12

RFuzzy: an easy and expressive tool for modelling the cognitive layer in RoboCupSoccer

Susana Muñoz Hernández
Technical University of Madrid
Spain

1. Introduction

The idea of robot playing soccer has been developed since early 90s Chen et al. (2003). Soccer environment is a dynamically changing environment which requires individual skill as well as team skill and therefore is an interesting research field on Artificial Intelligence and robotics. Prolog is a programming language that represent logic reasoning. Is is a perfect tool to represent human reasoning, so it seems to be a good choice for implementing the cognitive layer of soccer players that is a simulation of human behaviour related to this game. For example, applying the rule "if the goal keeper is not at the goal then kick to ball". But many of the most important decisions that are made by soccer players deal with non-crisp issues. They are related to fuzziness (e.g. "if other player of my team is FAR from me then don't pass him/her the ball"), uncertainty (e.g. "if I CAN get the goal then kick the ball"), or incompleteness (e.g. "if I cannot see the position of a player, by default I'm not going to pass him the ball").
In this work we are going to provide a programming framework to Robot Soccer programmers to model robot control in an expressive but simple way. We propose the possibility of using fuzzy concepts for this modelization and we are going to provide some conclusions about this tool based on a bench of practical experiments that we have done and that we describe here.
In the rest of this section we introduce RoboCupSoccer field (section 1.1) and we discuss some previous fuzzy approaches in logic programming (sections 1.2 and 1.3. In section 2 we describe our framework, *RFuzzy* enumerating the features that characterize its expressivity for modelling problems in general. From the following section we focus on the Robot Soccer use of our tool. Section 3 describes the environment for our experimentation (the general arquitecture at section 3.1 and the particular Prolog code arquitecture at section 3.2). We have described in detail our experiments in section 4. We provide information about the decision making analysis that we have done (section 4.1.1) and the action execution analysis (section 4.1.2). We finally conclude at section 5.

1.1 RoboCupSoccer

RoboCup is an international annual event promoting research on Artificial Intelligence, robotics, and related field. The original motivation of RoboCup is RoboCupSoccer. As the nature of soccer game, autonomous robots participating in RoboCupSoccer should have individual ability such as moving and kicking the ball, cooperative ability such as coordinating with team mates, and of course, the ability to deal with dynamic environment.

RoboCupSoccer consists of several leagues, providing test beds for various research scale: Simulation League, Small Size Robot League (F-180), Middle Size Robot League (f-2000), Four-Legged Robot League, Humanoid League, E-League and RoboCup Commentator Exhibition. The first E-League was held at RoboCup 2004.This league is a simplified version of Small Size Robot League, where vision processing and communications are factored out, thus provided by the league. Each team in this league consists of four small sized autonomous robots, one of whom can be a goalkeeper. The match lasts for two equal periods of 10 minutes.
We employ RoboCupSoccer Simulation League for the sake of simplicity because we are just focused on the robot control layer.

1.2 Fuzzy Approaches in Logic Programming

Introducing Fuzzy Logic into Logic Programming has provided the development of several fuzzy systems over Prolog. These systems replace its inference mechanism, SLD-resolution, with a fuzzy variant that is able to handle partial truth. Most of these systems implement the fuzzy resolution introduced by Lee in Lee (1972), as the Prolog-Elf system Ishizuka & Kanai (1985), the FRIL Prolog system Baldwin et al. (1995) and the F-Prolog language Li & Liu (1990). However, there is no common method for fuzzifying Prolog, as noted in Shen et al. (1989).
Some of these Fuzzy Prolog systems only consider fuzziness on predicates whereas other systems consider fuzzy facts or fuzzy rules. There is no agreement about which fuzzy logic should be used. Most of them use min-max logic (for modelling the conjunction and disjunction operations) but other systems just use Łukasiewicz logic Klawonn & Kruse (1994).
There is also an extension of constraint logic programming Bistarelli et al. (2001), which can model logics based on semiring structures. This framework can model min-max fuzzy logic, which is the only logic with semiring structure. Another theoretical model for fuzzy logic programming without negation has been proposed by Vojtáš in Vojtas (2001), which deals with many-valued implications.

1.3 Fuzzy Prolog

One of the most promising fuzzy tools for Prolog was the "Fuzzy Prolog" system Vaucheret et al. (2002); Guadarrama, Munoz-Hernandez & Vaucheret (2004). The most important advantages against the other approaches are:

1. A truth value is represented as a finite union of sub-intervals on $[0,1]$. An interval is a particular case of union of one element, and a unique truth value (a real number) is a particular case of having an interval with only one element.
2. A truth value is propagated through the rules by means of an *aggregation operator*. The definition of this *aggregation operator* is general and it subsumes conjunctive operators (triangular norms Klement et al. (n.d.) like min, prod, etc.), disjunctive operators Trillas et al. (1995) (triangular co-norms, like max, sum, etc.), average operators (averages as arithmetic average, quasi-linear average, etc) and hybrid operators (combinations of the above operators) Pradera et al. (2002)).
3. Crisp and fuzzy reasoning are consistently combined Munoz-Hernandez et al. (2002).

Fuzzy Prolog adds fuzziness to a Prolog compiler using CLP($\mathcal{R}$) instead of implementing a new fuzzy resolution method, as other former fuzzy Prologs do. It represents intervals as constraints over real numbers and *aggregation operators* as operations with these constraints, so it uses Prolog built-in inference mechanism to handle the concept of partial truth. From the implementation point of view, Fuzzy Prolog is implemented over Ciao

Fig. 1. *RFuzzy* architecture.

