic algorithms and neural nets can learn the rules for a fuzzy system. Here a combined genetic fuzzy system is documented which uses a context-free grammar as interface. A neuro fuzzy system approach is also described.

Key words

Fuzzy logic; fuzzy control; genetic algorithms; neural nets; combined systems; machine learning; technical decision support; genetic fuzzy system; neuro fuzzy learning; steel-making application;

Contents

1 Introduction **11**

1.1 Artificial intelligence directions 11

1.2 Properties of artificial intelligence methods 11

1.3 Selection of methods 13

1.4 Future 14

1.5 Structure of this thesis 14

2 Project description **15**

2.1 Hot metal desulfurization 15

2.2 Way of the hot metal 15

2.3 Controlling the process 16

2.3.1 Chemical background knowledge 16

2.3.2 Monitoring the relevant influences 17

2.3.3 Analyzing the data 17

2.3.4 Process models 17

2.3.5 Improvements 17

2.4 Difficulties 17

2.4.1 Influences not measurable 18

2.4.2 Unknown importance 18

2.4.3 Time dependent influences 18

3 Fuzzy logic **19**

3.1 Simple idea 19

3.2 Computation with words 19
3.3 Compatibility 20
3.4 Typical fields of application 20
3.5 Fuzzy Sets 21
3.6 Linguistic terms 22
3.7 Linguistic variables 23
3.8 Instanciation 24
3.9 Rules 25
3.10 Evaluation 26
3.10.1 Fuzzification 26
3.10.2 Application of importance 27
3.10.3 Aggregation 27
3.10.4 Certification 28
3.10.5 Inference 28
3.10.6 Accumulation 29
3.10.7 Defuzzification 29
3.11 Operators 29
3.11.1 Aggregation operators 30
3.11.2 Classification of fuzzy operators 31
3.11.3 Defuzzy operators 36
3.11.4 Moderators 38
3.11.5 Algorithmic operators 38

4 FLIP++ **39**

4.1 ∗FLIP++ project 39
4.1.1 FLIP++ as part of a scheduling project 39
4.2 Representation of knowledge 40
4.2.1 Qualitative representation 40
4.2.2 Quantitative representation 41
4.2.3 Parameter sets 42
4.2.4 Instanciation of variables 43
4.3 Relations within qualitative knowledge 43
4.3.1 The relation in the example 45
4.3.2 Constraint tree - modularity 46
4.4 Operators 47
4.4.1 ACOtan - an Associative Compensatory Operator . . . 48
4.5 Combination of already-represented knowledge 49
4.6 Reasoning by evaluating knowledge 51
4.7 Conclusion 51

5 Example **52**

5.1 Problem description 52
5.1.1 Variables 52
5.1.2 Function 52
5.2 Fuzzy solution 54
5.2.1 Linguistic variables 54
5.2.2 Rule set 55
5.3 Comparison 58

6 Genetic algorithms **60**

6.1 Initialization 61

6.2 Cycle of the generations 61

6.2.1 Calculation of fitness 61

6.2.2 Reproduction 62

6.2.3 Mutation 62

6.2.4 Crossover 62

6.3 Combination genetic algorithm - fuzzy logic 63

7 Genetic programming kernel **65**

7.1 Context-free grammar interface 65

7.2 Training circle 66

7.2.1 Parsing 66

7.2.2 Translating 67

7.3 Generation of a population 67

7.4 Parsing the individual 68

7.5 Translating the word into task's syntax 68

7.6 Evaluation 68

7.7 Returning fitness value 69

7.8 Applying best the individual 69

8 Structure of the genetic fuzzy system **70**

8.1 Input for training 70

8.2 Training 70

8.3 Training cycle 71

8.4 Translating the result 71
8.5 Independent fuzzy system 72

9 Building up the system **73**

9.1 Designing a grammar 73
9.2 Initialization from an existing rule set 74
9.3 Raw fitness calculation 75
9.4 Parser for grammar words 77
9.5 Translating grammar word 79

10 Training the system **80**

10.1 Initialization 80
10.2 Training cycle 81
10.3 Retransformation 82

11 Neuro fuzzy system **83**

11.1 Task 83
11.2 The model 84
11.3 Historic data 85
11.4 Learning 86
11.4.1 Finding out relevant variables 86
11.4.2 Building up a hierarchy 87
11.4.3 Dividing variables into linguistic terms 87
11.4.4 Creating all possible rules 88
11.4.5 Classifying the rules 88

11.4.6 Selecting more reliable rules 89
11.5 Conclusion 89

12 Conclusion **90**

A IJCAI'95 Report **94**
A.1 The Workshop 94
A.1.1 Medical applications 94
A.1.2 Image processing 94
A.1.3 Case based reasoning 95
A.1.4 Fuzzy expert systems 95
A.1.5 Fundamentals 95
A.1.6 Membership functions 95
A.1.7 Further Information on the FLAI'95 Workshop 96
A.2 The IJCAI'95 Conference 96

B Fuzzy tools comparison **97**
B.1 Criteria for the selection of a fuzzy logic tool 98
B.2 MATLAB Fuzzy Logic Tool Box 100
B.3 fuzzyTECH 102
B.4 SieFuzzy 105
B.5 TILShell 107
B.6 NEFCLASS 110
B.7 FUZZLE 111
B.8 Mathematica Toolbox 112

B.9 DataEngine 112
B.10 FOOL & FOX 113
B.11 FUZZ-C 115
B.12 FIDE 115
B.13 DBFuzzy 116
B.14 FRIL 116
B.15 FuzzySoft 116
B.16 Fuzzy Systems MATLAB-Toolbox 117
B.17 FuzzyControlDesk, FuzzyDecisionDesk, FuzzyPatternDesk . . 117
B.18 NeuFuz 118
B.19 Fuzzy Box 118
B.20 IlmenauerFuzzyTool 119
B.21 FLIP++ 120
B.22 Other Fuzzy Logic and Neuro Fuzzy Software 121
B.23 Conclusion 123

C FLIP++ header files **125**

C.1 flip.h 125
C.2 flippara.h 126
C.3 flipcvar.h 134
C.4 flipoper.h 145
C.5 fliprule.h 152

List of Figures

1 Linguistic variable with linguistic terms and membership functions . . . 23

2 Set of instanciated membership functions of a linguistic variable 24

3 Minimum operator . . . 32

4 Drastic product operator . . . 32

5 Algebraic product operator . . . 33

6 Maximum operator . . . 34

7 Drastic sum operator . . . 34

8 Algebraic sum operator . . . 35

9 Arithmetic mean operator . . . 36

10 Membership function for percentage difference of aluminum . . . 42

11 Set of parameters for membership functions in FLIP++ . . . 44

12 Graphical user interface with rules and variables . . . 45

13 General tree of constraints . . . 46

14 Default operator set in FLIP++ . . . 48

15 ACOtan - A associative compensatory operator . . . 50

16 Distribution of input variable A . . . 53

17 Distribution of input variable B . . . 53

18 Functionality of $C = \log (A / B)$ . . . 54

19 Linguistic variable A . . . 55

20 Linguistic variable B . . . 56

21 Linguistic variable C . . . 57

22 Rule set for formula $C = \log (A / B)$ . . . 58

23 Functionality of the rule set 58

24 Comparison of formula and rule set 59

25 Distribution of the deviations between formula and rule set . . 60

1 Introduction

This chapter introduces some artificial intelligence methods and explains their advantages for real life solutions. Furthermore, an overview of the structure of this thesis is given.

1.1 Artificial intelligence directions

There are mainly two different directions of methods in artificial intelligence. The one group is more concerned with the application of logic theories. And the second one tries to understand human thinking and rebuild at least single aspects of the vague understanding of the human way of information processing within some artificial intelligence methods.

Part of the theoretical approach is to create logic that might fit into models of real life tasks. Thus, the characteristics of the model and the logic are analyzed. If they show similarities, the logic can be applied on the problem. The greatest area of applied logic lies within data base applications.

On the side of the human-like artificial intelligence the first step is to understand the human brain and thinking processes. Because these two fields are still quite unexplored and not well understood, it is hard to rebuild these methods within computer systems. It is hard to make a copy of a document if the original is only available in fragments. It is the same with human thinking.

What we know already about thinking processes, allows us already to develop methods that function similarly to the human way of information processing. This approach is the one I will mostly refer to as artificial intelligence.

1.2 Properties of artificial intelligence methods

The fact that many artificial intelligence methods are derived from processes within nature or within human thinking makes them, in some way, familiar to persons applying them on their tasks.

Neural nets are derived from the networks of nerve cells in living bodies. They use our vague understanding of the functions of nerve cells to create

some methods that can work in a comparable way. Even this simple copy of our poor insight into the processes within the brain allows us to build up methods that can be successfully applied to problem-solving. For more information on neural nets see [11].

Genetic algorithms try to make available our understanding of the evolutionary process to computational problems. Here it is the same as with neural nets. What we currently know about the development of species allows us to build methods that can successfully be applied to learning tasks.

Fuzzy logic tries to bring computational information processing nearer to the way we do. Most people do not think digitally or in terms of real numbers. In many cases we think in linguistic terms or words. And many of the time thinking is not such hard work for our brain that it would cause us headaches as consequence of a too large thought load. It is obvious therefore to try to use this way of information processing also in machines. For the reason that we describe our real life problems mostly in words, it is useful to bring the computer system to a level where it can understand our descriptions of a task. It is practical, too, if the machine can operate with our description of environment and processes.

All three methods have in common the use of quite simple ideas that we already know from nature. Because we are more or less familiar with these ideas, it is easier for us to understand, accept, and apply them. The tools can by applied without a complet theoretical understanding. The ideas of these methods are easy to apply to real life problems because they also come from reality and they are not only theoretical models, for which it is often hard to find representations in nature.

All three ideas are very uncomplicated and have natural origins. Therefore, they can easily be combined. By using a combination of methods, drawbacks of single methods can be compensated by other methods. We can, for example, add to a representation in fuzzy logic the learning abilities of genetic algorithm or neural nets like described in [10, 9]. There are many ways to combine these methods. They can be used at the same time, one after another, or more than one can be used and the best result then applied.

These three methods are often used if other more conventional methods of information processing have failed, or are too expensive, or inefficient.

Fuzzy logic allows us to build up models of a process from expert knowledge and experience even where no exact theoretical model is available. It also can be operated without full information or with imprecise informa-

tion. Neural nets and genetic algorithms can learn relations within tasks that would be unable in other ways. Therefore, these methods widen the field where computers can be applied and bring constant results where before results would only be guessed or estimated at best.

1.3 Selection of methods

The important point about methods selection is the usability for a given task. The value of a method can only be seen in connection with the environment where it is to be applied. The nature and the size of a task are important parameters for the right selection of a method to solve a problem within a game, simulation, or real life problem.

The nature of a problem determines the principle selection of a method. On one hand, we need a model of knowledge in order to apply this knowledge to a given problem. On the other hand, we need models in order to gain new knowledge from either observing and analyzing a process, or from a human expert, or from evaluating historic data. In many cases, we need both. Therefore, we have to combine models of knowledge with adequate learning or training methods.

An equally important influence is the expected size of a problem. The need of computer resources are limited in many cases. Even though computers have become faster and faster, and larger memories are available, these resources are still a limited. Many artificial intelligence methods consume large amounts of computational resources. They are mostly developed for solving problems for which no simple solutions are known. For the reason that these methods are not a-priori intelligent, they try to find solutions by searching the possible search space in various ways. This fact is common to most of the artificial intelligence methods. They vary mainly in their model of the search space. Therefore, they have a great appetite for calculation time. And even in future they will not have difficulty to consume all processor resources they can get. Already relatively simple games or simulations have huge search spaces. Solutions for real-world problems with even more dimensions, will also be limited in the near future. Hopefully massively parallel processing and adapted methods will lead to a next step in the use of artificial intelligence.

Other criteria for the selection of artificial intelligence methods are more practical reasons. One argument for the selection of a method can be the availability of an expert for a method. Other considerations are the tools

that can be used to apply these methods; which tools are available; and which tools fit into the current environment; and how high are the estimated costs for the single methods and tools? Often user preferences have to be considered. If a certain method for a certain application is planned, and there is no important reason for another decision, the user's preferences are taken as the basis for the selection of a particular tool.

1.4 Future

On one hand, the above-mentioned artificial intelligence methods bring advantages where they are applied. On the other hand, their use provides feedback for improving the methods or their combinations. The more these methods are used for solving real life problems, the more we will learn about the needs of real life tasks. The feedback from applications gives necessary experience and motivation for further development of methods.

These further steps also concern the theoretical side. A theory is necessary for proving the reliability of these methods. This has to be granted for safety-relevant applications. The theory is also important for a better understanding of the methods to create more efficient and reliable real-world solutions.

1.5 Structure of this thesis

This work starts with a general introduction into aritificial intellingence methods. The selection of such methods is also discused there. A description of the steel-making project which as a basis for this work is given in Chapter 2. Chapter 3 and 4 are concerned with fuzzy logic respectively the FLIP++ fuzzy logic library. Then follows an example used as basis for a learning genetic fuzzy system in Chapter 5. In Chapter 6 and 7 genetic algorithms and a genetic programming kernel are introduced. Chapter 8, 9, and 10 describe the structure, the generation, and the training of a genetic fuzzy system. In Chapter 11 an approach for a neuro fuzzy system is explained. The last chapter summarizes the single methods.

The appendix holds a report of the IJCAI'95 conference in Montreal, followed by a fuzzy tools comparison, and the header files of the FLIP++ library.

2 Project description

The practical application for which this learning fuzzy system was developed is the control of the desulfurization process of liquid raw metal. In this chapter a short description of the process and its control is given.

2.1 Hot metal desulfurization

The purpose of the desulfurization of hot metal is to reduce the sulfur content in liquid hot metal to about one third of the original content depending on the demands of the steel plant. This is done by adding a powder containing mainly a carbide that reduces the sulfur. The reduction product gets part of the slag that swims on the surface of the hot metal.

The technical execution of the process takes place in the hot metal desulfurization unit. The hot metal is delivered in tube wagons on tracks. The tube wagons have a hole on the top, through which a lance can plunge into the liquid hot metal. Through this lance the carbide powder is blown in for some time.

2.2 Way of the hot metal

The letting out of the liquid hot metal out of the blast furnace is called tap. At a tap, the hot metal flows through a channel out of heat resistant material into the prepared tube wagons. Usually one tap fills more than one wagon. Each tap receives a unique number within its data. The blast furnace is also recorded in the data. The hot metal analysis, listed in the data, are made from the channel sample, which is taken from the hot metal flowing in the channel. Only the sulfur content is then again determined within the hot metal desulfurization unit. The temperature within the channel is also determined. On the wagon the weight of the filled-in hot metal is measured. The tubes are usually not filled to the same extent. The heat resistant material inside the tubes becomes thinner over time. On the other hand, depositions similar to slag grow inside. These two facts make it difficult to determine the exact amount of hot metal in the tube. The depositions can also have influence on the content of sulfur in the hot metal. The shape of the depositions influences the circulation of the liquid metal and, therefore, also the efficiency of the desulfurization process.

After the trap the tube wagons get on tracks to the hot metal desulfurization unit. Another sample of the hot metal is taken there to determine the sulfur content. The temperature is also measured. Afterwards the carbide powder is blown into the liquid metal through a lance. The target sulfur content is defined by the steel plant. The calculation of the required carbide amount is executed at the control desk of the hot metal desulfurization unit.

After finishing this treatment the tube wagon is brought to the steel plant without taking further samples. Once there the hot metal is poured into pans. The slag is taken out of the pans and the total weight is determined. Also the net weight of the hot metal without slag is determined by adding together the weights of the single pans. From one pan a sample for the determination of the achieved sulfur content is taken. The result of this sample is added to the collection of data.

2.3 Controlling the process

There are few steps to get to a successful control of the desulfurization process. Some facts make it on the other hand difficult to build up an efficient control model for this process.

2.3.1 Chemical background knowledge

To control a chemical process, a certain understanding of the different chemical properties and reactions is necessary. It is important to know about the properties of steel and how to get a certain steel quality. The properties the product steel should have, are determined by the people who buy the steel and make other products out of it. It is then up to the metallurgists to find ways to produce steel with the demanded properties. One part of this task is finding the right chemical composition for the steel and ways to receive steel with such a composition.

In this case the problem is to get near to the target sulfur content with low costs. The sulfur mainly comes from the coal used for melting the ore. Too much sulfur in the hot metal leads to poor steel qualities. Therefore it is reduced by adding a carbide powder to the hot metal. This additive reduces the sulfur and falls out in the slag.

2.3.2 Monitoring the relevant influences

To get a better understanding of a process it helps to see the parameters that might be important within the task. During the control of the process it is necessary to have the relevant input values to determine the needed control values. For these two reasons a great number of possibly relevant information is collected. Among them are weights, temperatures, chemical analysis, and some indicators for influences that cannot be measured.

2.3.3 Analyzing the data

The collection of information is evaluated with statistical methods. This shows up the relations among the collected data. In operation different control models can then also be analyzed and their results can be compared with the actual results of the process itself.

2.3.4 Process models

From the analysis of the data and models new or improved models can be build. From the known importance of the different influences and their correlation, it is possible to built up models that might have a similar behavior as the process itself.

2.3.5 Improvements

With respect to older models and their comparison to the process behavior new improved models can be designed and implemented. All the field knowledge and results from statistical evaluations as well as the experience of the experts is necessary to build models that come close to reality and are stable and elegant solutions.

2.4 Difficulties

Most of the difficulties in controlling the desulfurization process concern the influences.

2.4.1 Influences not measurable

Many relevant influences cannot be measured. This is the case mostly because there is no possibility to position sensors in the concerned area. For instance, it is not possible to get much information from inside the furnace. Other very important influences concern the depositions within the tube wagons. No significant information can be received about the depositions inside the tube wagons, their influence on the circulation during the desulfurization process, and on the influence on the sulfur content of the hot metal.

In these cases it is only possibl to find models to estimate correcting factors for the main models. Such influences make it difficult to evaluate other influences or models.

2.4.2 Unknown importance

The above-mentioned difficulties in the evaluation processes also lead to uncertain importances of influences. The importance of single parameters are basis for the representation within the model. Therefore, evaluations over long periods and the expert's experience are needed to build working models despite this lack of known importances.

2.4.3 Time dependent influences

Some influences change over time without the possibility of monitoring these changes. This leads to deviations of the control results according to these changed influences. Here again the depositions inside the tube wagons play a major part. These changes are continuous for each single wagon. Other changes over time are unmeasured chemical influences that can occur at the supply of single additives. These are sudden changes that can only be detected by their effect on the control result. Another sudden change is the removing of the depositions inside the tubes. This one is known to the operator, or can also be derived from the cycle time of a certain wagon.

All these time dependent influences can be included into the control model in separate submodels that try to give a good estimation for these changes over time. The number of such separate models should be kept small. They weaken the main model and its evaluability. If these compensation models work well and give exact predictions for the time dependent influences, and they can improve the fitting of the central model and its stability significantly.

3 Fuzzy logic

In this chapter a short introduction into the principles of fuzzy logic is given, insofar as it is needed for the understanding of this work. Further insights into the way fuzzy logic works are included in the description of the FLIP++ library or can be seen in [18, 14]. For fuzzy logic in artificial intelligence see [8].

3.1 Simple idea

An advantage of fuzzy logic is its simplicity. It is a simple idea that works and is applicable to a great number of problems. It is easy to understand even without extended theoretical education. There is almost no understanding necessary of the principles behind it to apply this method to real life problems. Fuzzy logic is almost easier to use than many word processing programs or spread sheets. Everybody who is capable of playing adventure games or strategy games on computers should have no problems to use fuzzy tools for simple problems he is usually concerned.

Fuzzy logic brings computers very near to the human way of information processing. Many things people think can be easily put into a fuzzy system. People do not have to learn how a computer works, or learn any computer language, or find their way through a huge number of buttons or possibilities. They can more or less only write down their problem description and for some unexplainable reason the computer pretends to understand them and can evaluate their thoughts. That is the way computers should be operated, and not the other way round. This close relation to the human way of thinking makes it a lot easier to accept a system and then to apply it.

3.2 Computation with words

Most computational usage is done with numbers. On the other hand, a great deal of human thinking about certain problems is done with words. Many task descriptions are also written in words. Fuzzy logic is one approach in using words to solve problems with computers. It is not possible simply to refer to single items of the real word with their original name, the properties of these items can also be describe with words. Internally these properties are represented through membership functions that can be easily built and with which exact calculations can be executed.

The relations among the items and their properties and the relations within a set of such items can also be represented in words. Rules are used mainly for this purpose. Mostly Rules in form of *IF THEN* rules are used. This is an easy to understand model of relations among items.

It is also possible to use items represented by membership functions for calculations. It is just necessary to use implementations of therefore adapted operators. Fuzzy logic allows the use of rules and arithmetic operations at the same time. It is not a problem for a system to evaluate such a mixture of approaches.

3.3 Compatibility

Even real numbers can be included into a fuzzy system either by giving them a representation with a membership function or by implementing the evaluation steps in a way that they accept such real numbers. This can be used to build an extension for calculators that also can handle vagueness or uncertainty.

Fuzzy logic can easily be combined with other artificial intelligence methods. It is no problem to build connections to neural nets, genetic algorithms, or expert systems. One purpose of such combinations can be that fuzzy logic is used as interface between various other artificial intelligence approaches like in [4]. Another frequent use of such combined or hybrid systems is adding learning abilities to the widely applicable fuzzy systems. In this field there is still a lot to be developed, analyzed, thought through and tried out. When more experience with self training fuzzy systems is available fuzzy logic will make a giant step into a by far greater number of applications.

3.4 Typical fields of application

If other more conventional methods of information processing fail, or too expensive, or inefficient, fuzzy logic allows to build up models of a process from expert knowledge and experience even if there is no exact theoretical model. Fuzzy logic can handle imprecise and vague knowledge. If the information comes from human observation or from not very reliable sensors, it is possible still to handle this input data in a correct way. Precise inputs are, of course also not a problem to handle.

A widespread field of applications lies within fuzzy control. It is quite

easy to build up control schemes for small machines like cameras, washing machines or vacuum cleaners. A far more positive effect can be reached in controlling complex tasks in industry. Many processes have no precise model or cannot be monitored well. Both situations make it hard to control these processes with conventional methods. Here fuzzy logic can help out and supply efficient and reliable solutions. Some applications in this field concern the cement industry; or many also come from the steel sector, either from steel plants or from furnaces.

Within transport systems companies also try to use fuzzy logic to control vehicles. One of the first famous applications was the control of the subway system of Sendai. Now ways to build some kind of driving assistance for cars are being tried. This is a much more complicated task. Fuzzy logic is on way to reach this aim and get reliable solutions.

A great number of expert systems use fuzzy logic for their derivations and calculation. Many of them are medical applications. Fuzzy logic as part of such a system can improve the practical applicability through its handling of imprecise knowledge. Experts systems that handle imprecise knowledge in this way and are able to learn and become experience, can be at least as competent as inexperienced experts. They are probably more stable and have higher availability.

A growing field of applications is concerned with decision analysis and decision-making. Many models for making decisions are based on rules. They are using vague input information or uncertain rules. This is an ideal field for fuzzy logic to be applied. One example from this group is scheduling. Here we often have the problem of a huge search space that makes it almost impossible to find optimal solutions with any method. If we have to deal additionally with soft constraints in the application, this almost requires the use of fuzzy logic. The FLIP++ library was developed in such a problem. It was designed to do the evaluation of hard and soft constraints within a scheduling application for a steel plant.

3.5 Fuzzy Sets

Fuzzy sets are sets of values that have membership values from 0 to 1. The difference to ordinary sets is that these membership values can also lay between 0 and 1. This means that the concerned element is partially a member of the set.

- A membership value of 0 means that this element does not belong to the set.
- A membership value of 1 means that this element is member of the set.
- A membership value of e.g. 0.3 means that this element belongs to the set with a probability of 30%. With a probability of 70% it does not belong to this set.

Fuzzy sets can be described as a list of elements together with their membership degrees. This approach is used within the FLIP++ library and many other fuzzy tools. These sets can be seen as point lists representing a membership function. This point list can be used to give an illustration of the distribution of the membership values.

A typical shape of such a membership distribution has some value or interval where the membership degree is 1 and above and below this the membership degrees fall down to 0 in a more or less steep descent. For the rest of the elements the membership remains 0.

Another way to describe a fuzzy set is to define a membership function as arithmetic function. This function gives every possible value within a defined range a membership degree from 0 to 1. Internally such a membership function can also be represented as point list that holds the edges and characteristic points.

3.6 Linguistic terms

Each fuzzy has a name to refer to it. This is the name of a property of an item. The fuzzy set is the internal model of this property, which is used for the evaluations of the fuzzy system. A name of a property will be stated in linguistic terms in the following:

Linguistic terms can be: *small*, *medium*, *big*, *hot*, *late*, *deep*, *dark*, and so on. They are all adjectives that describe a single property like the size, temperature, altitude, etc., of an item such as a person, a piece of wood, the air outside, or a mountain.

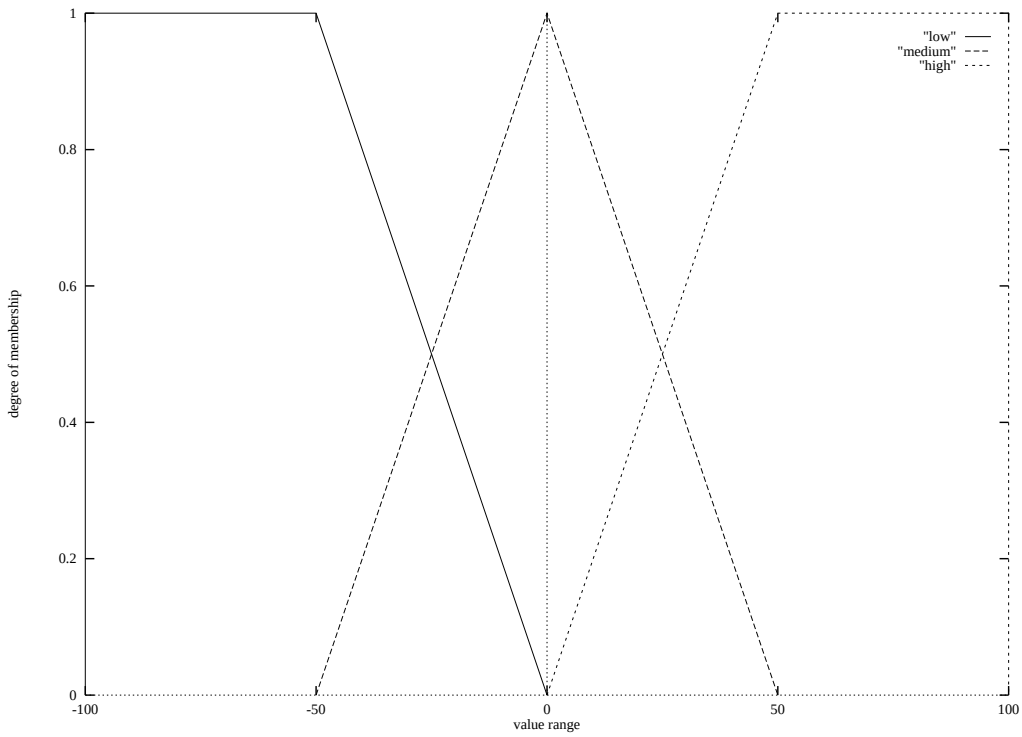

Figure 1: Linguistic variable with linguistic terms and membership functions

3.7 Linguistic variables

The properties that are of interest for a task description are represented by linguistic variables. These are variables that can take a certain value represented by a number of fuzzy sets as shown in Figure 1. Linguistic variables can, for example, hold the size of a person. The single possible values this variable can take are predefined at the declaration of this variable. This declaration is similar to the declaration of any other type like integer, real, or string. At the declaration of a linguistic variable, the possible linguistic terms are declared. And for these terms, the corresponding fuzzy sets are defined. With this, the allowed value range is also defined.

It is possible to create a completely different type for each linguistic variable. If there are similarities between a number of variables, it is useful to create types for such variables that can then be used for the declaration of new linguistic variables whenever needed. Such types are defined by parameter sets within the FLIP++ library. They are not supported by many fuzzy tools, but they are very helpful when building up larger fuzzy systems. What information such a parameter set contains can be seen in the respected figure in the library description.

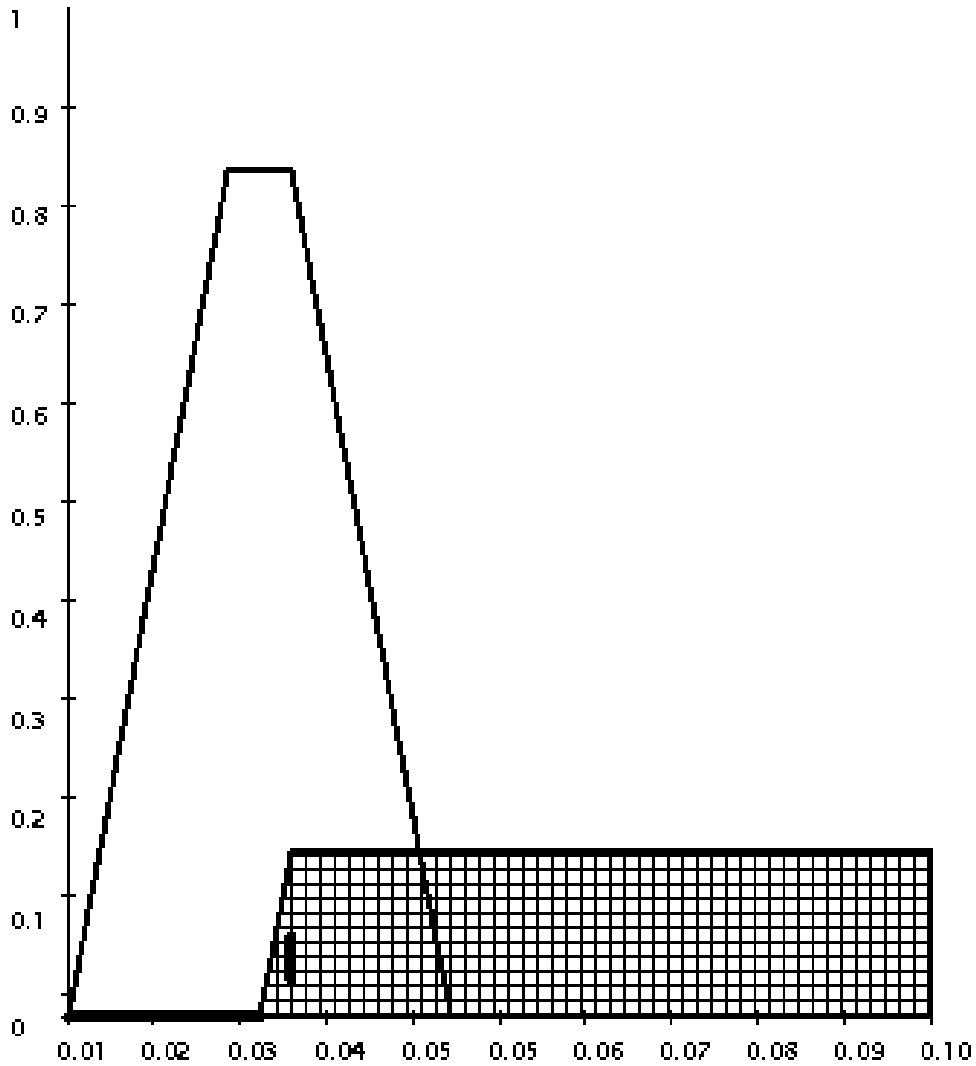

Figure 2: Set of instanciated membership functions of a linguistic variable

3.8 Instanciation

After declaring a linguistic variable it still does not hold any value. A linguistic variable can be set to any value contained in any of its linguistic terms. In each fuzzy set of the linguistic variable this certain value is searched and each linguistic term is assigned to the degree of membership of this value. Each fuzzy set is then additively or multiplicatively cut off at this height. The meaning of such an instanciated linguistic term is that it is fulfilled to the degree of its height. Such an instanciated linguistic term can be seen in the Figure 2. This step is usually called fuzzification.

