# Rfuzzy Framework

Víctor Pablos Ceruelo, Susana Muñoz-Hernandez and Hannes Straß

Babel Group, UPM, Spain

December 2008

# Index


1. Motivation
2. Credibility and Certainty
3. Rfuzzy syntax
4. Queries and Constructivity
5. Implementation details
6. Conclusions
7. Applications and Current work.

# Motivation

The Fuzzy prolog system was one of the most promising frameworks for modelling credibility and uncertainty.

Main problems with it are its complexity, due to the use of intervals between real numbers to represent truth values, its answers to queries with constraints and the necessity to take care of some variables to manage truth values.

Rfuzzy tries to reduce its complexity in some aspects:

- Use real numbers instead of intervals between real numbers to represent truth values.
- Answer queries with direct values instead of constraints.
- Avoid the necessity to code variables to manage truth values.

# Credibility and Uncertainty

- Certainty: How fast went the car?
  - Mom says car went fast with a truth value of *0.9*.
  - Dad says car went fast with a truth value of *0.6*.
- Credibility: Mom vs Dad fast idea.
  - If we use mom's truth value in a rule, we could assign that rule a credibility of *0.7*
  - If we use dad's truth value in a rule, we could assign that rule a credibility of *0.8*

# Rfuzzy syntax (I): Type definition

Types are the correct way to define the individuals we use in our programs. It enables constructivity.

$$\textbf{:- set\_prop}\ pred/ar\ =>\ type\_pred\_1/1\ [,\ type\_pred\_2/1\ ]^{*}\ . \qquad (1)$$

where *set_prop* is a reserved word, *pred* is the name of the typed predicate, *ar* is its arity and *type_pred_{n}* is the predicate used to assure that the value given to the argument in the position *n* of a call to *pred/ar* is correctly typed. Predicate *type_pred_{n}* must have arity 1
Example:

```
:- set_prop has_lower_price/2 => car/1, car/1.
car(vw_caddy).
car(alfa_romeo_gt).
car(aston_martin_bulldog).
car(lamborghini_urraco).
```

# Rfuzzy syntax (II): Fact truth value

To define fact truth values in programs we use the following 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.
Example:

```
expensive_car(alfa_romeo_gt) value 0.6 .
```

# Rfuzzy syntax (III): Functional Truth Value (I)

Fact truth value definition is worth for a finite (and relative small) number of individuals. As we may want to define a big amount of individuals, we need more than this.

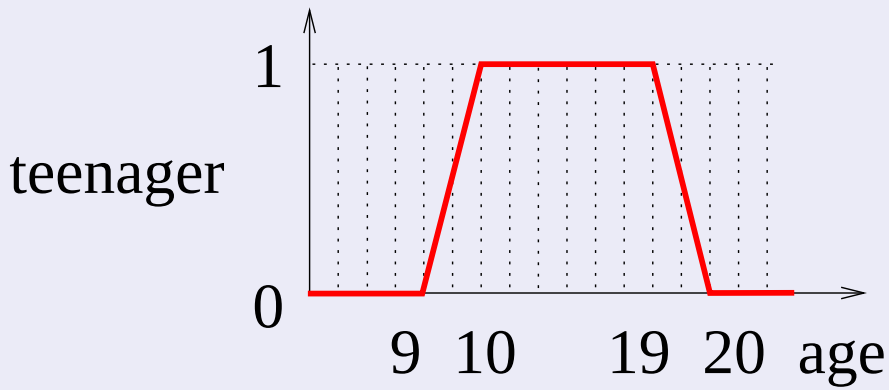


Figura: Teenager credibility.

# Rfuzzy syntax (III): Functional Truth Value (II)

$$pred :\# ([(arg1, truth\_val1),\ (arg2, truth\_val2)\ [,\ (arg3, truth\_val3)\ ]^*])\ . \tag{3}$$

*arg1, ..., argN* should be ground terms (numbers) and *truth_val1, ..., truth_valN* should be border truth values. The truth value of the rest of the elements (apart from the border elements) is obtained by interpolation.
Example:

```
:- set_prop teenager/1 => people_age/1.
:- default(teenager/1, 0).
teenager :# ([ (9, 0), (10, 1), (19, 1), (20, 0) ]) .
```

# Rfuzzy syntax (IV): Rule truth value

Credibility is used to express how much we trust the rule we write.