Prolog CLIP Lab (n.d.). The Ciao Prolog System offers a complete Prolog system supporting ISO-Prolog. Its modular design allows restriction and extension of the language both syntactically and semantically. The Ciao Prolog Development System provides many libraries including a constraint logic programming system and interfaces to some programming languages. In Ciao Prolog terminology, a library is implemented as either a module or a package. Fuzzy Prolog described in Guadarrama, S.Muñoz & C.Vaucheret (2004) and *Extending Prolog with Incomplete Fuzzy Information* (2005);
Default values to handel Incomplete Fuzzy Information (2006) is implemented as the package "fuzzy.pl", a syntactic extension of the CLP($\mathcal{R}$) system in the Ciao Prolog System.

2. *RFuzzy* tool expressiveness

Besides the advantages of Fuzzy Prolog, it truth value representatio based on constraints is too general that it is complex to interpret for regular users. That was the reason for implementing a simpler variant that we called *RFuzzy*. In *RFuzzy* the truth value is represented by a simple real number.
RFuzzy is implemented as a Ciao Prolog CLIP Lab (n.d.) package because Ciao Prolog offers the possibility of dealing with a higher order compilation through the implementation of Ciao packages.
The compilation process of a *RFuzzy* program has two pre-compilation steps: (1) the *RFuzzy* program is translated into CLP($\mathcal{R}$) constraints by means of the *RFuzzy* package and (2) the program with constraints is translated into ISO Prolog by using the CLP($\mathcal{R}$) package. Fig. 1 shows the whole process.
As the motivation of *RFuzzy* was providing a tool for practical application, it was loaded with many nice features that represent an adventage with respect to previous fuzzy tools to model real problems. In this section we enumerate and describe some of the most interesting characteristics of *RFuzzy* expressiveness through its syntax (to show its simplicity that is the other advantage of *RFuzzy*). For the examples we are going to use intuitive concepts related to soccer vocabulary although many of ther are not use for the simulator because it use just simple variables of position and speed but they are more ilustrative in the interest of concepts understanding.

2.1 Types definition

Prolog does not have types. The problem of not having types is that it is impossible to return constructive answers but using constraints. *RFuzzy* does not use constrains because they are not friendly to return constructive results and that is the reason for having types instead.
In *RFuzzy* types are defined according to (1) syntax.

$$\textbf{:- set\_prop } pred/ar \texttt{ => } type\_pred\_1/1 \; [, \; type\_pred\_n/1 \;]^* \; . \tag{1}$$

where *set_prop* is a reserved word, *pred* is the name of the typed predicate, *ar* is its arity and *type_pred_1*, *type_pred_n* ($n \in 2,3,\ldots,ar$) are predicates used to define types for each argument of *pred*. They must have arity 1. The definition is constraining the values of the n_th argument

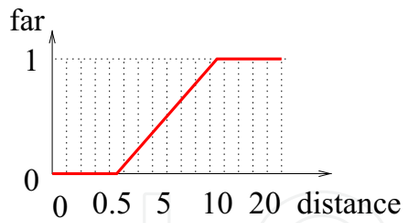

Fig. 2. Far truth value continuous representation

of *pred* to the values of the type *type_pred_n*. This definition of types ensures that the values assigned to the arguments of *pred* are correctly typed.
The example below shows that the arguments of predicates *is_striker/1* and *is_faster_than/2* have to be of type *player/1*. The domain of type *player* is enumerated.

$$:-set\_prop\ is\_striker/1 \Rightarrow player/1.$$
$$:-set\_prop\ is\_faster\_than/2 \Rightarrow player/1,\ player/1.$$
$$player(robot1). \quad player(robot2). \quad player(robot3).$$
$$player(robot4). \quad player(robot5).$$

2.2 Simple truth value assignment

It is possible to assign a truth value to an individual using fuzzy facts. Their syntax, that we can see in (2), is different than regular Prolog facts syntax.

$$pred(args)\ \textbf{value}\ truth\_val. \tag{2}$$

Arguments, *args*, should be ground and the truth value, *truth_val*, must be a real number between 0 and 1. The example below defines that the player *robot3* is a *fast_player* with a truth value 0.9.

$$fast\_player(robot3)\ value\ 0.9.$$

2.3 Continuous function to represent truth values

Facts definition (see subsection 2.2) is worth for a finite (and relative small) number of individuals. Nevertheless, it is very common to represent fuzzy truth using continuous functions. Fig. 2 shows an example in which the continuous function assigns the truth value of being *far* to a distance.
Functions used to define the truth value of some group of individuals are usually continuous and linear over intervals. To define those functions there is no necessity to write down the value assigned to each element in their domains. We have to take into account that the domain can be infinite.
RFuzzy provides the syntax for defining functions by stretches. This syntax is shown in (3). External brackets represent the Prolog list symbols and internal brackets represent cardinality in the formula notation. Predicate *pred* has arity 1, *val1, ..., valN* should be ground terms representing numbers of the domain (they are possible values of the argument of *pred*) and *truth_val1, ..., truth_valN* should be the truth values associated to these numbers. The truth value of the rest of the elements is obtained by interpolation.

$$pred\ \textbf{:\#}\ ([(val1, truth\_val1),\ (val2, truth\_val2)\ [,\ (valn, truth\_valn)\ ]^*]). \tag{3}$$

The *RFuzzy* syntax for the predicate *far/1* (represented in Fig.2) is:

$$teenager : \#([(0.5,0),(10,1)]).$$

2.4 Rule definition with truth values and credibility

A tool which only allows the user to define truth values through functions and facts lacks on allowing him to combine those truth values for representing more complex situations. A rule is the tool to combine the truth values of facts, functions, and other rules.
Rules allow the user to combine truth values in the correct way (by means of aggregation operators, like *minimum*, *maximum*, *product*, etc.). The aggregation operator combines the truth values of the subgoals of the body of the rule to obtain the truth value of the head of the rule. Appart from this, rules are assigned a credibility value to obtain the final truth value for the head of the clause. Credibility is used to express how much we trust a rule. It is used another opperator to aggregate the truth value obtained (from the aggregation of the subgoals of the body) with the rule's credibility.
RFuzzy offers a simple syntax for representing these rules, defined in (5). There are two aggregation operators, *op2* for combining the truth values of the subgoals of the rule body and *op1* for combining the previous result with the rule's credibility. The user can choose for any of them an aggregation operator from the list of the available ones[1] or define his/her own aggregation operator.

