14

Fuzzy Prolog: Default Values to Represent Missing Information

Susana Munoz-Hernandez[1] and Claudio Vaucheret[2]

[1] Departamento de Lenguajes, Sistemas de la Información e Ingeniería del Software, Facultad de Informática - Universidad Politécnica de Madrid, Campus de Montegancedo 28660 Madrid, Spain
susana@fi.upm.es

[2] Departamento de Ciencias de la Computación, Facultad de Economía y Administración, Universidad Nacional del Comahue, Universidad Politécnica de Madrid, Buenos Aires 1400 (8300), Neuquen, Argentina
vaucheret@gmail.com

Summary. Incomplete information is a problem in many aspects of actual environments. Furthermore, in many scenarios the knowledge is not represented in a crisp way. It is common to find fuzzy concepts or problems with some level of uncertainty. There are not many practical systems which handle fuzziness and uncertainty and the few examples that we can find are used by a minority. To extend a popular system (which many programmers are using) with the ability of combining crisp and fuzzy knowledge representations seems to be an interesting issue.

Fuzzy Prolog [5] is a language that models fuzziness and uncertainty. In this chapter we enhance Fuzzy Prolog by using default knowledge to represent incomplete information in Logic Programming. We also provide the implementation of this new framework. This new release of Fuzzy Prolog handles incomplete information, it has a complete semantics (the previous one was incomplete as Prolog) and moreover it is able to combine crisp and fuzzy logic in Prolog programs. Therefore, new Fuzzy Prolog is more expressive to represent real world.

Fuzzy Prolog inherited from Prolog its incompleteness. The incorporation of default reasoning to Fuzzy Prolog removes this problem and requires a richer semantics which it is discussed.

14.1 Introduction

World information is not represented in a crisp way. Its representation is imperfect, fuzzy, etc., so that the management of uncertainty is very important

This research was partly supported by the Spanish MCYT project TIC2003-01036.

in knowledge representation. There are multiple frameworks for incorporating uncertainty in logic programming:

- fuzzy set theory
- probability theory
- multi-valued logic
- possibilistic logic

In [10] a general framework was proposed that generalizes many of the previous approaches. At the same time an analogous theoretical framework was provided and a prototype for Prolog was implemented [21]. Basically, a rule is of the form $A \leftarrow B_1, \ldots, B_n$, where the assignment I of certainties is taken from a certainty lattice, to the B_is. The certainty of A is computed by taking the set of the certainties $I(B_i)$ and then they are propagated using the function F that is an aggregation operator. This is a very flexible approach and in [5, 22] practical examples in a Prolog framework are presented.

In this work we extend the approach of [5] with arbitrary assignments of default certainty values (non-uniform default assumptions). The usual semantics of logic programs can be obtained through a unique computation method, but using different assumptions in a uniform way to assign the same default truth-value to all the atoms. The most well known assumptions are:

- the *Closed World Assumption* (CWA), which asserts that any atom whose truth-value cannot be inferred from the facts and clauses of the program is supposed to be false (i.e. certainty 0). It is used in stable models [2, 3] and well-founded semantics [12, 13, 15],
- the *Open World Assumption* (OWA), which asserts that any atom whose truth-value cannot be inferred from the facts and clauses of the program is supposed to be undefined or unknown (i.e. certainty in $[0, 1]$). It is used in [10].

There are also some approaches [23, 24] where both assumptions can be combined and some atoms can be interpreted assuming CWA while others follows OWA. Anyway, what seems really interesting is not only to combine both assumptions but to generalize the use of a default value. The aim is working with incomplete information with more guarantees.

The rest of the paper is organized as follows. Section 14.2 introduces the Fuzzy Prolog language. A complete description of the new semantics of Fuzzy Prolog is provided in Section 14.3. Section 14.4 completes the details about the improved implementation using CLP($\mathcal{R}$) with the extension to handle default knowledge. Some illustrating examples are provided in section 14.5. Finally, we conclude and discuss some future work in section 14.6.

14.2 Fuzzy Prolog

In this section we are going to summarize the main characteristics of the Fuzzy Prolog that we proposed in [5] and that is the basis of the work presented here.

Fuzzy Prolog is more general than previous approaches to introduce fuzziness in Prolog in some respects:

1. A truth value will be a finite union of closed sub-intervals on $[0, 1]$. This is represented by Borel algebra, $\mathcal{B}([0, 1])$, while the algebra $\mathcal{E}([0, 1])$ only considers intervals. A single interval is a special case of union of intervals with only one element, and a unique truth value is a particular case of having an interval with only one element.
2. A truth value will be propagated through the rules by means of an *aggregation operator*. The definition of *aggregation operator* is general in the sense that it subsumes conjunctive operators (triangular norms [9] like min, prod, etc.), disjunctive operators [19] (triangular co-norms, like max, sum, etc.), average operators (like arithmetic average, quasi-linear average, etc) and hybrid operators (combinations of the above operators [17]). In [11][3] a resolution-Like Strategy based on a Lattice-Value Logic is proposed, as in our approach, although it is limited to the Lukasiewicz's implication operator.
3. The declarative and procedural semantics for Fuzzy Logic programs are given and their equivalence is proved.
4. An implementation of the proposed language is presented. A *fuzzy program* is a finite set of
   - *fuzzy facts* ($A \leftarrow v$, where A is an atom and v, a truth value, is an element in $\mathcal{B}([0, 1])$ expressed as constraints over the domain $[0, 1]$), and
   - *fuzzy clauses* ($A \leftarrow_F B_1, \ldots, B_n$, where $A, B_1, \ldots, B_n$ are atoms, and F is an interval-aggregation operator, which induces a union-aggregation, as by Definition 14.2, $\mathcal{F}$ of truth values in $\mathcal{B}([0, 1])$ represented as constraints over the domain $[0, 1]$).

   We obtain information from the program through *fuzzy queries or fuzzy goals* ($v \leftarrow A$? where A is an atom, and v is a variable, possibly instantiated, that represents a truth value in $\mathcal{B}([0, 1])$).

Programs are defined as usual but handling truth values in $\mathcal{B}([0, 1])$ (the Borel algebra over the real interval $[0, 1]$ that deals with unions of intervals) represented as constraints. We refer, for example, to expressions as: $(v \geq 0.5 \ \wedge \ v \leq 0.7) \ \vee \ (v \geq 0.8 \ \wedge \ v \leq 0.9)$ to represent a truth value in $[0.5, 0.7] \bigcup [0.8, 0.9]$.

A lot of everyday situations can only be represented by this general representation of truth value. There are some examples in [5].

The truth value of a goal will depend on the truth value of the subgoals which are in the body of the clauses of its definition. Fuzzy Prolog [5] uses *aggregation operators* [20] in order to propagate the truth value by means

[3] This work discusses the resolution based on a Lattice-Valued Logic for the Prolog language at theoretical level. In our approach we provide also an operational semantics and an implementation using an extended Prolog with constraints.

of the fuzzy rules. Fuzzy sets *aggregation* is done using the application of a numeric operator of the form $f : [0,1]^n \to [0,1]$. An *aggregation operator* must verify $f(0,\ldots,0) = 0$ and $f(1,\ldots,1) = 1$, and in addition it should be monotonic and continuous. If we deal with the definition of fuzzy sets as intervals it is necessary to generalize from *aggregation operators* of numbers to *aggregation operators* of intervals. Following the theorem proved by Nguyen and Walker in [16] to extend T-norms and T-conorms to intervals, we propose the following definitions.

