A tabulation procedure for first-order residuated logic programs*

C.V. Damásio
cd@di.fct.unl.pt
Centro Inteligência Artificial
Univ. Nova de Lisboa.
Portugal

J. Medina
jmedina@ctima.uma.es
Dept. Matemática Aplicada
Univ. de Málaga
Spain

M. Ojeda-Aciego
aciego@ctima.uma.es
Dept. Matemática Aplicada
Univ. de Málaga
Spain

Abstract

Residuated logic programs have shown to be a generalisation of a number of approaches to logic programming under uncertainty. Regarding automated deduction, a tabulation procedure was recently introduced for the propositional version. In this paper, we introduce a sound and complete tabulation-based proof procedure for the first-order extension of residuated logic programs.

1 Introduction

Several different approaches to the so-called inexact or fuzzy or approximate reasoning have been proposed in the recent years, these approaches involve either fuzzy or annotated or probabilistic or similarity-based logic programming [1, 2, 9–11, 15–17, 19, 20].

Monotonic and residuated logic programs have shown to be a generalisation of a number of the approaches above. In this paper, we will focus on the framework of residuated logic programming. The semantics of a residuated program is characterised, as usual, by the post-fixpoints of the immediate consequence operator $T_{\mathbb{P}}$, which is proved to be *monotonic* and continuous under very general hypotheses.

In this paper we aim at the use of tabulation (tabling, or memoizing), a technique which is receiving increasing attention in the logic programming and deductive database communities [3, 4, 17, 18]. Tabulation has better termination properties and can be substantially more efficient than the SLD resolution usually found in Prolog systems. In fact, for some dramatic cases query evaluation with tabulation mechanisms takes polynomial time while Prolog systems take exponential time. Existing Prolog systems like XSB and YAP complement ordinary Prolog query evaluation with tabulation mechanisms increasing declarativity of developed programs/knowledge bases. For this paper we are more interested in the better termination properties of tabulation since the use of many-valued character of our truth-value space introduces substantial complexity in soundness and completeness results. Efficiency is also obtained, but requires further enquiry and optimization of the basic procedure presented in this paper.

In this work, we provide a tabulation goal-oriented query procedure for first-order monotone and residuated logic programs, which generalises the given one for the propositional case in [7]. The underlying idea is, essentially, that atoms of selected tabled predicates as well as their answers are stored in a table. When an identical atom is recursively called, the selected atom is not resolved against program clauses; instead, all corresponding answers computed so far are looked up in the table and the associated answer substitutions are applied to the atom. The process is repeated for all subsequent computed answer

* This research has been partially funded by European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net) and by the Spanish Project TIC2003-9001-C02-01.

substitutions corresponding to the atom.

The structure of the paper is as follows: in Section 2, the syntax and semantics of our logic programs are summarized; Section 3 introduces a non-deterministic procedure for tabulation. The soundness and completeness of the tabulation procedure are stated in Section 4, length restrictions do not allow to introduce the proof of the claims. The paper finishes with some conclusions and pointers to future work.

2 Syntax and semantics

In this section the essentials of first order monotonic logic programming are reviewed. The reader might consult [5] for the propositional version or [13] for a first-order (multi-adjoint) language.

The mathematical structure underlying residuated logic programs is that of residuated lattice, which provides an abstraction of the usual conjunction and implication and the modus ponens inference rule. The formal definition is given below:

Definition 1 *A* residuated lattice $\mathcal{L}$ *is a tuple* $(L, \leftarrow, \&)$ *satisfying the following items:*

1. $\langle L, \preceq \rangle$ *is a bounded lattice, i.e. it has bottom and top elements, denoted 0 and 1;*
2. $(L, \&, 1)$ *is a commutative monoid;*
3. $(\&, \leftarrow)$ *is an* adjoint pair *in* $\langle L, \preceq \rangle$*; i.e.*
   (a) *Operation* $\&$ *is increasing in both arguments,*
   (b) *Operation* $\leftarrow$ *is increasing in the first argument and decreasing in the second,*
   (c) *For any* $x, y, z \in P$*, we have*

$$x \preceq (y \leftarrow z) \text{ if and only if } (x \,\&\, z) \preceq y$$

Definition 2 *A* program *over a residuated complete lattice* $\langle L, \leftarrow, \otimes \rangle$ *is a finite set of rules* $A \leftarrow \mathcal{B}$ *satisfying:*

1. *The* head *of the rule* A *is an atom.*
2. *The* body *formula* $\mathcal{B}$ *is a formula built from atoms or elements of the lattice* $B_1, \ldots, B_n$ *(with* $n \geq 0$*) by the use of arbitrary monotonic operators, also denoted by* $\mathcal{B}[B_1, \ldots, B_n]$*.*

A query *is an atom intended as a question* $?A$ *prompting the system.*

In the following, we will use the term *residuated program* as a short name for definite program over a complete residuated lattice.