$$\begin{aligned} pred(arg1\ [,\ argn]^*\ )\ [\ \textbf{cred (}op1,value1\textbf{)}\ ] :\sim op2 \\ pred1(args\_pred\_1)\ [,\ predm(args\_pred\_m)]\ . \end{aligned} \tag{4}$$

The following example uses the operator *prod* for aggregating truth values of the subgoals of the body and *min* to aggregate the result with the credibility of the rule (which is 0.8). "**cred (***op*1,*value*1**)**" can only appear 0 or 1 times.

$$good\_player(J)\ cred(min,0.8) :\sim prod\ swift(J),\ agile(J),\ has\_experience(J).$$

2.5 General and Conditioned Default Truth Values

Unfortunately, information provided by the user is not complete in general. So there are many cases in which we have no information about the truth value for a fuzzy predicate of an individual or a set of them. This happend many times in Robot soccer (not in the simulator but in games with real robots) when the camara does not detect correctly any player of the ball position. Nevertheless, it is interesting not to stop a complex query evaluation just because we have no information about one or more subgoals if we can use a reasonable approximation. A solution to this problem is using default truth values for these cases. The *RFuzzy* extension to define a default truth value for a predicate when applied to individuals for which the user has not defined an explicit truth value is named *general default truth value*. The syntax for defining a general default truth value is shown in (5).
Conditioned default truth value is used when the default truth value only applies to a subset of the domain. This subset is defined by a membership predicate which is true only when an individual belongs to the subset. The membership predicate (*membership_predicate/ar*) and the

[1] Aggregation operators available are: *min* for minimum, *max* for maximum, *prod* for the product, *luka* for the Łukasiewicz operator, *dprod* for the inverse product, *dluka* for the inverse Łukasiewicz operator and *complement*.

predicate to which it is applied (*pred/ar*) need to have the same arity (*ar*). The syntax is shown in (6).

$$\textbf{:- default(}pred/ar,\ truth\_value\textbf{) .} \tag{5}$$

$$\textbf{:- default(}pred/ar,\ truth\_value\textbf{) =>}\ membership\_predicate/ar\textbf{.} \tag{6}$$

pred/ar is in both cases the predicate to which we are defining default values. As expected, when defining the three cases (explicit, conditioned and default truth value) only one will be given back when doing a query. The precedence when looking for the truth value goes from the most concrete to the least one.
The code from the example below joint with the code from examples in subsections 2.1 and 2.2 assigns to the predicate *fast_player* a truth value of 0.8 for *robot2* (default truth value), 0.6 when it is *robot1* (conditioned default truth value for the goal keeper) and 0.9 when it is *robot3* (explicit truth value).

$$:-default(fast\_player/1, 0.6) => goal\_keeper/1.$$
$$:-default(fast\_player/1, 0.8).$$
$$goal\_keeper(robot1).$$

2.6 Constructive Answers

A very interesting characteristic for a fuzzy tool is being able to provide constructive answers for queries. The regular (easy) questions ask for the truth value of an element. For example, how fast is *robot3*? (See left hand side example below)

$?-fast\_player(robot3, V).$	$?-fast\_player(X, V), V > 0.7.$
$V = 0.9?;$	$V = 0.9, X = robot3?;$
no	$V = 0.8, X = robot2?;$
	$V = 0.8, X = robot4?;$
	$V = 0.8, X = robot5?;$
	no

But the really interesting queries are the ones that ask for values that satisfy constraints over the truth value. For example, which players are very fast? (See right hand side example above). *RFuzzy* provides this constructive functionality.

3. Environment

In this work we have prepared a complete framework with all the interfaces necessary for someone interested in defining strategies for robot control. In this section we are going to describe all the components of the environment that we have used for our experiments and we provide them for their free use.

3.1 Architecture and Implementation Details

Based on agent system architecture proposed by García et al. (2004), in Hernndez & Wiguna (2007) we proposed a generic system architecture for RoboCup offering flexibility on choice of programming language and minimal modification to switch between leagues. This architecture is shown in figure 3. Prolog is proposed for cognitive layer, and in Hernndez & Wiguna (2007) we use Fuzzy Prolog Guadarrama, Munoz-Hernandez & Vaucheret (2004) for implementing the cognitive layer and we use the Atan library and a RoboCupSoccer Simulator.

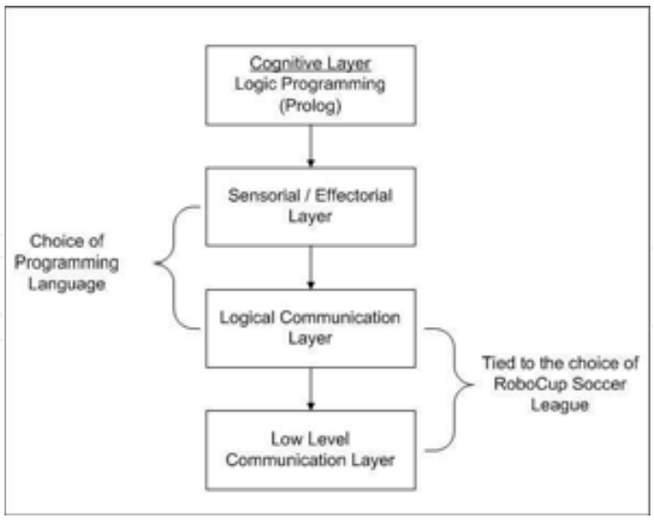

Fig. 3. Generic System Architecture

3.1.1 Low Level Communication Layer

As the name suggests, this is the lowest layer of our architecture. This layer includes all hardwares and softwares provided by the league. The robots, infrared transmitter, video camera, communication network, and vision systems belong to this layer. Different leagues in RoboCupSoccer are represented by different Low Level Communication Layer. E-League has the robots, Doraemon vision package, and communication server as part of this layer, whereas Simulation League has only The RoboCup Soccer Simulator as part of this layer.