Definition 14.1 (interval-aggregation) *Given an aggregation* $f : [0,1]^n \to [0,1]$*, an interval-aggregation* $F : \mathcal{E}([0,1])^n \to \mathcal{E}([0,1])$ *is defined as follows:*

$$F([x_1^l, x_1^u], ..., [x_n^l, x_n^u]) = [f(x_1^l, ..., x_n^l), f(x_1^u, ..., x_n^u)].$$

Actually, we work with union of intervals and propose the definition:

Definition 14.2 (union-aggregation) *Given an interval-aggregation* $F : \mathcal{E}([0,1])^n \to \mathcal{E}([0,1])$ *defined over intervals, a union-aggregation* $\mathcal{F} : \mathcal{B}([0,1])^n \to \mathcal{B}([0,1])$ *is defined over union of intervals as follows:*

$$\mathcal{F}(B_1, \ldots, B_n) = \cup\{F(\mathcal{E}_1, ..., \mathcal{E}_n) \mid \mathcal{E}_i \in B_i\}.$$

In the presentation of the theory of possibility [25], Zadeh considers that fuzzy sets act as an elastic constraint on the values of a variable and fuzzy inference as constraint propagation.

In [5] (and furthermore in the extension that we presented in this paper), truth values and the result of aggregations are represented by constraints. A constraint is a Σ*-formula* where Σ is a signature that contains the real numbers, the binary function symbols $+$ and $*$, and the binary predicate symbols $=$, $<$ and $\leq$. If the constraint c has solution in the domain of real numbers in the interval $[0,1]$ then c is *consistent*, and is denoted as *solvable*(c).

14.3 Semantics

This section contains a reformulation of the semantics of Fuzzy Prolog. This new semantics is complete thanks to the inclusion of default value.

14.3.1 Least model semantics

The *Herbrand universe* U is the set of all ground *terms*, which can be made up with the constants and function symbols of a program, and the *Herbrand base* B is the set of all ground atoms which can be formed by using the predicate symbols of the program with ground *terms* (of the *Herbrand universe*) as arguments.

Definition 14.3 (default value) *We assume there is a function* default *which implement the Default Knowledge Assumptions. It assigns an element of* $\mathcal{B}([0,1])$ *to each element of the Herbrand Base. If the Closed World Assumption is used, then default*$(A) = [0,0]$ *for all* A *in Herbrand Base. If Open World Assumption is used instead, default*$(A) = [0,1]$ *for all* A *in Herbrand Base.*

Definition 14.4 (interpretation) *An* interpretation $I = \langle B_I, V_I \rangle$ *consists of the following:*

1. *a subset* B_I *of the* Herbrand Base*,*
2. *a mapping* V_I*, to assign*
   a) *a truth value, in* $\mathcal{B}([0,1])$*, to each element of* B_I*, or*
   b) *default*(A)*, if* A *does not belong to* B_I*.*

Definition 14.5 (interval inclusion $\subseteq_{II}$) *Given two intervals* $I_1 = [a,b]$*,* $I_2 = [c,d]$ *in* $\mathcal{E}([0,1])$*,* $I_1 \subseteq_{II} I_2$ *if and only if* $c \leq a$ *and* $b \leq d$*.*

Definition 14.6 (Borel inclusion $\subseteq_{BI}$) *Given two unions of intervals* $U = I_1 \cup \cdots \cup I_N$*,* $U' = I'_1 \cup \cdots \cup I'_M$ *in* $\mathcal{B}([0,1])$*,* $U \subseteq_{BI} U'$ *if and only if* $\forall I_i \in U$*,* $i \in 1..N$*,* I_i *can be partitioned in to intervals* $J_{i1}, ..., J_{iL}$*, i.e.* $J_{i1} \cup ... \cup J_{iL} = I_i$*,* $J_{i1} \cap ... \cap J_{iL}$ *is the set of the border elements of the intervals (except the lower limit of* J_{i1} *and the upper limit of* J_{iL}*) and for all* $k \in 1..L$*,* $\exists J'_{jk} \in U'$ *.* $J_{ik} \subseteq_{II} J'_{jk}$ *where* $jk \in 1..M$*.*

The Borel algebra $\mathcal{B}([0,1])$ is a complete lattice under $\subseteq_{BI}$ (Borel inclusion), and the Herbrand base is a complete lattice under $\subseteq$ (set inclusion) and so the set of all *interpretations* forms a complete lattice under the relation $\sqsubseteq$ defined as follows.
Notice that we have redefined interpretation and Borel inclusion with respect to the definitions in [5]. We will also redefine the operational semantics and therefore the internal implementation of the Fuzzy Prolog library. Sections below are completely new too. For uniformity reasons we have kept the same syntax that was used in [5] in fuzzy programs.

Definition 14.7 (interpretation inclusion $\sqsubseteq$) *Let* $I = \langle B_I, V_I \rangle$ *and* $I' = \langle B_{I'}, V_{I'} \rangle$ *be interpretations.* $I \sqsubseteq I'$ *if and only if* $B_I \subseteq B_{I'}$ *and for all* $B \in B_I$*,* $V_I(B) \subseteq_{BI} V_{I'}(B)$*.*

Definition 14.8 (valuation) *A* valuation σ *of an atom* A *is an assignment of elements of* U *to variables of* A*. So* $\sigma(A) \in B$ *is a ground atom.*

In the Herbrand context, a valuation is the same as a substitution.

Definition 14.9 (model) *Given an* interpretation $I = \langle B_I, V_I \rangle$*,*

- I *is a* model *for a* fuzzy fact $A \leftarrow v$*, if for all valuations* σ*,* $\sigma(A) \in B_I$ *and* $v \subseteq_{BI} V_I(\sigma(A))$*.*

- *I is a* model *for a clause* $A \leftarrow_F B_1, \ldots, B_n$ *when the following holds: for all valuations* σ, $\sigma(A) \in B_I$ *and* $v \subseteq_{BI} V_I(\sigma(A))$, *where* $v = \mathcal{F}(V_I(\sigma(B_1)), \ldots, V_I(\sigma(B_n)))$ *and* $\mathcal{F}$ *is the union aggregation obtained from* F.
- *I is a* model *of a* fuzzy program, *if it is a* model *for the facts and clauses of the program.*

Every program has a least model which is usually regarded as the intended interpretation of the program since it is the most conservative model. Let $\cap$ (that appears in the following theorem) be the meet operator on the lattice of interpretations $(I, \sqsubseteq)$. We can prove the following result.

Theorem 14.1 (model intersection property) *Let* $I_1 = \langle B_{I_1}, V_{I_1} \rangle$, $I_2 = \langle B_{I_1}, V_{I_1} \rangle$ *be models of a fuzzy program* P. *Then* $I_1 \cap I_2$ *is a model of* P.