An *interpretation* is a mapping I from the Herbrand base of the program to L. Moreover, since elements of the lattice can be used in the body of the rules, it should be noticed that they are interpretated by themselves, i.e. $I(a) = a$ for all $a \in L$.

Note that each of these interpretations can be uniquely extended to the set of body formulas, in this case it is denoted $\hat{I}$. The ordering $\preceq$ on the underlying lattice can also be easily extended pointwise to the set of interpretations, inheriting a structure of complete lattice.

Definition 3

1. *An interpretation* I satisfies $A \leftarrow \mathcal{B}$ *if and only if* $\hat{I}(\mathcal{B}\eta) \preceq I(A\eta)$ *for all grounding substitution* η*.*
2. *An interpretation* I *is a* model *of* $\mathbb{P}$ *iff all its rules are satisfied by* I*.*

The immediate consequences operator, given by van Emden and Kowalski, can be easily generalised to the framework of residuated logic programs.

Definition 4 *Let* $\mathbb{P}$ *be a residuated program over a complete lattice* L*. The* immediate consequences operator $T_{\mathbb{P}}$ *maps interpretations to interpretations and, for an interpretation* I *and a ground atom* A*,* $T_{\mathbb{P}}(I)(A)$ *is defined as*

$$\sup\{\hat{I}(\mathcal{B}\eta) \mid C\eta \leftarrow \mathcal{B}\eta \in \mathcal{G}(\mathbb{P}) \text{ and } A = C\eta\}$$

where $\mathcal{G}(\mathbb{P})$ *denotes the grounding of* $\mathbb{P}$*.*

The semantics of a residuated logic program can be characterised, as usual, by the post-fixpoints of $T_{\mathbb{P}}$; that is, an interpretation I is a model of a residuated logic program $\mathbb{P}$ iff $T_{\mathbb{P}}(I)(A) \preceq I(A)$ for all ground atom A.

The $T_{\mathbb{P}}$ operator is proved to be monotonic and continuous under very general hypotheses, see [12], and it is remarkable that these results are true even for non-commutative and non-associative conjunctors. In particular, by continuity, the least model can be reached in at most countably many iterations of $T_{\mathbb{P}}$ on the least interpretation, denoted $T_{\mathbb{P}} \uparrow^{\omega}$. In what follows, we will assume this behaviour for all our programs, together with finite dependency.

3 The tabulation Procedure

In this section we describe a simple version of the first-order tabulation proof procedure for residuated logic programs which allows to obtain more easily the proofs of soundness and completeness.

R1: Create New Tree.

Given an atom A, let $\mathbb{P}(A)$ be the finite set of rules $\langle C_j \leftarrow \mathcal{B}_j \rangle$ of $\mathbb{P}$, with variables renamed apart, such that there exists a substitution θ_j satisfying $C_j\theta_j = A\theta_j$, where $j = 1, \ldots, m$.

Construct the following tree with root A

$$\begin{array}{c} A\colon \{A\colon 0\} \\ \diagup \quad \ldots \quad \diagdown \\ A\theta_1 \leftarrow \mathcal{B}_1\theta_1 \qquad A\theta_m \leftarrow \mathcal{B}_m\theta_m \end{array}$$

and append it to the current forest. If the forest does not exist, then create a new forest containing this single tree.

R2: New Subgoal.

Select a non-tabulated atom C occurring in a leaf of some tree (note that non-tabulated means that there is no tree in the forest with root containing a variant of C), then create a new tree as indicated in Rule 1, and append it to the forest.

R3: Answer Return.