3.1.2 Logical Communication Layer

This layer acts as the interface between low level communication layer and the upper layers. It is intended to hide physical structure of the environment from the upper layer. As long as the interface of the services offered by this layer remain unchanged, then the rest of the upper layer can also remain unchanged García et al. (2004). Basic services that should be offered for E-league are :

- Reading the packets generated by video server.
- Establishing communication with the communication server.
- Continuous sensing for the referee decision.

We have used the Simuro environment FIRA (n.d.). SimuroSot consists of a server which has the soccer game environments (playground, robots, score board, etc.) and two client programs with the game strategies. A 3D color graphic screen displays the match. Teams can make their own strategies and compete with each other without hardware.

3.1.3 Sensorial/Effectorial Layer

This layer serves as a bridging layer between the logical communication layer and the cognitive layer. It translates visual information into the representation needed by cognitive layer, and also translates output from cognitive layer into basic action to be performed by the robots. In our implementation for Simulation League which use Prolog programs as cognitive layer

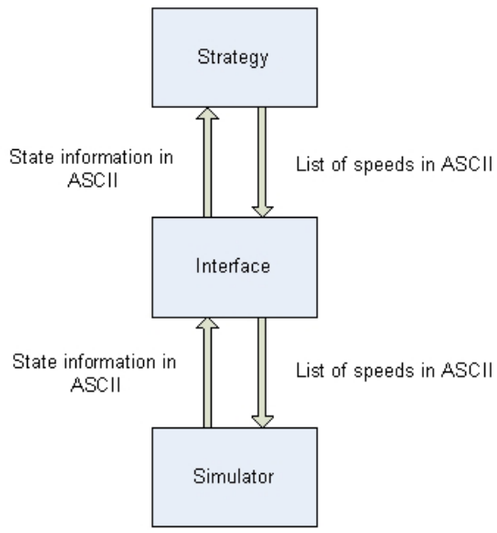

Fig. 4. Environment Architecture

and Java library as logical communication layer, this means translating visual information into prolog predicates and interpreting prolog query result. We use the Rakiduam UNCOMA (2006) interface with the dll files ("tcpb.dll" and "tcpy.dll") that are necessary in between the simulator and the Prolog compiler.

3.1.4 Cognitive Layer

Cognitive layer is where the strategy is implemented. It is the highest level layer. Our work is focused in this layer where we employ The Ciao Prolog System Hermenegildo et al. (1999), and in particulas the RFuzzy Prolog library, to reason over the provided information. Our approach is providing the capability of handling fuzzy, uncertain and incomplete information at the cognitive layer. This information is very close to the human reasoning, so this framework is improving the human-like control of this layer. A strategy can be easily implemented on this layer without having to put effort on low level technical details more related to the machine than to the human mind.
In this contribution we have changed (with respect to Hernndez & Wiguna (2007)) the fuzzy library and the RoboCupSoccer simulator to obtain more precise results related the use of fuzzy and crisp rules in the control of the robots. The fuzzy library that we use here is RFuzzy (that is decribed in detail in section 2), and for the simulation we use Rakiduam UNCOMA (2006). We can see in figure 4 the environment architecture.

3.2 Prolog Code Arquitecture

For this comparative study we have implemented two main modules and a set of auxiliary modules. We have also used a couple of communication modules from UNCOMA (2006).
The complete modules arquitecture is represented in figure 5 whose internal running is described below.

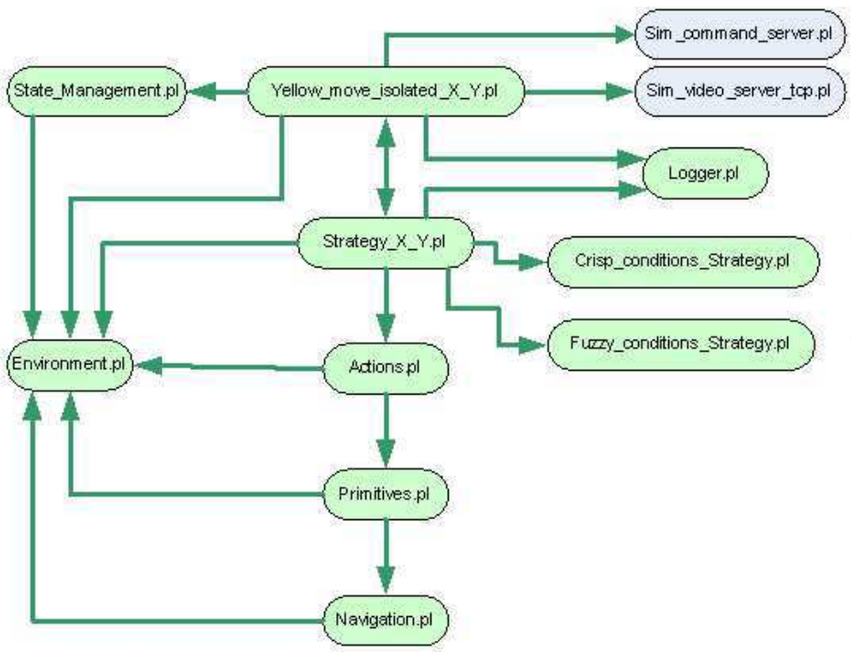

Fig. 5. Prolog code Architecture

3.2.1 Main Modules

- The module **yellow_move_isolated_X_Y.pl** starts the comunication in between the interface and the team strategy. After stablishing the connection goes into a loop for deciding (with the help of module strategy_X_Y.pl) an action for each robot. These actions should be transmited to the simulator server as a list of speeds of the set of robots. During the loop a trace of the game is generated also.
- The module **strategy_X_Y.pl** provide the strategy information to the above module to take a decision about the robots action.