Proof. Let $M = \langle B_M, V_M \rangle = I_1 \cap I_2$. Since I_1 and I_2 are models of P, they are models for each fact and clause of P. Then for all valuations σ we have

- for all facts $A \leftarrow v$ in P,
  - $\sigma(A) \subseteq B_{I_1}$ and $\sigma(A) \in B_{I_2}$, and so $\sigma(A) \in B_{I_1} \cap B_{I_2} = B_M$,
  - $v \subseteq_{BI} V_{I_1}(\sigma(A))$ and $v \subseteq_{BI} V_{I_2}(\sigma(A))$, and so hence $v \subseteq_{BI} V_{I_1}(\sigma(A)) \cap V_{I_2}(\sigma(A)) = V_M(\sigma(A))$

  therefore M is a model for $A \leftarrow v$
- and for all clauses $A \leftarrow_F B_1, \ldots, B_n$ in P
  - since $\sigma(A) \in B_{I_1}$ and $\sigma(A) \in B_{I_2}$, hence $\sigma(A) \in B_{I_1} \cap B_{I_2} = B_M$.
  - if $v = \mathcal{F}(V_M(\sigma(B_1)), \ldots, V_M(\sigma(B_n)))$, since F is monotonic, $v \subseteq_{BI} V_{I_1}(\sigma(A))$ and $v \subseteq_{BI} V_{I_2}(\sigma(A))$, hence $v \subseteq_{BI} V_{I_1}(\sigma(A)) \cap V_{I_2}(\sigma(A)) = V_M(\sigma(A))$

  therefore M is a model for $A \leftarrow_F B_1, \ldots, B_n$

and M is model of P.□

Remark 14.1 (Least model semantic). If we let $\mathbf{M}$ be the set of all models of a program P, the intersection of all of these models, $\bigcap \mathbf{M}$, is a model and it is the least model of P. We denote the least model of a program P by $lm(P)$.

Example 14.1 *Let's see an example (from [5]). Suppose we have the following program* P:

$tall(peter) \leftarrow [0.6, 0.7] \vee 0.8$
$tall(john) \leftarrow 0.7$
$swift(john) \leftarrow [0.6, 0.8]$
$good\_player(X) \leftarrow_{luka} tall(X), swift(X)$

Here, we have two facts, $tall(john)$ *and* $swift(john)$ *whose truth values are the unitary interval* $[0.7, 0.7]$ *and the interval* $[0.6, 0.8]$, *respectively, and a clause for the* $good\_player$ *predicate whose* aggregation operator *is the Lukasiewicz T-norm.*

The following interpretation $I = \langle B, V \rangle$ *is a model for* P*, where*
$B = \{tall(john), tall(peter), swift(john),$
$good\_player(john), good\_player(peter)\}$ *and*

$$\begin{aligned} V(tall(john)) &= [0.7, 1] \\ V(swift(john)) &= [0.5, 0.8] \\ V(tall(peter)) &= [0.6, 0.7] \vee [0.8, 0.8] \\ V(good\_player(john)) &= [0.2, 0.9] \\ V(good\_player(peter)) &= [0.5, 0.9] \end{aligned}$$

note that for instance if $V(good\_player(john)) = [0.2, 0.5]$ $I = \langle B, V \rangle$ *cannot be a model of* P*, the reason is that* $v = luka([0.7, 1], [0.5, 0.8]) = [0.7 + 0.5 - 1, 1 + 0.8 - 1] = [0.2, 0.8] \not\subseteq_{II} [0.2, 0.5]$.

The least model of P *is the intersection of all models of* P *which is* $M = \langle B_M, V_M \rangle$ *where*
$B_M = \{tall(john), tall(peter), swift(john),$
$good\_player(john)\}$ *and*

$$\begin{aligned} V_M(tall(john)) &= [0.7, 0.7] \\ V_M(swift(john)) &= [0.6, 0.8] \\ V_M(tall(peter)) &= [0.6, 0.7] \vee [0.8, 0.8] \\ V_M(good\_player(john)) &= [0.3, 0.5]. \end{aligned}$$

14.3.2 Fixed-point semantics

The fixed-point semantics we present is based on a one-step consequence operator T_P. The least fixed-point $lfp(T_P) = I$ (i.e. $T_P(I) = I$) is the declarative meaning of the program P, so is equal to $lm(P)$. We include it here for clarity reasons although it is the same that in [5].

Let P be a fuzzy program and B_P the Herbrand base of P; then the *mapping* T_P over *interpretations* is defined as follows:

Let $I = \langle B_I, V_I \rangle$ be a fuzzy *interpretation*, then $T_P(I) = I'$, $I' = \langle B_{I'}, V_{I'} \rangle$, $B_{I'} = \{A \in B_P \mid Cond\}, V_{I'}(A) = \bigcup\{v \in \mathcal{B}([0,1]) \mid Cond\}$

where

$$\begin{aligned} Cond = \; & (A \leftarrow v \text{ is a ground instance of a fact in } P \text{ and} \\ & solvable(v)) \text{ or} \\ & (A \leftarrow_F A_1, \ldots, A_n \text{ is a ground instance of a} \\ & \text{clause in } P, \text{ and} \\ & solvable(v), v = \mathcal{F}(V_I(A_1), \ldots, V_I(A_n))). \end{aligned}$$

Note that since I' must be an interpretation, $V_{I'}(A) = default(A)$ for all $A \notin B_{I'}$.

The set of interpretations forms a complete lattice, so that T_P it is continuous.

Recall (from [5]) the definition of the *ordinal powers* of a function G over a complete lattice X:

$$G \uparrow \alpha = \begin{cases} \bigcup\{G \uparrow \alpha' \mid \alpha' < \alpha\} & \text{if } \alpha \text{ is a limit ordinal,} \\ G(G \uparrow (\alpha - 1)) & \text{if } \alpha \text{ is a successor ordinal,} \end{cases}$$

and dually,

$$G \downarrow \alpha = \begin{cases} \bigcap\{G \downarrow \alpha' \mid \alpha' < \alpha\} & \text{if } \alpha \text{ is a limit ordinal,} \\ G(G \downarrow (\alpha - 1)) & \text{if } \alpha \text{ is a successor ordinal,} \end{cases}$$

Since the first limit ordinal is 0, it follows that $G \uparrow 0 = \bot_X$ (the bottom element of the lattice X) and $G \downarrow 0 = \top_X$ (the top element). From Kleene's fixed point theorem we know that the least fixed-point of any continuous operator is reached at the first infinite ordinal ω. Hence $lfp(T_P) = T_P \uparrow \omega$.

Example 14.2 *Consider the same program P of the example 14.1 (from [5]), the ordinal powers of T_P are*
$T_P \uparrow 0 = \{\}$
$T_P \uparrow 1 = \{tall(john), swift(john), tall(peter)\}$ *and*

$$\begin{aligned} V(tall(john)) &= [0.7, 0.7] \\ V(swift(john)) &= [0.6, 0.8] \\ V(tall(peter)) &= [0.6, 0.7] \vee [0.8, 0.8] \end{aligned}$$

$T_P \uparrow 2 = \{tall(john), swift(john), tall(peter), good\_player(john)\}$ *and*

$$\begin{aligned} V(tall(john)) &= [0.7, 0.7] \\ V(swift(john)) &= [0.6, 0.8] \\ V(tall(peter)) &= [0.6, 0.7] \vee [0.8, 0.8] \\ V(good\_player(john)) &= [0.3, 0.5] \end{aligned}$$

$T_P \uparrow 3 = T_P \uparrow 2.$

Lemma 14.1 *Let P a fuzzy program. Then M is a model of P if and only if M is a pre-fixpoint of T_P, that is $T_P(M) \sqsubseteq M$.*

Proof. Let $M = \langle B_M, V_M \rangle$ and $T_P(M) = \langle B_{T_P}, V_{T_P} \rangle$.