3.9 Rules

The most commonly used form of expressing relations among linguistic variables is to write rules. Such rules in most systems have the form of *IF THEN* rules.

- Between *IF* and *THEN* the premise holds the conditions for this rule to fire. The premise holds one or more clauses. These consist of a reference to a linguistic variable followed by *IS* for better readability. As last part of such a clause the linguistic term that is of interest in this clause is named. This linguistic term can additionally be influenced by a linguistic moderator in form of adverbs such as *VERY*, *ABOUT*, *ABOVE*, *UP TO*, or *NOT*.

  Such a clause means that the degree of belief of this single, eventually moderated, linguistic term of the concerned variable is of interest and should be returned at its evaluation. The whole premise is the collection of such clauses. The single clauses are connected by aggregation operators. This can be *AND* or *OR*. The exact meanings of these operators are defined in the selection of the operators for the aggregation step.

- After *THEN* follows the conclusion of a rule. This is a single clause containing the resulting linguistic variable.

- Eventually an expression for the belief in this rule can be added. This can be a real number or again a linguistic term. The certainty of a rule is a way to differ between certain rules and relations that are only estimated or of less reliable origin or should have less influence to the result. In the following we will use this expression in the belief of a rule through adding *WITH (CERTAINTY)* and a real number from 0 to 1.

  - 0 means no belief in this rule and thereforee no influence into the rest of the evaluation process of the fuzzy system. Such rules are normally not added to a set of rules at all. For temporally disabling a rule without completely removing it, the certainty can be set to 0.
  - A value between 0 and 1 means a partial belief into the rule or a reduced importance of it. Such certainty values are helpful in building up hierarchies with inequally important sub-trees.
  - 1 means that there is no doubt in this rule and it is seen as absolutely true.

A rule built according to the above syntax could be:

IF Appetite IS ABOVE average AND Cook IS excellent THEN Time-for-Dinner IS long WITH CERTAINTY 1.

- *Appetite*, *Cook*, and *Time-for-Dinner* are linguistic variables.
- *average*, *excellent*, and *long* are linguistic terms.
- *ABOVE* is a moderator.
- 1 is a real number for expressing the certainty of the rule.
- *IF*, *IS*, *THEN*, *WITH CERTAINTY*, and . are symbols of the language used for describing the task within the fuzzy system.

3.10 Evaluation

In this chapter the single steps of the evaluation of a set of rules are described. The way information is written into linguistic variables, the influence of different importance of variables, the aggregation of single clauses of the premise, the certification of single rules, the inference process itself, and defuzzification of the result are explained.

3.10.1 Fuzzification

Before the actual evaluation of the rules can start all referenced linguistic variables must hold a value. Either the linguistic terms are directly set or a crisp or real value is used.

Such a value is used as input for the fuzzification step. This step is executed for each linguistic term separately. For each fuzzy set or membership function, a membership value or instanciated membership function is calculated. These are then the actual values of the linguistic variable.

The support of a fuzzy set or membership function is the interval where the degree of membership is higher than 0. If the crisp value does not lay within the support of a fuzzy set, its membership value remains 0. Otherwise, the membership value of the position of the crisp input value is taken as membership value for this linguistic variable. The membership function is then cut off at the level of this membership value. This is done for each single linguistic term of a linguistic variable.

3.10.2 Application of importance

If linguistic variables with different importance are part of the task description, these differences can be included within the evaluation process. Either the importance is already respected within the instanciation step when determining the membership values for the variable, or it is respected as first step within the actual evaluation process. The importance can be applied with any aggregation operator.

A possible selection could be the minimum operator. This would have the effect that the minimum of the importance of the variable itself and the membership value of the linguistic term are used when evaluating the clause in which this linguistic variable is referenced.

If a fuzzy system allows this step, but it is not planned to actually apply it in the current task, it can be switched of by selecting some kind of default operator that does nothing and therefore has no effect on the evaluation process. This way is especially recommended when including of the importance should be let out only temporarily. Another way is to set the importance of the single linguistic variables to a neutral element for a certain operator. This would have the same effect. The setting to the neutral element could already take place within the generation of the linguistic variable.

3.10.3 Aggregation

This step evaluates the clauses within the premise of a rule. First each single clause of the premise is evaluated. Within a clause the membership value is taken from the linguistic term of the referenced linguistic variable. This value is returned as the result of one single clause. The return values of all clauses of the premise are then put together with the aggregate operators. The selected operators for the *AND* and *OR* conjunction are executed on the set of results of the single clauses.

If there are two clauses that should be connected with *AND* in the premise of a rule and the minimum operator is selected to perform the *AND* operation, the two membership values from the two linguistic terms of the linguistic variables in the premise are taken, and then the smaller one is selected by the minimum operator and returned as result of this step.

3.10.4 Certification

The result of the aggregation step can be connected with the certainty value of the rule. With which operator this step should take place is also defined in the selection of the operators for the single steps before the evaluation process is started. For the certification, the same operators as in the previous steps can be used.

A possible choice is the minimum operator. The lower value of the result of the premise and of the certainty is taken in this case. Therefore, the certainty value should be in the some range as the possible values for membership. This is mostly from 0 to 1.

After this step the evaluation of the premise is corrected by the reliability of the rule.

3.10.5 Inference

In the inference step the corrected result of the premise is put into the conclusion variable. The conclusion also has the form of a clause like in the premise. A linguistic variable is referenced and a single linguistic term of this variable is selected to be set. This linguistic term is then the value of the variable. The linguistic term is valid to the extend of the value of the certified premise. The exact setting of the membership value of this linguistic term is again executed by a fuzzy operator.

For this step, usually the maximum or the product operator are applied. In case of the maximum operator the value of the evaluation of the premise is simply taken as membership value in the first appearance of the respected linguistic term of this variable. If a previous rule fired already to the same term of the same variable, the higher value of the old membership value and the new value is taken.

With this step the evaluation of a single rule is finished. The premise is worked through and the result is written into the variable within the conclusion.

3.10.6 Accumulation

The single rules always influence on single linguistic term of a certain variable. The actual result of the evaluation process of the whole set of rules is the collection of these linguistic terms of the conclusion variable.

This is no real step because there is no operation to be executed at this point. It always has to be remembered that only all membership functions taken together are the value of a variable.

3.10.7 Defuzzification

Some applications like decision-making problems can skip this step. If it is enough to know which linguistic term of a variable has the highest membership value, the defuzzification can be omitted. In the evaluation process of complex tree structures this step can be neglected as well, at least within the intermediate levels of the hierarchy.

For the application in the field of fuzzy control the resulting variables have to hold crisp real values for controlling the task. To receive a result in the demanded representation, the collection of the single linguistic terms of the variables haves to be transformed into single real values.

For this purpose a defuzzy operator is selected that does the transformation work then. The operator selected for this step depends on the domain. Some selection criteria are mentioned within the description of the defuzzification operators. In most cases the center of gravity operator is a good chose.

With this step the end of the evaluation process is reached and the result in either fuzzy or crisp form can be used within the application.

3.11 Operators

For handling the linguistic variables a number of different operators are needed. The operators have to take care of the intended use of the linguistic variables and operators and also consider the internal representation of the variables. In many of the single steps within a fuzzy system it is possible to select fitting operators. Most fuzzy tools allow only in a few steps to select the way this step is executed. In many cases this is the better way to handle

a fuzzy system.

To allow a wider range of applications the user should have more influence on the way the tool handles the information. The kind of inputs vary among the fields of application. Therefore the treatment of the available information should also be adapted to the application.

For instance there can be importance or certainties at different levels of the application. This additional information should be included to build up a representation of the system in reality. There are different needs for fuzzy control or for decision analysis or for add-ons to calculators. The different needs should be respected within a greater freedom for the user of a fuzzy system to adapt his system to the needs of his task.

Four major types of operators are listed below.

3.11.1 Aggregation operators

The aggregation operators are used for *AND / OR* operations. This is the most-often considered use of operators in the field of fuzzy logic. The point here is to find a fitting representation of the *AND / OR* conjunction within rules. It should have the same functionality as the intention of the conjunction within the task description.

Other tasks within a fuzzy system, where these operators can be applied, are the connection of the importance of linguistic variables with their use within rules. Such a step is one way to differentiate between more and less important influences.

The next step after the aggregation of the single clauses of a rule is the application of the certainty of a single rule within the evaluation process. The certainty of a single rule holds the belief into the correctness of the rule. It allows us to differentiate between certain rules and rules that hold weak or only estimated relations.

In the inference step the operator determines the way in which the result of the evaluation of the premises is put into the linguistic variable within the conclusion of the rule. This step is mostly done with the minimum operator. Sometimes better results can be reached by using the product operator.

3.11.2 Classification of fuzzy operators

Three major classes of fuzzy operators exist. These are t-norms, t-conorms, and compensatory operators. T-norms are intersection operators. T-conorms are union operators. They are sometimes also called s-norms. The compensatory operators lay between the intersection and union operators. They are used to represent the human idea of *AND* respectively *OR*, which always includes more or less compensation in the aggregation of arguments. The intersection operators are mainly used to function as *AND* operator. The union operators are used as *OR*.

In each class also exist operators which can be adjusted by setting a parameter that helps tune the function to the current needs. The t-norms and the t-conorms are commutative and associative. Therefore they are easy to handle even with more than two arguments and within hierarchies of rules, where the order of evaluation should have no influence on the result.

Examples for t-norms are:

- Minimum
  This operator was already recommended in [17]. It is the simplest and most commonly used operator. It is shown in Figure 3.
- Drastic product
  This operator is the hardest t-norm. In cases where one value is 1, it returns the other one. In the rest of the possible cases it returns 0. Therefore, this operator is not continuous. It is shown in Figure 4.
- Algebraic product
  This operator performs the product of the input variables. It lays between drastic product and minimum. It is shown in Figure 5.
- Yager intersection
  This operator can be adjusted by a parameter to range in its functionality from the minimum to the drastic product. It covers the whole possible functionality for a t-norm.

Examples for t-conorms are:

- Maximum
  This operator is the opposite to the minimum operator. It is shown in Figure 6.

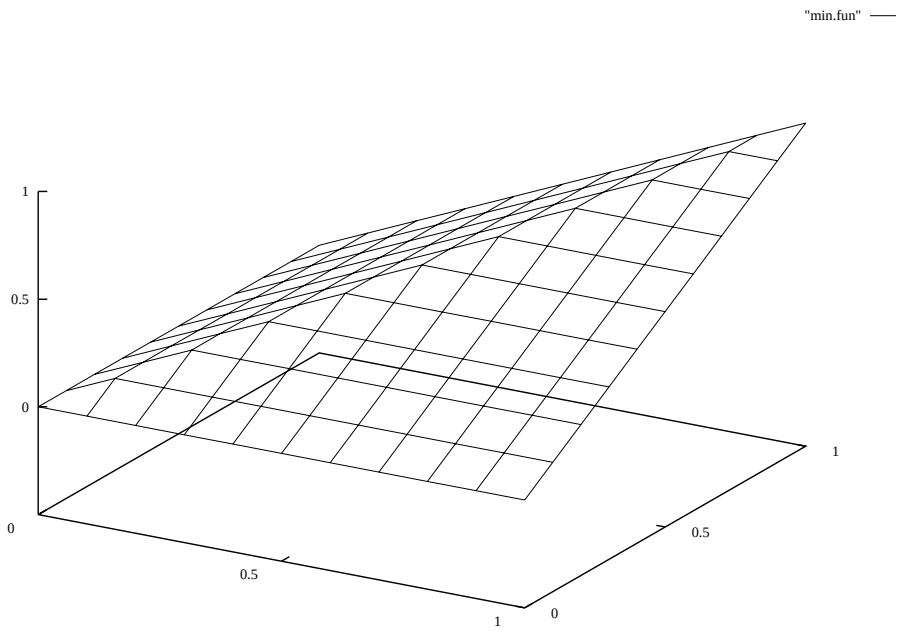

Figure 3: Minimum operator

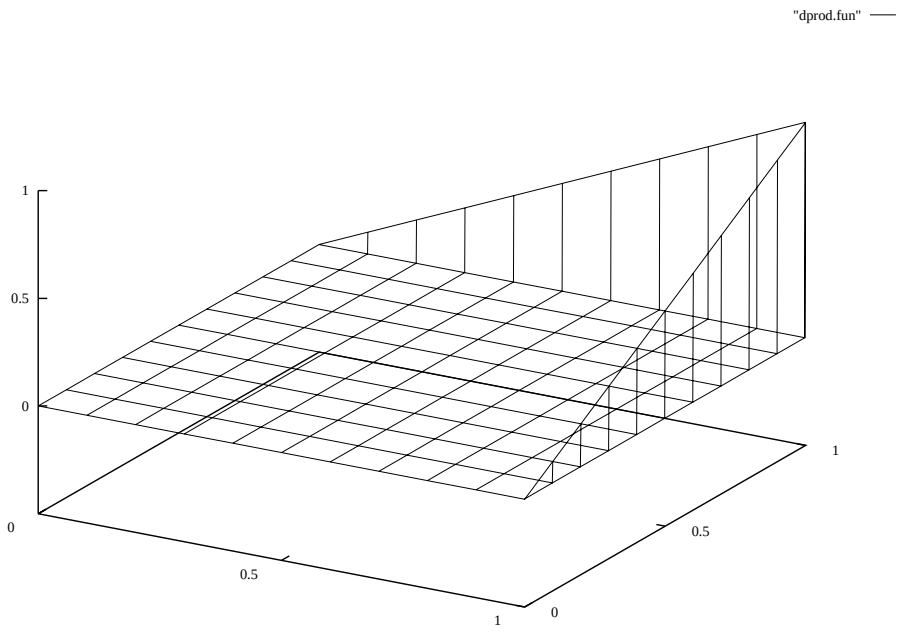

Figure 4: Drastic product operator

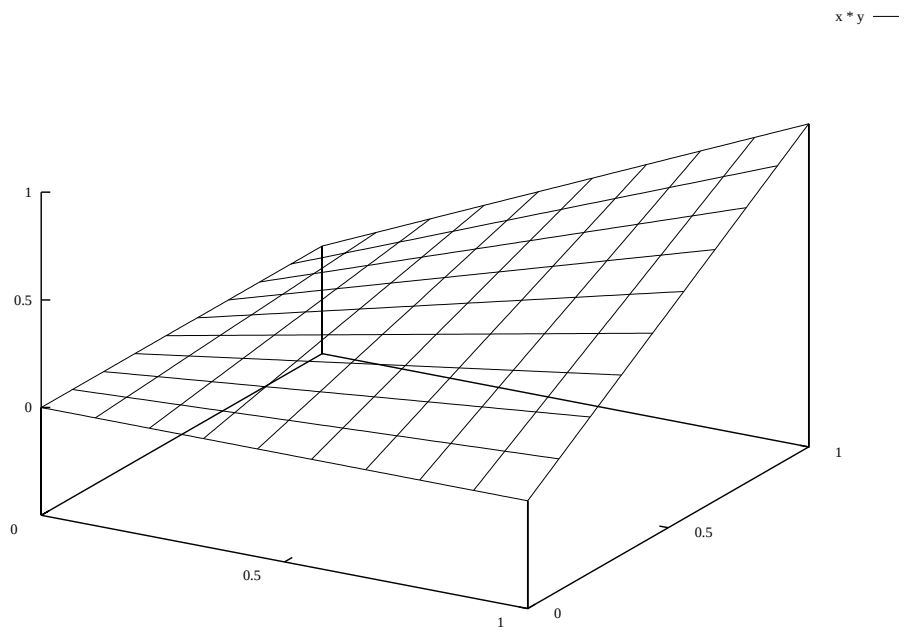

Figure 5: Algebraic product operator

- Drastic sum
  This operator is the softest t-conorm. In most cases it returns 1, and in cases where one value is 0, it returns the other one. This operator is not continuous. It is shown in Figure 7.

- Algebraic sum
  This operator is the counterpart to the algebraic product. It returns the difference between the sum and the product of the input values. It is shown in Figure 8.

- Yager union
  This operator can be adjusted by a parameter to range in its functionality from the maximum to the drastic sum. It covers the whole possible functionality for a t-conorm.

Examples for compensatory operators are:

- Arithmetic mean
  This operator returns the middle between two values. It is a compen-

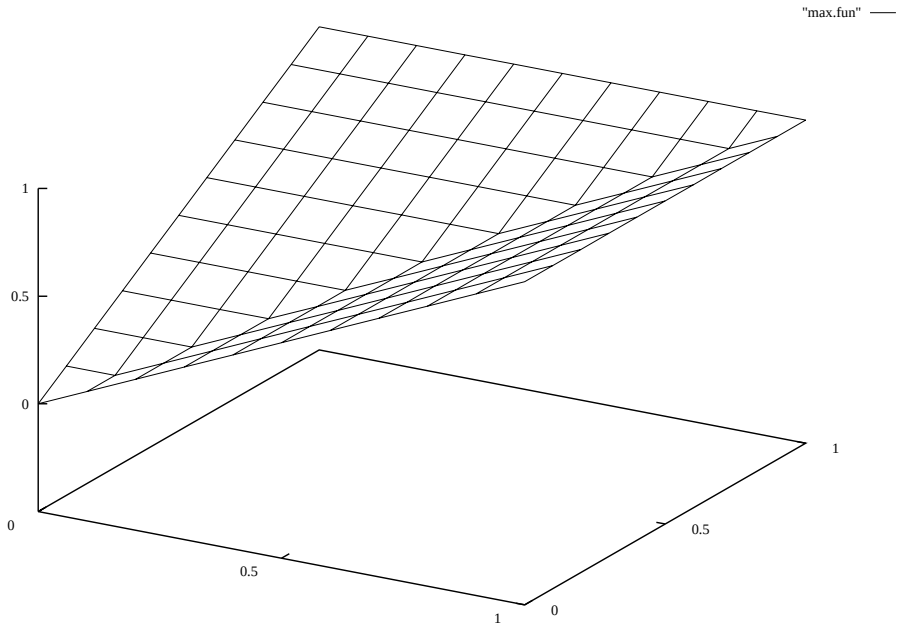

Figure 6: Maximum operator

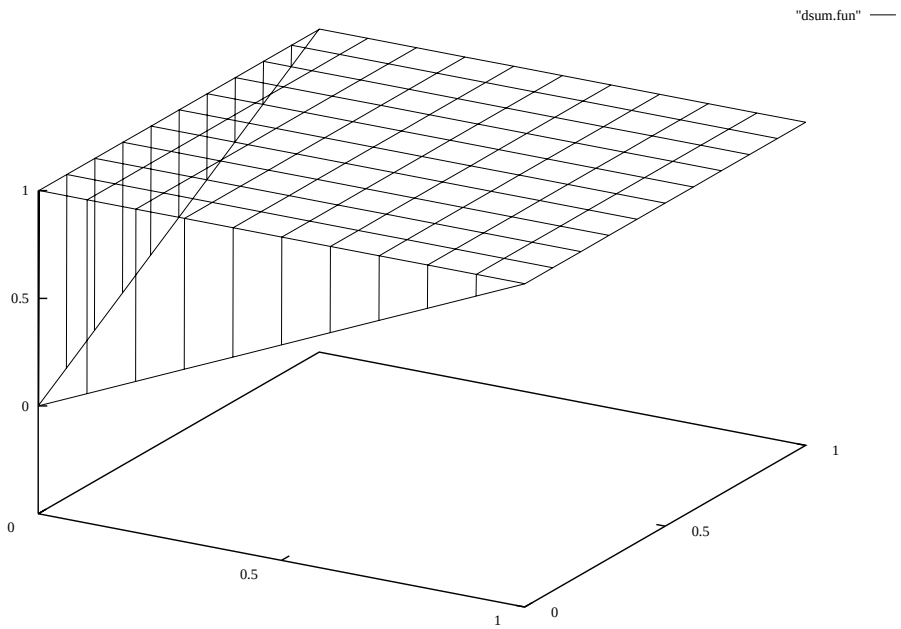

Figure 7: Drastic sum operator

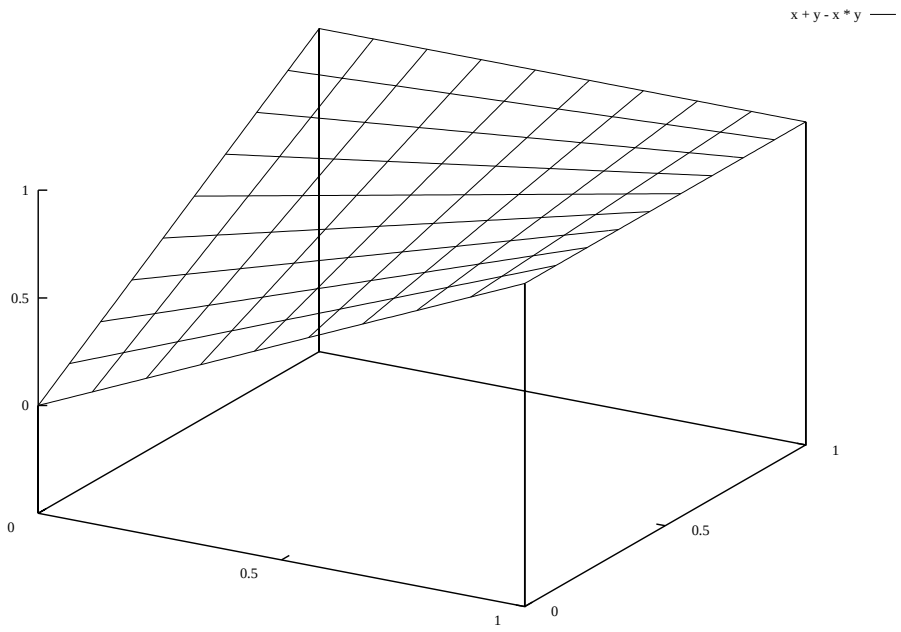

Figure 8: Algebraic sum operator

sation between two inputs. Other kinds of mean operators can also be used as compensatory operator. It is shown in Figure 9.

- Fuzzy and / fuzzy or
  These operators range from minimum to mean respectively from mean to maximum. Their exact functionality is again defined by an additional parameter.

- Gamma operator
  This operator ranges from algebraic product to algebraic sum. It covers thereforee parts of the t-norm functionality, the t-conorm functionality, and the area of the compensatory operators. It can be adjusted by a parameter gamma ranging from 0 to 1.

- Associative compensatory operators
  One such operator is shown in Figure 15. All compensatory operators are not associative. For that reason in [7] a new idea for associative compensatory operators was introduced. One such operator is used in the FLIP++ library and is named ACOtan.

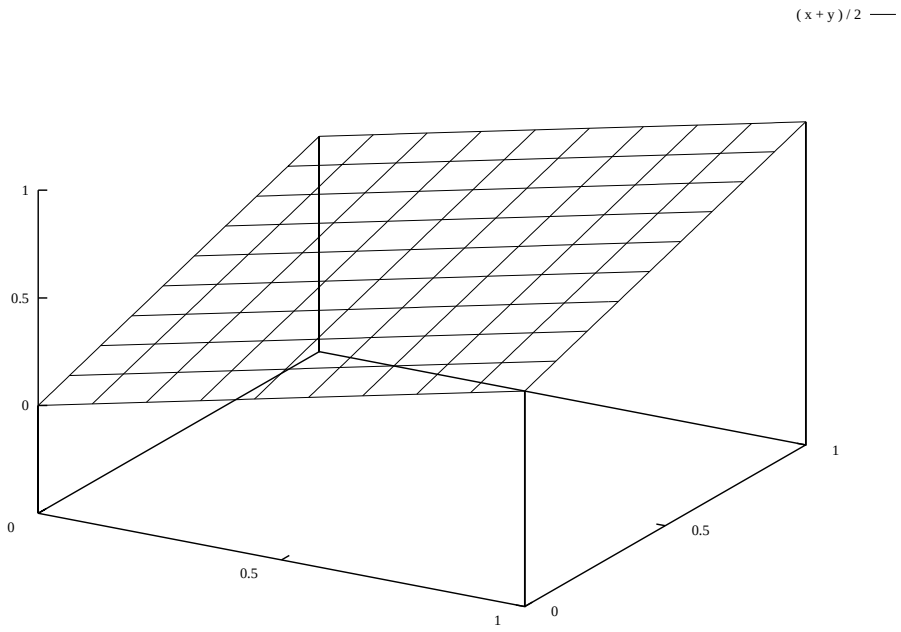

Figure 9: Arithmetic mean operator

3.11.3 Defuzzy operators

After the evaluation of the rules the results are contained in one or more linguistic variables. For some applications this representation of the result is useful. For other applications, especially in fuzzy control, the result should have the form of a real value. For that reason an operator is needed which translates the fuzzy set representation into a real number that can then be used for controlling the task.

There are a few methods to get one value out of a number of membership functions. Three commonly used operators are maximum, mean of maximum, and center of gravity.

- Maximum
  This operator returns the highest degree of membership as result. If we look at the number of fuzzy sets as point lists, it returns the x-value of the highest y-value among these points as result. If there is not just one point with highest amplitude, a way of determining one value to return has to be defined. There are two possible cases for such ambiguity.

- First the maximum membership value could be the same for a number of x-values or for an interval.
- The second case is that two membership functions have the same maximum height.

One method of determining an x-value to return is to take the first appearance of this maximum amplitude. This would have the effect that the real number representation would hold a value that is lower than the intended one in all ambiguous cases. The maximum method is mostly used in applications with resource limits. In such a case it makes sense to take the simplest method and accept deviations.

- Mean of maximum
  A maximum method that supplies the same functionality in unambiguous cases, as the above described standard maximum method, is the mean of maximum method. In cases of more than one maximum value this method calculates the mean of the current maximum values. This method returns a value that has less deviation from the meaning of the original linguistic variable. It needs a little more calculation time within the ambiguous cases.

  A drawback of this method might be in some cases that the mean of two maximums could lie in an area with a very low degree of membership or with a membership function value of zero. Whether this is a tolerable compromise, has to decided in respect to the application.

- Center of gravity
  The center of gravity method is in most cases the best-fitting method. It takes the areas below all membership functions and subtracts the areas that lay below more than one such membership function. From this area the center of gravity is then calculated.

  The drawback of this method is that it is relatively complex in comparison to the first two mentioned methods. Depending on the internal representation of the membership functions, it is more or less work to do. In usual applications this greater need of computational time should not be a real disadvantage. If this method has to be used within the training of a fuzzy system, the greater complexity can be recognized in the time needed for training. This is the only real drawback of this method.

3.11.4 Moderators

Moderators are operators that perform the translation of adverbs like very, about, above, up to, or not form their meaning within the task description into the internal representation within the fuzzy sets. These adverbs can be added to linguistic terms to change their meaning in a minor or major way. The respected operators change the membership function of the concerned linguistic term in the way expressed within the moderator term.

Such operators are mainly used where the task description comes directly from human field experts. If the expert uses such moderation terms in his task description it makes sense to apply such moderators also within the fuzzy system. Within automatically generated fuzzy systems such operators can be omitted. Such systems should have as a simple structure as possible to avoid getting a too fuzzy and loose meaning.

3.11.5 Algorithmic operators

Fuzzy logic can also be included into calculations. If there are arithmetic formulas using linguistic variables, operators are needed that can execute algorithmic operations on fuzzy sets. A way to implement such operators is to apply the intended operations pointwise for the different levels of membership.

It is also possible to mix up linguistic variables with real numbers. In this case the real number is pointwise applied with the corresponding arithmetic operator to the concerned fuzzy sets of the linguistic variable.

Such operators allow the build up of systems that can handle precise information with precise methods and as well handle vague information and uncertainties with appropriated methods. Within such a hybrid system all available information is treated with the best-fitting methods. The user does not have to pay attention to the type of knowledge he is using if the arithmetic operators are additionally defined for linguistic variables and the fuzzy system is well able to handle precise information.

4 FLIP++

FLIP++ is a fuzzy logic inference processor library. This C++ library was developed to perform calculations with uncertain data and priorities. It evaluates various types of rules, having different levels of importance, from a hierarchy of rules applied to uncertain or precise data. It evaluates only those parts of the hierarchy needed for the result. The result itself has two forms: first a list of membership functions, and second a crisp value for fuzzy control. The library also contains the whole functionality needed to manage one or more sets of rules on all data. It is built to be easily adapted to new applications. Due to its quite general concept of classes it is possible to include it, for the purpose of fuzzy computations, in other systems.

4.1 ∗FLIP++ project

FLIP++ (first introduced in [2]) is a fuzzy control library together with its graphical user interface from the InterFLIP++ library. Additionally it can be linked together with any other software where fuzzy calculations are needed.

Moreover, FLIP++ is part of the ∗FLIP++ library [12, 13] for real-world decision-making. This allows optimizing under vague constraints of different importance using uncertain data, where compromises between antagonistic criteria can be modeled. Typical application areas include scheduling, design, configuration, planning, and classification.

First, we give an overview of how to represent knowledge both qualitatively and quantitatively. Then we describe the relationships within the qualitative representation of knowledge using an example in the steelmaking process. In this section we also emphasize the concept of modularity in complex systems. Furthermore,, we show which operators FLIP++ applies and what purpose they are used for. Here we also introduce the associative compensatory operator ACOtan. How to combine already-represented knowledge follows in the next section. Afterwards a short description of the reasoning process is given. Finally, the key points of the chapter are summarized.

4.1.1 FLIP++ as part of a scheduling project

∗FLIP++ is composed of the following layered sub-libraries:

- FLIP++: the basic fuzzy logic inference processor library.
- ConFLIP++: the static fuzzy constraint library.
- DynaFLIP++: the dynamic fuzzy constraint generation library.
- DomFLIP++: the domain knowledge representation library.
- OptiFLIP++: the heuristic optimizing library; several heuristics have so far been implemented and tested, namely:
  - a tabu list min-conflicts repair based hill climbing heuristic,
  - a min-conflicts repair based genetic algorithm heuristic.
- CheckFLIP++: the knowledge-change consistency checker library that also allows fine-tuning of the configuration parameters of a problem.
- InterFLIP++: the graphical user interface for all other libraries.
- DocuFLIP++: the on-line documentation available separately for end-users, knowledge-engineers, and programmers, and accessible via the World-Wide-Web as HTML documents.
- ReaFLIP++: the reactive optimizer as an extension for DomFLIP++.
- TestFLIP++: the version control and test environment for the complete library set.
- SimFLIP++: the simulation toolkit library.