Select in any non-root node an atom C which is tabulated (i.e. there is a tree with root C' which is a variant of C). Let $C'\theta'\colon r$ be an element of the answer list of C', which unifies with C, and was not consumed before. Let Θ be the most general unifier of C and $C'\theta'$. Then, add a new successor node

$$\begin{array}{c} A\theta \leftarrow \mathcal{B}[\ldots, C, \ldots] \\ \vert \\ A\theta\Theta \leftarrow \mathcal{B}[\ldots, r, \ldots]\Theta \end{array}$$

R4: Value Update.

Given a leaf in the tree for an atom C, having the form $C\theta \leftarrow \mathcal{B}[s_1, \ldots, s_m]\theta$, where $\mathcal{B}$ does not contain atoms. Then, evaluate the corresponding arithmetic formula in the body of the rule $\mathcal{B}[s_1, \ldots, s_m]$, assume that its value is, say, s. If there is a variant of $C\theta$ with same value s in the answer list of C then we do nothing, otherwise we add the new answer $C\theta\colon s$.

R5: Answer merging.

Let $A_1\colon s_1$ and $A_2\colon s_2$ be two instances in an answer list, which *unify* with mgu θ. Then, add the answer $A_1\theta\colon \sup\{s_1, s_2\}$ whenever $A_1\theta\colon \sup\{s_1, s_2\}$ is not in the answer list (modulo renaming of variables).

Moreover, if the answer list contains two *variants* $A_1\colon s_1$ and $A_2\colon s_2$ for which $s_1 \preceq s_2$ then $A_1 : s_1$ is removed from the list.

Remark: Note that a list of "computed" answers is attached to the root of each tree in the forest, in terms of a substitution and a value in L. The answer list of a root C is denoted by $\mathfrak{AL}(C)$.

Recall that the only rules which change the values in the answer list of the roots of the trees in the forest are R4 and R5.

A non-deterministic procedure for tabulation

Now, we can state the general non-deterministic procedure for calculating the

answer to a given query by using a tabulation technique in terms of the previous rules.

Initial step Create the initial forest by applying R1 to the query.

Next steps Non-deterministically select an atom and apply one of the rules R2, R3, R4 or R5.

There are several improvements that can be made to the basic tabulation proof procedure, for instance, by considering subsumption-based tabulation instead of variants, but we are not concerned with efficiency in this paper, but in showing soundness and completeness.

4 Soundness and Completeness

We start by taking care of the soundness proof of the tabulation proof procedure. In intuitive terms, it is shown that every answer in the answer list in a tree for some atom is a correct answer.

Definition 5

1. *Given a program $\mathbb{P}$ and a query A, a* computed answer for A in $\mathbb{P}$ *is a pair (θ, ϑ) where θ is a substitution and ϑ a value in L such that $A\theta\colon \vartheta$ belongs to the answer list of the tree for A.*
2. *Given a program $\mathbb{P}$ and a query A, a* correct answer for A in $\mathbb{P}$ *is a pair (θ, ϑ) where θ is a substitution and ϑ a value in L such that $\vartheta \leq M(A\theta\eta)$ for all Herbrand models M of $\mathbb{P}$ and grounding substitution η.*

We can give an equivalent definition of correct answer in terms of the $T_{\mathbb{P}}$ operator as follows: (θ, ϑ) is a correct answer for A in $\mathbb{P}$ if for every grounding substitution η we have that $\vartheta \leq T_{\mathbb{P}}{\uparrow}^{\omega}(A\theta\eta)$.

Theorem 1 (Soundness) *Let $\mathbb{P}$ be a program and a tabulation forest for a given query. Then, every computed answer for a tabulated atom A in the forest is a correct answer for A in $\mathbb{P}$.*

In order to prove completeness, we need a suitable extension of the well-known lifting lemma. In its statement, we need to introduce the notion of *super-forest* $\mathfrak{F}'$ of a given forest $\mathfrak{F}$: every tree in the forest $\mathfrak{F}$ is subsumed by another tree in $\mathfrak{F}'$ which, moreover, whose nodes are labelled by formulas more general than those in $\mathfrak{F}$.

In addition, we say that computed values are *preserved by the super-forest* if for every element $Q\theta\eta\colon s$ in the answer list of $Q\theta$ in $\mathfrak{F}$ there exists a substitution η' such that $Q\eta'\colon s$ is the answer list for Q in $\mathfrak{F}'$.