We first prove the "only if" ($\rightarrow$) direction. Let A be an element of Herbrand Base, if $A \in B_{T_P}$, then by definition of T_P there exists a ground instance of a fact of P, $A \leftarrow v$, or a ground instance of a clause of P, $A \leftarrow_F A_1, \ldots, A_n$ where $\{A_1, \ldots, A_n\} \subseteq B_M$ and $v = \mathcal{F}(V_M(A_1), \ldots, V_M(A_n))$. Since M is a model of P, $A \in B_M$, and each $v \subseteq_{BI} V_M(A)$, then $V_{T_P}(A) \subseteq_{BI} V_M(A)$ and then $T_P(M) \sqsubseteq M$. If $A \notin B_{T_P}$ then $V_{T_P}(A) = \mathit{default}(A) \subseteq_{BI} V_M(A)$.

Analogously, for the "if" ($\leftarrow$) direction, for each ground instance
$v = \mathcal{F}(V_M(A_1), \ldots, V_M(A_n))$, $A \in B_{T_P}$ and $v \subseteq_{BI} V_{T_P}(A)$, but as $T_P(M) \sqsubseteq M$, $B_{T_P} \subseteq B_M$ and $V_{T_P}(A) \subseteq_{BI} V_M(A)$. Then $A \in B_M$ and $v \subseteq_{BI} V_M(A)$ therefore M is a model of P. $\square$

Given this relationship, it is straightforward to prove that the least model of a program P is also the least fixed-point of T_P.

Theorem 14.2 *Let P be a fuzzy program. Then $lm(P) = lfp(T_P)$.*

Proof.

$$\begin{aligned} lm(P) &= \bigcap\{M \mid M \text{ is a model of } P\} \\ &= \bigcap\{M \mid M \text{ is a pre-fixpoint of } P\} \\ &\quad \text{from lemma 14.1} \\ &= lfp(T_P) \\ &\quad \text{by the Knaster-Tarski} \\ &\quad \text{Fixpoint Theorem [18].}\square \end{aligned}$$

14.3.3 Operational semantics

The improvement of Fuzzy Prolog is remarkable in its new procedural semantics that is interpreted as a sequence of transitions between different states of a system. We represent the state of a *transition system* in a computation as a tuple $\langle A, \sigma, S\rangle$ where A is the goal, σ is a substitution representing the instantiation of variables needed to get to this state from the initial one and S is a constraint that represents the truth value of the goal at this state.

When computation starts, A is the initial goal, $\sigma = \emptyset$ and S is true (if there are neither previous instantiations nor initial constraints). When we get to a state where the first argument is empty then we have finished the computation and the other two arguments represent the answer.

Definition 14.10 (Transition) *A* transition *in the* transition system *is defined as:*

1. $\langle A \cup a, \sigma, S\rangle \rightarrow \langle A\theta, \sigma \cdot \theta, S \wedge \mu_a = v\rangle$
   if $h \leftarrow v$ is a fact of the program P, θ is the mgu of a and h, μ_a is the truth value for a and solvable$(S \wedge \mu_a = v)$.
2. $\langle A \cup a, \sigma, S\rangle \rightarrow \langle (A \cup B)\theta, \sigma \cdot \theta, S \wedge c\rangle$
   if $h \leftarrow_F B$ is a rule of the program P, θ is the mgu of a and h, c is the constraint that represents the truth value obtained applying the union-aggregation $\mathcal{F}$ to the truth values of B, and solvable$(S \wedge c)$.
3. $\langle A \cup a, \sigma, S\rangle \rightarrow \langle A, \sigma, S \wedge \mu_a = v\rangle$
   if none of the above are applicable and solvable$(S \wedge \mu_a = v)$ where $\mu_a =$ default(a).

Definition 14.11 (Success set) *The success set $SS(P)$ collects the answers to simple goals $p(\widehat{x})$. It is defined as follows:* $SS(P) = \langle B, V\rangle$
where $B = \{p(\widehat{x})\sigma | \langle p(\widehat{x}), \emptyset, true\rangle \rightarrow^* \langle \emptyset, \sigma, S\rangle\}$ is the set of elements of the Herbrand Base that are instantiated and that have succeeded; and $V(p(\widehat{x})) = \cup\{v | \langle p(\widehat{x}), \emptyset, true\rangle \rightarrow^* \langle \emptyset, \sigma, S\rangle$, and v is the solution of $S\}$ is the set of truth values of the elements of B that is the union (got by backtracking) of truth values that are obtained from the set of constraints provided by the program P while query $p(\widehat{x})$ is computed.

Example 14.3 *Let P be the program of example 14.1 (from [5]). Consider the fuzzy goal*

$$\mu \leftarrow good\_player(X) \quad ?$$

the first transition *in the computation is*

$$\langle \{(good\_player(X)\}, \epsilon, true\rangle \rightarrow \\ \langle \{tall(X), swift(X)\}, \epsilon, \mu = max(0, \mu_{tall} + \mu_{swift} - 1)\rangle$$

unifying the goal with the clause and adding the constraint corresponding to Lukasiewicz T-norm. The next transition *leads to the state:*

$$\langle \{swift(X)\}, \{X = john\}, \mu = max(0, \mu_{tall} + \mu_{swift} - 1) \wedge \mu_{tall} = 0.7\rangle$$

after unifying $tall(X)$ with $tall(john)$ and adding the constraint regarding the truth value of the fact. The computation ends with:

$\langle \{\}, \{X = john\}, \mu = max(0, \mu_{tall} + \mu_{swift} - 1) \wedge \mu_{tall} = 0.7 \wedge 0.6 \leq \mu_{swift} \wedge \mu_{swift} \leq 0.8\rangle$

As $\mu = max(0, \mu_{tall} + \mu_{swift} - 1) \wedge \mu_{tall} = 0.7 \wedge 0.6 \leq \mu_{swift} \wedge \mu_{swift} \leq 0.8$ *entails* $\mu \in [0.3, 0.5]$*, the answer to the query $good\_player(X)$ is $X = john$ with truth value the interval* $[0.3, 0.5]$.

In order to prove the equivalence between operational semantic and fixed-point semantic, it is useful to introduce a type of canonical top-down evaluation strategy. In this strategy all literals are reduced at each step in a derivation. For obvious reasons, such a derivation is called *breadth-first.*

Definition 14.12 (Breadth-first transition) *Given the following set of valid transitions:*

$$\begin{array}{l} \langle \{A_1, \ldots, A_n\}, \sigma, S\rangle \rightarrow \langle \{A_2, \ldots, A_n\} \cup B_1, \sigma \cdot \theta_1, S \wedge c_1\rangle \\ \langle \{A_1, \ldots, A_n\}, \sigma, S\rangle \rightarrow \langle \{A_1, A_3 \ldots, A_n\} \cup B_2, \sigma \cdot \theta_2, S \wedge c_2\rangle \\ \qquad\qquad\qquad\qquad \ldots \\ \langle \{A_1, \ldots, A_n\}, \sigma, S\rangle \rightarrow \langle \{A_1, \ldots, A_{n-1}\} \cup B_n, \sigma \cdot \theta_n, S \wedge c_n\rangle \end{array}$$

a breadth-first transition *is defined as*
$\langle \{A_1, \ldots, A_n\}, \sigma, S\rangle \rightarrow_{BF} \langle B_1 \cup \ldots \cup B_n, \sigma \cdot \theta_1 \cdot \ldots \cdot \theta_n, S \wedge c_1 \wedge \ldots \wedge c_n\rangle$
in which all literals are reduced at one step.