4.2 Representation of knowledge

A fundamental concept of fuzzy systems is the translation of qualitative knowledge, e.g. represented in human, language-like terms, into a machine-usable, quantitative representation.

4.2.1 Qualitative representation

It is often not possible to express knowledge in numbers or other forms usable on a computer. If the knowledge consists only of terms in human language, like *large* or *small*, it is impossible to do calculations with these terms. With fuzzy logic such a representation of knowledge is possible. A second advantage is that, like in the human usage of such terms, the exact meaning and

range of values is not the same but depends on the context in which the terms are used. The connection to a context can be made later, or the context can be adapted to new circumstances. Therefore, such qualitative variables do not have numbers as possible values, but only terms as mentioned above and are an ideal representation for vague or uncertain knowledge.

In this chapter the representation of knowledge will be shown in an example that is typical for the use of the whole ∗FLIP++ project, which FLIP++ is a part of. The example consists of constraints needed for scheduling a steelmaking plant. Here, three different types of constraints are used: first the chemical compatibility constraints, which will be shown in detail, second the temporal constraints for the representation of delivery dates that have to be fulfilled, and third the capacity constraints for human, machinery, and other common restrictions. Each of these three constraints consists of many different subconstraints. The chemical compatibility constraint, for example, consists of constraints for every measured chemical element. It is derived from the difference between the contents of an element in the old and the new load. Residuals of chemical elements from previous loads influence the quality of subsequent loads to various degrees, depending on the specific chemical element. An overall chemical compatibility constraint determines degree to which a certain sequence of two loads is desirable. This difference in content is also a qualitative variable with terms as value to express the difference. Here we will use the percentage difference for the chemical element aluminum (Al) as qualitative variable with the allowed terms: *negative*, *zero*, *positive*. Figure 10 shows the corresponding membership functions for such a representation where term *zero* is selected for graphical editing. We will later derive from this variable the constraint of the compatibility for aluminum by using rules.

4.2.2 Quantitative representation

For the evaluation of rules that contain qualitative variables, we assign a quantitative representation to the used terms so that ordinary calculation can be performed on them. Each term is described by a membership function that states the membership of this term for each crisp value in a range of possible values. The membership function can have values from 0 to 1. A value of the membership function of 1 means that the concerned crisp values fully represent the term. A membership function value of 0 states that the crisp values in these areas have nothing to do with this term. All other function values between 0 and 1 express a partial membership of the crisp value to the term.

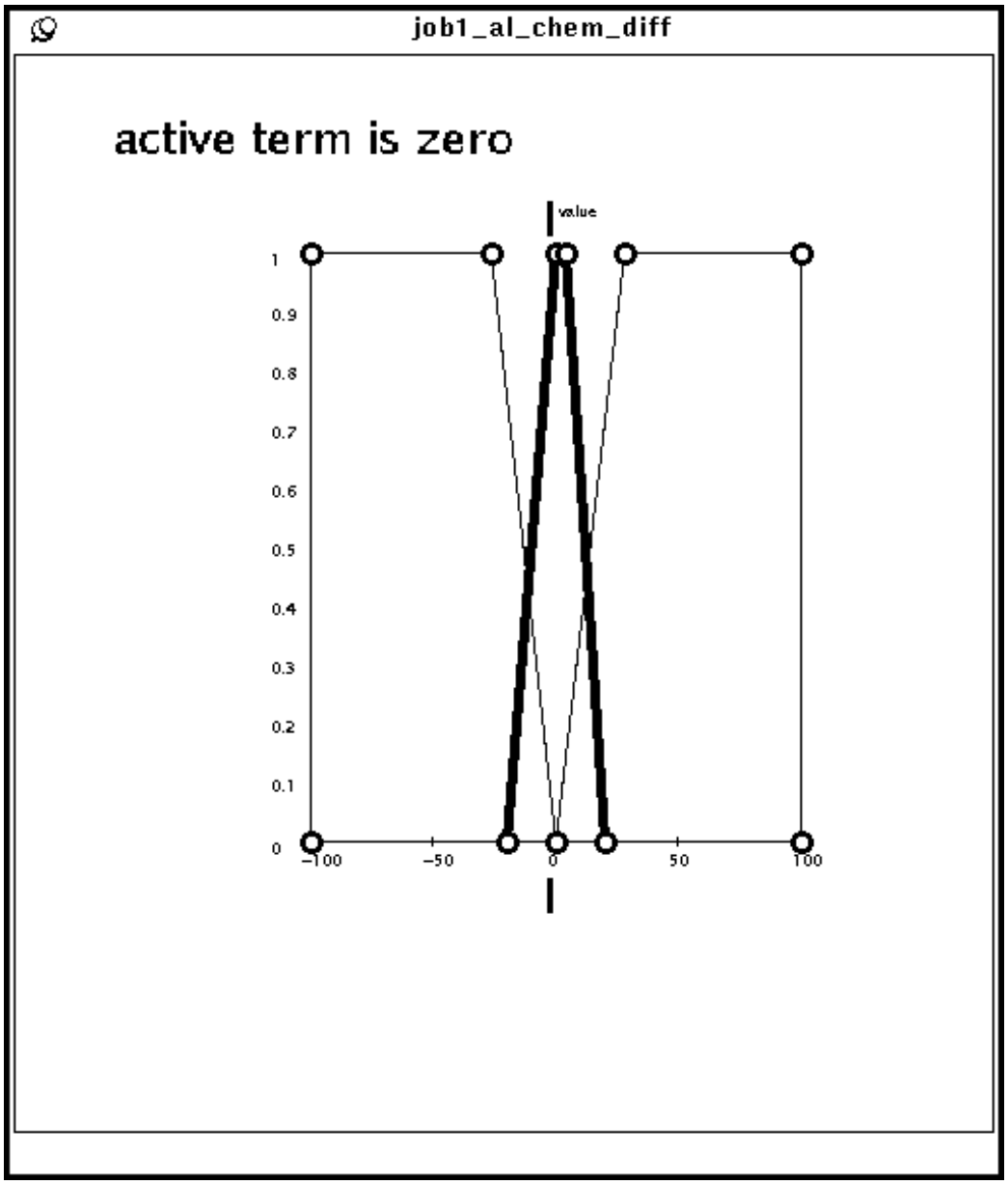

Figure 10: Membership function for percentage difference of aluminum

4.2.3 Parameter sets

The parameters that define a qualitative variable are shown in Figure 11. Parameter sets are used to ease the generation of variables by defining the number of terms, naming the terms, and determining parameters that are used to shape and position the membership functions. The figure shows the parameter set that can be used for variables representing chemical differences.

A parameter set generates a flexible structure. The membership functions can be directly manipulated according to the specific needs associated with the variable within the graphical user interface.

One idea behind this concept of parameter sets is that a parameter set provides a basis for one type of variable. Such a default variable can then be easily generated to provide a first-hand impression before integrating it into the system.

4.2.4 Instanciation of variables

When fuzzy inference *IF THEN* rules are used to evaluate some given knowledge, this knowledge is contained in variables. The variables on the condition part of the rule have to be instanciated to get a result.

One way is to directly give the variable a value by setting a certain crisp value or by attaching an instanciated membership function to it. The other way is to write rules that create an instanciation for this variable.

When a variable is evaluated, the system will first check whether the variable is already instanciated. If not, it will recursively try to find all rules that evaluate this variable.

4.3 Relations within qualitative knowledge

The representation of knowledge in the form of qualitative variables alone will not deliver results. To gain new knowledge from what we know, it is necessary to include links between qualitative variables by formulating rules. By applying these rules, new results are derived from instanciated variables and written into the dependent variable that is to be evaluated. The rules contain the known relations concerning two or more variables. All clauses on the condition part of such a rule consist of a variable and a term. Each clause is evaluated by determining the degree of membership of the quantitative representation of its term. All results on the condition part are aggregated and, in a further step, used to generate new knowledge on the conclusion part. The value of the conclusion part is obtained by applying the aggregated degree of membership of the condition part to its quantitative representation.

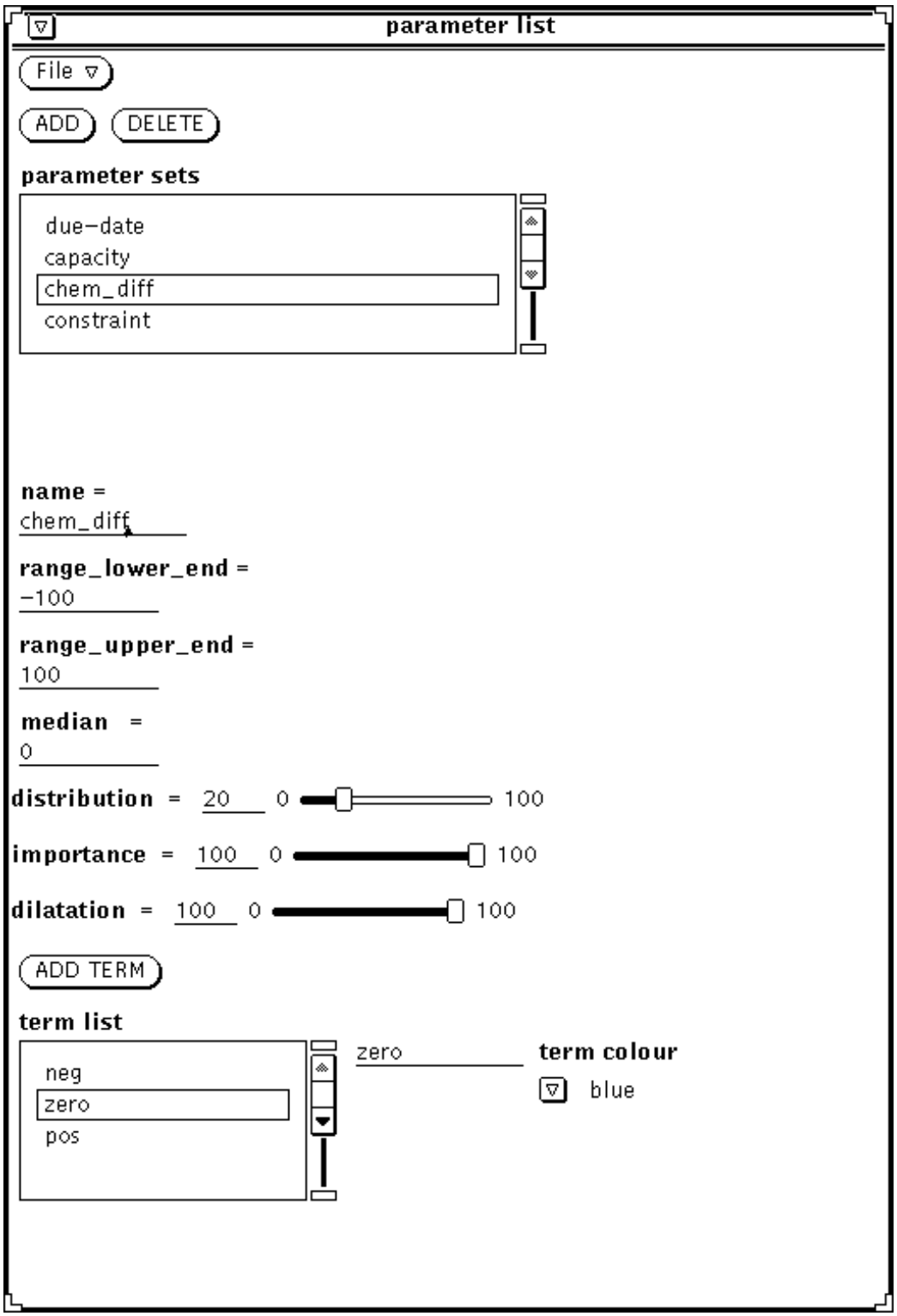

Figure 11: Set of parameters for membership functions in FLIP++

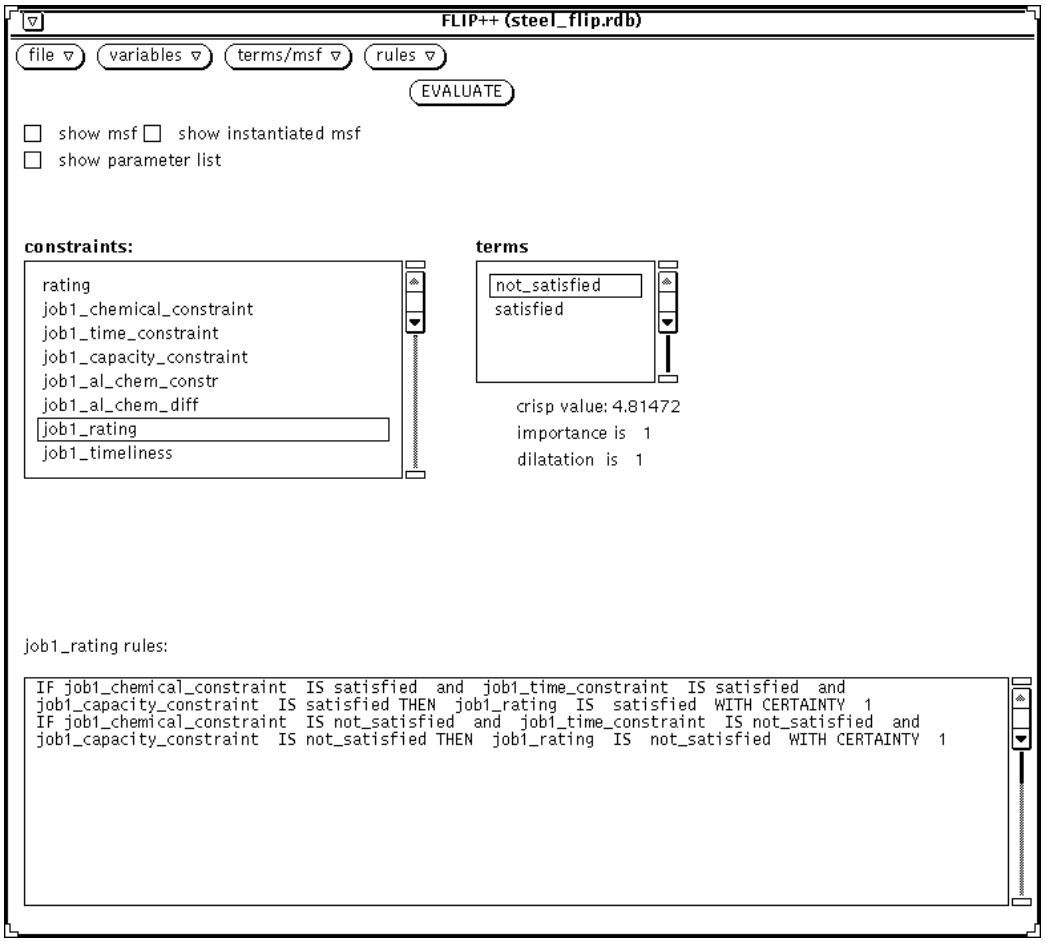

Figure 12: Graphical user interface with rules and variables

4.3.1 The relation in the example

The degree of satisfaction of the requirements for a certain chemical element is represented through a constraint. The aluminum chemical constraint depends in our example on the difference of contents in two consecutive loads, the chemical difference of aluminum. The same scheme is applied for other chemical elements relevant for the steelmaking process. In a further step, the specific chemical compatibility constraints of all chemical elements involved in one job, relative to its preceding job, are the input variables for the global chemical constraint of the same job. Subsequently, we also aggregate all constraints of a job - capacity, chemical, and time - to obtain an overall rating for the job that is exemplified in the rules window of Figure 12. Finally, a global constraint node variable for an overall rating of the production schedule is built by aggregating all individual job ratings.

Figure 13: General tree of constraints

4.3.2 Constraint tree - modularity

It is easier and safer to create a complex system of constraints in different modules and then combine them. A major aim of fuzzy logic is to bring computer systems nearer to reality and human thinking. This makes more sense than trying to reduce reality to a computer-readable form. The latter approach is almost always complicated and of uncertain success. It also has less to do with human problem-solving techniques.

Building smaller modules makes them easier to handle and verifiable with less effort. For more complex systems it is recommended to build a hierarchy of modules. This makes even intricate problems understandable.

In FLIP++ each level in a hierarchy has the same methods and is directly accessible and evaluable. The hierarchy is built up automatically through the implicit dependencies in the rules. It can be created explicitly by the use of smaller modules that can be combined to a more complex module.

The simplified steelplant example consists of many small modules, or set of rules, for the single constraints. All these are combined into several modules: the first for the chemical compatibility constraints, the second for the temporal delivery date constraints, and the last for the capacity constraints. Their aggregation builds the overall set of rules, or the highest level in the hierarchy, as exemplified in Figure 13.

To build up such a complex set of constraints requires the ability to combine small set of rules and variables, i.e., small modules, to a greater set of rules. This was one of the major aims of FLIP++. The fuzzy conjunction

operators of Figure 13 that allow this aggregation are illustrated in Section 4.4.

4.4 Operators

In the FLIP++ library we use fuzzy operators for different aspects of the evaluation of given rules and input information. The first place where a fuzzy operator is used, is the application of the importance of a linguistic variable within a rule. The importance of a variable can be used to differentiate between the linguistic variables at the level of the knowledge. In the next step the single clauses in the condition part of a rule are connected with help of an *AND* or *OR* operator. Then the certainty of the rule is applied to the result of this action. This is the second level where uncertainty of the given knowledge can be included in the evaluation. After that, the calculated membership degree is used on the linguistic variable of the clause in the conclusion part of a rule. The fuzzy operators we implemented so far in the library are: as t-norm the minimum operator, as t-co-norm the maximum operator, two parameterizable operators fuzzy_an and fuzzy_or, and an associative compensatory operator as described below. Figure 14 shows the operators we have included in the system.

An important requirement for operators is also derived from the use of hierarchies mentioned in Section 4.3.2. Reasoning with such variables and rules creates a need for associativity in the evaluation of rules. Otherwise, the order of single modules in each level of the hierarchy would influence the result.

Fuzzy operators are used for the aggregation of rules combining more than one variable to a variable of a higher level in the hierarchy. The abovementioned need for associativity is one of two major requirements for operators. T-norms such as minimum fulfill this requirement; and they are often used because of their pleasing mathematical behavior [6].

A second requirement is that it should represent an aggregation that is similar to the one that a human could use. Since the rules represent the human sight of a problem, the combination of such knowledge should be done in a way comparable to human thinking. Human understanding of operators like *AND* or *OR* is not a strict logical one, but lies between the logical *AND* or *OR*. Such aggregations can be done with compensatory operators. The drawback of such compensatory operators is that they normally are not associative [18], and thereforee, would not fulfill our requirements.

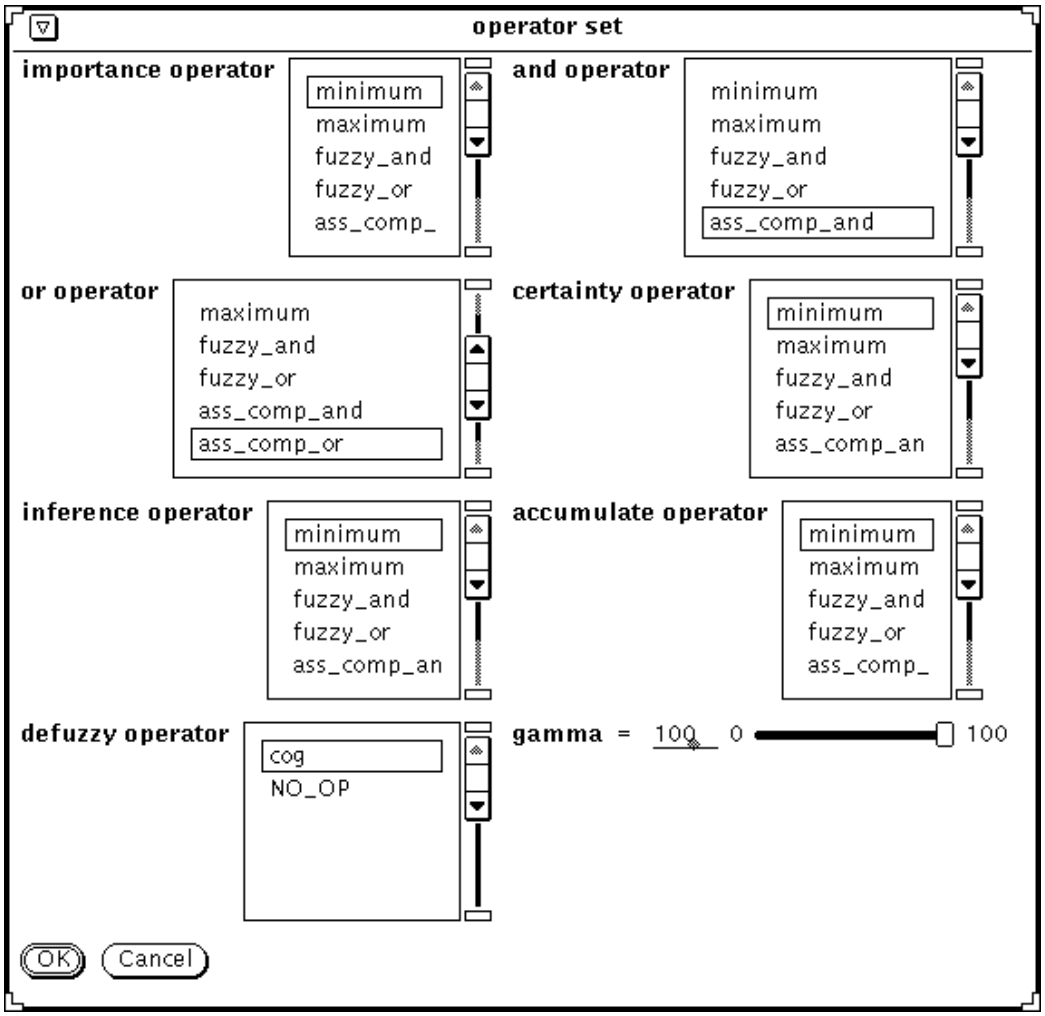

Figure 14: Default operator set in FLIP++

4.4.1 ACOtan - an Associative Compensatory Operator

A special operator that holds both above explained requirements is an associative compensatory operator designed according to a paper by Erich Peter Klement, Radko Mesiar, and Endre Pap [7]. From Theorem 1 they derive their Definition 1 for an associative compensatory operator with multiplicative generator f.

Theorem 1 [7] *Let* $C : [0,1]^2 \to [0,1]$ *be a binary operation on* $[0,1]$. *The following are equivalent:*

(a) The operation C is continuous on $[0,1]^2\backslash\{(0,1),(1,0)\}$, commutative, associative, non-decreasing in both components, and satisfies the cancellation law on $]0,1[$ and the boundary conditions $C(0,0) = 0$ and $C(1,1) = 1$.

(b) There exists a closed subinterval I of $[0,+\infty]$ and a continuous, strictly increasing and surjective function $f : [0,1] \rightarrow I$ with $f(\{0,1\})\epsilon\{0,1,+\infty\}$ such that for all (x,y) element $[0,1]^2\backslash\{(0,1),(1,0)\}$

$$C(x,y) = f^{-1}(f(x) \cdot f(y)).$$

Definition 1 [7] *Let $f : [0,1] \rightarrow [0,+\infty]$ be a continuous, strictly increasing function with $f(0) = 0$ and $f(1) = +\infty$. Then the binary operation $C : [0,1]^2 \rightarrow [0,1]$ defined by*

$$C(x,y) = \begin{cases} f^{-1}(f(x) \cdot f(y)) & if\{x,y\} \neq \{0,1\} \\ 0 & otherwise, \end{cases}$$

is called an associative compensatory operator with multiplicative generator f.

The duals of such operators are shown to be again associative compensatory operators, and a characterization of self-dual operators is given.

To create such an operator for FLIP++ with a tangent function as multiplicative generator, we took

$$f(x) = \tan(\pi/2 \cdot x)$$

and

$$f^{-1}(z) = \arctan(z) \cdot 2/\pi.$$

The characteristic plot of this fuzzy aggregation operator ACOtan can be seen in Figure 15.

4.5 Combination of already-represented knowledge

A major requirement for modularity is the ability to handle single parts of a used hierarchy. Thus it is possible to build up a complex system of qualitative variables and rules in small and verifiable steps that are easy to understand.

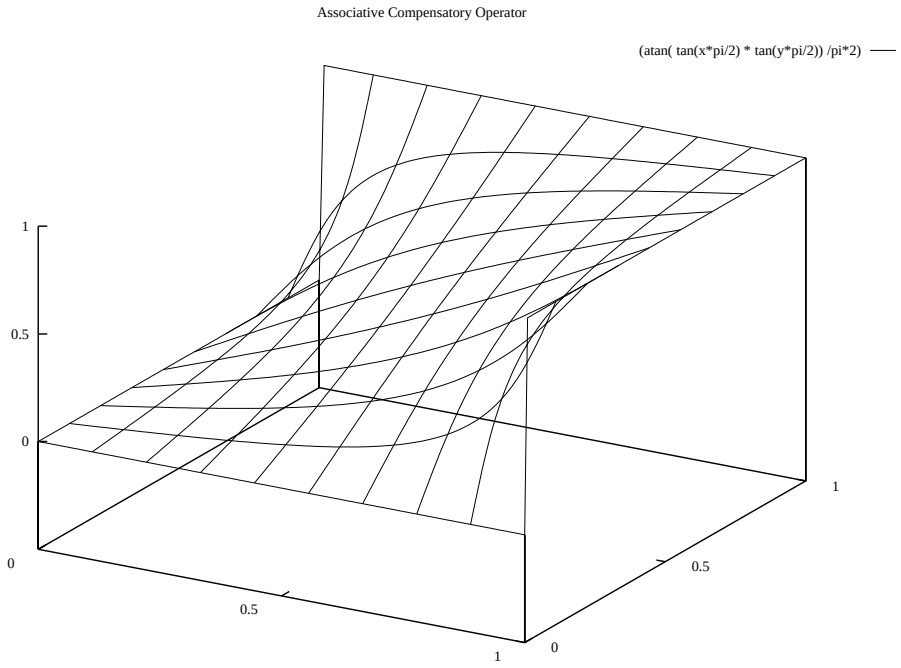

Figure 15: ACOtan - A associative compensatory operator

Two or more single sets of rules with the concerned variables can be combined by adding the variables and rules from one such system to the other. Alternatively, all can be put into a new module. When they are put together in one system, there is still no connection between the used variables and rules. The linking of the separated parts is done by adding new rules that express their relationships. This is the same step as only combining qualitative variables. In this way complex systems of variables and rules can be built. The only restrictions for the size of the module come from the computer system on which FLIP++ is running.

To compensate for these restrictions, it is possible to evaluate single modules. The connections between them are then expressed in a new module having new rules, with the output variables as the new inputs. In this case, it is important to evaluate the overall module again after all the changes and new evaluations in lower modules have been done.

4.6 Reasoning by evaluating knowledge

Reasoning in FLIP++ is done by evaluating the rules and input values whatever their representation is, whether crisp value or instanciated membership function. The rules represent the knowledge about the connections of the qualitative variables, which in turn represent the static knowledge of the domain in form of terms and their quantitative representation. Evaluation is done for one single variable, and all rules which match this variable are searched among all rules. For all variables used in the conclusion part of such a rule the same is done recursively, so that all the static knowledge and all the known relations for the resulting variable are searched in the hierarchy and then evaluated. During the evaluation, all the uncertainties about variables and the uncertainties about the relations are used in the calculations. The rules can be evaluated by applying fuzzy methods on the quantitative representations of the variables. As a result, a degree of membership is attached to each possible term of the resulting variable. For control applications, this representation of the result can be interpreted in different ways to get some crisp value for controlling machinery.

4.7 Conclusion

To make the FLIP++ library fit better for different applications, there is the possibility to add new operators. To combine it with other systems one can easily attach a new interface as a front end to it since it is clearly structured and well-documented.

Within the *FLIP++ library FLIP++ serves also as the basis for a scheduling application to solve shift planning problems. The FLIP++ library together with a graphical interface for DOS PCs is now used for determining bridge stability from optical damages.

5 Example

In the following parts of this work an example from the steel production is used to show the application of fuzzy control and genetic algorithms.

5.1 Problem description

A small part of a formula from the desulfurization process is taken to demonstrate the application of these methods. First a fuzzy system is built by hand according the model represented by a formula. In following chapters such a fuzzy system is trained with a genetic algorithm. Later ideas for using neural net for training such a system are given.

5.1.1 Variables

The variables used here are:

- A: This is the first input variable. It has a value range from 0.01 up to 0.1.

  Its distribution is shown in Figure 16.

- B: This is the second input variable. It has a value range from 0.001 up to 0.01.

  Its distribution is shown in Figure 17.

- C: This is the output variable that holds the result of the evaluation of the formula respectively the rule set. It has a value range from 0 up to 5.

5.1.2 Function

The model to be represented by fuzzy rules are included within the following formula:

$$C = \log(A/B)$$

Applied on the value range of the input variables it has a functionality as shown in Figure 18.