3.2.2 Auxiliary Modules

- In the module **stage_management.pl** the value of the stage variable (that contains the data of the environment as for example the position of the players and the ball.
- All predicates related with loading stored data from the stage variable and creating new variables for the strategy are in the module **environment.pl**. For example getting the ball position or calculating its speed from its two last positions.
- The module **crisp_conditions_strategy.pl** contains the predicates that evaluate the conditions of the environment in each cycle and, acording to them, determine (using crisp rules) the action to develope by each robot.
- The module **fuzzy_conditions_strategy.pl** contains the predicates that evaluate the conditions of the environment in each cycle and, acording to them, determine (using fuzzy rules) the action to develope by each robot.
- The set of high level actions (e.g. shoot, pass, etc.) that the robots can developed are implemente in the module **actions.pl**.

Fig. 6. Robots positions

- The set of medium level actions (e.g. moving the robot to a concrete position) that the robots can developed are implemente in the module **primitives.pl**.
- The set of medium level actions (e.g. moving the robot to a concrete position) that the robots can developed are implemente in the module **navegation.pl**.
- The module **logger.pl** provide the trace of the the game

3.2.3 Communication Modules

- The module **sim_video_server_tcp.pl** abstracts the strategy programming of the communication of the video server. It receives the ambient data from the video server and it transforms them into logic rules in Prolog.
- The module **sim_command_server.pl** abstract the logic programming to the communication with the interface. It maintains the communication with the command server and decodify the Prolog logic rules for being understandable by the command server.

4. Comparative Study

For testing the use of different logics we have define the structure of the comparative study. We have design and implement all modules (described in section 3.2) that are necessary to model the basic strategy of a math, the decission making, the actions execution and the test cases.
Figure 6 identifies the name of the position of each robot to reference them in the rest of the paper.
In this section we are going to describe some test cases. Some of them are simple moves and others are strategies in matches.

4.1 Simple Movements

For a set of simple movements we have study the behavior of the players in two aspects: decision making and action execution. We compare the control of a player implemented using crisp rules and the control of a player implemented using fuzzy rules for both aspects:

- **Decision Making**: Which one is the best action to chose. The crisp and the fuzzy variants will take different decisions sometimes. This is analyzed in the comparative study. When the decision is the same, the execution of the action will be the same because this experiments just use basic actions that are not taking into account the environment conditions. The code for this part is in the module **conditions_X_strategy.pl** where X can be crisp or fuzzy.
- **Action Execution**: How the action is executed. We have chosen some actions that depend on some factors as speed or direction of the ball. The execution of the action is examined but not the previous decision that have taken us to do it. The code for defining the execution of the basic actions (executed after the decision making process of the first experiments) and the actions that are programmed using crisp and fuzzy logic (for their comparison) is in module **actions.pl**.

The tests are clasified attending to the action that is expected to do the robot (shoot, clear or pass). We have used for the bench of tests that we have made the same structure for an action X:

1. Action X: description of the players that participate and showing (through an image) the initial position of the robots.
2. Analysis of the crisp logic in Action X
3. Analysis of the fuzzy logic in Action C
4. Crisp and Fuzzy logic comparison

4.1.1 Decision Making Analysis

The action that is made for a robot is chosen attending to a set of variables. These variables describe the environment of the game. The variables used for decision making using crip logic are different from the set of variables that are used for decision making through fuzzy logic. In figure 7 there are some of these variables that are used by module **conditions_X_strategy.pl** to decide which action to perform by the robot.
Close to the definition of values for the variables is the distribution of areas in which we have divided the game field. It is represented in figure 8.
We are going to provide a brief description of these variables to understand their relevance in the comparative study:

- **Relative position** is the position of the ball in the field attending to the position of the players with respect the bal. It is the same in fuzzy and crisp logic. There are five possible values (ofensive left side area, ofensive right side area, ofensive centre, ofensive closure, defensive area).
- **Crisp ball position** is the area from 8 where the ball is. It has twelve possible values.
- **Crisp goal direction** is the angle (in grades and always positive) that if defined in between the trayectory of the ball and the segment that joins the ball with the centre of the opposite goal area. This variable provides information about the direction of the ball with respect to the opposite goal area. The possibles values are in the following ranges: $[0^o..45^o]$ or $[135^o..180^o]$ if it is in the direction of the goal area, and $[45^o..135^o]$ if it is not.

Crisp logic variables	Fuzzy logic variables
Relative position	Relative position
Crisp ball position	Fuzzy ball position X
	Fuzzy ball position Y
Crisp goal direction	Fuzzy ball direction
Crisp ball speed	Fuzzy ball speed

Fig. 7. Decision making variables

- **Crisp ball speed** is the speed of the ball that is calculated as the distance that the ball is able to cover during a server cycle in the simulator. The possible values are: slow (less than 0.5), regular (into 0.5 and 1) and fast (greater than 1).
- **Fuzzy ball position X** is the distance from the ball to the back line of the own field. The values are in the range [0..86].
- **Fuzzy ball position Y**is the distance from the ball to the left line of the own field. The values are in the range [0..70].
- **Fuzzy ball direction** is the same concept that the crisp goal direction. The values belong the range $[0^o..180^o]$
- **Fuzzy ball speed** is the same concept that crisp ball speed but the domain is continue (speed $\in R^+$).