Theorem 14.3 *Given an ordinal number n and $T_P \uparrow n = \langle B_{T_{P_n}}, V_{T_{P_n}}\rangle$. There is a successful breadth-first derivation of length less or equal to $n+1$ for a program P, $\langle \{A_1, \ldots, A_k\}, \sigma, S_1\rangle \rightarrow^*_{BF} \langle \emptyset, \theta, S_2\rangle$ iff $A_i\theta \in B_{T_{P_n}}$ and $solvable(S \wedge \mu_{A_i} = v_i)$ and $v_i \subseteq_{BI} V_{T_{P_n}}(A_i\theta)$.*

Proof. The proof is by induction on n. For the base case, all the literals are reduced using the first type of transitions or the last one, that is, for each literal A_i, it exits a fact $h_i \leftarrow v_i$ such that θ_i is the mgu of A_i and h_i, and μ_{A_i} is the truth variable for A_i, and $solvable(S_1 \wedge \mu_{A_i} = v_i)$ or $\mu_{A_i} = default(A_i)$. By definition of T_P, each $v_i \subseteq_{BI} V_{T_{P_1}}(A_i\theta)$ where $\langle B_{T_{P_1}}, V_{T_{P_1}}\rangle = T_P \uparrow 1$.

For the general case, consider the successful derivation,
$\langle\{A_1,\ldots,A_k\},\sigma_1,S_1\rangle \rightarrow_{BF} \langle B,\sigma_2,S_2\rangle \rightarrow_{BF} \ldots \rightarrow_{BF} \langle\emptyset,\sigma_n,S_n\rangle$
the transition $\langle\{A_1,\ldots,A_k\},\sigma_1,S_1\rangle \rightarrow_{BF} \langle B,\sigma_2,S_2\rangle$

When a literal A_i is reduced using a fact or there is not rule for A_i, the result is the same as in the base case. Otherwise there is a clause $h_i \leftarrow_F B_{1_i},\ldots,B_{m_i}$ in P such that θ_i is the mgu of A_i and $h_i \in B\sigma_2$ and $B_{j_i}\theta_i \in B\sigma_2$, by the induction hypothesis $B\sigma_2 \subseteq B_{T_{P_{n-1}}}$ and $solvable(S_2 \wedge \mu_{B_{j_i}} = v_{j_i})$ and $v_{j_i} \subseteq_{BI} V_{T_{P_{n-1}}}(B_{j_i}\sigma_2)$ then $B_{j_i}\theta_i \subseteq B_{T_{P_{n-1}}}$ and by definition of T_P, $A_i\theta_i \in B_{T_{P_n}}$ and $solvable(S_1 \wedge \mu_{A_i} = v_i)$ and $v_i = \subseteq_{BI} V_{T_{P_n}}(A_i\sigma_1)$. □

Theorem 14.4 *For a program P there is a successful derivation*

$$\langle p(\widehat{x}),\emptyset,true\rangle \rightarrow^* \langle\emptyset,\sigma,S\rangle$$

iff $p(\widehat{x})\sigma \in B$ and v is the solution of S and $v \subseteq_{BI} V(p(\widehat{x})\sigma)$ where $lfp(T_P) = \langle B,V\rangle$.

Proof. It follows from the fact that $lfp(T_P) = T_P \uparrow \omega$ and from the Theorem 14.3. □

Theorem 14.5 *For a fuzzy program P the three semantics are equivalent, i.e.*

$$SS(P) = lfp(TP) = lm(P).$$

Proof. The first equivalence follows from Theorem 14.4 and the second from Theorem 14.2. □

14.4 Implementation and Syntax

14.4.1 CLP($\mathcal{R}$)

Constraint Logic Programming [7] began as a natural merging of two declarative paradigms: constraint solving and logic programming. This combination helps make CLP programs both expressive and flexible, and in some cases, more efficient than other kinds of logic programs. CLP($\mathcal{R}$) [8] has linear arithmetic constraints and computes over the real numbers.

Fuzzy Prolog was implemented in [5] as a syntactic extension of a CLP($\mathcal{R}$) system. CLP($\mathcal{R}$) was incorporated as a library in the Ciao Prolog system[4].

Ciao Prolog is a next-generation logic programming system which, among other features, has been designed with modular incremental compilation in

[4] The Ciao system [1] including our Fuzzy Prolog implementation can be downloaded from `http://www.clip.dia.fi.upm.es/Software/Ciao`

mind. Its module system [1] permits having classical modules and fuzzy modules in the same program and it incorporates CLP($\mathcal{R}$).

Many Prolog systems have included the possibility of changing or expanding the syntax of the source code. One way is using the `op/3` built-in and another is defining *expansions* of the source code by allowing the user to define a predicate typically called `term_expansion/2`. Ciao has redesigned these features so that it is possible to define source translations and operators that are local to the module or user file defining them. Another advantage of the module system of Ciao is that it allows separating code that will be used at compilation time from code which will be used at run-time.

The *fuzzy* library (or *package* in the Ciao Prolog terminology) which implements the interpreter of our Fuzzy Prolog language [5] has been modified to handle default reasoning.

14.4.2 Syntax

Let us recall, from [5], the syntax of Fuzzy Prolog. Each Fuzzy Prolog *clause* has an additional argument in the head which represents its truth value in terms of the truth values of the subgoals of the body of the clause. A *fact* $A \leftarrow v$ is represented by a Fuzzy Prolog fact that describes the range of values of v with a union of intervals (which can be only an interval or even a real number in particular cases). The following examples illustrate the concrete syntax of programs:

$$\begin{array}{l} youth(45) \leftarrow\ [0.2, 0.5] \bigcup [0.8, 1] \\ tall(john) \leftarrow 0.7 \\ swift(john) \leftarrow\ [0.6, 0.8] \\ good\_player(X) \leftarrow_{min}\ tall(X), \\ \qquad\qquad\qquad\qquad\quad swift(X) \end{array}$$

```
youth(45):~ [0.2,0.5]v[0.8,1]
tall(john):~ 0.7
swift(john):~ [0.6,0.8]
good_player(X):~min tall(X),
                    swift(X)
```

These clauses are expanded at compilation time to constrained clauses that are managed by CLP($\mathcal{R}$) at run-time. Predicates $. = ./2$, $. < ./2$, $. <= ./2$, $. > ./2$ and $. >= ./2$ are the Ciao CLP($\mathcal{R}$) operators for representing constraint inequalities, we will use them in the code of predicates definitions (while we will use the common operators $=$, $<$, $\leq$, $>$, $\geq$ for theoretical definitions). For example the first fuzzy fact is expanded to these Prolog clauses with constraints

```
youth(45,V):-   V .>=. 0.2, V .<=. 0.5.
youth(45,V):-   V .>=. 0.8, V .<.  1.
```

And the fuzzy clause

```
good_player(X) :~ min tall(X),swift(X).
```

is expanded to

```
good_player(X,Vp) :- tall(X,Vq),
                     swift(X,Vr),
                     minim([Vq,Vr],Vp),
                     Vp .>=. 0,
                     Vp .=<. 1.
```

The predicate `minim/2` is included as run-time code by the library. Its function is adding constraints to the truth value variables in order to implement the T-norm *min*.

```
minim([],_).
minim([X],X).
minim([X,Y|Rest],Min):-
            min(X,Y,M),
            minim([M|Rest],Min).

min(X,Y,Z):- X .=<. Y , Z .=. X.
min(X,Y,Z):- X .>. Y, Z .=. Y .
```

We have implemented several *aggregation operators* as `prod`, `max`, `luka` (Lukasiewicz operator), etc. and in a similar way any other operator can be added to the system without any effort. The system is extensible by the user simply adding the code for new *aggregation operators* to the library.