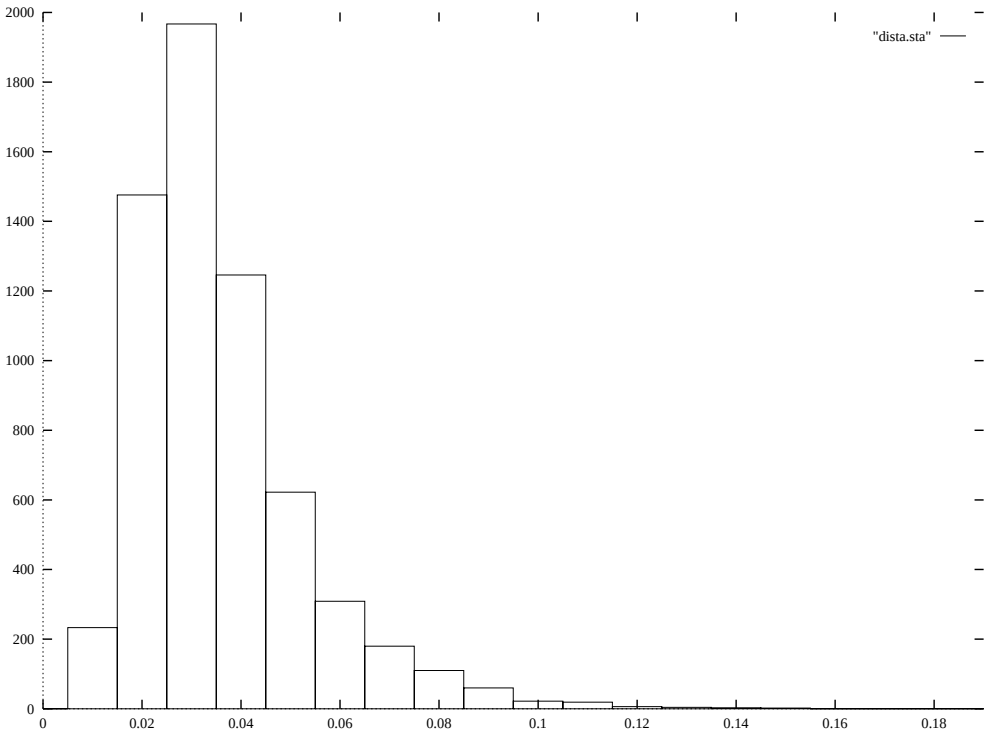

Figure 16: Distribution of input variable A

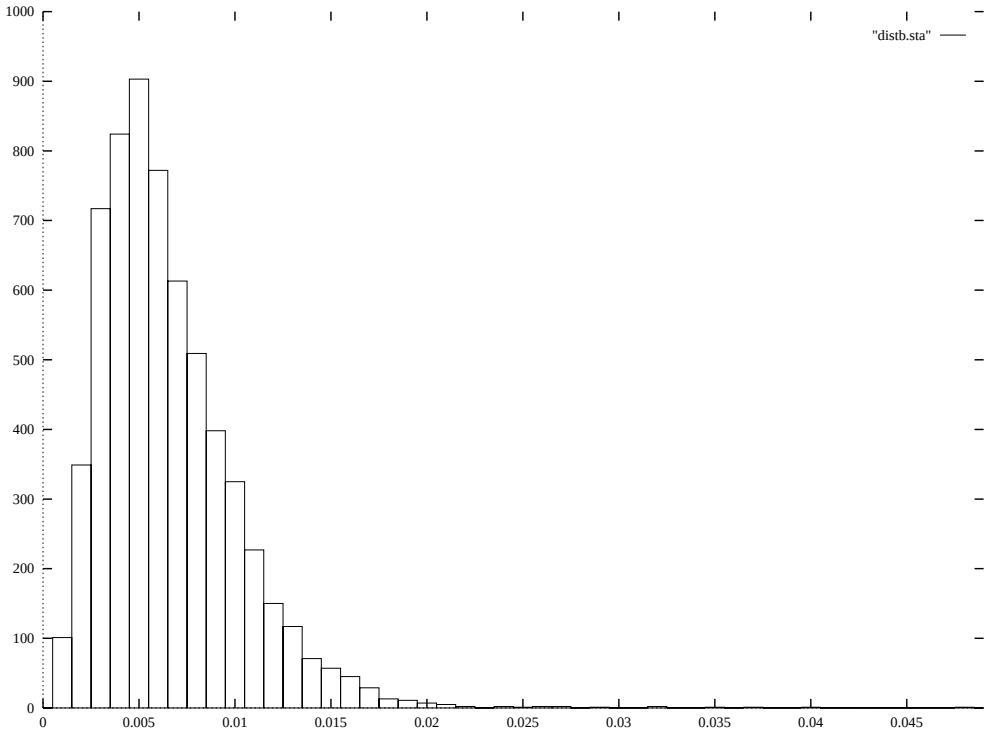

Figure 17: Distribution of input variable B

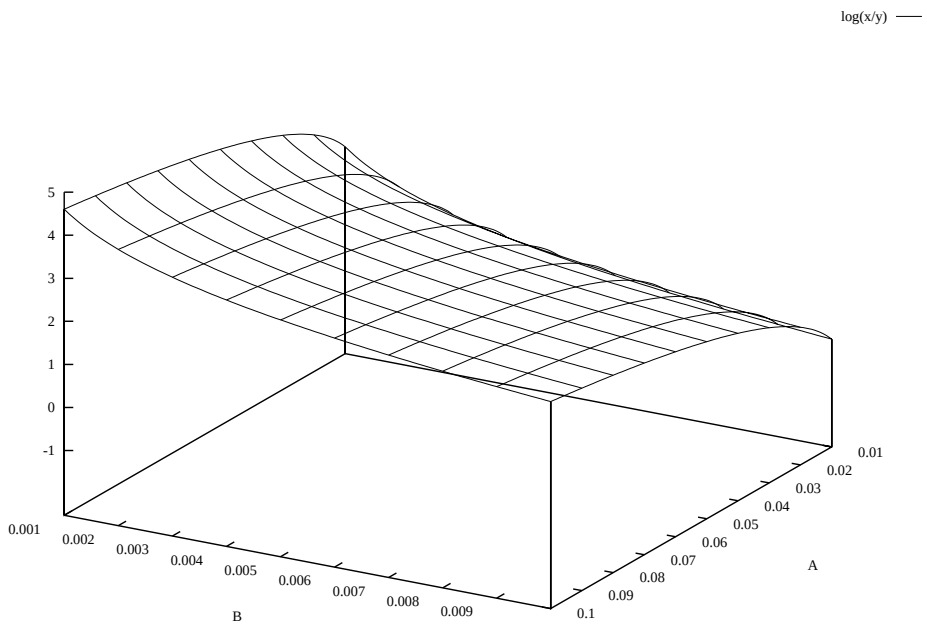

Figure 18: Functionality of $C = \log\ (\ A\ /\ B\ )$

5.2 Fuzzy solution

A solution for the above documented problem can look as described below. To build up a fuzzy solution the linguistic variables have to be defined first. After that the relation between the linguistic terms of these variables has to be fixed by defining rules.

5.2.1 Linguistic variables

The linguistic variables correspond to the variables of the problem description. Their names and their value range are the same. The definition of the linguistic terms follows the distribution of the actual values of the single variables. For all three variables the terms low, medium, and high are used. The membership functions for the single terms are defined according to the distribution of the possible values for the input variables A, see Figure 19, and B, see Figure 20. The terms of the output variable C are equally distributed as shown in Figure 21.

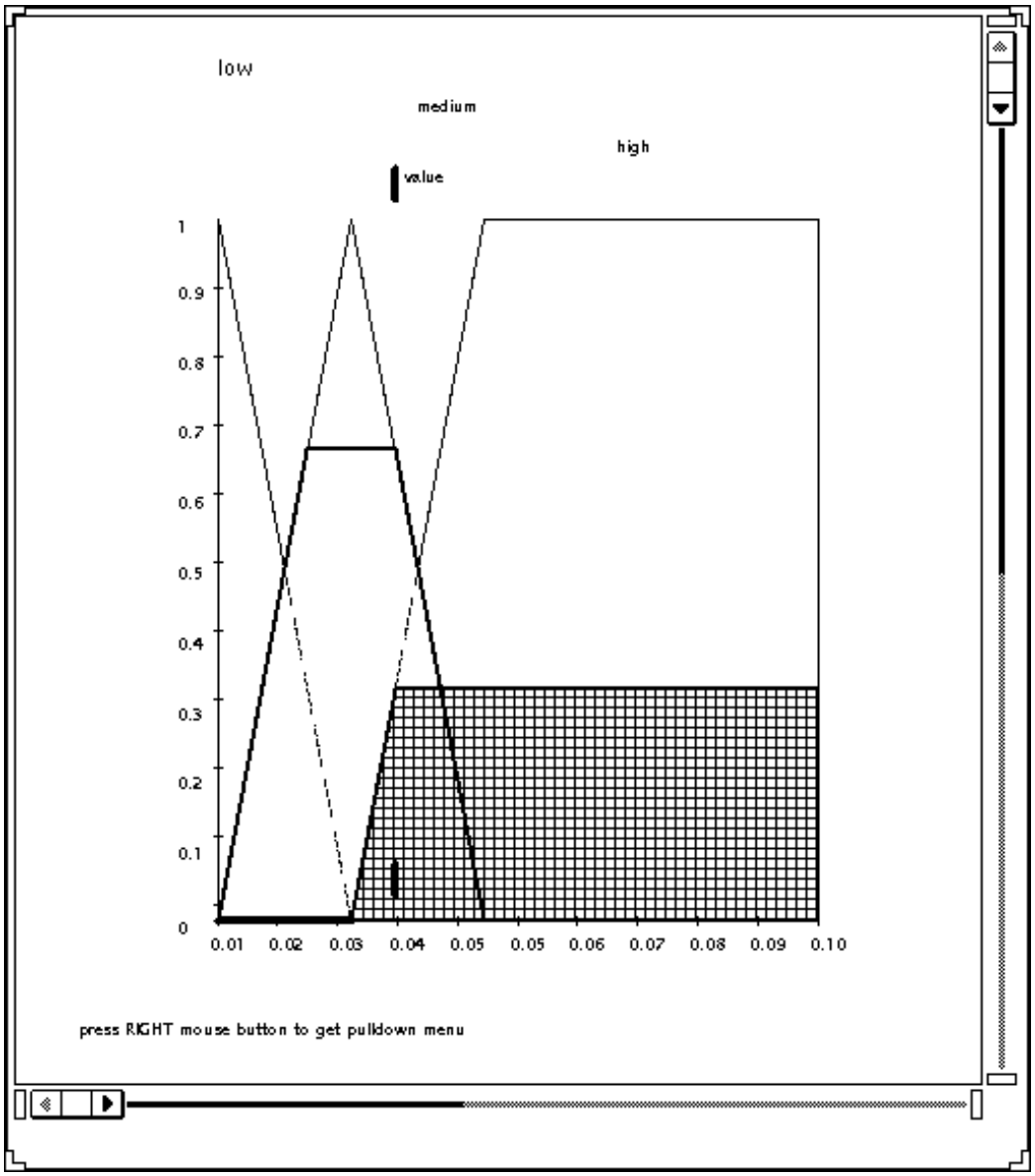

Figure 19: Linguistic variable A

5.2.2 Rule set

The model documented in the formula can be approximated by fuzzy rules. These rules try to imitate the functionality of the formula by applying *IF THEN* rules on the linguistic variables.

The linguistic variables consist of three linguistic terms each. For a definition of the whole value space, nine rules would have been necessary. The

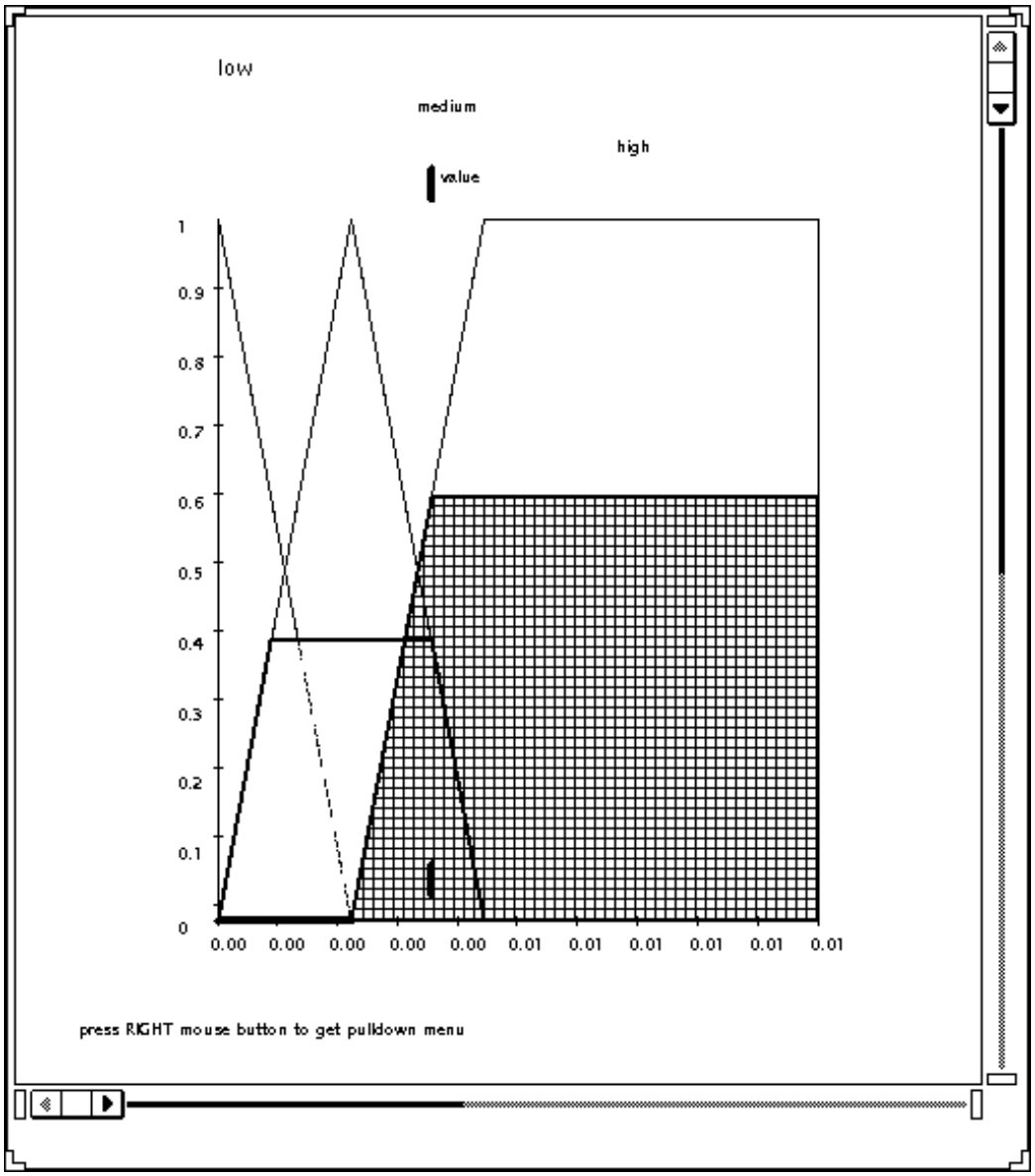

Figure 20: Linguistic variable B

membership functions of the linguistic terms overlap their neighbors. Therefore the whole value space can already be covered with four rules in this case.

The number of the rules does not say anything about the quality of the rule set anyway. Here are four rules defined for a first full coverage of the value space. For each combination of the extreme input variable terms a reasonable value of the output variable is searched and written into an *IF*

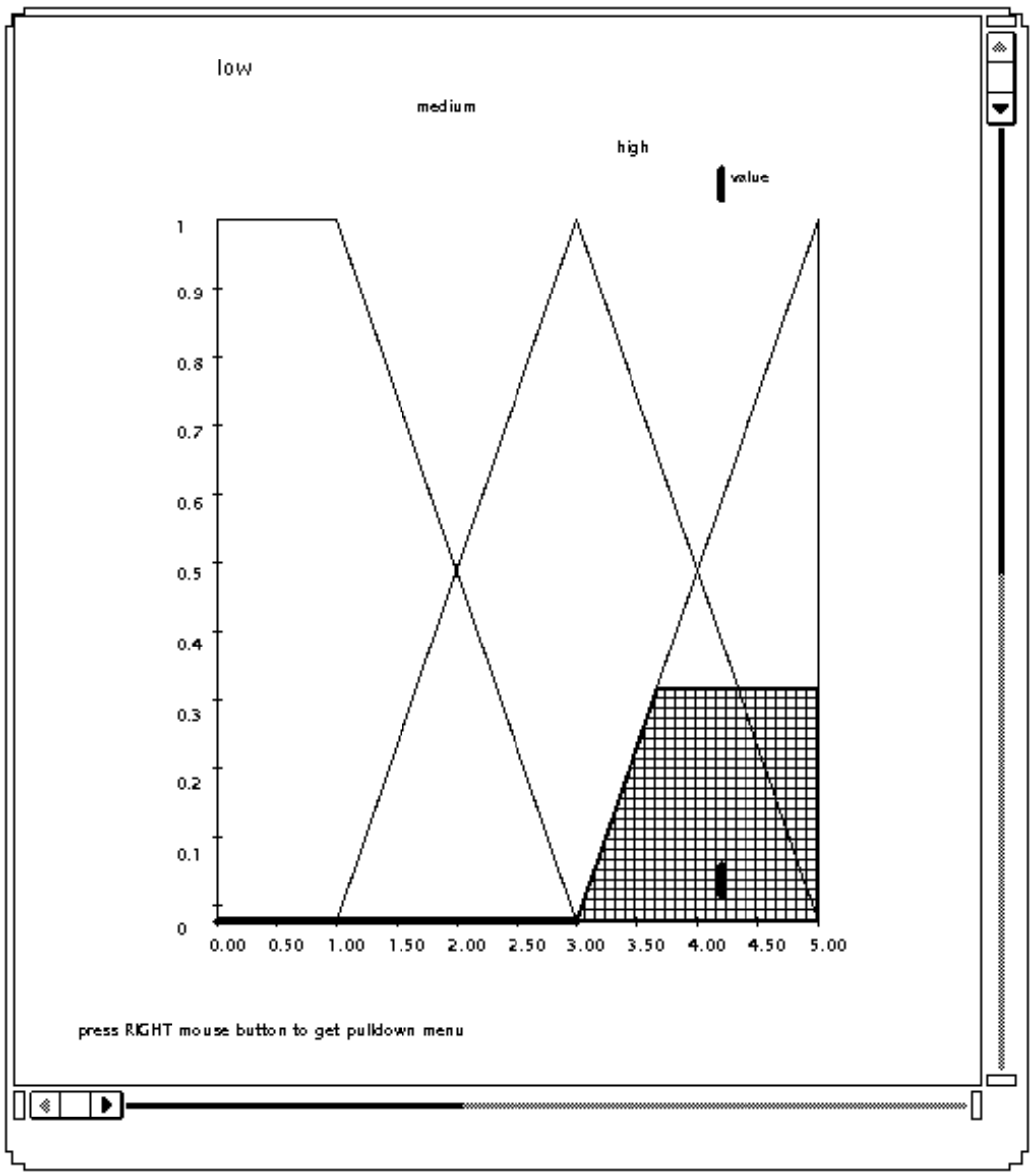

Figure 21: Linguistic variable C

THEN rule.

These rules are shown in Figure 22.

Applied on the value range of the input variables it has a functionality as shown in Figure 23.

C rules:

```
IF A  IS low  and  B  IS low THEN  C  IS  medium  WITH CERTAINTY  1.00
IF A  IS low  and  B  IS high THEN  C  IS  low  WITH CERTAINTY  1.00
IF A  IS high  and  B  IS low THEN  C  IS  high  WITH CERTAINTY  1.00
IF A  IS high  and  B  IS high THEN  C  IS  medium  WITH CERTAINTY  1.00
```

Figure 22: Rule set for formula $C = \log\ (\ A\ /\ B\ )$

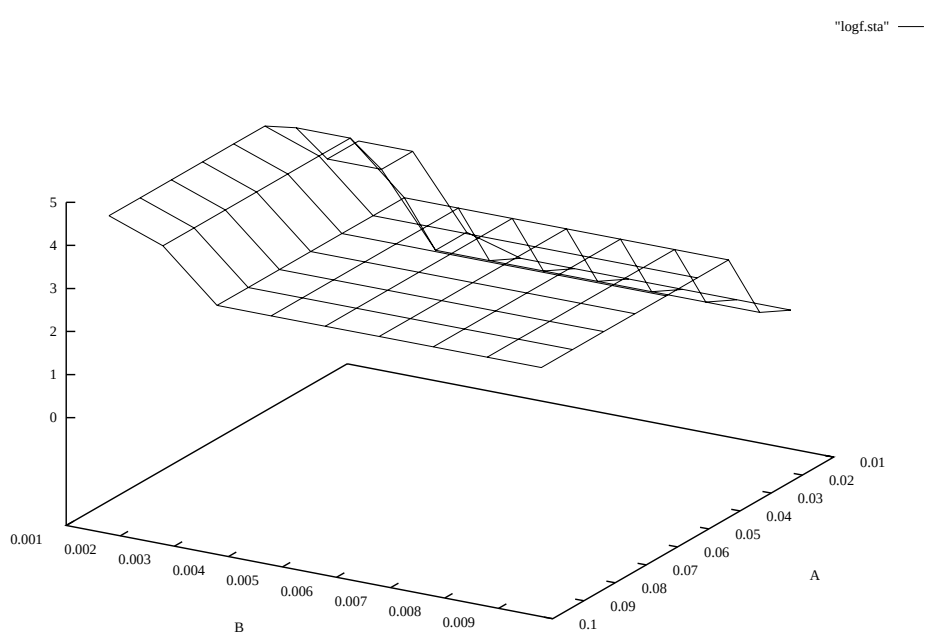

Figure 23: Functionality of the rule set

5.3 Comparison

The functionality of the formula and the rule set are shown above. In Figure 24 the difference between these two approaches can be seen. In Figure 25 the distribution of the deviations between the two models of the relations of

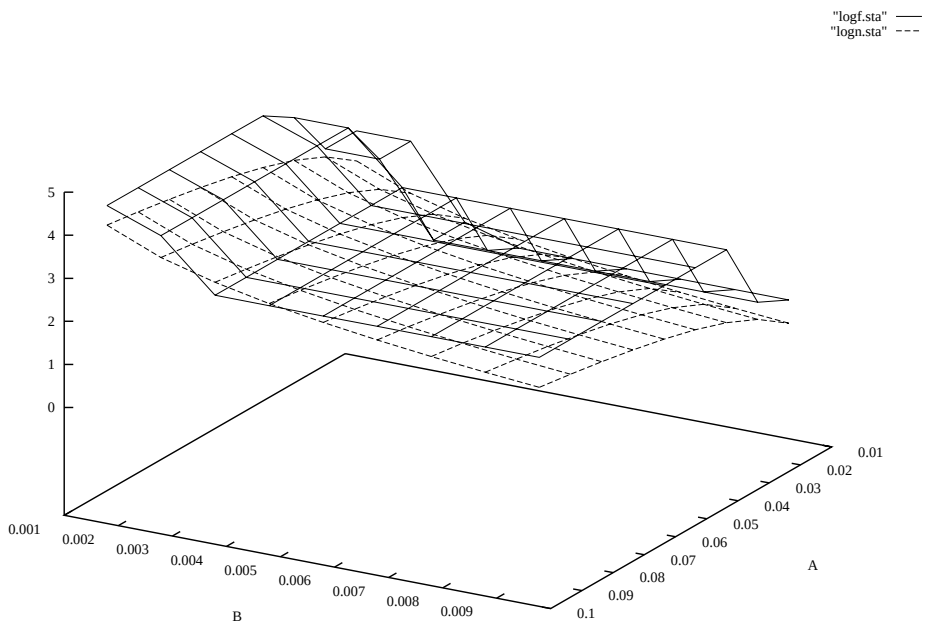

Figure 24: Comparison of formula and rule set

the two variables is shown.

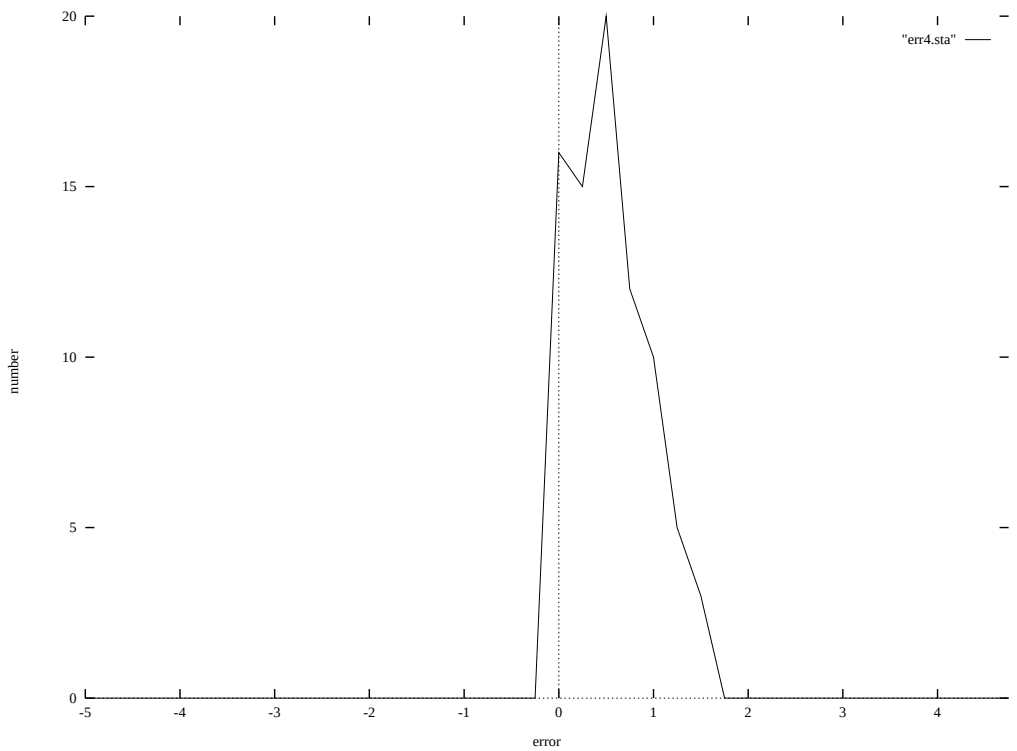

Figure 25: Distribution of the deviations between formula and rule set

6 Genetic algorithms

In this chapter the idea of genetic algorithms is introduced as described in [3]. Genetic algorithms are methods for finding solutions within a great area of possible combinations of single elements. Genetic algorithms simulated the natural evolution process. They can be applied within computer systems for learning tasks. They simulate the natural process reproduction, crossover, and mutation.

Within an implementation the algorithm mainly consists of state transitions. Needed are a representations of the single populations and some parameters for settings of the single evolutionary steps. There are simple mechanisms that perform the simulation of the evolution processes within the computer system. In the following, the single steps of such an algorithm are described.

6.1 Initialization

Within this step, a first population is generated according to the task description. This step can either be executed randomly or can translate domain knowledge into a first population. This allows a faster development into the demanded direction. If there is already some usable knowledge it makes sense to apply it. The a-priori knowledge does not get better by reinventing it. Thus, it is preferable to start at the current state of knowledge and try to improve it. Since genetic algorithms are still search methods within a mostly huge search space, it saves much time if the process can start at a point in the right direction.

6.2 Cycle of the generations

After initialization, the actual genetic algorithm starts. The simulation of the evolutionary process begins. One generation after another is generated and has to proof its fitness within the task. The most successful properties of the single populations will survive more probably than others that do not fit well into the given environment. First the fitness values for all individuals are calculated and then the next generation is breeded in three steps.

6.2.1 Calculation of fitness

The criteria for the survival of the genes of an individual are determined by a function that is closely related to the task. Within the problem description a function has to be defined that determines the value of a single individual. This value should be related to how near the individual is to the desired solution.

The fitness function has to be able to handle the representation of the individuals of a population. During the cycle of the generations this part provides the only direct interface between task and genetic algorithm. On the side of the environment it represents the goal of the use of the genetic algorithm. The algorithm will develop the generations into the direction of this function. This is the most important thing the algorithm knows about the field of application.

The fitness function has to be accessible for the genetic algorithm. It is the major not genetic algorithm function within the application of a genetic

algorithm. The task is described in some type of formal language within the genetic algorithm. The fitness function should operate on this syntax to avoid translations of the function into the task environment and then a translation back into the task description within the genetic algorithm.

Like all other methods and functions it should be as simple as possible to consume little computation time. The cycle of the generations is very often run through depending on the task. Therefore it is important to make each part of the cycle very short and fast.

6.2.2 Reproduction

The fitness function gives each individual a value of how near it is to the solution. To receive individuals that are nearer to the solution, the next generation should mainly be derived from the better-fitting individuals. Therefore the fitness value should be used as likelihood for the single individuals to be reproduced. The better-fitting individuals generate more children then the less well adapted ones.

This step lets the evolutionary process develop into the direction of the goal. It raises the average fitness value and gives the following steps a good chance to go further towards the solution.

6.2.3 Mutation

This step generates random changes of single features of the individuals. The value of these changes can be seen at the fitness calculation within the next generation cycle. It helps getting out of local maximums and brings in new properties that can remain if they are successful. This step is the major source for new ideas.

To avoid too chaotic development the rate of mutation is kept low. Else it would destroy the positive developments of the previous step and make any major development in direction of the solution impossible.

6.2.4 Crossover

This step represents the sexual generation of a next generation. The genes of two parents are taken and mixed up. It allows to build combinations

of well-fitting properties. If one child receives the positive features of both parents, it overcomes their individual drawbacks and will receive a better-fitting value at the generation of the next population. Such a combination of positive properties will have a far higher chance to survive in the next generation. Many individuals in the following generations will include these properties in their genes.

If the crossover produces less-fit individuals, they will more likely die out and with them the weak combination of properties.

This step is the major difference to other random search methods. Not only individuals are selected and used for further search. Here are also single elements of such individuals are verified and they are used for generating new individuals. This step is also the major advantage in finding solutions for genetic algorithms. It gives this method some kind of intelligence by going further than searching and evaluating. The attempt to generate better-fitting individuals from well-fit parents exceeds pure search methods.

6.3 Combination genetic algorithm - fuzzy logic

Fuzzy logic, as well as genetic algorithms, have very positive aspects for handling complex tasks. Fuzzy logic is very near to human task description and handling. It is also capable of dealing with imprecise and vague information. A genetic algorithm is a powerful method of finding solutions for problems. The combination of both ideas can widen the fields of application. It results in a more thorough method for problem handling and solving.

A contradiction lies within the implementation of the two ideas. Fuzzy logic uses a quite extended representation of information to allow the including of imprecise and vague knowledge. Genetic algorithms need short and simple representations of the information to guaranty short handling times within a cycle. The search for a good solution needs many runs through these cycles. That results in a need for very short execution times for the single steps of the generation cycles.

Within a cycle, the calculation of the fitness value is the only part that is concerned with the fuzzy logic part. The fuzzy logic handling of information is not oriented very much towards short execution times. Therefore the fuzzy evaluation of a task is usually too slow for the use within the cycles of the genetic algorithm.

To overcome this problem a time-oriented implementation of a fuzzy sys-

tem is needed to build a hybrid genetic fuzzy system that is applicable for real-world problems. If such a shorter representation and a faster evaluation can be found, the permanent translation and retranslation of information and relations between fuzzy system and genetic algorithm can be avoided.

7 Genetic programming kernel

The Vienna University of Economics genetic programming kernel is a tool for learning tasks. This tool was implemented by Helmut Hörner at the Abteilung für Angewandte Informatik insbesondere Betriebsinformatik, Institut für Informationsverarbeitung und Informationswirtschaft at the Vienna University of Economics. The genetic programming kernel is a C++ library. For more information of the ideas of this tool see [3].

This tool was developed to operate on context-free grammars. Any problem that is described in such a way can be solved with this tool. For details on handling such descriptions see the dragon book [1].

7.1 Context-free grammar interface

Designing the required context-free grammar is the most sensible step in applying the kernel. The description must have the form of a context-free grammar, yet still precisely represent the actual task. To write an exact context-free description for possible relations within a fuzzy system is, in some way, a contradiction to the fuzzy logic approach. There the relations are simply written down in *IF THEN* rules and other linguistic terms. There are nearly no constraints for expressing the relations within the environment.