Example: How much do we trust the following sentence?

*If it is windy and cold, it never rains*

*Rfuzzy* offers the user a concrete syntax to define combinations of truth values by means of aggregation operations, and assign to that rules a credibility.

$$pred(arg1\ [,\ arg2]^*\ )\ [\ \textbf{cred}\ (op1, value1)\ ]^{0,1}\textbf{:} \sim$$
$$op2\ pred1(arg1\ [,\ arg2]^*\ )\ [,\ pred2(arg1\ [,\ arg2]^*\ )]\ . \qquad (4)$$

We can choose two aggregation operators, *op2* for combining the truth values of the subgoals of the rule's body and *op1* for combining the previous result with the credibility of the rule.

# Rfuzzy syntax (V): Rule truth value

Example 1:

```
tempting_restaurant(R) :~ prod low_distance(R), cheap(R),
                          traditional(R).
```

Example 2:

```
good_player(J) cred (min, 0.8) :~ prod swift(J), tall(J),
                                  experience(J).
```

Example 3:

```
never_rains(X) cred (prod, 0.6) :~ prod windy(X), cold(X).
```

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

# Rfuzzy (VI): General and Conditioned default truth values

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*.

*Conditioned default truth value* is used when the default truth value only applies to a subset of the function's domain. This subset is defined by a membership predicate which is true only when an individual belongs to the subset.

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

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

NOTE: The membership predicate ( *membership_predicate/ar* ) and the predicate to which it is applied ( *pred/ar* ) need to be have the same arity ( *ar* ). If not, an error message will be shown at compilation time.

# Rfuzzy (VII): General and Conditioned default truth values

The example below shows how to assign a default truth value of 0.5 to all cars that do not have an explicit truth value nor have a default conditioned truth value. Besides, it shows how to assign a conditioned default truth value to all cars belonging to a small subset and not having an explicit truth value.

Example:

```
:- set_prop expensive_car/1 => car/1.
:- default(expensive_car/1, 0.9) => expensive_type/1.
:- default(expensive_car/1, 0.5).

expensive_type(lamborghini_urraco).
expensive_type(aston_martin_bulldog).
```

# Queries and Constructivity (I)

- Queries:
  Indeed the program has to be run. When compiling, *Rfuzzy* adds a new argument to the arguments list of each fuzzy predicate. This argument serves for querying about the predicate truth value. It can be seen as syntactic sugar, as truth value is not part of the predicate arguments, but metadata information.
- Constructivity:
  A fuzzy tool should be able to provide constructive answers for queries. The regular (easy) questions are asking for the truth value of an element. But the really interesting queries are the ones that ask for values that satisfy constraints over the truth value. RFuzzy provides this constructive functionality.

Example:
Non-constructive answer: how expensive is an *alfa_romeo_gt*?
Constructive answer: which cars are very expensive?

# Queries and Constructivity (II)

Non-constructive answer example:

```
?- expensive_car(alfa_romeo_gt,V).
V = 0.6 ? ;
no
```

Constructive answer example:

```
?- expensive_car(X,V), V > 0.8.
V = 0.9, X = aston_martin_bulldog ? ;
V = 0.9, X = lamborghini_urraco ? ;
no
```

First example simply looks for the truth value of the individual, but second one tries which each individuals until it finds one with a truth value bigger than *0.8*.

# Implementation details (I)

It is implemented as a Ciao Prolog package because Ciao Prolog offers the possibility of dealing with a higher order compilation through the implementation of Ciao packages. Those packages serve as input for the *"Ciao System Preprocessor"* (CiaoPP), a tool able to perform source-to-source transformations. The reason beyond the

implementation of *Rfuzzy* as a Ciao Prolog package is that the resultant code has to deal with two kinds of queries:

- queries in which the user asks for the truth value of an individual (example I), and
- queries in which the user asks for an individual with a concrete truth value (example II).