14.5 Combining Crisp and Fuzzy Logic

14.5.1 Example: Teenager student

In order to use definitions of fuzzy predicates that include crisp subgoals we must define properly their semantics with respect to the Prolog Close World Assumption (CWA) [4]. We will present a motivating example from [5].

Fuzzy clauses usually use crisp predicate calls as requirements that data have to satisfy to verify the definition in a level superior to 0, i.e. crisp predicates are usually tests that data should satisfy in the body of fuzzy clauses. For example, if we can say that a teenager student is a student whose age is about 15 then we can define the fuzzy predicate *teenager_student*/2 in Fuzzy Prolog as

```
teenager_student(X,V):~
                 student(X),
                 age_about_15(X,V2).
```

In this example we pretend the goal *teenager_student*(X, V) provides:

- $V = 0$ if the value of X is not the name of a student.
- The corresponding truth value V if the value of X is the name of a student and we know that his age is about 15 in a certain level.
- Unknown if the value of X is the name of a student but we do not know anything about his/her age.

Note that we can face the risk of unsoundness unless the semantics of crisp and fuzzy predicates is properly defined. CWA means that all non-explicit information is false. E.g., if we have the predicate definition of *student*/1 as

```
student(john).
student(peter).
```

then we have that the goal *student*(X) succeeds with $X = john$ or with $X = peter$ but fails with any other value different from these; i.e:

```
?- student(john).
yes

?- student(nick).
no
```

which means that *john* is a student and *nick* is not. This is the semantics of Prolog and it is the one we are going to adopt for crisp predicates because we want our system to be compatible with conventional Prolog reasoning. But what about fuzzy predicates? According to human reasoning we should assume OWA (non explicit information in unknown). Consider the following definition of *age_about*_15/2

```
age_about_15(john,1):~ .
age_about_15(susan,0.7):~ .
```

The goal *age_about*_15(X, V) succeeds with $X = john$ and $V = 1$ or with $X = susan$ and $V = 0.7$. If we want to work with the CWA, like crisp predicates do, then we will obtain $V = 0$ for any other value of X different from *john* and *susan*. The meaning is that the predicate is defined for all values and the membership value will be 0 if the predicate is not explicitly defined with other value. In this example we know that the age of *john* is 15 and *susan*'s age is about 15 and with CWA we are also saying that the rest of the people are not about 15. This is the equivalent semantics to the one in crisp definitions but we think that we usually prefer to mean something different, i.e. in this case we can mean that we know that *john* and *susan* are about 15 and that we have no information about the age of the rest of people.

Therefore we do not know if the age of *peter* is about 15 or not; and we know that *nick*'s age is definitely not about 15. We can explicitly declare

```
age_about_15(nick,0):~ .
```

We are going to work with this semantics for fuzzy predicates because we think it is the most alike to human reasoning. So a fuzzy goal can be true (value 1), false (value 0) or having other membership value. We have added the concept of `unknown` to represent no explicit knowledge in fuzzy definitions. We understand that if we don't have any information about a truth value V then its value is something (a value, an interval or a union of intervals) in the interval $\{0, 1\}$, so the most general assumption is the whole interval $[0, 1]$. The interval is represented by its corresponding constraints $V \geq 0$ and $V \leq 1$.

Our way to introduce crisp subgoals into the body of fuzzy clauses is translating the crisp predicate into the respective fuzzy predicate. In the example the way to obtain it is by overcoming the CWA behavior of the crisp predicate *student*/1 to obtain the truth value 0 for *student*(*susan*). The solution is to fuzzify crisp predicates when they are in the body of fuzzy clauses.

For each crisp predicate in the definition of fuzzy predicate, the compiler will generate a fuzzy version to replace the original one in the body of the clause. For the example above of crisp predicate *student*/1, the compiler will produce the predicate *f_student*/2 that is an equivalent fuzzy predicate to the crisp one. For our example we obtain the following Prolog definition of *teenager_student*/2.

```
teenager_student(X,V):~
               f_student(X,V1),
               age_about_15(X,V2).
```

Where the default truth value of a crisp predicate is 0.

```
f_student(X,1):- student(X).
:-default(f_student/2,0).
```

Nevertheless, we consider for *age_about_15*/2 and *teenager_student*/2 that the default value is unknown (the whole interval $[0, 1]$).

```
:-default(age_about_15/2,[0,1]).
:-default(teenager_student/2,[0,1]).
```

Observe the following consults:

```
?- age_about_15(john,X).
X = 1

?- age_about_15(nick,X).
X = 0

?- age_about_15(peter,X).
X .>=. 0, X .<=. 1
```

This means *john*'s age is about 15, *nick*'s age is not about 15 and we have no data about *peter*'s age.

We expect the same behavior with the fuzzy predicate *teenager_student*/2, i.e.:

```
?- teenager_student(john,V).
V .=. 1

?- teenager_student(susan,V).
V .=. 0

?- teenager_student(peter,V).
V .>=. 0, V .<=. 1
```

as *john* is a "teenager student" (he is a student and his age is about 15), *susan* is not a "teenager student" (she is not a student) and we do not know the value of maturity of *peter* as student because although he is a student, we do not know if his age is about 15.

Now the internal fuzzy resolution is simple, sound and very homogeneous because we only consider fuzzy subgoals in the body of the clause.

14.5.2 Example: Timetable compatibility

Another real example could be the problem of compatibility of a couple of shifts in a work place. For example teachers that work in different class timetables, telephone operators, etc. Imagine a company where the work is divided in shifts of 4 hours per week. Many workers have to combine a couple of shifts in the same week and a predicate *compatible*/2 is necessary to check if two shifts are compatible or to obtain which couples of shifts are compatible. Two shifts are compatible when both are correct (working days from Monday to Friday, hours between 8 a.m. and 18 p.m. and there are no repetitions of the same hour in a shift) and in addition when the shifts are disjoint.

```
compatible(T1,T2):-
                correct_shift(T1),
                correct_shift(T2),
                disjoint(T1,T2).
```

But there are so many compatible combinations of shifts that it would be useful to define the concept of compatibility in a fuzzy way instead of in the crisp way it is defined above. It would express that two shifts could be incompatible if one of them is not correct or if they are not disjoint but when they are compatible, they can be more or less compatible. They can have a level of compatibility. Two shifts will be more compatible if the working hours are concentrated (the employee has to go to work few days during the week). Also, two shifts will be more compatible if there are few free hours between the busy hours of the working days of the timetable.