For such a representation a formal description must be found. The main source for this is the specification or the implementation of the fuzzy system. Everything that is allowed by the program can also be allowed for the description. For this purpose the specification of the fuzzy tool can be used. If it is exact enough and is consistent with the actual implementation, it is only a formal translation of the specification.

A further influence for building grammar is the application itself. All information about the environment should be included within the context-free grammar as well. The grammar should produce individuals who are possible in the reality of the task. It makes no sense to produce individuals by deriving the grammar with the genetic algorithm that cannot or, respectively, do not exist in the task's environment. Such a description is very specific for each single application. All changes within the environment or the goal have to be respected within the grammar.

The use of the genetic algorithm reasons a very frequent execution of the evaluation of the grammar. Therefore it has to be evaluable very fast.

That means that there is no place for generality within the description. The grammar has to fit to the task as tightly as possible. The evaluation process of this grammar has to be very short because of its frequent calls. Already at the designing phase of the grammar the execution time has to respected. The grammar should contain few clauses with few alternatives. It makes more sense to adapt the grammar to each minor change within the environment than to build in generality which leads to an execution time that makes the genetic algorithm impractical.

If the above-mentioned points are respected, the use of a context-free grammar is a very elegant and widely applicable way of defining an interface between an application of the learning tool and the genetic algorithm itself. The representation as grammar allows a fast development and efficient evaluation methods for the calculation of the fitness values of the single individuals.

Context-free grammars are widely known forms of representation. There is no need to learn a new syntax for applying this tool. Some problems may already be described in such a grammar. In these cases the translation of the task description can be skipped.

7.2 Training circle

The training cycle is the main part of the genetic algorithm. It is usually executed a few thousand times while training. The actual number of executions depends on the complexity of the task and the start point.

There are two alternatives for the training cycle depending on the implementation of the training system. Single steps of the two branches are used in both of them. The alternative taken, has no effect on the result. Just the execution time and the effort of the implementation depending on the already available methods are different. The single steps including the generation of a new population are explained below separately.

7.2.1 Parsing

This is the faster alternative in determining the fitness of a single individual. The single steps used here are:

- Interpreting the generated individual

- Returning the fitness value
- Translating the best individual at the end of the training process
- Applying the found solution within the task

7.2.2 Translating

- Translating the individual into the tasks syntax
- Evaluating the individual within the task environment
- Returning the fitness value
- Applying the found solution within the task

7.3 Generation of a population

The first step within this cycle is the generation of a population through the genetic algorithm. A given number of individuals are generated according to the methods described in the previous chapter. The individuals are one single derivation of the given grammar. They are represented as derivation tree or some related string form.

The derivation tree form is the most efficient version for use within a computer system. It is very sensitive to faults within the grammar. It needs lot of effort to receive a working learning system. Such a system would then be the fastest version.

A short string form consists only of the numbers of the alternatives of the derivation of the grammar. These numbers are separated by points. This form is the most recommended one for applying within an application because it is short and easy to interpret within an application program.

A long string form additionally includes the alternatives themselves. Therefore they are easier to read for people. It lies between a good representation for machines and an easy to read version for the human expert.

In any case, the single individuals of the generated population will have to be evaluated. Each individual has to be tested for its fitness within the task's environment. A value has to be determined that shows how near this individual is to the desired solution of the problem.

7.4 Parsing the individual

This step determines the fitness value by testing the individual's properties with regard to the goals of the problem. An interpreter for one of the three possible models of the generated individuals looks at the individual as an application program would do and calculates a corresponding value.

The method for this step is mainly a transformation of the evaluation process of the application to the syntax of the genetic tool. This adaptation to the syntax of the grammar derivation allows a faster evaluation of the individual. This results in the major advantages of the parsing branch.

This method has to be implemented for the special definition of the grammar and the syntax of task. At minimum it's call has to be implemented within the genetic algorithm tool.

7.5 Translating the word into task's syntax

This method transforms the individual's representation from the syntax of the genetic tool to the syntax of the application program. It is applied either in case of the translation branch at every cycle, or at the parsing branch to transform the best individual into the application's form.

The word taken from the derivation of the grammar corresponds to some individual of the task because the grammar is another description syntax for the task. A derivation of the this grammar should, therefore, also contain individuals of the task's environment.

The result of this translation is a word, a statement, or an individual of the task's world.

This method has to be implemented for the special definition of the grammar and the syntax of the task. At minimum it's call has to be implemented within the genetic algorithm tool.

7.6 Evaluation

The calculation of the value of an individual is related to the task and not to the genetic algorithm tool. Therefore, it cannot be an a-priori part of the tool. This function has to be taken from the application that uses the genetic

programming tool. If no such function is available, one must be implemented for this purpose.

If there already exists a method which can determine the value of an individual it can be included in the genetic programming tool. It is likely that such a function already exists. It is mainly an evaluation method of the item that should be trained. If the system that applies the genetic kernel is also written in C or C++, the respective function needs only to be called within the kernel, and therefore, such a call has to be included in the genetic tool.

7.7 Returning fitness value

The determined fitness value of either branch has to be returned to the control of the training cycles of the genetic programming tool. This value is needed for the generation of the next generation of individuals. The end of the training process can also be determined by this value. If it reaches a pre-defined value the cycle is stopped and the best individual is returned.

7.8 Applying best the individual

The individual with the best fitness value can then be applied to the application. This individual is the goal of the training process. The program that used the genetic algorithm tool for learning something can from now on continue execution without the genetic tool.

8 Structure of the genetic fuzzy system

In this chapter the structure of the genetic fuzzy system for learning relations between the linguistic variables is described. The order in which the single tools and administrational functions are applied is listed below. The single steps performed by the fuzzy tool or the genetic algorithm tool are also introduced.

8.1 Input for training

The input information for training the systems can either be generated data material or real data from monitoring a process. For the structure of the system it makes no difference what kind of data is used for training.

- Generation of values
  In a case where there are no monitored process data available but, there is a model that represents the relation between the used variables, this model can be applied to the input data to generate output data for training a fuzzy system. As input data, the whole possible value space, for which the final application should be defined, can be used. On these input values a model or formula can be applied to receive the needed output values.

  This approach is used in the example that is used within this work.

- Monitored data
  This is the case for real-world applications where the process is already running for a longer period and the process data are monitored and collected. The data that are monitored under the same circumstances as the trained system should run within, can then be taken for evaluating the generated individual of a population within the training cycles of the genetic algorithm.

8.2 Training

The training process consists of the following steps:

- Designing of a grammar for the task

- Generating an initial population

  This can happen in two ways:

  - An initial population is generated randomly.
  - A process model is translated into a syntax readable for the genetic algorithm tool. This transformed model can be put into the genetic tool and used as initial population. This alternative allows a far quicker development of the training process into the direction of the solution.

    This way of initialization is applied within the example. The existing rule set is used for generating a first population by transforming it into the short string representation of the genetic tool. The method for transforming the relations from the rule set of the fuzzy tool to the short string representation of the genetic tool is related to the method that is used for generating the rule set out of the result of the training process.

8.3 Training cycle

The main training process consists of an alternative generation of new populations and the calculation of the fitness values for the single individuals of this population. It works as described in the previous chapters.

8.4 Translating the result

The result of the training process is one individual which received the best fitness value during the training process. This individual is encoded in the short string form that consists only of the number of the alternative of the derivation of the single clauses of the grammar separated by points. This representation has then to be translated into *IF THEN* rules by methods called at the end of the training process.

The resulting rule set can then be used for any rule-based system that operates on the same linguistic variables and uses similar operators.

8.5 Independent fuzzy system

The newly generated rule set can now be applied to the task it is trained for. The genetic algorithm tool is not needed during the application of the rule set. The resulting rule set is completely independent from the training methods. It can be taken to any machine or system that allows the evaluation of *IF THEN* rules using linguistic variables represented by membership functions.

The fuzzy system applied within the task can at any time be put again into the training process of the genetic algorithm. This makes sense especially if the environment of the task or the task itself has changed. Such a repeated training process should allow new optimization of the rule set to lead to a rule-based system that fulfills the goals of the task in an optimal way.

Within this example the rule set is also interpreted by the FLIP++ library. Within the desulfurization project the machine and system used for training and for controlling the process are different ones. There only the rule set and the linguistic variables with their membership functions are transferred to the control system. They are interpreted for controlling the process by a version of FLIP++ prepared for a different platform. If there are changes within the environment or the requirements, the fuzzy system is again adapted to the new needs on the machine where the genetic algorithm tool is installed.

9 Building up the system

This chapter shows the single steps of implementing a genetic fuzzy system. The FLIP++ library is used as fuzzy tool. The genetic algorithm methods are taken from the genetic programming library that was implemented by Helmut Hörner and described in [5].

9.1 Designing a grammar

The first thing needed to apply genetic algorithm methods is a context-free grammar that describes the task. To receive shorter words as result of the derivation of the grammar, a shorter syntax for modeling the rules is used. The words included in the language of the fuzzy tool are omitted here. Therefore, the rules do not include *IF*, *THEN*, *IS*, and *WITH CERTAINTY*. The structure is the same as in FLIP++.

The terminal symbols within the syntax are variables. They are:

- Linguistic variables: A or B within the premise and C within the conclusion
- Linguistic terms: *low*, *medium*, and *high* for premise as well as conclusion
- Operators: Within this example only *AND* is used.
- Importance: Only 0.5 and 1.0 are allowed here.

The non-terminal symbols are names of clauses. The whole grammar is shown below.

```
S := <rs> ;
<rs> := <rb> | <rs> <rb> ;
<rb> := <lvc> <rl>;
<rl> := <r> | <rl> <r> ;
<r> := <br> | <cr> ;
<br> := <tp> <ltc> <c> ;
<cr> := <o> <tlp> <ltc> <c> ;
<tlp> := <tp> | <tlp> <tp> ;
```

```
<tp> := <lvp> <ltp> ;
<lvp> := "A " | "B " ;
<lvc> := "C " ;
<ltp> := "low " | "medium " | "high " ;
<ltc> := "low " | "medium " | "high " ;
<o> := "AND " ;
<c> := "0.5 " | "1.0 " ;
```

The semantics of the single clauses are:

- The start symbol leads to a rule set.
- A rule set can consist of one or more rule blocks.
- A rule block holds a linguistic variable for the conclusion of all rules that are included in the following rule list.
- A rule list consists of one or more rules.
- A rule can either be a basic rule or a complex rule.
- A basic rule holds a term for the premise, a linguistic term for the conclusion, and a certainty value.
- A complex rule holds an operator, a term list for the premise, a linguistic term for the conclusion, and a certainty value.
- A term list for the premise consists of one or more premises.
- A term for the premise consists of linguistic variable and a linguistic term.
- The rest are replacement causes for terminal symbols.

9.2 Initialization from an existing rule set

To gain a shorter training process and to apply already existing knowledge, methods are implemented that can translate a rule set into the short string representation of the genetic programming tool. This transformed rule set is then used as part of the initial population of the training process.

Below shown is the transformation method for the term of a premise.

```
void Term::make_ShortString(strstream *ss) {
// <tp> := <lvp> <ltp> ;
// <lvp> := "A " | "B " ;
// <ltp> := "low " | "medium " | "high " ;

*ss << "1.";
if (strcmp(cv->get_cv_name(),"A") == 0) *ss << "1.";
else *ss << "2.";
if (strcmp(lt,"low") == 0)  *ss << "1.";
else if (strcmp(lt,"medium") == 0) *ss << "2.";
else  *ss << "3."; };
```

This is a method of the class Term with the fliprule.cc file. It writes its short string transformation on a string stream. In the same way, methods in the higher levels within a rule set write their transformation onto the stream. This stream can then be transformed into a character string and used to generate an initial population.

9.3 Raw fitness calculation

The following method determines the raw fitness value for individuals of a population. This value is a basis for the breeding of the next generation.

```
float StartNode::ObjFuncVal() {
//  In-tree implementation for calculating the raw
//  fitness for trees of flip language.

float af, bf, log_f, log_n, log_diff, log_diff_sum;
char s[STRLENGTH];
strcpy(s,GenoTypeString(SHORT));
  int sl=strlen(s);
static int mode=ios::out;
static RuleSet *rs;
static ifstream *input  = new ifstream;
static CVList *cvl;
static OperatorSet *os;
static CombinedVariable *SA;
static CombinedVariable *S_SOLL;
static CombinedVariable *LOG;
```

```
static name rs_name;

if (rs == NULL) {
input->open("example.rdb");
rs = new RuleSet;
rs->read(input);
input->close();
delete input; }
else {
strcpy(rs_name,*(rs->get_rs_name()));
if (strcmp(rs_name,"empty_rule_set")==0) {
input->open("example.rdb");
rs->read(input);
input->close();
delete input; }; };

if (cvl == NULL) cvl=rs->get_cv_list();
if (os  == NULL)  os=rs->get_operator_set();

if (A == NULL) A = cvl->find_cv("A");
if (B == NULL) B = cvl->find_cv("B");
if (C == NULL) C = cvl->find_cv("C");

log_diff_sum = 0.0;

for (int ai = 1; ai <= 10; ai++) {
for (int bi = 1; bi <= 10; bi++) {
strstream   *ss=new strstream(s,sl,mode);
af=0.01   * ai;
bf=0.001 * bi;
SA->set_value(af);
S_SOLL->set_value(bf);
log_f = Evaluate_rs(ss,cvl,os);
log_n = log(af/bf);
log_diff = log_f - log_n;
if (log_diff<0) log_diff = log_diff * (-1);
log_diff_sum += log_diff;
delete ss;  }; };

return log_diff_sum; };
```

This method of the class StartNode generates a short string of the derived tree structure. This string is the main source for the calculation of the raw fitness value.

As second source, the list of linguistic variables is read in from the file containing the rule set. The membership functions of the linguistic variables used within the derived words of the grammar are needed for the evaluation of these individuals. From the same file the operator set is read in. It holds the information about which operators have to be used for the different steps within the evaluation process.

For a set of possible combinations of generated input values the numeric formula is calculated. The result is stored in the *log_n* variable. On the same set the individual generated by the genetic algorithm is evaluated. The single results are stored as *log_f*. The differences between these two values are added to the *log_diff_sum* variable within each run of the double for-loop. After termination of this loop, *log_diff_sum* is returned as raw fitness value.

9.4 Parser for grammar words

The actual evaluation for each single pair of input values for a single individual is shown within this chapter. The method *Evaluate_rs* of the class *TreeNode* is called from the process which determines the raw fitness value. Used as arguments are a string stream holding the individual in short string representation, a list of linguistic variables, and an operator set.

The short string representation consists of numbers divided by points. A number is the number of the alternative within a clause. If a clause has no alternative derivations, it is represented by a 1 followed by a point. After that the derivations of the single non-terminal symbols of this clause follow. The single non-terminal symbols are derived from the left side to the right within a clause.

The evaluation process works its way through the derivation of the grammar from the left to the right within the short string representation in the stream. Each number determines which clause has to be evaluated next, therefore, which evaluation method will also be called next.

Below the evaluation method at the lowest level is shown. It is called when a term of the premise has to be evaluated. It returns the membership value of the referenced linguistic term for calculation at higher levels.

```
float TreeNode::Evaluate_tp(strstream *ss, CVList *cvl, OperatorSet *os) {
int in, fs_index;
float imp, mv, imv;
char inc;
name lt, cvpn;
CombinedVariable *cvp;
PlimFuzzySet *fs;

*ss >> inc; // first case
in=int(inc)-48;
*ss >> inc; // .
if (in != 1) cerr << "Error in Evaluate_tp: in=" << in << space;

*ss >> inc; // <cvp>
in=int(inc)-48;
*ss >> inc; // .
if  (in == 1) strcpy(cvpn,"A");
else if (in == 2) strcpy(cvpn,"SB");
else cerr << "Error in Evaluate_tp: in=" << in << endl;
cvp = cvl->find_cv(cvpn);

*ss >> inc; // <ltp>
in=int(inc)-48;
*ss >> inc; // .
if  (in == 1) strcpy(lt,"low");
else if (in == 2) strcpy(lt,"medium");
else if (in == 3) strcpy(lt,"high");
else cerr << "Error in Evaluate_cr: in=" << in << endl;

fs_index = cvp->get_fs_index(lt);
fs = cvp->find(fs_index);
mv = fs->get_match_value();
imp = cvp->get_importance();
imv = join_mv_with_importance(mv,imp,os);

return imv; };
```

The evaluation method for a term of the premise reads in 1 for the alternative first. Then the number for the linguistic variable is analyzed and a reference to the respective variable is searched. Next the number for the linguistic term is evaluated. Now the linguistic variable and the linguistic

term are known.

Within the linguistic variable, the fuzzy set of the linguistic term is taken. From this fuzzy set the membership value is searched. From the variable itself the importance value is taken. The membership value and the importance value are then connected by the operator mentioned in the operator set. This happens within the join_mv_with_importance method. After that, the compensated membership value is returned to a higher level.

9.5 Translating grammar word

After the training process has terminated, the result is an individual that received the best fitness value. This individual is a single derivation of a tree structure. There are two additional representations available. One is the above used short string representation. Another one is a long string form that is also the basis for a similar form used by a printing method. All listed methods are not useable as rule set for the FLIP++ library. Therefore, transformation into a set of fuzzy rules is required.

This method functions similarly to the evaluation process. Both methods work through the short string representation of an individual in the same way. The main difference is the output. The evaluation methods calculate a fitness value and the transformation method generates fuzzy rules by applying the appropriate methods from the FLIP++ library for generating rules.

10 Training the system

The training process consists of an initialization step, a training cycle, and the retransformation of the result. The main program is included in the file ga.cc. The single methods can also be included in other programs.

10.1 Initialization

The first steps concern mostly the building of an initial population. This is generated randomly. The rule set from the fuzzy set is converted and included into the first population. Some parameters for the training process itself are also set.

```
#include "ga.h"
#include "flipeval.cc"

#define MAXDERIVDEPTH 200
#define POPSIZE 10
#define HISTORY 2
#define LANGUAGEDEF "flip.def"

int main()
{
ifstream *input  = new ifstream;
static RuleSet *rs = new RuleSet();
input->open("example.rdb");
rs->read(input);
input->close();
delete input;

char cp[STRLENGTH];
strstream *ss = new strstream(cp,STRLENGTH,ios::trunc);
rs->make_ShortString(ss);

Farm MainFarm(POPSIZE, HISTORY, SIZEUNIF, LANGUAGEDEF, MAXDERIVDEPTH);
MainFarm.SetFitnessScaling(0,MINIMIZE);
MainFarm.SetFitnessAdjustment(One_DividedBy_OnePlusX);
MainFarm.SetSelection(LINEAR_RANK,0);
MainFarm.SetSampling(STOCH_SAMPL_WITH_REPL, POPSIZE);
```

```
MainFarm.SetMating(RANDOM_PERMUTATION);
MainFarm.SetSelectionHeuristic(pow4);
MainFarm.SetCrossOver(1,0.85);
MainFarm.SetMutation(0.2);
MainFarm.SetSaveBest(YES);
MainFarm.SetSmallestSubTreeOfVirtualPop(30);

MainFarm.Current->Actual[0].Set(ss->str());
delete &ss;

MainFarm.Current->GenerateVirtualPop();
MainFarm.ComputeFitnessValues();
```

MAXDERIVDEPTH holds the maximum derivation depth. *POPSIZE* holds the initial population size. *HISTORY* states how many generations can be traced back. *LANGUAGEDEF* names the file containing the definition of the grammar.

From the file example.rdb a rule set for the initial population is read in. The rules are converted into the short string representation of the genetic programming tool and written into a string stream.

Then an initial farm is generated. A farm can hold more than one population. Then a few parameters for the training process must be set.

The transformed rule set is then included into the initial population. The first fitness value calculation is started.

10.2 Training cycle

This cycle is run through a pre-defined number of times or it stops if a certain quality of the generated individual is reached.

```
unsigned int i = 0;
while(i<500)
{ i++;

MainFarm.Current->BestIndividual->RawFitness << ": " << endl;
MainFarm.Current->BestIndividual->Print();
```

```
if(MainFarm.Current->BestIndividual->RawFitness < 500)
break;

MainFarm.NextStep(); }
```

The raw fitness of the best individual is tested on the requirements. If it satisfies the criteria the training cycle is stopped. Else, a new population is generated, by calling the *NextStep* method for the farm.

A next population is generated by applying the methods as described in chapter 6. Through selection, mutation, and crossover new individuals are built.

10.3 Retransformation

The best individual after the training cycle is the result of the training process. This individual is transformed into a needed representation at the end of the training. In case of the given example the short string representation is transformed into *IF THEN* rules for simulating a simple formula. It can now be applied on the given task. The program for which the training process was run can from now on run independently till a new training process is needed.

11 Neuro fuzzy system

This chapter describes a second approach for learning rules from existing data. For building up a connection from our fuzzy tool to a neural net simulator, I work together with Dipl.Ing. Friedrich Leisch from the Institut für Statistik und Wahrscheinlichkeitstheorie.

The control of complex, nonlinear processes is difficult to make for human experts. In our case it is also not possible to find an efficient solution for automatic control of the process. Therefore, we develop a system that should, on one hand use all available expert knowledge, and on the other learn the remaining relations within the task with a neural net. Both types of knowledge should then be combined to achieve better results by bringing together experts and AI methods.

11.1 Task

The control of a complex dynamical chemical task within the steel-making plant is difficult. It is hard for human experts to make always the optimal control decisions. Also, most times these experts are not available for controlling the task. Simple mathematical models are not satisfying because of the highly nonlinear task and because many important influences on the process cannot be exactly measured. Often they can only be estimated from previous processes. Therefore we need a system that holds the expert knowledge and controls the process. This step would already improve the over all efficiency of the control by having expert knowledge available all the time. We want to improve this knowledge by including also information that could be learned from previous decisions.

We do not want to simply learn what experts already know, but we want to gain new knowledge and have a better understanding of the process from the information we already possess. We do not want to work independently from the experts and try to imitate their behavior in order to replace them, but we want to give them a tool, a partner, which allows them to learn and get a better understanding of their work. They can then bring their improved knowledge into the system and make the system just a little bit more efficient and reliable. There always should be a bi-directional learning process. Each part should see the other part as a kind of teacher. By helping the teacher to become more competent, one has the chance to learn more from an improved teacher. So we would like to build up a neuro-fuzzy system that can be used

in such a man machine cooperation. Here we still need the human computer expert as a connection between the task expert and the computer system, and in some steps to at least include the expert knowledge in the control system. We want to demonstrate that a connection of intelligent control methods and human experts brings better solutions than either human or machine can do alone.

11.2 The model

The system we need to create to control such process requires to include the expert knowledge and to learn from existing data material.

In this project we will contribute expert knowledge in the single steps of building and training the system. We try to use directly the experts as advisors when we define the structure of the final control system. Part of their knowledge will be included in each learning step. Some explicit relations that can be easier implemented in some preparation steps will be put into a data input interface in order to obtain values that can be directly used for fuzzy control.

In two learning steps we will use SNNS, the Stuttgarter Neural Net Simulator, to build neural nets for training tasks. The knowledge we gain from the SNNS will be translated into a fuzzy rule-base. This happens in an interface to FLIP++, a fuzzy control library.

FLIP++ was developed by us to evaluate constraints within a scheduling project for a steel plant. It is part of the ∗FLIP++ library [12, 13] for real-world decision-making. This allows optimizing under vague constraints of different importance using uncertain data where compromises between antagonistic criteria can be modeled. Typical application areas include scheduling, control, design, configuration, planning, and classification. FLIP++ (first introduced in [2]) is only one part, the basis part of this project, and can be independently used for other fuzzy inference problems. For this C++ library we developed the InterFLIP++ library as graphical user interface. This library will also be used for the new project in an adapted version.

Additionally we plan to use the Gnu-plot library to graphically display the functionality of the system to have a better handle on the control system. We can use it also to display the process to make it easier and faster for the operator to get information about the process itself.

11.3 Historic data

Not all information we can get from the process is trainable. Some input variables are only IDs for some items. We have no direct information about these items other than their references. We know only that these hidden influences are not constant and can change over time. There is no information a neural net or any other learning utility can use to train a system and derive rules therefrom.

Examples therefore are the IDs of the furnaces or the wagons.

We have continuous and sudden changes over time for the different influences represented by the IDs. Continuous changes are e.g. growing displacements. Sudden changes can be the removing of these displacements. From the continuous changes we have functions that approximate their influences, determined by statistical methods. With these linear models we can make an estimation of the actual value of the concerned variable. In some cases we can assume sudden changes and set the variable to an a-priori defined reset value. If this is not possible, the operator must tell the system the occurrence of such an unpredictable change and trigger the setting of this variable to a reset value. These estimations can then be included in controlling the whole task together with measurable variables.

The relevant historic data about each variable represented by some identifications are stored in a separate matrix for each ID. This historic information is then used to predict a continuous or sudden change of the concerned variable and to give an estimation of its actual value. After each use of a certain identification, the according values in the matrix are updated by the newly received values from the control task. By reason of either known linear relations or unpredictable sudden changes over time, we will use simple functions and checks to include the a-priori known expert knowledge. This estimation of the values of input variables will be implemented in the data read-in interface.

This separation of immeasurable time sensitive information from the other input data allows us to simplify the learning process by neglecting the time influence on the process. So we do not have to include lots of historic data in the training. The second positive effect is to achieve a clear structure in the whole system by always treating the information with adequate methods. First, we use algebraic and boolean methods to prepare the values we cannot measure and then we use fuzzy logic [18] to control the system through a rule-base trained by neural nets.

11.4 Learning

The main idea is not to learn by trying out all possibilities within the existing data. This would not work because of the great number of influences and great number of old data we have to learn from. On the other hand, we need a great number of samples to cover the complexity of the task. Even if we successfully try once to learn something from the existing data, the learned model would not work well for a long time. The reason is the permanent change in some influences through time like the growing displacements. We do not have direct information about this time dependency. If we would try to include enough historic information in the learning process to cover these changes over time, the learning task would become too large and complicated to be solved. To find a solution, we structure the task and the learning process.

First we need to find out which influences are really important and which can be neglected. This is a very important step if we want an efficient and transparent system. If we find out the relevant input variables and their relations, we have the chance to build up a system that works more efficiently than the human experts now do with statistical methods.

Then we have to try to get the time dependent information out of the learning process by assuming some default values while learning. During the operation, the change over time is calculated with numerical methods. The models used are created by experts at the plant through observing the process and evaluating the results with statistical methods. After this preparation of the time relevant inputs, we can use them together with the time constant values optimize the process. We structured the learning process into six steps as follows:

11.4.1 Finding out relevant variables

This is a very important step in the learning process. We need to find out which variables are relevant for the result and which can be neglected with minimal effect on the result. Here we can reduce the problem by about 80%. Just as in the whole project, we try here to use as much expert knowledge as possible to shorten the learning process and to always have some control over the direction in which the learning process is developing. At the same time, the experts themselves will be able instantly learn from our results, even during the learning process.

We begin with the influences the experts gave us and checked the relations between them with neural nets created with SNNS, the Stuttgarter Neural Net Simulator. Thus we can get approximations for the importances of single input variables.

11.4.2 Building up a hierarchy

In this step we will try to build up a hierarchy with the relevant variables. This is done according to their influence on the process. The most important variables stand in a higher level of this hierarchy than less significant influences. Another rule for the structuring is the relation between the single influences. Related variables like chemical elements or weights are put together if they are of similar importance for the process. The single subtrees are put togehter within intermediate varibales. Thus a structure is created with the aim of gaining simpler rules with fewer variables. This should make the resulting rule-base easier to read and understand for human experts.

As in the first step we will try to use expert knowledge to build up a structure that we cannot learn from the data. At the same time we give the experts feedback of what we learned until now and, vice versa, we will be checked by the experts. These steps are not done automatically. We have to follow the importance learned in the first step aswell as some plausibility rules. If we would need to use these steps more often, we could implement some tools that support the building of this hierarchy according to the information from the variable evaluation. But it would be difficult to automatically include the expert knowledge if we have no expert system with this knowledge which supports our problem.

11.4.3 Dividing variables into linguistic terms

Dividing the single variables selected by the first step into a number of linguistic terms is the next step. This step is supported by the FLIP++ library by using predefined variable types and adjusting them to the actual needs using parameter sets. Here we try to use as few terms as necessary to keep the effort in the following steps relatively small and built up a system which does not include more complexity than the task itself. According to the data and the insights of the experts, the membership functions for the single linguistic terms are created. The main reason we do not want to learn the membership function, at least for the time being, is, because of the correspondence of the membership functions to the experts imaginations concerning the repre-

sented linguistic terms. The experts are human beings, who then are most likely read the resulting rule-base. The terms used in the rules should represent as closely as possible human sight of these linguistic terms. This is necessary in order understand the rules and gain new knowledge out of the trained neuro fuzzy system.

11.4.4 Creating all possible rules

After we have built a model of all relevant variables in form of fuzzy variables with linguistic terms and corresponding membership functions, we can create a rule-base using these variables. We can create all possible rules that can be built using the variables we have. According to the hierarchy from step two, we build rules as combinations of all linguistic terms of all variables for each layer and branch of the hierarchy. The nodes in this hierarchy are intermediate variables designed according to the variables they are derived from. The hierarchy with its intermediate variables reduces the number of possible rules by putting intermediate results together. In the next level we have only one and not many variables to consider when building all possible rules.

11.4.5 Classifying the rules

From all created rules we have to select only those which have a representation in the existing data. Therefore, we must classify all these rules and then delete those rules with no strong representation in the data. For this classification we have an importance field in each rule which holds the information about the credibility of this single rule. Here we use a value ranging from 0 to 1. Zero means there is no reason to trust this rule and 1 means that this rule represents the relation of the included variables. Values in-between indicate a partial trust in this rule.

Now we have to find out which rules of all the possible rules with the relevant data are a representation of actual data material. For all rules which are a representation of given data, the importance is set to the relation between given data and linguistic value of the conclusion part of this rule. This can be done by using the difference between the fuzzified resulting value and the membership value of the linguistic term of the rule. To improve the performance and not implement a new learning method, we must develop an interface to SNNS. We can then use this interface to build up an adequate representation of our fuzzy rules as a neural net. We can train this net

with the collected information about the process. The trained net is then retranslated into fuzzy rules. This means that all rules receive an importance value representing the credibility of each single rule according to the data material.

11.4.6 Selecting more reliable rules

We now have lots of rules where most importances are still set to some default value. This means that there have been no examples in the data saying anything about the particular rule. There are rules with different importance values in the range of 0 to 1 that are learned from a neural net using the information collected from the process.

To build up a working fuzzy system we have to eliminate all irrelevant rules. We can do this by selecting only rules with an importance above a certain threshold. If we take, for example, a value of 0.7 as threshold, only rules with a learned reliability of more than 70% are accepted and all other rules are neglected. The threshold determines the number of rules we then have in the rule-base. The threshold should be selected in a way such that the number of rules is kept as small as absolutely necessary and still a more or less continuous and smooth functionality of the whole system is received. The interpolation of the examples in the data material should be soft, smooth, and defined on the whole work space.