# Implementation details (II)

Example (I):

```
?- A is 1, B is 2, C is A + B.

A = 1, B = 2, C = 3 ? .
yes
```

Example (II):

```
?- C is 3, C is A + B.
{ERROR: illegal arithmetic expression}
{ERROR: illegal arithmetic expression}
no
?-
```

# Implementation details (III)

The global compilation process has two preprocessor steps:

1. the Rfuzzy program is translated into CLP($\mathcal{R}$) constraints by means of the Rfuzzy package and
2. those constraints are translated into ISO Prolog by using the CLP($\mathcal{R}$) package.

The following Figure shows the whole process.

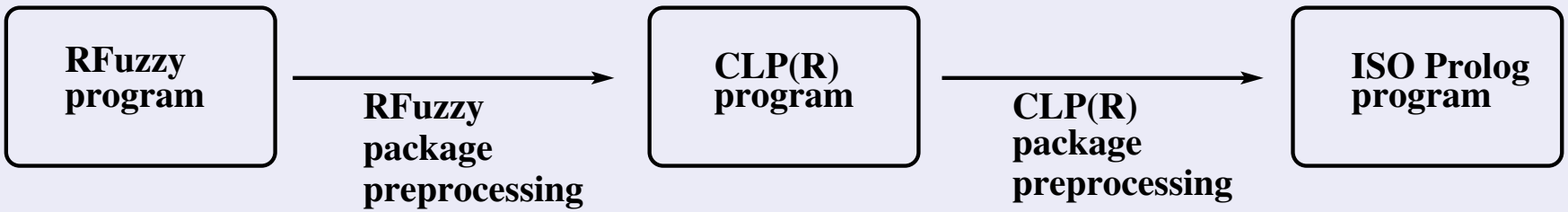


Figura: Rfuzzy architecture.

# Implementation details (IV)

Small Rfuzzy program:

```
% Rfuzzy program
tempting_restaurant(R) :~ prod low_distance(R), cheap(R),
                             traditional(R).
```

Small program is translated into a CLP($\mathcal{R}$) program:

```
% CLP(R) program
rfuzzy_rule_tempting_restaurant(R,_1) :-
        low_distance(R,_2),
        cheap(R,_3),
        traditional(R,_4),
        inject([_2,_3,_4],prod,_1),
        _1 .>=. 0,
        _1 .=<. 1.
```

# Implementation details (V)

Small program is translated into a ISO Prolog program:

```
% ISO Prolog program
rfuzzy_rule_tempting_restaurant(R,_1) :-
        low_distance(R,_2),
        cheap(R,_3),
        traditional(R,_4),
        inject([_2,_3,_4],prod,_1),
        solve_generic_1(le,0,_1,-1),
        solve_generic_1(le,-1,_1,1) .
```

# Implementation details (VI)

*Rfuzzy package simplified skeleton*

```
Main :- Types, ( Normal ; Default)

Normal :- ( Fact ;
            (\+(Fact_Aux), Function) ;
            (\+(Fact_Aux), \+(Function_Aux), Rule)
          )

Default :- \+(Fact_Aux), \+(Function_Aux), \+(Rule_Aux),
             ( Cl_With_Cond ;
               (\+(Cl_With_Cond_Aux), Cl_With_No_Cond)
             )
```

# Conclusions

- *Rfuzzy* offers to the users a new framework to represent fuzzy problems over real numbers.
- Main *Rfuzzy* advantages over *Fuzzy Prolog* are a simpler syntax and the elimination of answers with constraints.
  - Its fuzzy values are simple real numbers instead of intervals between real numbers,
  - it hides the management of truth value variables and
  - as it does not answer queries with constraints, the user always receives ground terms. This is a more human-readable way of answering questions.
- Extensions added to *Prolog* by *Rfuzzy* are: Types, default truth values (conditioned or not), assignment of truth values to individuals by means of facts, functions or rules, and assignment of credibility to the rules.

# Applications and Current work

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, and Coding Rules (where the violation of one rule can be given a truth value).

Current work on Rfuzzy tries to apply constructive negation to the engine. RFuzzy needs to define types in a constructive way (by means of predicates that are able to generate all their individuals by backtracking) so we cannot use constraints. Future research is being done in this line for widening the definition of types.

# Questions