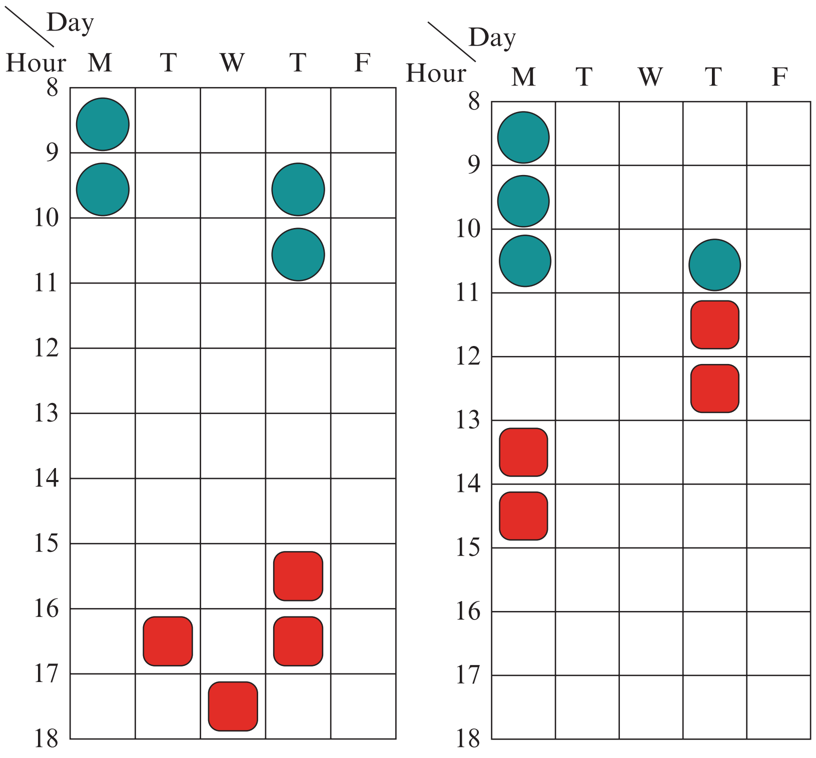

Fig. 14.1. Timetable 1 and 2

Therefore, we are handling crisp concepts (*correct_shift*/1, *disjoint*/2) besides fuzzy concepts (*without_gaps*/2, *few_days*/2). Their definitions, represented in Fig. 14.3 and Fig. 14.4, are expressed in our language in this simple way (using the operator ": #" for function definitions and the reserved word "*fuzzy_predicate*"):

```
few_days :# fuzzy_predicate([(0,1),
                   (1,0.8),(2,0.6),
                   (3,0.4),(4,0.2),
                   (5,0)]).

without_gaps :# fuzzy_predicate([(0,1),
                    (1,0.8),(5,0.3),
                    (7,0.1),(8,0)]).
```

A simple implementation in Fuzzy Prolog combining both types of predicates could be:

```
compatible(T1,T2,V):~ min
```

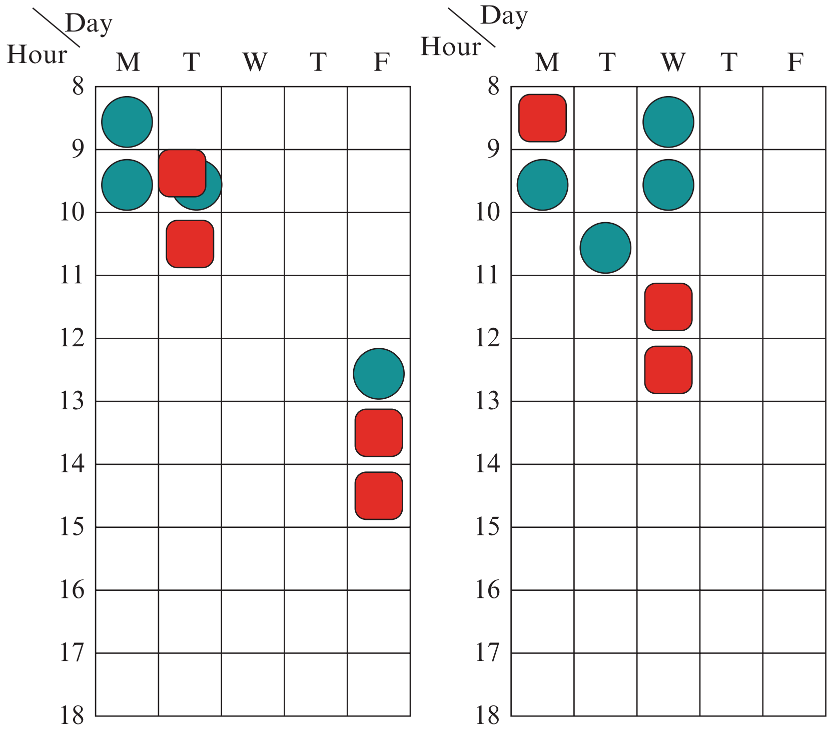

Fig. 14.2. Timetable 3 and 4

```
f_correct_shift(T1,V1),
f_correct_shift(T2,V2),
f_disjoint(T1,T2,V3),
f_append(T1,T2,T,V4),
f_number_of_days(T,D,V5),
few_days(D,V6),
f_number_of_free_hours(T,H,V7),
without_gaps(H,V8).
```

Here *append*/3 gives the total weekly timetable of 8 hours from joining two shifts, *number_of_days*/3 obtains the total number of working days of a weekly timetable and *number_of_free_hours*/2 returns the number of free one-hour gaps that the weekly timetable has during the working days. The `f_`*predicates* are the corresponding fuzzified crisp predicates. The aggregation operator *min* will aggregate the value of V from $V6$ and $V8$ checking that $V1$, $V2$, $V3$, $V4$, $V5$ and $V7$ are equal to 1, otherwise it fails. Observe the timetables in Fig. 14.1 and Fig. 14.2. We can obtain the compatibility between the couple of shifts, T1 and T2, represented in each timetable asking the

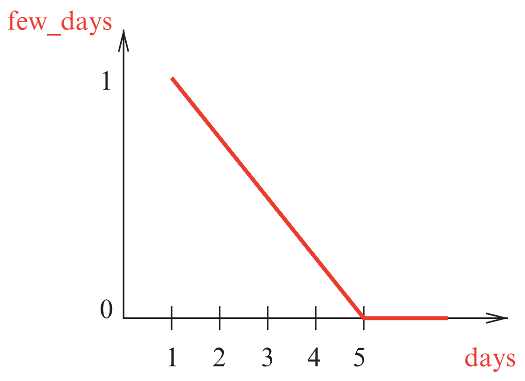

Fig. 14.3. Fuzzy predicate few_days/2

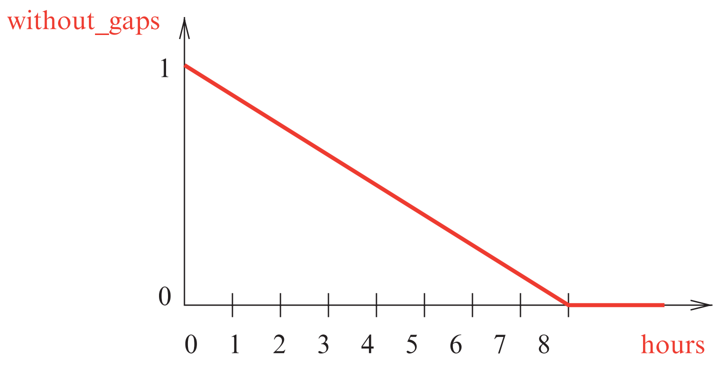

Fig. 14.4. Fuzzy predicate without_gaps/2

subgoal *compatible* $(T1, T2, V)$. The result is $V = 0.2$ for the timetable 1, $V = 0.6$ for the timetable 2, and $V = 0$ for the timetable 3 (because the shifts are incompatible).