11.5 Conclusion

We now have to prove that our idea works that the combination of human experts and AI methods delivers better results and more efficient solutions for complex problems. After this control problem we can try this idea on other projects within the *FLIP++ library, for instance on management problems. One could be personal planning for which a company has already shown interest.

To design and implement some kind of expert system that manages the man machine cooperation, the information flow from the expert to the machine can be the next step.

12 Conclusion

In this last chapter of this work, the major results gained in working on this thesis and the related steel making project are put together. The single artificial intelligence methods and their combination are again analyzed mainly on their advantages. All their drawbacks can be overcome and are, therefore, not of great interest.

Fuzzy logic is an approach for solving problems. It is easy to apply and understand which makes it easy to accept. Fuzzy Logic needs less computer resources compared to its advantages and can therefore be applied in practically all environments. It is applicable in many fields of computer usage such as control, decision making, as well as within expert systems, etc. It can handle human task descriptions and use them for solving problems. These contain often vague and uncertain knowledge or have to operate on imprecise information. These are the major reasons for the application of fuzzy logic. The solutions reached by the use of fuzzy logic are usually efficient and stable, which are good reasons for use in real applications.

Its greatest drawback is that it includes a-priori no learning method and can only use knowledge which is already available.

In the future, fuzzy logic can be utilized to a wider extent as a feature of expert systems, mathematical tools, general or task specific problem solvers, database applications for queries, and recognition problems like voice recognition or pattern recognition to help computers get their own view of their environment and make them widely applicable. Fuzzy logic can become part of many computer applications or of general purpose chips just like a numeric processing unit for handling vague knowledge and imprecise information. It will then be no longer an artificial intelligence method as a major point of concern, but a part of most computers thereby allowing then to think about problems on a far higher level.

The transparence and the stability of fuzzy solutions allow far more reliable solutions even in safety relevant applications. Thus its use is far more a help rather than a risk through to complex design causing unpredictable behavior.

Genetic algorithms are the usage of the evolutionary process to find solutions with the help of computers. They are based on a simple idea, and allow easy and elegant handling of problems. They are applicable in any field where well-fitting descriptions of the task can be made which are then

the basis for the searching for a good solution. With genetic algorithms it was possible to find solutions for certain problems that were better than any solutions known for these tasks until that time.

Finding solutions in more complex tasks, the description has to fit very tightly to the special case of the task and allow no generality. Whenever the environment changes the learning system has to be adapted. Else the effort of the training process would become too large. For more complex problems a lot of time for the training process is needed. The increase in processor speed will help loosen this problem and widen the applicability of genetic algorithms.

Neural nets are another method of finding solutions within a task. They imitate the functionality of nerve cells with a lot of success. Here also is the uncomplicated idea behind which makes neural nets applicable to many problems. They are also relatively easily to handle and work well in finding unknown relations within a task.

The knowledge learned by neural nets lies within the net and can hardly be used independent of the net. If the knowledge should be separated from the net, restrictions will have to be made for the net to be able to retrieve the learned relations out of the net. This step reduces the power of neural nets. Further experience in the application of hybrid systems will allow users to find a way to handle this problem.

By combining such artificial intelligence methods the drawbacks can be overcome. What form these combinations for the different types of applications should have is still a subject of studies. By building such combined or hybrid systems something like intelligent systems can be generated.

References

[1] Jeffrey D. Ullman Alfred V. Aho, Ravi Sethi. *Compilers principles, techniques, and tools*. Addison-Wesley, 1987.

[2] Markus Bonner, Stefan Mayer, Andreas Raggl, and Wolfgang Slany. FLIP++: A fuzzy logic inference processor library. In *Workshop Notes of the IJCAI'95 Workshop on Fuzzy Logic in Artificial Intelligence*, pages 83–92, Montreal, Canada, August 1995. URL: ftp://ftp.dbai.tuwien.ac.at/pub/papers/slany/ijcai95wsflai1.ps.gz.

[3] Andreas Geyer-Schulz. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*. Physica-Verlag Heidelberg, 1995.

[4] Christoph S. Herrmann. Fuzzy logic as interfacing technique in hybrid ai systems. In *Workshop Notes of the IJCAI'95 Workshop on Fuzzy Logic in Artificial Intelligence*, 1995.

[5] Helmut Hörner. Ein kern für genetisches programmieren in c++. Master's thesis, Vienna University of Economics, 1996. In German.

[6] Erich P. Klement. Operations on fuzzy sets and fuzzy numbers related to triangular norms. In *Proceedings of the Eleventh International Symposium on Multiple-Valued Logic*, pages 218–225, Norman, 1981. IEEE, New York.

[7] Erich Peter Klement, Radko Mesiar, and Endre Pap. On the relationship of associative compensatory operators to triangular norms and conorms. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 4(2):129–144, April 1996.

[8] Erich Peter Klement and Wolfgang Slany, editors. *Fuzzy Logic in Artificial Intelligence. Proceedings of the 8th Austrian Artificial Intelligence Conference, FLAI'93, Linz, Austria, June 1993*, volume 695 of *Lecture Notes in Artificial Intelligence*. Springer Verlag Berlin Heidelberg, 1993. A conference report is available on the net: URL: ftp://ftp.dbai.tuwien.ac.at/pub/papers/slany/flai.txt.

[9] Garnino A. Korn. *Neural Networks and Fuzzy-Logic Control on Personal Computers and Workstations*. The MIT Press, 1995.

[10] Bart Kosko. *Neural Networks and Fuzzy Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1992.

[11] Raúl Rojas. *Theorie der neuronalen Netze eine systematische Einführung.* Springer-Verlag Berlin Heidelberg New York, 1993. In German.

[12] Wolfgang Slany. Fuzzy scheduling. CD-Technical Report 94/66, Christian Doppler Laboratory for Expert Systems, Technical University of Vienna, 1994. URL: ftp://ftp.dbai.tuwien.ac.at/pub/papers/slany/cd-tr9466.ps.gz.

[13] Wolfgang Slany. Scheduling as a fuzzy multiple criteria optimization problem. *Fuzzy Sets and Systems*, 78:197–222, March 1996. Issue 2. Special Issue on Fuzzy Multiple Criteria Decision Making; URL: ftp://ftp.dbai.tuwien.ac.at/pub/papers/slany/fss96.ps.gz.

[14] Thomas Tilli. *Fuzzy-Logik Grundlagen, anwendungen, Hard- und Software.* Franzis-Verlag GmbH & Co.KG, München, 1992.

[15] Constantin von Altrock. *Fuzzy Logic and NeuroFuzzy Applications Expained.* Prentice Hall PTR, 1995.

[16] R. R. Yager, S. Ovchinnikov, R. M. Tong, and H. T. Nguyen, editors. *Fuzzy Sets and Applications: Selected Papers by L. A. Zadeh.* John Wiley & Sons, 1987.

[17] Lotfi A. Zadeh. Fuzzy sets. *Information and Control, New York: Academic Press.*, 8:338–353, 1965. Republished in [16].

[18] Hans-Jürgen Zimmermann. *Fuzzy Set Theory — and Its Applications.* Kluwer Academic Publishers, 2nd, revised edition, 1991.

A IJCAI'95 Report

Report on the Workshop on Fuzzy Logic in Artificial Intelligence (FLAI'95) held at IJCAI'95, Montreal, Canada

For a course on practical programming of my computer science studies at the Vienna University of Technology, I wrote a fuzzy logic inference library named FLIP++. At the beginning of 1995 I submitted, together with two colleagues, Stefan Mayer and Andreas Raggl, as well our teacher Wolfgang Slany, an article about this library and we were invited to a workshop on Fuzzy Logic in Artificial Intelligence at IJCAI'95, which my two colleagues and I attended in August 1995.

A.1 The Workshop

At the workshop on Fuzzy Logic in Artificial Intelligence, I presented our paper. There I could see the 'state of the art' of the scientific research in our field. In the following I will give a brief description of the workshop presentations ordered by topics:

A.1.1 Medical applications

Christoph S. Herrmann talked about a hybrid system making diagnoses based on EEGs. He uses fuzzy logic as an interface between different AI tools. He trained the system mainly by learning rules.

Another medical application was Lawrence O. Hall's KB-system, which used magnetic resonance slices for brain tumor diagnoses. It checks MR pictures for certain characteristics using fuzzy clustering.

A.1.2 Image processing

Anca L. Ralescu, who was one of the organizers of the workshop, described her work concerning object cognition, an algorithm for grouping line segments. Kenneth H.L. Ho and Noboru Ohnishi prepared another work on image processing: edge detection using fuzzy categorization and classification.

Scheduling

Alexander Ptuskin and Eugene Levner prepared a paper dealing with scheduling of a transportation robot in a production flow line with incomplete knowledge modeled by fuzzy numbers.

A.1.3 Case based reasoning

Neil Hurley presented his case-based reasoning approach for scaling numerical simulation systems, where he tries to find an understandable qualitative symbolic description that properly encapsulates the significant features of a complex mathematical model based on partial differential equations.

A.1.4 Fuzzy expert systems

Another field of application, namely fuzzy expert systems, was introduced by Eugene Roventa. In his talk he presented the way to generally structure an expert system using fuzzy sets to represent uncertain information, fuzzy inference mechanism, and constraint logic programming.

A.1.5 Fundamentals

Ronald R. Yager presented a talk in which he showed the way to apply ideas from nonmonotonic reasoning to improve the representational power of fuzzy systems. He built a hierarchical prioritized structure whereby priorities vary and special aggregation techniques must be applied.

Henri Prade found that fuzzy sets combined with possibility theory provide a basis for modeling both preferences and uncertainty. His approach, thus, is a qualitative and non-probabilistic view of decision making processes.

A.1.6 Membership functions

Taner Bilgic's presentation dealt with the problem measuring the membership relation within one linguistic variable or between two linguistic variables.

Another interesting topic concerning membership functions was presented

by Francesco Bergadano. He illustrated how membership functions can be learnt from positive and negative examples and from predefined set of classification rules.

Listening to these presentations was a invaluable experience for me. Furthermore, I had also the opportunity to get in touch with the researchers personally which led to some very interesting discussions.

A.1.7 Further Information on the FLAI'95 Workshop

It is planned by the organizers of the workshop to publish a collection of extended and revised versions of selected workshop papers after reviewing them with Springer Verlag in the Lecture Notes in AI series during the summer of 1996. Please contact the editors Anca Ralescu (ralescd@ucunix.san.uc.edu) and Trevor Martin (Trevor.Martin@bris.ac.uk) for details.

A.2 The IJCAI'95 Conference

Besides our workshop, I was able to atted many interesting talks in other fields of Artificial Intelligence such as Hybrid Systems, Learning, Knowledge Representation, and Autonomous Agents, as well as some more philosophical talks. Some of them were of interest because of my diploma thesis, some of them made me think about many of these other fields. It was also very interesting to be present when the pioneers of AI, like John McCarthy, Marvin L. Minsky, Edward A. Feigenbaum, and Herbert A. Simon all came together again for the first time in decades and talked about the principles of AI.

The conference was well-organized quite; if one had some special concerns, the organizers were very friendly and helped you immediately. The conference was located near the old town of Montreal, and it was always easy to find a place to relax after the conference program. Besides the conference, we got a chance to see a little bit of the city and countryside, and to get in contact with the people in Quebec. At the workshop I found many interesting ideas which will help me with my diploma thesis. And I also met some people who will support me with their experiences in Artificial Intelligence. The conference on the whole was very useful for me and I really enjoyed it. It provided a good motivation to write a good diploma thesis. And it gave me serious considerations remaining in the scientific community.

B Fuzzy tools comparison

Fuzzy logic simplifies engineering tasks in the rapidly growing area of microprocessor-controlled products, in industries such as automotive, aerospace, consumer electronics and appliances, industrial automation and process control equipment, and medical electronics. With the explosion of embedded microprocessors in consumer and industrial products, new systems and components become even more sophisticated and compact every year. In many situations, conventional design methods are neither cost-effective nor robust enough, which can hamper a company's ability to develop quickly the products consumers demand. Fuzzy logic technology simplifies the design and manufacturing of complex products by using simple English statements to specify rules for system behavior.

Unlike conventional programming logic or expert systems, fuzzy logic statements may be partially true, which results in more accurate representations of real-world systems. Fuzzy logic allows engineers to design products by specifying systems and data with attributes such as *warm*, *faster*, and *slippery*. This capability streamlines the engineering process and produces simpler, less expensive, and much more reliable products.

Fuzzy logic provides solutions whenever vague or conflicting information must be handled. This is often the case in control, in information processing, expert systems, financial forecasting and decision support, to name the prime examples. Fuzzy logic systems are rule-based systems that generally consist of a collection of rules of the following form:

$$\textit{If (A is HIGH) AND (B is MEDIUM) THEN C = MEDIUM}$$

where A and B are the input variables to the system, C is the output variable, and *HIGH* and *MEDIUM* are fuzzy sets, or membership functions, defined on the input and output variables.

First we will give a short overview about the reasons for the use of fuzzy logic within this project.

- Understandability

  The reason why we want to use fuzzy logic for the realization of the desulfurization control is to give the system a better understandability so that it is always possible to understand what the control is doing and to gain more insight into the process by following the automatic

process control. We want not a black box, where there is no possibility of human control over the system, we want to build up a system that is transparent to the field expert and will gain a higher degree of confidence by the user.

- Dealing with imprecise knowledge and nonlinear systems

  The process of desulfurization is very complex and nonlinear. To learn about this process we use a neural net and then we take a fuzzy controller to control this nonlinear task. Fuzzy logic allows us to control a complex nonlinear process by building up a representation of the functionality of the task through a number of rules that hold the relations learned by the neural net.

- Using human linguistic knowledge from experts

  All the a-priori expert knowledge can also be included into the process control by writing it down in linguistic rules and combining it with the relations learned by the neural net. So we can use experience which is successfully used now to control this process. Also, the expert can get a deeper insight into the process by learning from the relations and rules found by the neural net. This connection between expert and artificial intelligence methods should provide a very stable and reliable system for a successful and efficient control of the desulfurization process.

B.1 Criteria for the selection of a fuzzy logic tool

Below we list up some criteria that should be considered in the selection of a fuzzy tool. Some of these criteria are of more common nature, others are of special interest for the desulfurization project. Not all of these criteria of the same importance. I was unable to find out about all these criteria for all listed fuzzy tools.

- Are there learning facilities? Either neural nets or genetic algorithms or other learning tools. Here we need the possibility to learn relations between the recorded variables and their relevance from the existing data material.

- What can be learned? Is it possible to learn the rules, the membership functions, or other parameters? Or can all of them be trained? What has to be learned, however, depends on the model and the task.

- Is it possible to build a connection to SNNS (Stuttgarter Neural Net Simulator)? For the reason that we want to use SNNS within this project, we need a fuzzy logic tool, which can be connected to SNNS. A practical solution for the connection must be easy to handle. A complex solution for this problem would reduce the usability of the whole control system.

- How can such a connection be implemented? Is it possible to build up system with a single easy-to-use graphical user interface that can control both, SNNS and the fuzzy tool? Can the data transfer between them be hidden behind this GUI?

- Does the system have an easy-to-use graphical user interface at all? A GUI would enable to display always the relevant information about the control and the process. Also, in each situation the adequate functions and operations could be offered to the operator or expert.

- Is it possible to include expert knowledge within the control system? Not all of the knowledge can be learned from the collected data. Thus the information known by the field experts is not to be neglected. This information needs to be included too, in order to receive an efficiently working control system.

- Can time dependent information be evaluated? Some information can only be estimated as function of time or derived from previous control cycles. Are there methods to deal with time?

- Is it possible to include numeric methods? Some information can be easier handled with numeric methods. And such information should also be dealt in numerical methods in order to reduce the complexity of the rule-base. It makes no sense to implement numeric methods in fuzzy logic. For each kind of relation, the appropriate method should be used.

- Are automatic data read-in and output methods supported? The information about the process to be controlled should be automatically read-in from the environment through a network. There should also be the possibility to send the calculated control value to the process. Are these facilities available?

- What are the environment conditions for this tool? What hardware does the fuzzy tool require? And under what operating systems can it be run? SNNS will be run under Linux, thereforee the fuzzy logic tool should be able to run within the same environment.

- Is it reliable in operation? The system should run in a stable and correct manner for long time. If a failure occurs, it should be easy to restart it again. Is it possible to prepare tables for controlling the process even if the hardware is not available for some time?

- Is there support of the system for years? How stable is the company that provides the tool? Will the tool be supported by the provider after years in operation? Is it possible to include new releases of the tool if they could improve the system performance?

- Is the tool able to run simulations? Is it possible to feed the system with manipulated input data to simulate different situations?

- Is there a possibility of tracing the process values or rules for viewing while the process runs? Can the process control and the process itself be monitored for the operator? This would enable the operator to make the right decisions if some unforeseeable happens.

- Is it possible to run the system after changes without compiling? At least the changes during operation should be accepted without recompiling the system. The least possible maintenance work for the process control system should be required.

- Under which license does the system run and what is the price thereof? What are the legal and financial expects of the tool?

Below listed are some products about which I found some information. However, to be listed here does not mean that the product also meets the mentioned criteria. The order of this list of tools does not speak to their quality. After these descriptions, more fuzzy tools are listed without any further details. Most of the information comes from users, news groups, mailing lists, and the companies themselves. Very helpful has been work done by Martin Brown of the Image, Speech and Intelligent Systems research group at the Department for Electronics and Computer Science of the University of Southampton and by Dieter Korus and Bernhard Schmidt from the University of Marburg.

B.2 MATLAB Fuzzy Logic Tool Box

- Description:
  MATLAB works fairly well for designing and implementing simple fuzzy logic systems where the relationships between the variables are

well understood and the system can be designed with a minimum of testing and tuning iterations. Serious or complex systems, especially where the relationships are not well understood and may require lots of testing and tuning iterations, are better served by other tools.

- Environment:
  The MATLAB fuzzy logic toolbox has a particular limitation in that it runs under the MATLAB environment rather than in native machine code; this makes it less responsive than tools that are compiled and linked for a particular machine. However, its corresponding advantage is that it can be run on any platform that MATLAB can run. The Fuzzy Logic Toolbox integrates with and extends the MathWork's MATLAB Technical Computing Environment (TCE) and its SIMULINK graphical simulation environment, to allow customers to explore and apply fuzzy logic design while working in MATLAB framework. MATLAB is required to run the Fuzzy Logic Toolbox. The Fuzzy Logic Toolbox is available for Microsoft Windows-based personal computers, Apple Macintosh computers, and many popular UNIX workstations.

- Simulations:
  SIMULINK enables engineers to simulate fuzzy designs early in development, evaluate them in the context of a dynamic system model, and generate real-time C code with the SIMULINK real-time Workshop.

- Editors:
  Five GUI-driven editors for each phase of fuzzy logic design and analysis are available: The Fuzzy Interface System (FIS) Editor is used for graphical definition of overall system structure and variables, and serves as a starting point for other design activities. The Membership Function Editor is a tool for drawing the shape of the membership functions that specify fuzzy degrees of truth for each variable in the system. The Rule Editor enables the user to specify logical rules in plain English, and view them in English or in a more concise format.

  The Fuzzy Inference Viewer is an interactive tool that allows users to explore visually how the system responds to combinations of inputs or changes in inputs. The Surface Viewer displays the system output as a 3D surface, which enables users to view the overall system behavior.

- Inference:
  Adaptive Neuro-Fuzzy Inference Systems (ANFIS) technique is applied, which uses training data and neural network methods to improve the performance of fuzzy systems. Two methods of fuzzy clustering, for pattern recognition are fuzzy C-means and subtractive clustering (an

extension of Yager and Filev's Mountain Method). Sugeno style fuzzy inferencing is implemented which enables greater computational efficiency and more rigorous mathematical analysis.

- Output:
  The product generates machine independent, ANSI-standard C code, which can run on either real-time embedded hardware or general-purpose computers.

- Distribution:
  Scientific Computers GmbH, Franzstrasse 107, 52064 Aachen, Germany
  Phone:0241/47075-0, Fax: 0241/44983
  Price: from 2200DM for PC/Mac up to 6100DM for workstations
  email: info@scientific.de
  WWW: http://www.scientific.de

- Remarks:
  Even people using MATLAB are often not satisfied with the MATLAB fuzzy logic toolbox because of its complexity and lack of speed even on fast computers. The MATLAB fuzzy logic tool-box is not a good choice for a tool to do fuzzy logic development because it was developed primarily as an add-on package to MATLAB.

B.3 fuzzyTECH

- Applications:
  Fuzzy control, data analysis, XPS, decision-making

- Description:
  Graphical developing environment for fuzzy systems for PC and micro-controller, additional packages for neuro-fuzzy and data analysis

- Environment:
  MS-Windows, at least 386, 10MB HD, 4MB RAM

- Membership Functions:
  Polygon, Gauss, user-defined

- Inference:
  Minimum/maximum, min-average, min-max, FAM

- Editors:
  During design phase, fuzzyTECH's graphical editors allow the user to specify a system by point-and-click. Initial optimization of the system

design is supported by its analyzer and simulation tools in the debugging phase. Interactive 'what-if' tracing is provided for optimization. fuzzyTECH's browsers are allowed to verify the design with real process data.

- Representations:
  Graphical-interactive generation of the fuzzy system, representation of the system as structogramm

- Output:
  To implement the developed system, fuzzyTECH provides multiple code generators for C and assembly as well as interfaces to most PLC, VME, and industrial process control systems. Also, dedicated fuzzy logic processors are supported.

- Interfaces:
  fuzzyTECH is based on FTL, the hardware-independent fuzzy logic system description language that provides open links to other design tools, programming languages and simulation environments. FTL is the basic technology behind upcoming new fuzzy systems standard by the IEC board of standardization. In addition, fuzzyTECH provides built-in interfaces to most other MS-Windows-based software by DDE, DLL and remote control interfaces. For some software packages, special interfaces are available in addition to standard DDE/DLL interfaces:

  A VisualBasic source module allows the user to tie in fuzzy logic functions and even complete editors and analyzers of fuzzyTECH by 'point-and-click'. fuzzyTECH comes with an Application Programming Interface that allows users to remotely control most functions in fuzzyTECH from other software. This API is provided as a DLL called FTRCU (fuzzyTECH Remote Control Unit) which contains all the functions needed to call in the software to completely integrate fuzzyTECH within another program. The built-in M code generation provided by all fuzzyTECH Editions generate fuzzy logic control blocks that can be used with most control systems simulation software such as VisSim and MATLAB/SIMULINK.

  An interactive interface allows users to tie in the editors and analyzers of fuzzyTECH as building blocks in software such as VisSim and MS-Excel. For VisSim, the complete interface modules are provided with every version of VisSim. After installing the two software packages on the same PC, both software packages automatically install the necessary functionality.

For MS-Excel, fuzzyTECH comes with the "fuzzyTECH Assistant", an Add-In module that if installed within MS-Excel, allows users to link cells and arrays of cells by fuzzy logic functions the same way as MS-Excel's mathematical functions are used to link cells.

- Documentation:
  Reference hand-book, 210 pages, extra manual for additional modules (English, German), online help

- Distribution:
  INFORM GmbH,
  Pascalstr. 23, Aachen, Germany;
  Various editions are available: "Precompiler", "Online", "Business", as well as for the generation of special Code for micro-controller the "MCU- *- Editions",
  demo version: not directly available, included in the book "Fuzzy Logic - Werkzeuge", v. Altrock (v. Altrock is member of INFORM GmbH) et. al., Oldenburg Verlag 1994, ISBN 3-486-22693-2 or in [15].

- Learning:
  Term definitions of the linguistic variables can be learned from examples. A neuro-fuzzy module allows users to learn the correct rules from existing data. Neural network technologies are integrated to automatically generate or optimize complete fuzzy logic systems out of sample data with fuzzyTECH's neuro-fuzzy module. Dedicated for fuzzy data analysis applications, the data-analyzer module provides all tools for conventional signal processing and data analysis.

- Remarks:
  Different editions of fuzzyTECH come pre-configured to compliment the specific needs of the target hardware platform:

  - fuzzyTECH Precompiler Edition:

    For all applications in engineering, control, and technical data analysis. Generates portable C code.

  - fuzzyTECH Online Edition:

    In addition to the Precompiler Edition functionality, the Online Edition allows for the modification of the running system on the target hardware without halting the process.

  - fuzzyTECH MCU Editions:

    Developed in cooperation with the leading silicon manufacturers, the MCU Editions provide optimized assembly code generation of

the fuzzy logic system on standard micro-controllers, DSPs, and fuzzy processors.

- fuzzyTECH Editions for Special Hardware:
  In addition, specialized fuzzyTECH Editions are available for industrial process controller, PLC, and VME components as well as for commercially available fuzzy processing ASICS (Fuzzy Processors).

Inform Software Corporation has released a new fuzzy-cluster module to go with their line of fuzzyTECH software development tools. The module incorporates two clustering algorithms. One is a standard clustering algorithm known as IsoData clustering and the other is Inform's own fuzzy-cluster algorithm. The later clustering technique uses a fuzzy logic inference to cluster data.

B.4 SieFuzzy

- SieFuzzy/TILShell:
  SieFuzzy and the beta version of TILShell were jointly developed by Siemens and Togai InfraLogic. However, the commercial release of TILShell (after the joint development agreement expired) includes many additional features not in SieFuzzy – the most significant of which is the TILGen neural network tool that is in TILShell. Siemens has recently added some new features to SieFuzzy but it still is not as fully featured as TILShell.

- Applications:
  Fuzzy control

- Description:
  SieFuzzy is a CAE Tool from Siemens for fuzzy system design. It is a software development for designing fuzzy logic systems, for testing these systems through simulation, for online testing (compiled code), and for compiling the resulting system into any variant of C code.

- Environment/Interfaces:
  SieFuzzy is for IBM-Windows targets only. However, the resultant code from the tool can be used on virtually any machine or system with an ANSI-C compiler. SieFuzzy runs under Windows 3.x and Windows 95. A Linux version is not available.

  It supports Windows DLL and DDE links for integration into the graphical user interface with other packages. It offers a mex file generator

for executing fuzzy systems within MATLAB. It can still be tuned by routines as fminu and a DDE interface for interactive combination of MATLAB/SIMULINK and SieFuzzy on one screen. SieFuzzy includes features for interfacing to standard software packages (e.g. MATLAB/SIMULINK and Mathematica) and for user extensions (e.g. user defined operators, user defined objects). SieFuzzy 2.0 is the compatible revised update version of SieFuzzy 1.0. SieFuzzy 2.0 is also upward compatible with TILShell 3.0 and all earlier versions of TILShell.

The resulting fuzzy system can either take the shape of a stand-alone application or it can be embedded into an existing application. In SieFuzzy, the description of a fuzzy system is represented in the Fuzzy Programming Language (FPL). Each time a change is made through one of the editors, the FPL description of the system is updated to reflect the changes made. Both SieFuzzy and FPL support unlimited complexity of fuzzy systems, e.g. an arbitrary number of variables, membership functions and rules can be used.

FPL is a registered trademark of Togai InfraLogic, Inc.

- Editors:
  SieFuzzy comes with graphical editors for each part of a fuzzy logic knowledge base: variables, membership functions, and rules. These editors allow users to enter and modify descriptions of variables, membership functions, rules, and the connections between variables and rule-basis in a graphical point and click environment.

- Representations/Simulation:
  Once the system has been completely described, it can be tested using the various debugging and tuning facilities. These include static and dynamic testing facilities. For static testing, it is possible to enter specific values for the inputs of a system and then observe the values of the outputs (Quick Test), or plot a 3-dimensional representation of the relationship between two inputs and one output of the system over a given range of input values (Control Surface). For dynamic testing, it is possible to test a system as it runs with a simulation of the final application and observe the behavior of each rule, membership function, and their interactions. Or as a final test, it is possible to link SieFuzzy to an external processor that is actually running the final application, and make changes to the system "online" and observe the results.

- Learning:
  SieFuzzy includes features for automatic optimization of fuzzy systems. It is possible to learn the rules from existing data. SieFuzzy has a neuro

fuzzy component. An initial fuzzy system is transformed 1:1 into a neural net, this can be trained with existing data and then transformed back into an optimized fuzzy system.

- Documentation:
  SieFuzzy comes with a 400 page manual containing a tutorial and reference.

- Distribution:
  SieFuzzy is available as fully featured Version 2.0 or as highly-discounted Version 2.0 Educational without the online capability (universities only).

B.5 TILShell

- Applications:
  Fuzzy Control

- Description:
  TILShell is a graphical developing environment for the generation and simulation of fuzzy systems basing on FPL (Fuzzy Programming Language). The controlled system can be mathematically described and simulated. It is a Windows-based software development tool providing users with a way to design, debug, and test fuzzy logic expert systems, including control systems.

  The TILShell family offers the following features:

  - Real-time on-line debugging and tuning of rules, membership functions and rule weights, including addition and deletion of rules;
  - Fully integrated graphical simulation of fuzzy systems and conventional methods;
  - ANSI and K&R C code generation from the fuzzy-C compiler;

- Environment/Interfaces:
  TILShell is for IBM-Windows targets only. The TILShell runs under Windows 3.1 or Windows 95 and it does not run under Linux.

  However, the resultant code from the tool can be used on virtually any machine or system with an ANSI-C compiler.

  TILShell supports Windows DLL and DDE links for integration into the graphical user interface with other packages. TILShell includes a

number of examples for adding DLL and DDE links. DDE and DLL are standard Windows interfaces.

TILShell provides the source code for the fuzzy logic algorithms in a library format so they can be recompiled for use on virtually any target system. What the tool generates is the specific rule base, membership function, and inferencing logic and subroutine calls to the various library routines. This keeps the individual fuzzy source files small – once the libraries are compiled for a particular target, they need only be linked with a small source file that has to be recompiled for each iteration.

- Membership Functions:
  Polygon, s-form, singleton, analytical; Membership functions can be analytically described with almost all commonly used functions.

- Inference:
  Max-Min, Max-Prod, Mamdani, Tagaki-Sugano, analytically user-defined

- Operators:
  Min, max, complement, analytically user-defined

- Defuzzification:
  Center of gravity, height, analytical user-defined

- Editors:
  The TILShell offers tuning of rules, membership functions and rule weights, including addition and deletion of rules. It offers a graphical object-based "point and click" CASE tool and 3-D color rule-base and operator visualization tools. TILShell provides editors for the generation of a simulations system and input mask for the check of the simulation model through input values. Furthermore,, there are input masks for the definition of the whole fuzzy systems and editors for the description through FPL.

- Representations:
  TILShell provides graphical representation of the (not-fuzzy) in- and output values at a simulation, 3D-representation of the membership functions, and representation of the fuzzy system as a structure tree.

- Learning:
  TILShell has the inherent ability to learn rules from existing data using the TILGen neural network tool. It is possible to accomplish this with the other tools and external neural network tools, but it requires

external programming and work. The TILGen feature extracts rules from data vectors.