For the comparison of the decision making we have considered a set of crisp rules and a set of fuzzy rules to determine the best action for the robots in each situation. The rules are different for each robot depending on its position (goal keeper, striker, midfielder, etc.) Let's see a couple of examples of rules.
Crisp rules (implemented in Prolog) are of the form:

$$shoot\_striker \leftarrow ofensive\_area \wedge opposite\_area\_ball$$

where it is said that the striker should shoot if it is in the ofensive area and the ball is in the opposite area.
The list of crisp rules should be ordered according to priority because in each situation it is executed the first one that is satisfied. If any of them is satisfied, then the robot should mantain the base position that is calculated as the average point into the position of the ball and the own goal place. But to observed better the cases in which the player is not deciding to do any action we have change this base position to the same position. So, the robot that is not making any action is going to maintain its position.
Fuzzy rules (implemented in RFuzzy) are of the form:
$shoot\_striker(V) \leftarrow_{prod} relative\_position\_good\_shoot(V1), ball\_position\_X\_good\_shoot(V2),$
$ball\_position\_Y\_good\_shoot(V3), ball\_direction\_good\_shoot(V4), ball\_speed\_good\_shoot(V5)$

Fig. 8. Areas of the field

where it is said that it is going to be calculated the truth value for the action of shooting for a striker. The truth value will be a value in the range [0..1](0 means that it is a bad action for the player in its situation and 1 means that it is the perfect action for the player). Intermediate values have intermediate meanings. In this case the truth value of a set of concepts will be calculated first (if the relative position of the player is good for shooting, if the position of the ball is good for shooting, if the direcction of the ball is good for shooting and if the speed of the ball is godd for shooting) and them the truth value for shooting, V, will be calculated as the aggregation (using product operation in this case) of all the truth values previously calculated (V1, V2, V3, V4 and V5).

In figure 9 we can see the fuzzy functions that represent the concepts that are involved in the definition of the rule that we have defined above. They represent the truth value for each value of the environment.

In the case of fuzzy rules, all rules will be executed and the best action for the player situation (the one with higher truth value) will be executed. The option by default of keeping the base position has a default value of 0.2.

In a particular case, we can have all players and the ball in a particular position (as in figure 10) and then try to find out the best action to perform by a particular robot. For our experiments we use crisp and fuzzy batery of rules and we compare the results.

For the rest of the study we will analyse always a robot of the left team of the screen, so the yellow team.

We have studied different starting positions of the game that lead the robots to do the actions of shooting, passing and clearing. The results of the comparison in between the crisp and the fuzzy control for these tests are detailed in section 5.

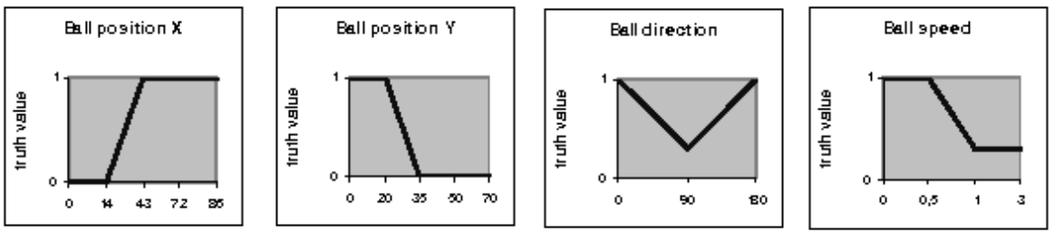

Fig. 9. Fuzzy functions to represent concepts related shooting

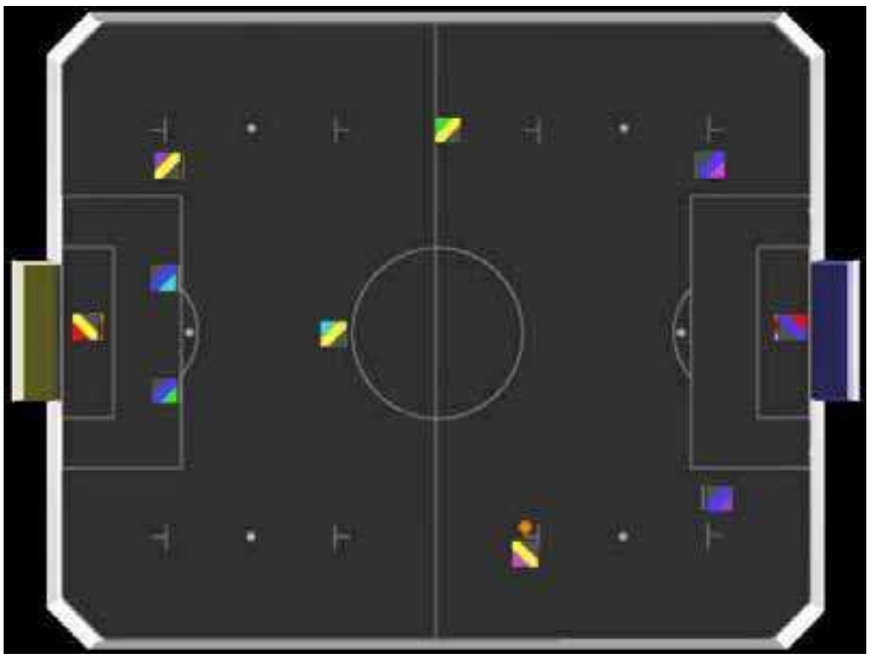

Fig. 10. Starting positions for shooting

4.1.2 Action Execution Analysis

In the analysis of the action execution we have studied starting situations in which the robot has already decided to make a particular action. Then we have observed the result when it do it using crisp and fuzzy execution rules.

The variables that we have used for shooting are listed in figure 11. Ball distance is the same in the crisp and in the fuzzy logic. The variables that we have used for passing are shown in figure 12.

The speed and direction variables are the same that the ones used in section 4.1.1 for decision making analysis. The rest of variables are defined as follow:

- **Ball distance** is the distance in between the robot that is shooting and the ball (distance $\in R^+$).

Crisp logic variables	Fuzzy logic variables
Crisp goal direction	Fuzzy ball direction
Crisp ball speed	Fuzzy ball speed
Ball distance	Ball distance

Fig. 11. Shoot action variables

Crisp logic variables	Fuzzy logic variables
Crisp centre forward position	Fuzzy centre forward position

Fig. 12. Pass action variables

- **Crisp centre forward position** is the distance from the striker that is passing the ball and the opposite passing point. It is calculated as the difference (always positive) of the striker X coordinate and the X coordinate of the opposite passing point (that is aproximately in the penalty point).
- **Fuzzy centre forward position** is the same concept that the crisp centre forward position but the values are in the range [0..75].