Regarding compatibility of shifts in a weekly timetable, we are going to ask some questions about the shifts T1 and T2 of timetable 4 of Fig. 14.2. One hour of T2 is not fixed yet.

We can note: the days of the week as mo, tu, we, th and fr; the slice of time of one hour as the time of its beginning from 8 a.m. till 17 p.m.; one hour of the week timetable as a pair of day and hour and one shift as a list of 4 hours of the week.

If we want to fix the free hour of T2 in the slice 10-11 a.m. but with a compatibility not null, we obtain that only Tuesday is not compatible.:

```
?- compatible(
       [(mo,9), (tu,10), (we,8), (we,9)],
      [(mo,8), (we,11), (we,12), (D,10)], V),
   V .>. 0 .

D =/= tu
```

If we want to know how to complete the shift T2 given a level of compatibility higher than 70 %, we obtain the slice from 10 to 11 p.m. at Wednesday or Monday morning.

```
?- compatible(
     [(mo,9), (tu,10), (we,8), (we,9)],
     [(mo,8), (we,11), (we,12), (D,H)],
     V),
   V .>. 0.7 .

V = 0.9,  D = we, H = 10 ? ;
V = 0.75, D = mo, H = 10 ? ;
no
```

14.6 Conclusions and Future Work

Extending the expressivity of programming systems is very important for knowledge representation. We have chosen a practical and extended language for knowledge representation: Prolog.

Fuzzy Prolog presented in [5] is implemented over Prolog instead of implementing a new resolution system. This gives it a good potential for efficiency, more simplicity and flexibility. For example *aggregation operators* can be added with almost no effort. This extension to Prolog is realized by interpreting fuzzy reasoning as a set of constraints [25], and after that, translating fuzzy predicates into CLP($\mathcal{R}$) clauses. The rest of the computation is resolved by the compiler.

In this paper we propose to enrich Prolog with more expressivity by adding default reasoning and therefore the possibility of handling incomplete information that is one of the most worrying characteristics of data (i.e. all information that we need usually is not available but only one part of the information is available) and anyway searches, calculations, etc. should be done just with the information that we had.

We have developed a complete and sound semantics for handling incomplete fuzzy information and we have also provided a real implementation based in our former Fuzzy Prolog approach.

We have managed to combine crisp information (CWA) and fuzzy information (OWA or default) in the same program. This is a great advantage because it lets us model many problems using fuzzy programs. So we have extended the expressivity of the language and the possibility of applying it to solve real problems in which the information can be defined, fuzzy or incomplete.

Presently we are working in several related issues:

- Obtaining constructive answers to negative goals.
- Constructing the syntax to work with discrete fuzzy sets and its applications (recently published in [14]).
- Implementing a representation model using unions instead of using backtracking.
- Introducing domains of fuzzy sets using types. This seems to be an easy task considering that we are using a modern Prolog [6] where types are available.
- Implementing the expansion over other systems. We are studying now the advantages of an implementation in XSB system where tabling is used.
- Using our approach for the engine of robots in a RoboCup league in a joint project between our universities.

References

1. Cabeza D, Hermenegildo M (2000) A new module system for Prolog. LNAI 1861:131–148. Springer-Verlag
2. Gelfond M, Lifschitz V (1988) The stable model semantics for logic programming. In: Proc Fifth Intl Conf Symposium on Logic Programming pp 1070–1080
3. Gelfond M, Lifschitz V (1990) Logic programs with classical negation. In: Proc seventh Intl Conf on Logic Programming pp 579–597. MIT Press — Complete version in: New Generation Computing (1991) 9:365–387
4. Clark KL (1978) Negation as failure. In: Gallaire H, Minker J (eds) Logic and Data Bases pp 293–322. Plenum Press, New York, NY
5. 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
6. Hermenegildo M, Bueno F, Cabeza D, Carro M, García de la Banda M, López-García 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 pp 65–85. Nova Science Commack, NY
7. Jaffar J, Lassez JL (1987) Constraint logic programming. In: ACM Symp Principles of Programming Languages pp 111–119
8. Jaffar J, Michaylov S, Stuckey PJ, Yap RHC (1992) The clp(r) language and system. ACM Trans Programming Languages and Systems 14(3):339–395
9. Klement E, Mesiar R, Pap E (2000) Triangular Norms. Kluwer Academic Publishers
10. Lakshmanan L, Shiri N (2001) A parametric approach to deductive databases with uncertainty. IEEE Trans Knowledge and Data Engineering 13(4):554–570
11. Liu J, Ruan D, Xu Y, Song Z (2003) A resolution-like strategy based on a lattice-valued logic. IEEE Trans Fuzzy Systems 11(4):560–567
12. Loyer Y, Straccia U (2002) Uncertainty and partial non-uniform assumptions in parametric deductive databases. In: Proc JELIA, LNCS 2424:271–282

13. Lukasiewicz T (2001) Fixpoint characterizations for many-valued disjunctive logic programs with probabilistic semantics. In: Proc LPNMR 2173:336–350
14. Munoz-Hernandez S, Pérez JG (2005) Solving collaborative fuzzy agents problems with clp(fd). In: Hermenegildo M, Cabeza D (eds) Proc Intl Symp Practical Aspects of Declarative Languages (PADL) LNCS 3350:187–202
15. Ng R, Subrahmanian V (1991) Stable model semantics for probabilistic deductive databases. In: Proc ISMIS, LNCS 542:163–171
16. Nguyen HT, Walker EA (2000) A First Course in Fuzzy Logic. Chapman & Hall/CRC
17. 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
18. Tarski A (1955) A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5:285–309
19. Trillas E, Cubillo S, Castro JL (1995) Conjunction and disjunction on ([0,1],<=). Fuzzy Sets and Systems 72:155–165
20. Trillas E, Pradera A, Cubillo S (1999) A mathematical model for fuzzy connectives and its application to operators behavioural study. Information, uncertainty and fusion. In: Bouchon-Meunier B, Yager R, Zadeh L (eds) ser The Kluwer International Series in Engineering and Computer Sciences 516:307–318
21. Vaucheret C, Guadarrama S, Munoz-Hernandez S (2001) Fuzzy Prolog: a simple implementation using clp(r). Constraints and uncertainty. `http://www.clip.dia.fi.upm.es/clip/papers/fuzzy-lpar02.ps`
22. Vaucheret C, Guadarrama S, Munoz-Hernandez S (2002) Fuzzy Prolog: a simple general implementation using clp(r). In: Baaz M, Voronkov A (eds) Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) LNAI 2514:450–463
23. Wagner G (1997) A logical reconstruction of fuzzy inference in databases and logic programs. In: Proc IFSA
24. Wagner G (1998) Negation in fuzzy and possibilistic logic programs. Logic programming and soft computing. Research Studies Press
25. Zadeh L (1978) Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems 1(1):3–28