- Documentation:
  300 pages, English, online help

- Distribution:
  Togai InfraLogic GmbH,
  Rosenstr.7, 8000 München 2
  Phone:(089) 231138-67, Fax: -11
  email: info@til.com or
  fuzzy-server@til.com (with subject or message-body "help")
  ftp: ftp.hiof.no/pub/Fuzzy/togai
  TILShell family is available in 4 editions:

  - TILShell 3.0 Professional Edition
    Designed for the user involved in intelligent control or systems applications, TILShell Professional Edition (PE) allows on-line debugging, DDE support, user-defined extensions, as well as automatic rule-base generation.
  - TILShell 3.0 Standard Edition
    Designed for the user requiring an enhanced graphical software development environment for creating fuzzy expert systems, TILShell Standard Edition (SE) includes the fuzzy-C development system for generating C source code.
  - TILShell 3.0 Basic Edition
    Designed with the novice-programmer in mind, TILShell 3.0 Basic Edition (BE) makes it easier for these users to develop a fuzzy expert system by providing basic abilities to help design and implement a working fuzzy system.
  - TILShell +
    TILShell + has been the front-line TILShell family product for several years prior to the introduction of TILShell 3.0. It includes many of the features of TILShell 3.0 Standard Edition and still offers a powerful solution at an economical price for users with relatively simple design needs.

- The demo version of TILShell is very unstable.

- Remarks:
  SieFuzzy and the beta version of TILShell were jointly developed by Siemens and Togai InfraLogic. However, the commercial release of

TILShell (since the expiration of the joint development agreement) includes many additional features not in SieFuzzy – the most significant of which is the TILGen neural network tool that is in TILShell. Siemens has recently added some new features to SieFuzzy but it still is not as fully featured as TILShell.

B.6 NEFCLASS

- Applications:
  NEFCLASS is used for data analysis by neuro-fuzzy models.

- Description:
  NEFCLASS is an abbreviation for neural fuzzy classification. NEFCLASS-PC is software for classifying data by means of a neuro-fuzzy model NEFCLASS. NEFCLASS-PC is software for MS-DOS Personal Computers to develop, train and test neuro-fuzzy networks for classification. This approach is based on a generic model of a fuzzy perceptron that can be used to derive fuzzy neural networks or neural fuzzy systems for specific domains. It derives fuzzy rules from data to classify patterns into a number of (crisp) classes. NEFCLASS uses a supervised learning algorithm based on fuzzy error back-propagation that is used in other derivations of the fuzzy perceptron.

- Environment:
  NEFCLASS-PC is software for MS-DOS Personal Computers. VGA is needed for graphical display. It runs on MS-DOS PC (80286 and above).

- Membership Functions:
  Triangle

- Inference:
  Neuronal algorithm basing on fuzzy *IF THEN* rules

- Defuzzification:
  Maximum, center of gravity

- Editors:
  In tables

- Representations:
  Membership functions single and against each other;

- Learning:
  NEFCLASS uses a supervised learning algorithm based on fuzzy error back-propagation that is used in other derivations of the fuzzy perceptron.

- Documentation:
  Various papers for the representation of the theory and short introduction, online help

- Distribution:
  NEFCLASS-PC is freely available via WWW:
  WWW: http://www.cs.tu-bs.de/ñauck
  NEFCLASS comes without any warranty, and is copyrighted software.

- Remarks:
  NEFCLASS is usable only for analyzing smaller problems because of its net architecture. NEFCLASS-PC may be used free of charge for scientific, educational or personal purposes, but it is not allowed for commercial purposes.

B.7 FUZZLE

- Description:
  FUZZLE is an expert system development shell based on fuzzy logic. One of its main features is the generic execution module that does not require any programming effort or compiler. Once the development is completed, the user can execute the fuzzy inference engine by simply entering inputs via keyboard or mouse.

- Environment/Interfaces:
  FUZZLE runs on IBM compatible personal computers. It offers the capability of creating customized Windows software without using a compiler. Once a development is completed, the user can click the mouse on the customize button. It generates a new software by capturing the fuzzy inference engine developed by the user. It also provides source codes in C and FORTRAN for embedded programming.

- Editors:
  FUZZLE supports GUI to develop the basic elements of a fuzzy expert system such as membership functions and fuzzy rules. Another feature is the mixed input processing capability. This capability allows the user to enter linguistic inputs as well as numerical inputs within the same execution step. It is an environment computed with words.

- Distribution:
  FUZZLE is developed by MODICO, Inc.

B.8 Mathematica Toolbox

- Description:
  Add-on packet for Mathematica
- Environment:
  All platforms for which Mathematica is available
- Membership Functions:
  Triangular, trapezoid, Gaussian, user-defined
- Inference:
  Max-Min, Max-dot, Max-star,
- Operators:
  Average, Hamacher, Frank, Yager, ...

B.9 DataEngine

- Applications:
  Data analysis
- Description:
  DataEngine is a tool for the analysis of data with help of fuzzy controllers, fuzzy clustering methods, neuronal nets, and neuro-fuzzy systems. DataEngine V.i is an add-on module for LabVIEW MIT has developed DataEngine V.i as virtual instrument for fuzzy rule-based systems, fuzzy-clustering (fuzzy c-means) and Kohonen networks.
- Environment:
  PC (386/33 upwards, 8MB, recommended: 486/33, 16MB) or
  SUN SPARC II (Motif); This toolbox under LabVIEW is running on Windows and Mac.
- Membership Functions:
  Polygon, trapeze, triangle
- Operators:
  Min, max, product, sum, gamma

- Editors:
  Membership functions, rules in tables

- Representations:
  Structure of the fuzzy system, rule input, membership functions

- Output:
  Treated data and control system as C-code (with runtime license under DataEngine ADL)

- Documentation:
  200 pages, German

- Distribution:
  MIT Management Intelligenter Technologien GmbH, Karl Lieven,
  Promenade 9, 52076 Aachen, Germany,
  Phone: +49-2408-945810 Fax: +49-2408-94582
  E-mail: kl@mitgmbh.de
  WWW: http://www.mitgmbh.de
  test license (390DM, 30 days full version)
  full license from 5990DM for PC to 11900DM for workstations for DataEngine,
  990DM for a runtime license under DataEngine ADL,
  2490DM for DataEngine V.i (only PC)

- Remarks:
  No online help is available. The present version runs very unstable. Fuzzy Clustering (FCM), signal treatment also conventional, signal input through measuring value input card or ASCII-Files, are included. The use of the neuro-fuzzy-add-on modules (MLP, Kohonen, backpropagation, unsupervised) is not clear.

B.10 FOOL & FOX

- Applications:
  Fuzzy-control, simulation, teaching

- Description:
  FOOL & FOX is a fuzzy system development tool. This fuzzy-tool is a packet consisting of the products FOOL and FOX, where FOOL serves as graphical front-end for the fuzzy-controller FOX. FOOL is a GUI for the construction of a data basis, which determines the behavior of the fuzzy-controller FOX. Here the variables, rules and so on are

determined. The generated data basis is stored in a text file. FOX, the actual fuzzy-controller, calculates the output, with which the system is controlled. It uses the data-base-files and the state parameters delivered by the system to be controlled.

- Environment:
  UNIX systems (SunOS, Linux, ...)
- Membership Functions:
  Triangle, trapezoid, value-table, s-function, pi-function, z-function
- Inference:
  T-norm, t-conorm, average, 36 different types
- Defuzzification:
  Center of gravity, height of defuzzification, first of maximums, last of maximums, middle of maximums
- Output:
  Crisp, fuzzy, both
- Editors:
  Input of the membership functions, rules and so on through masks.
- Representations:
  Representation of the membership functions via xprism of KHOROS or Gnuplot
- Output:
  Database-file, which is interpreted by FOX, the actual controller.
- Documentation:
  Online manual-pages and hand-book as postscript file (about 100 pages, English)
- Distribution:
  Uni-Oldenburg, Germany;
  It is freely available with source code via ftp and has to be compiled with an ANSI-C compiler and xview-libraries.
  ftp: ftp.informatik.uni-oldenburg.de/pub/fool
  WWW: http://condor.informatik.uni-oldenburg.de/FOOL.html
  email: fool@informatik.uni-oldenburg.de
- Remarks:
  This tool is for a professional use to strongly restricted in its functionality. For example, no more-level fuzzy rules are possible. In the

teaching area, this tool holds for sure enough functions to get a good impression of fuzzy-logic. The hand-book for it is complete and easily understandable.

B.11 FUZZ-C

- Applications:
  Control
- Description:
  Pre-processor for the generation of C-Code
- Environment:
  IBM-PC
- Membership Functions:
  Singletons, triangular, trapezoid, polygon, analytical
- Defuzzification:
  Center of gravity, left-max, right-max, max-average, user-defined.
- Editors:
  Pure text editor
- Documentation:
  English
- Distribution:
  ByteCraft, Canada
  Accel Infotech Spose Pte Ltd.
  one year free Update;
  e-mail: accel@solomon.technet.sg
- Remarks:
  No online help, no graphical user interface,
  Review: AI-Expert, March 1993

B.12 FIDE

- Description:
  Graphical developing environment for fuzzy-expert-systems

- Output:
  Code-generator for different micro-controller
- Distribution:
  Aptronix, Inc., USA
- Remarks:
  No longer available.

B.13 DBFuzzy

- Applications:
  Fuzzy-data collection
- Description:
  Data-base application for dBase- data-basis
- Distribution:
  Aria Ltd., Slovakia

B.14 FRIL

- Description:
  Fuzzy-logic programming language
- Environment:
  UNIX, Mac, MS-DOS, Win 3.1
- Distribution:
  Developed by Jim Baldwin at Bristol University, UK.
  Fril Systems Ltd., Bristol Business Center, Maggs House,
  78 Queens Rd, Bristol BS8 1Qx, UK.

B.15 FuzzySoft

- Description:
  Generator for C-code, simulations possibilities
- Environment:
  MS-Windows

- Distribution:
  GTS Trautzl GmbH, Gottlieb-Daimler-Str.9,
  W-2358 Kaltenkirchen/Hamburg
  Phone:(04191) 8 -711, Fax: -8665

B.16 Fuzzy Systems MATLAB-Toolbox

- Applications:
  Decision making, control, data analysis
- Environment:
  PC, Mac
- Distribution:
  PWS Publishing Company
  Demo version ftp: ftp.pws.com/pws/engr/fuzzy/

B.17 FuzzyControlDesk, FuzzyDecisionDesk, FuzzyPatternDesk

- Applications:
  Fuzzy control
- Description:
  Fuzzy tool for the generation of C-code, also for micro-controller 80C51, 80C166
- Environment:
  PC
- Membership Functions:
  Polygon, analytic functions
- Inference:
  Max-min, max-product
- Defuzzification:
  Center of gravity
- Editors:
  Text, table, graphical rule editor

- Documentation:
  169 pages, German, online-help
- Distribution:
  Fuzzy-Demonstrations-Zentrum, Fuzzy Logik Systems GmbH,
  Joseph-von-Fraunhofer-Str.20, 44227 Dortmund
  Phone:
  (0231) 9700 -921, Fax: -929

B.18 NeuFuz

- Applications:
  Common description fuzzy tool for learning a fuzzy system with neuronal nets;
- Environment:
  PC
- Membership functions:
  Gauss
- Inference:
  Min
- Remarks:
  MLP, feed-forward, back-propagation-learning
- Distribution:
  National Semiconductor Corp., USA

B.19 Fuzzy Box

- Applications:
  Control
- Description:
  Fuzzy control tool for micro-controller
- Environment:
  PC
- Membership Functions:
  Trapezoid, triangular, singleton, Dil-, Kon-, Int-Op.

- Inference:
  Max-min; min and prod- operators
- Defuzzification:
  COM
- Representations:
  Structure-, membership function- and rule editor
- Documentation:
  40 pages, German, Micro 4 about 30 pages, German
- Distribution:
  ZeTec GmbH

B.20 IlmenauerFuzzyTool

- Applications:
  Control
- Description:
  Fuzzy Tool for the generation of C-code, micro-controller
- Environment:
  PC, MS-Windows
- Membership Functions:
  Triangular, trapezoid, singleton
- Inference:
  Max-min, max-prod
- Operators:
  Min, max, complement
- Defuzzification:
  Center of gravity
- Editors:
  Text
- Documentation:
  100 pages, German, online help
- Distribution:
  SEI GmbH Ilmenau

- Remarks:
  Optimizing algorithm for fuzzy system

B.21 FLIP++

- Description:
  FLIP++ is a fuzzy logic inference processor library. This C++ library was developed to perform calculations with uncertain data and priorities. It evaluates various types of rules, having different importance, from a hierarchy of rules for uncertain or precise data. It evaluates only those parts of the hierarchy needed for the result. The result itself has two forms: first a list of membership functions, and second a crisp value for fuzzy control. The library also contains the whole functionality needed to manage one or more sets of rules on all data. It is built to be easily adapted to new applications. Due to its quite general concept of classes it is possible to include it, for the purpose of fuzzy computations, in other systems.

- Environment/Interfaces:
  FLIP++ runs on unix platform. It is a C++ library and can be linked together with any other software where fuzzy calculations are needed. All the data needed for fuzzy calculations are store in FPL.

- Membership Functions:
  Point-list

- Operators:
  t-norm, t-conorm, compensatory, gamma, ACOtan associative-compensatory

- Defuzzification:
  Center of gravity

- Editors/Representation:
  The InterFLIP++ library is the GUI for the FLIP++ library for the rule base, the membership functions, and all other parameters.

- Learning:
  NeuroFLIP++ is the connection from FLIP++ to SNNS (the Stuttgarter Neural Net Simulator). It is possible to learn the rules from existing data with this tool.

- Documentation:
  DocuFLIP++ is documentation about FLIP++ and the other parts of the *FLIP++ library as library description, interface description, and sample code in English at WWW: http://www.dbai.tuwien.ac.at/StarFLIP/DocuFLIP.html

- Distribution:
  FLIP++ is released by the Department for Information Systems at the Technical University of Vienna/Austria. And it is supported by this institute. It is copyright software and released with the Gnu public license.
  mailto:wsi@dbai.tuwien.ac.at
  Phone: ++43-1-58001-6123

- Remark:
  FLIP++ is part of the *FLIP++ library for real-world decision-making. This allows optimizing under vague constraints of different importance using uncertain data, where compromises between antagonistic criteria can be modeled. Typical application areas include scheduling, design, configuration, planning, and classification.

B.22 Other Fuzzy Logic and Neuro Fuzzy Software

- The neuro-fuzzy toolbox for NeuFrame by Neural Computer Sciences.
- ECANSE Environment for Computer Aided Neural Software Engineering by Siemens.
- Assist Visual Designer a fuzzy knowledge based system generator for Windows.
- O'INCA Design Framework for windows.
- FLINT fuzzy logic tool-kit by Logic Programming Associates.
- CubiCalc products by HyperLogic:
  CubiCalc, CubiQuick, CubiCard, CubiCalc RTC, CubiCalc, RuleMaker
- Fuzzy Symbolic Connectionist Network by Universal Problem Solvers.
- Fuzzy Knowledge Builder, Fuzzy Judgment Maker and the Fuzzy Thought Amplifier from Fuzzy Systems Engineering, Poway, CA.

- NeuroForcaster, Genetic and Visual Data by NIBS Inc.
- FLDE Fuzzy Logic Development Environment by Syndesis Ltd.
- INSiGHT IIe Fuzzy Development System by Adaptive Logic.
- FuzzyEXPERT 3M, Illmenau, Germany
- Package of MATLAB RBF routines and manual by Mark Orr.
- AUSIM A Simulator for Neural Networks and Massively Parallel AI, at Queensland University of Technology.
- Fuzzy CLIPS Fuzzy logic Expert Systems shell developed at NASA.
- Fuzzy Logic in Exsys.
- FCTool Demo-software from Amira GmbH (in German).
- Fuzzy Logic software from University of New Mexico.
- FLASH Fuzzy Logic Applications Software Helper from Rigel Corporation.
- STB Software Catalog at NASA.
- CNCL Communication Networks Class Library.
- (Fuzzy) CLIPS at CMU's AI-expert system repository.
- OAK Repository, directory SimTel/msdos/ai/.
- Fuzzy clustering tool ICOT.
- FlexTool fuzzy and Genetic Algorithm toolboxes by Flexible Intelligent Group, LLC, USA
- FUDGE Motorola freeware packages.
- Fuzzy database query and face drawing demos by Omron.
- Fuzzy stock market candlesticks recognition software (with free demo version) by Science in Finance Ltd.
- Level 5 Quest data search and discovery tool using fuzzy logic.
- Non-Axiomatic Reasoning System demonstration by Pei Wang.
- General-Purpose Fuzzy Inference Library Ver. 3.0.

- FuNeGen 1.0 a fuzzy neural system capable of generating fuzzy classification systems (as C-code) from sample data.
- Fuzzy Arithmetic Library [VAX-based may have to FTP directly].
- PGAPack Parallel Genetic Algorithm Library.
- Decision Plus Nicesoft
- DORA-Fuzzy University of Dortmund, Germany
- FCMC Matlab-Toolbox Bogdan Kosanovic, Laboratory for Computational Neuroscience, University of Pittsburgh, USA
- FuziCalc FuziWare, Inc., USA
- Fuzzy Control Manager TransferTech GmbH, Braunschweig, Germany
- FuzzyNex Fuzzy Logic Systems GmbH, Dortmund, Germany
- FuzzySoft, FuzzySoft AG
- Hi-Flag HIWARE
- IFR-fuzzy Toolbox for MATLAB Institute of Flight Mechanics and Control, University of Stuttgart, Germany
- Mathematica Toolbox Wolfram Research, Inc., USA
- Metus Fuzzy Library The Metus Systems Group, USA
- NeuroGraph ASTRUM GmbH, Neuro-Fuzzy-Tools, Erlangen, Germany

B.23 Conclusion

With FLIP++ it is possible to fulfill the requirements for a successful and efficient control of the desulfurization process. We build up a connection to SNNS and then learn the rules of a complex and nonlinear system. With help of the field experts we include the expert knowledge in the rule-base. The time dependent input variables are prepared in an algorithmic data interface for their use in fuzzy control. Additionally we include the GNU plot library to support more graphical visualization of the control and the process.

All three parts, especially the SNNS and FLIP++ are C respectively C++ libraries and therefore can be easily connect and combined to one system.

With the InterFLIP++ library we have GUI for graphically representing and editing the control system. All mentioned libraries run on UNIX platform.

FLIP++ is released by the Department for Information Systems at the Technical University of Vienna. And it is supported by this institute. It is like SNNS copyright software and released with the Gnu public license.

C FLIP++ header files

C.1 flip.h

```
//
// $Id: flip.h,v 1.4 1996/09/22 11:25:55 bonner Exp $
//
// -------------
//  F L I P . H
// -------------
//
// flip.h includes fliprule.h includes flipoper.h includes flipcvar.h
//        includes flippara.h
// flippara.cc includes flippara.h
// flipcvar.cc includes flipcvar.h
// flipoper.cc includes flipoper.h
// fliprule.cc includes fliprule.h

#include "fliprule.h"
```

C.2 flippara.h

```
//
// $Id: flippara.h,v 1.7 1996/09/22 11:26:01 bonner Exp $
//
//
// ---------------------
//  F L I P P A R A . H
// ---------------------
//

#ifndef flippara
#define flippara

#include <iostream.h>
#include <fstream.h>
#include <strstream.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define tab '\t'
#define space ' '
#define sspace "  "

#define NAME_LENGTH 40
#define STRLENGTH 1000
typedef char name_string[NAME_LENGTH];
typedef name_string name;

enum colourtype {violet,blue,light_blue,turquoise,green,yellow,orange,
red,pink,white,black,grey,light_grey,dark_grey,dark_blue,light_green,
dark_green,light_red,dark_red,brown};

enum boolean {FALSE,TRUE};

//
//
// * * * * * * * * * *   N a m e   * * *
//
//

class Name {
```

```
friend class NList;

public:

// * * constructors:
Name(void) { strcpy(nam,"no_lingiustic_term"); next=NULL; };
Name(name init_name)  { strcpy(nam,init_name); next=NULL; };

// * * complex methods:
void show(void)              { cout << nam << space; };
int  write(ofstream *output) { *output << nam << space; return 1; };
int  read(ifstream *input)   { *input >> nam;           return 1; };

// * * simple methods to access data elements:
void set_name(name new_n)    { strcpy(nam,new_n); next=NULL; };

name *get_name(void)         { return &nam; };
Name *get_next(void)         { return next; };

protected:
// * * data element:
name nam;
Name *next;
};

//
//
// * * * * * * * * * * N L i s t  * * *
//
//
// N a m e L i s t

class NList {

public:

// * * constructors:
NList(void)       { first=NULL; last=NULL; number=0; };
NList(int number);

virtual ~NList(void);

// * * complex methods:
void show(void);
void show_NList(void)             { show(); };
```

```
int  write(ofstream *output);
int  write_NList(ofstream *output) { return write(output); };
int  read(ifstream *input);
int  read_NList(ifstream *input)   { return read(input); };
void add(name new_name="no_linguistic_term");
void add_NList(name new_n="no_linguistic_term") { add(new_n); };
void copy(NList *n_list);
void copy_NList(NList *n_list)     { copy(n_list); };
Name *find(int index);
Name *find_NList(int index)        { return find(index); };
name *find_n(int index)            { return &(find(index)->nam); };
name *operator [] (int index)      { return find_n(index); };
void remove_all(void);

// * * simple methods to access data elements:
void set_first(Name *nf)         { first=nf; };
void set_first_NList(Name *nf)   { first=nf; };
void set_last(Name *nl)          { last=nl; };
void set_last_NList(Name *nl)    { last=nl; };
void set_number(int nn)          { number=nn; };
void set_number_NList(int nn)    { number=nn; };

Name *get_first(void)       { return first; };
Name *get_first_NList(void) { return first; };
Name *get_last(void)        { return last; };
Name *get_last_NList(void)  { return last; };
int  get_number(void)       { return number; };
int  get_number_NList(void) { return number; };

protected:
// * * data element:
Name *first;
Name *last;
int number;
};

//
//
// * * * * * * * * * *   C o l o u r   * * *
//
//

class Colour {
friend class CList;
```

```
public:

// * * constructors:
Colour(void)                    { colour=white;       next=NULL; };
Colour(colourtype init_colour) { colour=init_colour; next=NULL; };

// * * complex methods:
void show(void);
int  write(ofstream *output);
int  read(ifstream *input);

// * * simple methods to access data elements:
void set_colour(colourtype new_c) { colour=new_c; };

colourtype *get_colour(void) { return &colour; };
Colour     *get_next(void)   { return next; };

protected:
// * * data elements:
colourtype colour;
Colour     *next;
};

//
//
// * * * * * * * * * * *   C L i s t   * * *
//
//

class CList {

public:

// * * constructors:
CList(void)  { first=NULL; last=NULL; number=0; };
CList(int number);

virtual ~CList(void);

// * * complex methods:
void       show(void);
void       show_CList(void) { show(); };
void       rshow(Colour *c);
int        write(ofstream *output);
int        write_CList(ofstream *output) { return write(output); };
```

```
int        read(ifstream *input);
int        read_CList(ifstream *input) { return read(input); };
void       add(colourtype new_colour = white);
void       add_CList(colourtype new_c = white) { add(new_c); };
void    copy(CList *c_list);
void    copy_CList(CList *c_list) { copy(c_list); };
Colour     *find(int index);
Colour     *find_CList(int index) { return find(index); };
colourtype *find_ct(int index)  { return &(find(index)->colour); };
colourtype *operator [] (int index) { return find_ct(index); };
void remove_all(void);

// * * simple methods to to access data elements:
void set_first(Colour *nf)       { first=nf; };
void set_first_CList(Colour *nf) { first=nf; };
void set_last(Colour *nl)        { last=nl; };
void set_last_CList(Colour *nl)  { last=nl; };
void set_number(int nn)          { number=nn; };
void set_number_CList(int nn)    { number=nn; };

Colour *get_first(void)       { return first; };
Colour *get_first_CList(void) { return first; };
Colour *get_last(void)        { return last; };
Colour *get_last_CList(void)  { return last; };
int    get_number(void)       { return number; };
int    get_number_CList(void) { return number; };

protected:
// * * data elements:
Colour *first;
Colour *last;
int    number;
};

//
//
// * * * * * * * * * * *  P a r a m e t e r S e t B a s e  * * *
//
//

class ParameterSetBase {

public:
// * * constructor:
ParameterSetBase(void)
```

```
{ range_lower_end=-100.0; range_upper_end=100.0;
  median=0.0; distribution=0.95; dilatation=0,7;
  importance=1; };

ParameterSetBase(float new_range_lower_end,
float new_range_upper_end, float new_median,
float new_distribution,    float new_dilatation,
float new_importance );

// * * complex methods:
void         show(void);
void         show_PSB(void)              { show(); };
int          write(ofstream *output);
int          write_PSB(ofstream *output) { return write(output); };
int      read(ifstream *input);
int      read_PSB(ifstream *input)   { return read(input); };
void         copy(ParameterSetBase *psb);
void         copy_PSB(ParameterSetBase *psb) { copy(psb); };

// * * simple methods to access data elements:
void set_range_lower_end(float nrle) { range_lower_end=nrle; };
void set_range_upper_end(float nrue) { range_upper_end=nrue; };
void set_median(float nm)            { median=nm; };
void set_distribution(float nd);
void set_dilatation(float nd);
void set_importance(float ni);

float      get_range_lower_end(void) { return range_lower_end; }
float      get_range_upper_end(void) { return range_upper_end; };
float      get_median(void)          { return median; };
float      get_distribution(void)    { return distribution; };
float      get_dilatation(void)      { return dilatation; };
float      get_importance(void)      { return importance; };

protected:
// * * data elements:
float     range_lower_end;
float     range_upper_end;
float     median;
float     distribution;
float     dilatation;
float     importance;
};

//
```

```
//
// * * * * * * * * * * *  P a r a m e t e r S e t  * * *
//
//

class ParameterSet : public ParameterSetBase, public CList, public NList {
friend class PSList;

public:
// * * constructor:
ParameterSet(void);
ParameterSet(name new_ps_name, float new_range_lower_end,
 float new_range_upper_end, float new_median,
 float new_distribution, float new_dilatation,
 float new_importance,
 CList *new_c_list, NList *new_n_list );

// * * complex methods:
void         show(void);
int          write(ofstream *output);
ParameterSet *read(ifstream *input);
void         copy(ParameterSet *ps);

// simple methods to access data elements:
void  set_ps_name(name nn) { strcpy(ps_name,nn); };

char             *get_ps_name(void)   { return ps_name; };
ParameterSetBase *get_PSB(void);
CList  *get_CList(void);
NList            *get_NList(void);
ParameterSet     *get_next(void)      { return next; };

protected:
// * * data elements:
name ps_name;
ParameterSet *next;
};

//
//
// * * * * * * * * * * *  P S L i s t  * * *
//
//

class PSList {
```

```
public:

// * * constructors:
PSList(void) { first=NULL; last=NULL; number=0; };

// * * destructor:
~PSList(void);

// * * complex methods:
void show(void);
int  write(ofstream *output);
int  read(ifstream *input);
void add(void);
void add(ParameterSet *ps);
void add(PSList *psl);
ParameterSet *find(int index);
void remove(int index);

// * * simple methods to access data elements:
ParameterSet *get_first(void)       { return first; };
ParameterSet *get_last(void)        { return last; };
int         get_number(void)       { return number; };

// * * data elements:
ParameterSet *first;
ParameterSet *last;
int number;
};


#endif
```