We have just study the most representative starting situations that lead the robot to shoot and pass. In section 5 are discussed the results of the comparison in between the crisp and the fuzzy control for these actions.

4.2 Matches

Besides studying single movements we have uses proof tests for complete matches. We have program in a different way each team. The crisp team uses crisp logic for decision making and also crips logic for action execution. The fuzzy team uses fuzzy logic for decision making and also crips logic for action execution. We will use also the Lingo team and the Rakiduam team. In the tests the yellow team is team 1 and the blue team is team 2.
For each scenario the test is a 30 seconds match where it is compared the number of gols of each team and the percentage of time that each team control the ball.
We have evaluated the mathes: crisp team vs Lingo team, crisp team vs Lingo team, crisp logic vs Rakiduam07, fuzzy logic vs Rakiduam07, and crisp team vs fuzzy team.

5. Conclusions

RFuzzy has many advantages related its expressivity and some advanced characteristics of *RFuzzy* are missing in other similar tools as FLOPERMoreno (2006); Morcillo & Moreno (2008). *RFuzzy* has been designed with a simple syntax to facilitate programmers from other fields (as

in this case from Robot Soccer programming) to model their problems in a simple way. This is the reason why *RFuzzy* is much more convenient that *Fuzzy Prolog* (that use constrains that are much more difficult to handle and understand that real numbers that are used in *RFuzzy*). Extensions added to *Prolog* by *RFuzzy* are: types (subsection 2.1), default truth values conditioned or general (subsection 2.5), assignment of truth values to individuals by means of facts (subsection 2.2), functions (subsection 2.3) or rules with credibility (subsection 2.4).
One of the most important consequences of these extensions is the constructivity of the answers with the possibility of constraining the truth value in the queries as we describe in section 2.6.
There are countless applications and research lines which can benefit from the advantages of using the fuzzy representations offered by *RFuzzy*. Some examples are: Search Engines, Knowledge Extraction (from databases, ontologies, etc.), Semantic Web, Business Rules, Coding Rules, etc.
In particular in this work we have studied the possibilites of this tool for modelling the robot control in Robot Soccer.
It is well known that logic programming is a perfect environment for dealing with the cognitive layer at RoboCupSoccer league as it is in general to implement cognitive and control issues in robotics.
Our goal is to provide a programming framework to Robot Soccer programmers to model robot control in an expressive but simple way.
After some preliminary groundwork Hernndez & Wiguna (2007) we have developed a better engine for rules execution (*RFuzzy* instead of the discrete constraint variant used in Hernndez & Wiguna (2007), called *dfuzzy*) and we have designed and provided a set of unitary test to compare the behaviour of a crisp and a fuzzy strategy for simple movements and for complete matches. Our goal is to provide this framework and some practical results (based in our experimentation) for its use in the strategy programming at Robot Soccer.
After evaluating some study tests we can provide the following conclusions:

- Using fuzzy logic we can model some concepts that are impossible to represent in an adequate way using crisp logic or other representation (i.e. fast, slow, close, far, etc.) Due to this the rules to define robot control are much more expressive and alike to human reasoning.
- *RFuzzy* lets us define continue functions over real numbers using syntactic sugar. Other tools requires to provide values for all elements of the domain. This is impossible for an infinite domain (that is the general case). So, a simple syntax is available.
- Using fuzzy logic we can distinguish the level of satisfaction of a rule. In crisp logic, rules can be satified or not. In *RFuzzy* we can obtain different truth vales of satisfaction for the set of rules that can be applied in a particular situation. So the robot can choose at any time the best rule (the one with highest truth value) that is suppose to provide it the best decision about which action to make.
- The tests comparing single movements show similar efectivenes in fuzzy logic and crisp logic. When both take the same decision, fuzzy logic is faster because it has not to wait till any limit to asign a truth value while crisp logic depend on when the ball across limit areas.
- Atending to decision making, fuzzy control is much better and the disadvantage comes from the speed. In this case it does not affect the experiments because it depends on the cycle and it comes determined by the simulator.

- The tests comparing complete match strategies show that fuzzy control is much better in taking decisions. Due to the importance of speed in this kind of game, an ofensive strategy can obtain better results even if it fails frecuently in the decisions.

Despite the results are good for our experiments in Robot Soccer, they are much better for scenarios in which it is more important to take the right decision that to decide fast. Do not fail in the decision is important in some parts of the Robot Soccer strategy but not in all of it because in much parts the speed is the decisive parameter.

6. References

Baldwin, J. F., Martin, T. P. & Pilsworth, B. W. (1995). *Fril: Fuzzy and Evidential Reasoning in Artificial Intelligence*, John Wiley & Sons.

Bistarelli, S., Montanari, U. & Rossi, F. (2001). Semiring-based constraint Logic Programming: syntax and semantics, *ACM TOPLAS*, Vol. 23, pp. 1–29.

Chen, M., K.Dorer & E.Foroughi (2003). *Users Manual RoboCup Soccer Server*.

CLIP Lab (n.d.). The ciao prolog development system www site.
URL: *http://www.clip.dia.fi.upm.es/Software/Ciao/*

Default values to handel Incomplete Fuzzy Information (2006). Vol. 14 of *IEEE Computational Intelligence Society Electronic Letter, ISSN 0-7803-9489-5*, IEEE.

Extending Prolog with Incomplete Fuzzy Information (2005). Proceedings of the 15th International Workshop on Logic Programming Environments.