Lemma 1 (Lifting lemma) *Let $\mathbb{P}$ be a program, Q an atom and θ a substitution. Given a finite tabulation forest $\mathfrak{F}$ for $Q\theta$, there exists a tabulation super-forest for Q which preserves computed answers in $\mathfrak{F}$.*

Definition 6 *Given a program $\mathbb{P}$, a terminated forest for $\mathbb{P}$ and a query, and a grounding substitution η, a* computed answer for A relative to η *in a tree for A, denoted $r_\eta(A)$, is the supremum of*

$$\{r_i \mid A\theta_i\colon r_i \in \mathfrak{AL}(A) \text{ where } A\theta_i \text{ and } A\eta \text{ unify}\}$$

Note that, by repeated application of Rule 5 and assuming finite termination of the forest construction, there must exist some answer $A\theta\colon r$ in the tree for A which unifies with the ground atom $A\eta$.

Theorem 2 *Consider a program $\mathbb{P}$ and a finite terminated forest for a ground atom A. Then $T_{\mathbb{P}} \uparrow {}^{n}(A) \leq r_\varepsilon(A)$ for all $n \in \mathbb{N}$, where ε denotes the identity substitution.*

Theorem 3 *Let $\mathbb{P}$ be a program $\mathbb{P}$, let η be a grounding substitution and consider a finite terminated forest for an atom A. Then*

$$T_{\mathbb{P}}{\uparrow}^{n}(A\eta) \leq r_\eta(A) \text{ for all } n \in \mathbb{N}$$

As a consequence of the definition of ω-th iteration step, we obtain straightforwardly that

$$T_{\mathbb{P}}{\uparrow}^{\omega}(A\eta) \leq r_\eta(A)$$

Corollary 1 (Completeness) *Let $\mathbb{P}$ be a program, a terminated forest for a given query*

and a tabulated atom A in the forest. For every correct answer (η, ϑ) for A, where η is grounding, the inequality $\vartheta \leq r_\eta(A)$ holds.

5 A working example

Let us consider an example which illustrates the tabulation procedure at work. Recall the well-known Gödel, Łukasiewicz and product t-norms on the real unit interval:

$$\begin{aligned} I(\&_P)(x,y) &= x \cdot y \\ I(\&_G)(x,y) &= \min(x,y) \\ I(\&_L)(x,y) &= \max(0, x+y-1) \end{aligned}$$

Assume the query $?P(x)$ against the following program:

$$\begin{aligned} P(a) &\leftarrow 0.8 \,\&_G\, Q(a,x) && (1)\\ P(a) &\leftarrow 0.7 \,\&_G\, S(y) && (2)\\ P(x) &\leftarrow 0.9 \,\&_G\, T(x) && (3)\\ Q(a,b) &\leftarrow 0.9 && (4)\\ S(a) &\leftarrow 0.6 \,\&_G\, T(a) && (5)\\ S(b) &\leftarrow 0.7 \,\&_L\, P(x) && (6)\\ T(a) &\leftarrow 0.8 \,\&_P\, S(x) && (7)\\ T(b) &\leftarrow 0.9 \,\&_L\, Q(a,x) && (8) \end{aligned}$$

To begin with, the procedure applies R1 and generates the initial tree in Fig. 1. Now, this tree evolves and grows as an application of the non-deterministic procedure. In order to clarify the application of the rules, we will consider rules R4 and R5 as soon they are applicable, the choice of atoms in R2 and R3 will be made on a lexicographical basis, and new information in the answer list is used as soon as possible.

The next step consists in an application of R2, in which the selected atom is $Q(a, x_1)$. Immediately R3 applies, and the answer list is updated as shown in Fig. 2.

The recently computed valued for $Q(a,b)$ is consumed in Fig. 1. This generates node (ii) and, by R4, the answer list is updated by the computed result 0.8 for $P(a)$.

Now, we consider $S(y_1)$ and apply R2: the tree in Fig. 3 is created; then, node (iii) is generated as an application of R3. As a result, a new computed value $S(b)\colon 0.5$ is appended to the answer list according to R4.

The value $S(b)\colon 0.5$ is consumed again in Fig. 1: node (iv) is generated and the answer list is updated with the value $P(a)\colon 0.5$. Rule R5 removes this new computed value, since is smaller than the current $P(a)\colon 0.8$.

Rule R2 is applied again, and generates a tree for $T(x_1)$, which is shown in Fig. 4. Node (v) is generated by consuming the answer corresponding to $Q(a,b)$ in Fig. 2, then the answer list is updated with $T(b)\colon 0.9$.