C.3 flipcvar.h

```
//
// $Id: flipcvar.h,v 1.10 1996/09/22 11:25:57 bonner Exp $
//
//
// ---------------------
//  F L I P C V A R . H
// ---------------------
//

#ifndef flipcvar
#define flipcvar

#include "flippara.h"

//
//
// * * * * * * * * * * F l o a t V a l u e   * * *
//
//

class FloatValue {

public:
// * * constructors:
FloatValue(float init_value=0)
{ value=init_value; next=NULL; };

// * * simple methods to access data elements:
void set_value(float new_v)         { value=new_v; };
void set_next(FloatValue *new_next) { next=new_next; };

float      get_value(void) { return value; };
FloatValue *get_next(void) { return next; };

protected:
// * * data elements:
float      value;
FloatValue *next; };

//
//
// * * * * * * * * * * F L i s t  * * *
```

```
//
//

class FList {

public:
// * * constructors:
FList(void) { first=NULL; last=NULL; number=0; };

// * * destructor:
~FList(void);

// * * complex methods:
void       add(float new_v=0);
FloatValue *find(int index);
float      find_v(int index);
float      operator [] (int index) { return find_v(index); };
int        remove(int index);
void     remove_all(void);
float    mean(void);

// * * simple methods to access data elements:
FloatValue *get_first(void) { return first; };
FloatValue *get_last(void)  { return last; };
int        get_number(void) { return number; };

protected:
// * * data elements:
FloatValue *first;
FloatValue *last;
int        number; };

//
//
// * * * * * * * * * * P o i n t  * * *
//
//

class Point {

friend class PList;

public:
// * * constructors:
Point(float init_x, float init_y)
```

```
{ x_value=init_x; y_value=init_y; next=NULL; };
Point() { x_value = 0.0; y_value = -1.0; next=NULL; };

// * * complex methods:
void show(void);
int write(ofstream *output);
int read(ifstream *input);

// * * simple methods to access data elements:
void set_x(float new_x)        { x_value=new_x; };
void set_y(float new_y)        { y_value=new_y; };
void set_next(Point *new_next) { next = new_next; };

float get_x(void)     { return x_value; };
float get_y(void)     { return y_value; };
Point *get_next(void) { return next; };

protected:
// * * data elements:
float x_value;
float y_value;
Point *next;
};

//
//
// * * * * * * * * * * P L i s t  * * *
//
//

class PList {

public:
// * * constructors:
PList(void) { first=NULL; last=NULL; number=0; };

// * * destructor:
~PList(void);

// * * complex methods:
void  show(void);
int   write(ofstream *output);
int   read(ifstream *input);
void  add      (float new_x=0, float new_y=-1);
void  add_PList(float new_x=0, float new_y=-1)
```

```
{ add(new_x,new_y);};
Point *find(int index);
float find_x(int index) { return find(index)->get_x(); };
float find_y(int index) { return find(index)->get_y(); };
int   find(float x,float y);
int   find(Point *p) { return find(p->get_x(),p->get_y()); };
Point *operator [] (int index) { return find(index); };
int   insert(float new_x=0, float new_y=-1);
void  insert(Point *p) { insert(p->get_x(),p->get_y()); };
void  remove(int index);
void  remove_all(void);
void  copy      (PList *p_list);
void  copy_PList(PList *p_list)     { copy(p_list); };
float get_max_y(void);
float get_first_x(float y);
void  get_x(FList *xl, float y);
void  clear_horizontal_line(void);
void  clear_vertical_line(void);
void  clear_lines(void) { clear_horizontal_line();
  clear_vertical_line(); };

// * * simple methods to access data elements:
Point *get_first(void) { return first; };
Point *get_last(void)  { return last; };
int   get_number(void) { return number; };

protected:
// * * data elements:
Point *first;
Point *last;
int   number;
};

//
//
// * * * * * * * * * * F u z z y S e t B a s e  * * *
//
//

class FuzzySetBase {
public:
// * * constructors:
FuzzySetBase(name init_linguistic_term, float init_lower_support_end,
float init_upper_support_end, colourtype init_fs_colour);
FuzzySetBase();
```

```
// * * complex methods:
void show(void);
void show_FSB(void)   { show(); };
int write(ofstream *output);
int write_FSB(ofstream *output)  { return write(output); };
int read(ifstream *input);
int read_FSB(ifstream *input)    { return read(input); };
void copy(FuzzySetBase *fsb);
void copy_FSB(FuzzySetBase *fsb) { copy(fsb); };

// * * simple methods to access data elements:
void set_linguistic_term(char *new_lt)
{ strcpy(linguistic_term,new_lt); };
void set_lower_support_end(float new_value)
{ lower_support_end=new_value; };
void set_upper_support_end(float new_value)
{ upper_support_end=new_value; };
void set_fs_colour(colourtype new_c)
{ fs_colour.set_colour(new_c); };
void set_fs_colour(Colour new_c)
{ fs_colour.set_colour(*(new_c.get_colour())); };

char *get_linguistic_term(void)   { return linguistic_term; };
float get_lower_support_end(void) { return lower_support_end; };
float get_upper_support_end(void) { return upper_support_end; };
colourtype *get_fs_colour(void)   { return fs_colour.get_colour(); };

protected:
name   linguistic_term;
float  lower_support_end;
float  upper_support_end;
Colour fs_colour; };

//
//
// * * * * * * * * * * *   P l i m F u z z y S e t   * * *
//
//
// P_oint L_ist I_nstanciated M_atched   FuzzySet

class PlimFuzzySet : public FuzzySetBase {
friend class PlimList;

public:
```

```
// * * constructors:
PlimFuzzySet(char *init_linguistic_term, float init_lower_support_end,
 float init_upper_support_end, colourtype init_fs_colour,
 name init_function);
PlimFuzzySet();

// * * complex methods:
void show(void);
int  write(ofstream *output);
int  read(ifstream *input);
void calculate_match_value(float crisp_input);
void build_imsf(void);
void add_msf_point(float new_x, float new_y);
int  delete_msf_point(int index);
int  get_msf_point_number(void)   { return msf.get_number(); };
int  get_imsf_point_number(void)  { return imsf.get_number(); };
void copy    (PlimFuzzySet *pfs);
void copy_PFS(PlimFuzzySet *pfs)  { copy(pfs); };
FuzzySetBase *get_FSB(void);
void set_msf_point_x(int index, float new_x);
void set_msf_point_y(int index, float new_y);

// * * simple methods to access lower class data elements:
float get_msf_point_x(int index)
{ return msf[index]->get_x(); };
float get_msf_point_y(int index)
{ return msf[index]->get_y(); };
float get_imsf_point_x(int index)
{ return imsf[index]->get_x(); };
float get_imsf_point_y(int index)
{ return imsf[index]->get_y(); };

// * * simple methods to access data elements:
void set_match_value(float new_mv) { match_value=new_mv; };
void set_function(name new_f)      { function.set_name(new_f); };

float get_match_value(void)        { return match_value; };
PlimFuzzySet *get_next(void)       { return next; };
PList *get_msf(void)    { return &msf; };
PList *get_imsf(void)    { return &imsf; };
name  *get_function(void)    { return function.get_name(); };

private:
PList msf;    // membership function
PList imsf;    // instanciated membership function
```

```
float match_value;
Name  function; // name of msf
PlimFuzzySet *next; };

//
//
// * * * * * * * * * *  P l i m L i s t  * * *
//
//

class PlimList {
public:
// * * constructors:
PlimList(void) { first=NULL; last=NULL; number=0; };

// * * destructor:
virtual ~PlimList(void);

// * * complex methods:
void show(void);
void show_PL(void)  { show(); };
int  write(ofstream *output);
int  write_PL(ofstream *output)  { return write(output); };
int  read(ifstream *input);
int  read_PL(ifstream *input)  { return read(input); };
void add(void);
void add_PFS(void) { add(); };
void add(name init_name,
       float lower_support_end,       float upper_support_end,
       float lower_dilatation_length, float upper_dilatation_length,
       colourtype colour);
void add_PFS(name init_name,
       float lower_support_end,       float upper_support_end,
       float lower_dilatation_length, float upper_dilatation_length,
       colourtype colour)
{ add(init_name, lower_support_end, upper_support_end,
      lower_dilatation_length, upper_dilatation_length,
      colour); };
PlimFuzzySet *find(int index);
PlimFuzzySet *operator [] (int index) { return find(index); };
int   get_fs_index(char *linguistic_term);
float get_membership(float x_value,int fs_index);
NList *get_names_of_vars(void);
int   check_match(void);
void  set_all_mvs(float mv);
```

```
int   add_msf_point(int fs_index, float new_x, float new_y)
{ find(fs_index)->add_msf_point(new_x, new_y); return 1; };
int   delete_msf_point(int fs_index, int point_index);
void  copy         (PlimList *plim_list);
void  copy_PlimList(PlimList *plim_list) { copy(plim_list); };
int   put_imsf_into_cover(PList *cover, int imsf_index);

int   above(float x,float y,float ax,float ay,float bx,float by);
int   above(Point *p, PList *l);
int   picture(float *pic,float x,float ax,float ay,float bx,float by);
int   crosspoint(float *cx,float *cy,
 float ax1,float ay1,float ax2,float ay2,
 float bx1,float by1,float bx2,float by2);
void  set_fs_lower_support_end(int fs_index,float new_value);
void  set_fs_upper_support_end(int fs_index, float new_value);
void  remove_all(void);

// * * simple methods to access lower class data elements:
void set_msf_x(int fs_index, int point_index, float new_x)
{ find(fs_index)->set_msf_point_x(point_index,new_x); };
void set_msf_y(int fs_index, int point_index, float new_y)
{ find(fs_index)->set_msf_point_y(point_index,new_y); };
void set_fs_match_value(int fs_index, float init_mv)
{ find(fs_index)->set_match_value(init_mv); };
void set_fs_colour(int fs_index, Colour new_c)
{ find(fs_index)->set_fs_colour(*(new_c.get_colour())); };

int   get_msf_point_number(int fs_index)
{ return find(fs_index)->msf.get_number(); };
int   get_imsf_point_number(int fs_index)
{ return find(fs_index)->imsf.get_number(); };
float get_msf_x(int fs_index, int point_index)
{ return find(fs_index)->get_msf_point_x(point_index); };
float get_msf_y(int fs_index, int point_index)
{ return find(fs_index)->get_msf_point_y(point_index); };
float get_imsf_x(int fs_index, int point_index)
{ return find(fs_index)->get_imsf_point_x(point_index); };
float get_imsf_y(int fs_index, int point_index)
{ return find(fs_index)->get_imsf_point_y(point_index); };
float get_fs_match_value( int fs_index)
{ return find(fs_index)->get_match_value(); };
float get_fs_lower_support_end(int fs_index)
{ return find(fs_index)->get_lower_support_end(); };
float get_fs_upper_support_end(int fs_index)
{ return find(fs_index)->get_upper_support_end(); };
```

```
colourtype *get_fs_colour(int fs_index)
{ return find(fs_index)->get_fs_colour(); };

// * * simple methods to access data elements:
void set_first(PlimFuzzySet *pfs) { first=pfs; };
void set_last(PlimFuzzySet *pfs)  { last=pfs; };
void set_number(int nn)           { number=nn; };

PlimFuzzySet *get_first(void) { return first; };
PlimFuzzySet *get_last(void)  { return last; };
int          get_number(void) { return number; };

protected:
// * * data elements:
PlimFuzzySet *first;
PlimFuzzySet *last;
int number;
};

//
//
// * * * * * * * * * * *  C o m b i n e d V a r i a b l e   * * *
//
//

class CombinedVariable : public ParameterSetBase, public PlimList {
friend class CVList;

public:
// * * constructors:
CombinedVariable(void);
CombinedVariable(ParameterSet *ps, name ncv_name="default_cv");

// * * complex methods:
void show(void);
int  write(ofstream *output);
int  read(ifstream *input);
int  set_value(float crisp_input);
void copy(CombinedVariable *cv);
ParameterSetBase *get_PSB(void);
void copy_PlimList(CombinedVariable *cv);

// * * simple methodes to access data elements
void  set_cv_name(char *new_cv_name) {strcpy(cv_name,new_cv_name);};
void  set_crisp_value(float ncv) {crisp_value=ncv; }; // dangerous
```

```
char  *get_cv_name(void)    { return cv_name; };
float get_crisp_value(void) { return crisp_value; };
CombinedVariable *get_next(void) { return next; };

protected:
name   cv_name;
float          crisp_value;
CombinedVariable *next; };

//
//
// * * * * * * * * * * *   C V L i s t   * * *
//
//

class CVList {
public:
// * * constructor:
CVList(void) { first=NULL; last=NULL; number=0; };

// * * destructor:
~CVList(void);

// * * complex methods:
void show(void);
int  write(ofstream *output);
int  read(ifstream *input);
void add(void);
void add(CombinedVariable *cv);
void add(CVList *cvl);
CombinedVariable *find(int index);
CombinedVariable *find_cv(name cv_name);
int  find_cv_by_pointer(CombinedVariable *cv);
int  find_cv_by_name   (CombinedVariable *cv);
CombinedVariable *operator [] (int index) { return find(index); };
void remove(int index);
void remove_all(void);

// * * simple methods to access data elements:
CombinedVariable *get_first(void)       { return first; };
CombinedVariable *get_last(void)        { return last; };
int     get_number(void)       { return number; };

private:
```

```
// * * data elements:
CombinedVariable *first;
CombinedVariable *last;
int number;
};


//
//
// * * * * * * * * * * *  o t h e r e  f u n c t i o n s  * * *
//
//

float min(float ,float );
float max(float ,float );
void  wait_for_key_press(void);

#endif
```

C.4 flipoper.h

```
//
// $Id: flipoper.h,v 1.5 1996/09/22 11:25:59 bonner Exp $
//
//
// ---------------------
//  F L I P O P E R . H
// ---------------------
//

#ifndef flipoper
#define flipoper

#include "flipcvar.h"

enum fuzzy_operator {minimum,maximum,fuzzy_and,fuzzy_or,
     ass_comp_and,ass_comp_or, product, sum,
     no_fop};  // to be continued
enum defuzzification_operator {cog,fimax,mom,no_dfop}; // to be continued
// center of gravity
// first maximum
// mean of maximum
// no defuzzy operator
enum concat_operator {c_and,c_or,AND,OR,no_cop}; // to select fuzzy_operator

const double pi=3.14159265359;

//
//
// * * * * * * * * * *   F u z z y O p e r a t o r   * * *
//
//

class FuzzyOperator {

public:

// * * constructors:
FuzzyOperator(void) { oper=no_fop; };
FuzzyOperator(fuzzy_operator op) { oper=op; };

// * * complex methods:
void show(void);
```

```
int  write(ofstream *output);
int  read(ifstream *input);

// * * simple methods to access data elements:
void set_oper(fuzzy_operator new_op) { oper=new_op; };
fuzzy_operator *get_oper(void)       { return &oper; };

protected:
// * * data elements:
fuzzy_operator oper; };

//
//
// * * * * * * * * * * *  D e F u z z y O p e r a t o r  * * *
//
//

class DeFuzzyOperator {

public:

// * * constructors:
DeFuzzyOperator(void)       { oper=no_dfop; };
DeFuzzyOperator(defuzzification_operator op) { oper=op; };

// * * complex methods:
void show(void);
int  write(ofstream *output);
int  read(ifstream *input);

// * * simple methods to access data elements:
void set_oper(defuzzification_operator new_op) {oper=new_op;};
defuzzification_operator *get_oper(void)       {return &oper;};

protected:
// * * data elements:
defuzzification_operator oper; };

//
//
// * * * * * * * * * * *  C o n c a t O p e r a t o r  * * *
//
//

class ConcatOperator {
```

```
public:

// * * constructors:
ConcatOperator(void)     { oper=no_cop; };
ConcatOperator(concat_operator op) { oper=op; };

// * * complex methods:
void show(void);
int  write(ofstream *output);
int  read(ifstream *input);
int  read(strstream *input);
int  compare(ConcatOperator *c1, ConcatOperator *c2);

// * * simple methods to access data elements:
void set_oper(concat_operator new_op) { oper=new_op; };
concat_operator *get_oper(void)       { return &oper; };

protected:
// * * data elements:
concat_operator oper; };

//
//
// * * * * * * * * * * *   t - n o r m   * * *
//
//

class TNorm {

public:
float min(FList *mvs);
float min(float oamv,  float cer);
void  min(float aocmv, PlimFuzzySet *fs);

float prod(FList *mvs);
float prod(float oamv,  float cer);
void  prod(float aocmv, PlimFuzzySet *fs); };

//
//
// * * * * * * * * * * *   t - c o - n o r m   * * *
//
//
```

```
class TCoNorm {

public:
float max(FList *mvs);
float max(float oamv,  float cer);
void  max(float aocmv, PlimFuzzySet *fs);

float addition(FList *mvs);
float addition(float oamv,  float cer);
void  addition(float aocmv, PlimFuzzySet *fs); };

//
//
//
* * * * *   c o m p a n s a t o r y   -   o p e r a t o r   * * *
//
//

class CompOp {

public:
float ass_comp_and(FList *mvs);
float ass_comp_and(float oamv,  float cer);
void  ass_comp_and(float aocmv, PlimFuzzySet *fs);

float ass_comp_or(FList *mvs);
float ass_comp_or(float oamv,  float cer);
void  ass_comp_or(float aocmv, PlimFuzzySet *fs);

float fuzzy_and(FList *mvs, float gamma);
float fuzzy_and(float oamv,  float cer, float gamma);
void  fuzzy_and(float aocmv, PlimFuzzySet *fs, float gamma);

float fuzzy_or(FList *mvs, float gamma);
float fuzzy_or(float oamv,  float cer, float gamma);
void  fuzzy_or(float aocmv, PlimFuzzySet *fs, float gamma); };

//
//
// * * * * * * * * * * *  i n t e r s e c t i o n  * * *
//
//

//
//
```

```
// * * * * * * * * * * d e f u z z y * * *
//
//

class DefOp {

public:
float cog(CombinedVariable *cv); // center of gravity
float first_max(CombinedVariable *cv); // maximum
float mom(CombinedVariable *cv); // mean of maximums
void  show_cover(PList cover, int cover_number);
float integrate(PList *cpl); };

//
//
// * * * * * * * * * * O p e r a t o r S e t * * *
//
//
class   OperatorSet {
friend  float aggregate(float mvs[], concat_operator agg_op,
OperatorSet *operator_set);
friend  float certify(float oamv, float certainty,
OperatorSet *operator_set);
friend  void  inference(float oacmv, PlimFuzzySet *fs,
OperatorSet *operator_set);
friend  float accumulate(CombinedVariable *cv,
OperatorSet *operator_set);


public:
// * * constructor:
OperatorSet(void)
{ importance_operator.set_oper(no_fop);
  and_operator.set_oper(minimum);
  or_operator.set_oper(maximum);
  certainty_operator.set_oper(minimum);
  inference_operator.set_oper(minimum);
  accumulate_operator.set_oper(minimum);
  defuzzy_operator.set_oper(cog);
  gamma=1;
  accuracy=10; };

// * * complex methods:
void show(void);
int  write(ofstream *output);
```

```
int  read(ifstream *input);

// simple methods to access data elements:
void set_importance_operator(fuzzy_operator op)
{ importance_operator.set_oper(op); };
void set_and_operator(fuzzy_operator op)
{ and_operator.set_oper(op); };
void set_or_operator(fuzzy_operator op)
{ or_operator.set_oper(op); };
void set_certainty_operator(fuzzy_operator op)
{ certainty_operator.set_oper(op); };
void set_inference_operator(fuzzy_operator op)
{ inference_operator.set_oper(op); };
void set_accumulate_operator(fuzzy_operator op)
{ accumulate_operator.set_oper(op); };
void set_defuzzy_operator(defuzzification_operator op)
{ defuzzy_operator.set_oper(op); };
void set_gamma(float g)     { gamma=g; };
void set_accuracy(int ac) { accuracy=ac; };

fuzzy_operator *get_importance_operator(void)
{ return importance_operator.get_oper(); };
fuzzy_operator *get_and_operator(void)
{ return and_operator.get_oper(); };
fuzzy_operator *get_or_operator(void)
{ return or_operator.get_oper(); };
fuzzy_operator *get_certainty_operator(void)
{ return certainty_operator.get_oper(); };
fuzzy_operator *get_inference_operator(void)
{ return inference_operator.get_oper(); };
fuzzy_operator *get_accumulate_operator(void)
{ return accumulate_operator.get_oper(); };
defuzzification_operator *get_defuzzy_operator(void)
{ return defuzzy_operator.get_oper(); };
float get_gamma(void)    { return gamma; };
int   get_accuracy(void) { return accuracy; };

private:
FuzzyOperator   importance_operator;
FuzzyOperator   and_operator;
FuzzyOperator   or_operator;
FuzzyOperator   certainty_operator;
FuzzyOperator   inference_operator;
FuzzyOperator   accumulate_operator;
DeFuzzyOperator defuzzy_operator;
```

```
float gamma;
int             accuracy; };

//
//
// * * * * * * * * * * *   i n t e r f a c e    * * *
//
//

float join_mv_with_importance(float mv, float imp,OperatorSet *operator_set);
float aggregate(FList *mvs,concat_operator *agg_op,OperatorSet *operator_set);
float certify(float oamv, float certainty, OperatorSet *operator_set);
void  inference(float oacmv, PlimFuzzySet *fs, OperatorSet *operator_set);
float accumulate(CombinedVariable *cv, OperatorSet *operator_set);
int   calculate(CombinedVariable *cvr, CombinedVariable *cva,
CombinedVariable *cvb, concat_operator *agg_op,
OperatorSet *operator_set);

#endif
```

C.5 fliprule.h

```
//
// $Id: fliprule.h,v 1.7 1996/09/22 11:26:02 bonner Exp $
//
//
// ---------------------
//  F L I P R U L E . H
// ---------------------
//

#ifndef fliprule
#define fliprule

#include "flipoper.h"

//
//
// * * * * * * * * * *   T e r m  * * *
//
//

class Term {
public:
// * * constructor:
Term(void) {  cv = NULL;
      strcpy(lt,"no_linguistic_term");
      next = NULL; };
~Term(void) { cv = NULL;
      strcpy(lt," ");
      next = NULL; };

// * * complex methods:
void show(void);
int  write(ofstream *output);
int  read(ifstream *input, CVList *cv_list);
void copy(Term *t);
int  compare(Term *t1, Term *t2);
void make_ShortString(strstream *ss);

// * * simple methods to access data elements:
void set_cv(CombinedVariable *ncv) { cv=ncv; };
void set_lt(name nlt)              { strcpy(lt,nlt); };
void set_next(Term *t)    { next=t; };
```

```
CombinedVariable *get_cv(void)     { return cv; };
name   *get_lt(void)     { return &lt; };
Term  *get_next(void)   { return next; };

private:
// * * data elements:
CombinedVariable  *cv;
name            lt;
Term   *next; };

//
//
// * * * * * * * * * *   T L i s t * * *
//
//

class TList {
public:
// * * constructor:
TList(void) { first=NULL; last=NULL; number=0; };
~TList(void);

// * * complex methods:
void show(void);
void show(ConcatOperator *op);
int  write(ofstream *output, ConcatOperator *op);
int  read(ifstream *input,   ConcatOperator *op, CVList *cv_list);
void add(void)    { add(NULL,"no_linguistic_term"); };
void add(Term *t);
void add(CombinedVariable *cv, name lt="no_linguistic_term");
Term *find(int index);
CombinedVariable *find_cv(int index) { return find(index)->get_cv();};
name *find_lt(int index)             { return find(index)->get_lt();};
Term *operator [] (int index)        { return find(index); };
void copy(TList *tl);
int  compare(TList *tl1, TList *tl2);
int  remove(int t_index);
void remove_all(void);
void make_ShortString(strstream *ss);

// * * simple methods to access lower class data elements:
void set_cv(CombinedVariable *ncv, int t_index)
{ find(t_index)->set_cv(ncv); };
void set_lt(name nlt, int t_index) { find(t_index)->set_lt(nlt); };
```

```
CombinedVariable *get_cv(int t_index)
{ return find(t_index)->get_cv(); };
name *get_lt(int t_index) { return find(t_index)->get_lt(); };

// * * simple methods to access data elements
Term *get_first(void)   { return first; };
Term *get_last(void)    { return last; };
int  get_number(void)   { return number; };

protected:
Term *first;
Term *last;
int number; };

//
//
// * * * * * * * * * *   R u l e  * * *
//
//
class Rule {
public:
// * * constructor:
Rule(void)
      { rs_term=new Term;
ls_t_list=new TList;
certainty=0;
aggregate_operator.set_oper(no_cop);
next=NULL;};
~Rule(void);

// * * complex methods:
void show(void);
int  write(ofstream *output);
int  read(ifstream *input, CVList *cv_list);
void copy(Rule *r);
int  compare(Rule *r1, Rule *r2);
void make_ShortString(strstream *ss);

// * * simple methods to access lower class data elements:
void set_left_side_cv(int op_index, CombinedVariable *cv)
{ ls_t_list->find(op_index)->set_cv(cv); };
void set_right_side_cv(CombinedVariable *cv) { rs_term->set_cv(cv); };
void set_left_side_lt(int op_index, name lt)
{ ls_t_list->find(op_index)->set_lt(lt); };
```

```
void set_right_side_lt(name lt) { rs_term->set_lt(lt); };
void set_aggregate_operator(concat_operator agg_op)
{ aggregate_operator.set_oper(agg_op); };

CombinedVariable *get_left_side_cv(int cv_number)
{ return ls_t_list->find(cv_number)->get_cv(); };
CombinedVariable *get_right_side_cv(void) {return rs_term->get_cv();};
name  *get_left_side_lt(int cv_number)
{ return ls_t_list->find(cv_number)->get_lt(); };
name  *get_right_side_lt(void) { return rs_term->get_lt(); };
concat_operator *get_aggregate_operator_op(void)
{ return aggregate_operator.get_oper(); };
int   get_ls_t_list_number(void)
{ return ls_t_list->get_number(); };

// * * simple methods to access data elements:
void set_rs_term(Term *t);
void set_ls_t_list(TList *tl);
void set_aggregate_operator(ConcatOperator *agg_op)
{ aggregate_operator=*agg_op; };
void set_certainty(float cer) { certainty=cer; };
void set_next(Rule *r)          { next=r; };

Term  *get_rs_term(void)        { return rs_term; };
TList *get_ls_t_list(void)      { return ls_t_list; };
float get_certainty(void)       { return certainty; };
ConcatOperator *get_aggregate_operator(void)
{ return &aggregate_operator; };
Rule  *get_next(void) { return next; };

protected:
Term           *rs_term;
TList          *ls_t_list;
float          certainty;
ConcatOperator aggregate_operator; // to combine single clauses
Rule *next; };

//
//
// * * * * * * * * * * *   R u l e B l o c k   * * *
//
//
class RuleBlock {
public:
// * * constructor:
```

```
RuleBlock(void) { first=NULL; last=NULL; number=0; next=NULL; };
~RuleBlock(void);

// * * complex methods:
void show(void);
int  write(ofstream *output);
int  read(ifstream  *input, CVList *cv_list);
void add(void)        {add(NULL);};
void add(Rule *r);
Rule *find(int index);
int  find(Rule *r);
CombinedVariable *find_cv(int index)
{ return find(index)->get_rs_term()->get_cv(); };
name *find_lt(int index)
{ return find(index)->get_rs_term()->get_lt();};
Rule *operator [] (int index) {return find(index);};
void copy(RuleBlock *rb);
int  remove(int r_index);
void remove_all(void);
void make_ShortString(strstream *ss);

// * * simple methods to access lower class data elements
void set_left_side_cv(int r_index,int op_index,CombinedVariable *cv)
{ find(r_index)->set_left_side_cv(op_index,cv); };
void set_right_side_cv(int r_index,CombinedVariable *cv)
{ find(r_index)->set_right_side_cv(cv); };
void set_left_side_lt(int r_index,int op_index,name lt)
{ find(r_index)->set_left_side_lt(op_index,lt); };
void set_right_side_lt(int r_index,name lt)
{ find(r_index)->set_right_side_lt(lt); };
void set_aggregate_operator(int r_index,concat_operator agg_op)
{ find(r_index)->set_aggregate_operator(agg_op); };
void set_certainty(int r_index,float cer)
{ find(r_index)->set_certainty(cer); };

CombinedVariable *get_left_side_cv(int r_index,int cv_number)
{ return find(r_index)->get_left_side_cv(cv_number); };
CombinedVariable *get_right_side_cv(int r_index=0)
{ return find(r_index)->get_right_side_cv(); };
name  *get_left_side_lt(int r_index,int cv_number)
{ return find(r_index)->get_left_side_lt(cv_number); };
name  *get_right_side_lt(int r_index)
{ return find(r_index)->get_right_side_lt(); };
concat_operator *get_aggregate_operator(int r_index)
{ return find(r_index)->get_aggregate_operator_op(); };
```

```
float get_certainty(int r_index)
{ return find(r_index)->get_certainty(); };
int   get_operand_number(int r_index)
{ return find(r_index)->get_ls_t_list_number(); };

// * * simple methods to access data elements
void      set_first(Rule *t)      { first=t; };
void      set_last(Rule *t)       { last=t; };
void      set_number(int n)       { number=n; };
void   set_next(RuleBlock *rb) { next=rb; };

Rule      *get_first(void)        { return first; };
Rule      *get_last(void)         { return last; };
int       get_number(void)        { return number; };
RuleBlock *get_next(void)         { return next; };

protected:
Rule      *first;
Rule      *last;
int       number;
RuleBlock *next; };

//
//
// * * * * * * * * * * *   R u l e S e t   * * *
//
//
class RuleSet {

public:
// * * constructor:
RuleSet(void) { ps_list = new PSList;
cv_list = new CVList;
operator_set = new OperatorSet;
first=NULL; last=NULL; number=0;
strcpy(rs_name,"empty_rule_set"); };

RuleSet(PSList *psl, CVList *cvl, OperatorSet *os) {
ps_list=psl; cv_list=cvl; operator_set=os;
first=NULL; last=NULL; number=0;
strcpy(rs_name,"empty_rule_set"); };
~RuleSet(void);

// * * complex methods:
void show(void);
```

```
int  write(ofstream *output);
int  read(ifstream  *input);
void add(RuleBlock *rb=NULL);
void add_all_RuleBlocks(RuleSet *rs);
int  add_Rule(TList *ls_tl,Term *rs_t,concat_operator op, float cer);
int  add_BasicRule(CombinedVariable *left, name left_ling,
CombinedVariable *right, name right_ling, float cer=1);
int  add_ComplexRule(CombinedVariable *left_cv_1, name left_lt_1,
    CombinedVariable *left_cv_2, name left_lt_2,
    CombinedVariable *right_cv, name right_lt,
    concat_operator agg_op, float cer=1);
RuleBlock *find(int index);
CombinedVariable *find_cv(int index)
{return find(index)->get_first()->get_rs_term()->get_cv();};
RuleBlock *operator [] (int index) {return find(index);};
int   search(CombinedVariable *cv);
int   remove(int rb_index);
int   remove(CombinedVariable *which);
void  remove_cv_and_its_approaches(CombinedVariable *which);
void  remove_all(void);
CombinedVariable *get_cv(name cv_name);

int   calculate(CombinedVariable *cva, CombinedVariable *cvb,
concat_operator *agg_op);;
float evaluate(CombinedVariable *cv);
int   internal_evaluate(CombinedVariable *cv);
void  make_ShortString(strstream *ss);

// * * simple methods to access lower class data elements:
void set_left_side_cv(int rb_index, int r_index, int op_index,
CombinedVariable *cv)
{ find(rb_index)->set_left_side_cv(r_index,op_index,cv); };
void set_left_side_lt(int rb_index,int r_index,int op_index,name lt)
{ find(rb_index)->set_left_side_lt(r_index,op_index,lt); };
void set_right_side_cv(int rb_index,int r_index,CombinedVariable *cv)
{ find(rb_index)->set_right_side_cv(r_index,cv); };
void set_right_side_lt(int rb_index, int r_index, name lt)
{ find(rb_index)->set_right_side_lt(r_index,lt); };
void set_aggregate_operator(int rb_index,int r_index,
concat_operator agg_op)
{ find(rb_index)->set_aggregate_operator(r_index,agg_op); };
void set_certainty(int rb_index, int r_index, float cer)
{ find(rb_index)->set_certainty(r_index,cer); };

CombinedVariable *get_left_side_cv(int rb_index,int r_index,
```

```
int op_index)
{ return find(rb_index)->get_left_side_cv(r_index,op_index);};
name *get_left_side_lt(int rb_index, int r_index, int op_index )
{ return find(rb_index)->get_left_side_lt(r_index,op_index);};
CombinedVariable *get_right_side_cv(int rb_index, int r_index=0 )
{ return find(rb_index)->get_right_side_cv(r_index); };
name *get_right_side_lt(int rb_index, int r_index)
{ return find(rb_index)->get_right_side_lt(r_index); };
concat_operator *get_aggregate_operator(int rb_index, int r_index)
{ return find(rb_index)->get_aggregate_operator(r_index); };
float get_certainty(int rb_index, int r_index)
{ return find(rb_index)->get_certainty(r_index); };
int get_operand_number(int rb_index, int r_index)
{ return find(rb_index)->get_operand_number(r_index); };
int get_rule_number(int rb_index)
{ return find(rb_index)->get_number(); };
int get_rule_block_number(void) { return number; };

// * * simple methods to access data elements
void  set_rs_name(name nn)               { strcpy(rs_name,nn); };
void  set_ps_list(PSList *npsl)          { ps_list=npsl; };
void  set_cv_list(CVList *ncvl)          { cv_list=ncvl; };
void  set_operator_set(OperatorSet *nos) { operator_set=nos; };
void  set_first(RuleBlock *t)           { first=t; };
void  set_last(RuleBlock *t)            { last=t; };
void  set_number(int n)                 { number=n; };

name     *get_rs_name(void)        { return &rs_name; };
PSList      *get_ps_list(void)       { return ps_list; };
CVList      *get_cv_list(void)       { return cv_list; };
OperatorSet *get_operator_set(void)  { return operator_set; };
RuleBlock   *get_first(void)         { return first; };
RuleBlock   *get_last(void)          { return last; };
int         get_number(void)         { return number; };

private:
// * * data elements:
name        rs_name;
PSList      *ps_list;
CVList      *cv_list;
OperatorSet *operator_set;
RuleBlock   *first;
RuleBlock   *last;
int         number; };
```

```
# endif
```