FIRA (n.d.). Simurosot environment for soccer game.
URL: *http://www.fira.net/soccer/simurosot/overview.html*

García, A., G.I.Simari & T.Delladio (2004). Designing an Agent System for Controlling a Robotic Soccer Team. Argentine Conference on Computer Science (CACIC 2004).
URL: *http://www.cs.umd.edu/ gisimari/publications/cacic2004GarciaSimariDelladio.pdf*

Guadarrama, S., Munoz-Hernandez, S. & Vaucheret, C. (2004). Fuzzy Prolog: A new approach using soft constraints propagation, *Fuzzy Sets and Systems* **144**(1): 127–150. ISSN 0165-0114.

Guadarrama, S., S.Muñoz & C.Vaucheret (2004). Fuzzy prolog: A new approach using soft constraints propagation, *Fuzzy Sets and Systems* **144**(1): 127–150.

Hermenegildo, M., Bueno, F., Cabeza, D., García de la Banda, M., López, P. & Puebla, G. (1999). The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems, *Parallelism and Implementation of Logic and Constraint Logic Programming*, Nova Science, Commack, NY, USA.

Hernndez, S. M. & Wiguna, W. S. (2007). Fuzzy cognitive layer in robocupsoccer, *Proceedings of the 12th International Fuzzy Systems Association World Congress (IFSA 2007). Foundations of Fuzzy Logic and Soft Computing*, Springer, Cancn, Mxico, pp. 635–645.

Ishizuka, M. & Kanai, N. (1985). Prolog-ELF incorporating fuzzy Logic, *International Joint Conference on Artificial Intelligence*, pp. 701–703.

Klawonn, F. & Kruse, R. (1994). A Łukasiewicz logic based Prolog, *Mathware & Soft Computing* **1**(1): 5–29.
URL: *citeseer.nj.nec.com/227289.html*

Klement, E., Mesiar, R. & Pap, E. (n.d.). Triangular norms, Kluwer Academic Publishers.

Lee, R. C. T. (1972). Fuzzy Logic and the resolution principle, *Journal of the Association for Computing Machinery* **19**(1): 119–129.

Li, D. & Liu, D. (1990). *A Fuzzy Prolog Database System*, John Wiley & Sons, New York.

Morcillo, P. & Moreno, G. (2008). Floper, a fuzzy logic programming environment for research, *Proceedings of the Spanish Conference on Programming and Computer Languages, PROLE 2008*, Gijón, Spain.

Moreno, G. (2006). Building a fuzzy transformation system., *SOFtware SEMinar 2006: Theory and Practice of Computer Science*, pp. 409–418.

Munoz-Hernandez, S., Vaucheret, C. & Guadarrama, S. (2002). Combining crisp and fuzzy Logic in a prolog compiler, *in* J. J. Moreno-Navarro & J. Mariño (eds), *Joint Conf. on Declarative Programming: APPIA-GULP-PRODE 2002*, Madrid, Spain, pp. 23–38.

Pradera, A., Trillas, E. & Calvo, T. (2002). A general class of triangular norm-based aggregation operators: quasi-linear t-s operators, *International Journal of Approximate Reasoning* **30**(1): 57–72.

Shen, Z., Ding, L. & Mukaidono, M. (1989). Fuzzy resolution principle, *Proc. of 18th International Symposium on Multiple-valued Logic*, Vol. 5.

Trillas, E., Cubillo, S. & Castro, J. L. (1995). Conjunction and disjunction on $([0,1], <=)$, *Fuzzy Sets and Systems* **72**: 155–165.

UNCOMA (2006). Diseño e implementación de un sistema multiagente: Un equipo de fútbol con robots.
URL: *http://code.google.com/p/rakiduam*

Vaucheret, C., Guadarrama, S. & Munoz-Hernandez, S. (2002). Fuzzy prolog: A simple general implementation using clp(r), *in* M. Baaz & A. Voronkov (eds), *Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2002*, number 2514 in *LNAI*, Springer-Verlag, Tbilisi, Georgia, pp. 450–463.

Vojtas, P. (2001). Fuzzy logic programming, *Fuzzy Sets and Systems* **124**(1): 361–370.

Robot Soccer
Edited by Vladan Papi

ISBN 978-953-307-036-0
Hard cover, 348 pages
Publisher InTech
Published online 01, January, 2010
Published in print edition January, 2010

The idea of using soccer game for promoting science and technology of artificial intelligence and robotics was presented in the early 90s of the last century. Researchers in many different scientific fields all over the world recognized this idea as an inspiring challenge. Robot soccer research is interdisciplinary, complex, demanding but most of all, fun and motivational. Obtained knowledge and results of research can easily be transferred and applied to numerous applications and projects dealing with relating fields such as robotics, electronics, mechanical engineering, artificial intelligence, etc. As a consequence, we are witnesses of rapid advancement in this field with numerous robot soccer competitions and a vast number of teams and team members. The best illustration is numbers from the RoboCup 2009 world championship held in Graz, Austria which gathered around 2300 participants in over 400 teams from 44 nations. Attendance numbers at various robot soccer events show that interest in robot soccer goes beyond the academic and R&D community. Several experts have been invited to present state of the art in this growing area. It was impossible to cover all aspects of the research in detail but through the chapters of this book, various topics were elaborated. Among them are hardware architecture and controllers, software design, sensor and information fusion, reasoning and control, development of more robust and intelligent robot soccer strategies, AI-based paradigms, robot communication and simulations as well as some other issues such as educational aspect. Some strict partition of chapter in this book hasn't been done because areas of research are overlapping and interweaving. However, it can be said that chapters at the beginning are more system-oriented with wider scope of presented research while later chapters generally deal with some more particular aspects of robot soccer.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Susana Munoz Hernandez (2010). RFuzzy: an Easy and Expressive Tool for Modelling the Cognitive Layer in RoboCupSoccer, Robot Soccer, Vladan Papi (Ed.), ISBN: 978-953-307-036-0, InTech, Available from: http://www.intechopen.com/books/robot-soccer/rfuzzy-an-easy-and-expressive-tool-for-modelling-the-cognitive-layer-in-robocupsoccer

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元

www.intechopen.com

Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen

IntechOpen