This value is used in Fig. 1 in order to generate node (vi) and update the answer list with $P(b)\colon 0.9$

There are still some atoms occurring in leafs, so R2 could still be applicable. Note that not all the atoms create a new tree in the forest, for instance, no new subgoal is generated from atom $S(x)$ in Fig. 4, since a variant of it has already been created. In fact, we can apply R3 and consume the answer $S(b)\colon 0.5$, generating node (vii) and updating the answer list with the value $T(a)\colon 0.4$.

This new computed answer cannot be applied to substitute node $T(a)$ in Fig. 3, since it belongs to the answer list of $T(x_1)$, which is not a variant of $T(a)$. Instead, it generates a new leaf, node $(viii)$, in the tree of Fig. 1, the detail is shown in Fig. 5. The computed value obtained for $P(a)$ is 0.4, which is appended to the answer list, and then removed by application of R5.

Now, the occurrence of $T(a)$ in Fig. 3 creates a new tree, since it has not a tabulated variant (see Fig. 6). The leaf $S(x)$ is tabulated, hence we can use R3 and extend the branch with node (ix); the answer list is then updated with $T(a)\colon 0.4$.

Once again, the recent computer answer allows to extend one tree in the forest; in this case, the tree in Fig. 3 generates node (x) and computes the answer $S(a)\colon 0.4$. There are three possible applications of R3 with this new value, in Fig. 1 nothing happens, since the new computed value is already in the an-

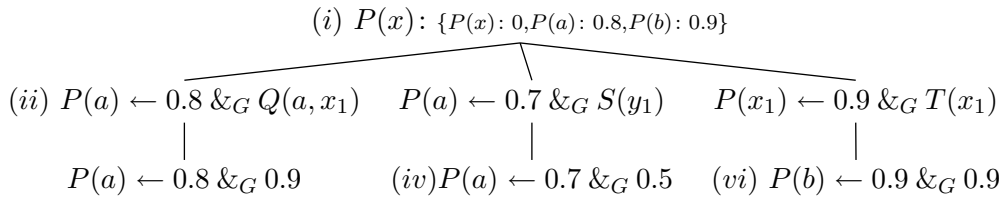

Figure 1: Tabled tree for $P(x)$.

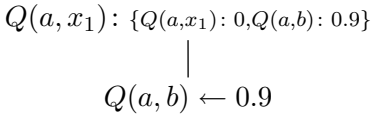

Figure 2: Tabled tree for $Q(a, x_1)$.

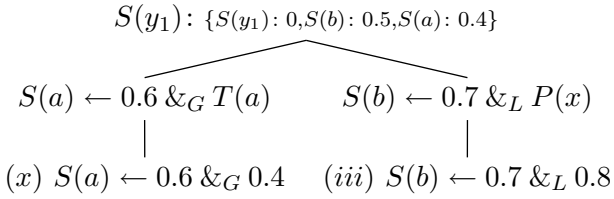

Figure 3: Tabled tree for $S(y_1)$.

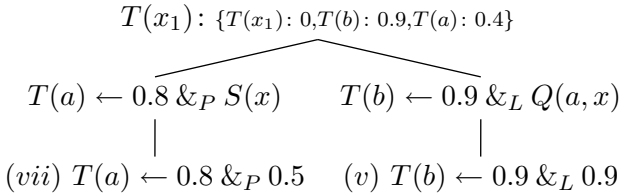

Figure 4: Tabled tree for $T(x_1)$.

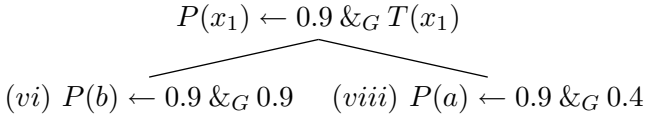

Figure 5: Detail from the initial tree.

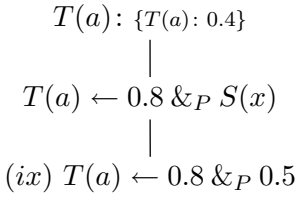

Figure 6: Tabled tree for $T(a)$.

swer list, in Fig. 4 and Fig. 6 a new computed valued $T(a)$: 0.32 is inserted in the answer list. Rule R5 discards the new computed values since they are smaller than the currently computed ones, hence the procedure terminates.

6 Conclusions and Future Work

A tabulation goal-oriented query procedure for first-order residuated logic programs has been introduced in order to generalise the given one for the propositional case in [7]. Note that this approach can be easily adapted so that it can be applied to any multi-adjoint program. Then, after introducing the non-deterministic procedure for tabulation, a thorough example is presented containing the most relevant features of the procedure. So far the authors are not aware of any other approach to the development of tabulation procedures for the extended framework of multi-adjoint logic programming;

However, some related approaches have been published quite recently. For instance, the development of a fold/unfold based transformation system for optimizing multi-adjoint logic programs has been presented in [14]. In this paper, a set of strongly correct transformation rules is presented (strongly correct means here that the semantics of computed substitutions and truth degrees is preserved) which is able to significantly improve the execution of goals against transformed programs. The fold/unfold methodology suggests a parallelism with partial evaluation of programs, which is essentially what is done in every tree of the computational forest.

There are a number of open issues regarding the basic procedure given here. To begin with, a interesting issue from a practical point of view is related to the efficiency of the method. The basic procedure presented here should be improved by means of some modifications to the rules (subsumption instead of variants, updating values instead of simply copying new ones, ...). Subsequently, new forms of rules will be considered whose behaviour should be proved equivalent to the basic ones.

On the other hand, note that termination of the procedure has been assumed in the statement of the completeness theorems. This problem is not an easy one even in the propositional case, as shown in [8]. Even for continuous operators in the bodies, the tabulation procedure might not terminate. Sufficient conditions for termination of the procedure are our target for continuing research on these topics.

A third line of research would be to consider an extension of this procedure for antitonic logic programs [6], which generalise the framework of monotonic and residuated logic programs to allow for rules with arbitrary antitonic bodies over general complete lattices, of which normal programs are a special case.

References

[1] T. Alsinet and L. Godo. Towards an automated deduction system for first-order possibilistic logic programming with fuzzy constants. *International Journal of Intelligent Systems*, 17(9), 2002.

[2] J. F. Baldwin, T. P. Martin, and B. W. Pilsworth. *Fril - Fuzzy and Evidential Reasoning in Artificial Intelligence*. Res. Studies Press, 1995.

[3] R. Bol and L. Degerstedt. The underlying search for magic templates and tabulation. In *Proc. of ICLP93*, pages 793–811, 1993.

[4] W. Chen, T. Swift, and D. S. Warren. Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming*, 24(3):161–199, 1995.

[5] C.V. Damásio and L. M. Pereira. Monotonic and residuated logic programs. *Lect. Notes in Artificial Intelligence* 2143, pp. 748–759, 2001.

[6] C.V. Damásio and L. M. Pereira. Antitonic logic programs. *Lect. Notes in Artificial Intelligence* 2173, pp. 379?393, 2001.

[7] C.V. Damásio, J. Medina and M. Ojeda-Aciego. A tabulation proof procedure for residuated logic programming. In Proc. ECAI'04. *Frontiers in AI and Applications 110:808–812*, 2004

[8] C.V. Damásio, J. Medina and M. Ojeda-Aciego. Termination of logic programs with imperfect information: applications and query procedure. *Journal of Applied Logic*, 2006. To appear.

[9] A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. *J. of Logic Programming*, 43:187–250, 2000.

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

[11] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming*, 12:335–367, 1992.

[12] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. *Lect. Notes in Artificial Intelligence* 2173, pp. 351–364, 2001.

[13] J. Medina, M. Ojeda-Aciego, P. Vojtáš. Similarity-based unification: a multi-adjoint approach. *Fuzzy sets and systems*, 146:43–62, 2004.

[14] G. Moreno. Building a fuzzy transformation system. *Lect. Notes in Computer Science* 3831, pp. 409–418, 2006.

[15] P. Rhodes and S. Merad-Menani. Towards a fuzzy logic programming system: a clausal form fuzzy logic. *Knowledge-Based Systems*, 8(4):174–182, 1995.

[16] M. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1–2):389–426, 2002.

[17] T. Swift. Tabling for non-monotonic programming. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):201–240, 1999.

[18] H. Tamaki and T. Sato. OLD resolution with tabulation. In *Proc. of ICLP'86*, pages 84–98, 1986.

[19] M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.

[20] P. Vojtáš. Fuzzy logic programming. *Fuzzy sets and systems*, 124(3):361–370, 2001.