# A Tabulation Proof Procedure for Residuated Logic Programming

**C.V. Damásio**[1] and **J. Medina**[2] and **M. Ojeda-Aciego**[3]

**Abstract.** Residuated Logic Programs allow to capture a spate of different semantics dealing with uncertainty and vagueness. In this work we provide a tabulation goal-oriented query procedure, and show that our tabulation query procedure terminates if and only if the sequence of iterations of the immediate consequences operator reaches the least fixpoint after only finitely-many steps. On the basis of this result we show that the tabulation procedure terminates for important classes of residuated logic programs, in particular for probabilistic deductive databases of Lakshmanan and Sadri.

## 1 INTRODUCTION

The interest in the development of logics for dealing with information which might be either vague or uncertain has increased in the recent years. Several different approaches to the so-called inexact or fuzzy or approximate reasoning have been proposed, involving either fuzzy or annotated or probabilistic or similarity-based logic programming [1, 2, 4, 8, 9, 10, 11, 12, 19, 20, 21, 23, 24].

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, see [16]. Following traditional techniques of logic programming, a procedural semantics was given in [17], in which non-determinism was discarded by using reductants.

In this paper we aim at the use of tabulation (tabling, or memoizing) methods to increase the efficiency of the previously proposed proof procedures. Tabulation is a technique which is receiving increasing attention in the logic programming and deductive database communities [3, 5, 21, 22]. 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 substitutions corresponding to the atom.

In this work, we provide a tabulation goal-oriented query procedure and show that it is terminating for all queries if and only if the sequence of iterations of the immediate consequences operator reaches the least fixpoint after only finitely-many steps. On the basis of this property, we show that the tabulation procedures terminates for a significant class of residuated logic programs. In particular, several fuzzy logic programming languages can be proven to terminate, namely residuated logic programs whose bodies consist of a repeated application of commutative conjunctors obeying to the boundary condition $v \otimes \top \preceq v$. Conditions for extending the termination property for cartesian products of lattices are presented, and applied to obtain the termination results for probabilistic deductive databases of Lakshmanan and Sadri [13].

The structure of the paper is as follows: in Section 2, the syntax and semantics of residuated logic programs are summarized; Section 3 introduces a non-deterministic procedure for tabulation. The soundness and completeness of the tabling procedure appear in Section 4, as well as independence of the selection ordering and termination properties. Next, we illustrate the proof procedure with an example from Probabilistic Deductive Databases. The paper finishes with some conclusions and pointers to future work. For lack of space, proofs are omitted.

## 2 PRELIMINARY DEFINITIONS

In this section the essentials of residuated logic programming are reviewed [6, 7]. To simplify the presentation, we consider only finite propositional programs over an arbitrary residuated complete lattice. Due to space restrictions we do not introduce the formal definition of a residuated lattice, just let us recall that is an abstract mathematical structure $\langle \mathfrak{R}, \longleftarrow, \otimes \rangle$ based on a lattice $\mathfrak{R}$ and endowed with two operations (an adjoint pair $\langle \longleftarrow, \otimes \rangle$) generalizing the usual conjunction and implication and the modus ponens inference rule (the adjoint condition $x \otimes y \preceq z$ iff $y \preceq z \longleftarrow x$).

**Definition:** A *residuated program* over a residuated complete lattice $\langle \mathfrak{R}, \longleftarrow, \otimes \rangle$ is a finite set of weighted rules $A \xleftarrow{\vartheta} \mathcal{B}$ satisfying:

1. The *head* of the rule $A$ is a propositional symbol.
2. The *body* formula $\mathcal{B}$ is a formula built from propositional symbols $B_1, \dots, B_n$ ($n \geq 0$) by the use of arbitrary monotonic operators, also denoted by $\mathcal{B}[B_1, \dots, B_n]$.
3. The weight $\vartheta$ is an element of $\mathfrak{R}$, interpreted as a truth-value.

*Facts* are rules with constant body $\top$ interpreted as the top element in $\mathfrak{R}$ (usually not written), and a *query* (or *goal*) is a propositional symbol intended as a question $?A$ prompting the system.

An *interpretation* is a mapping $I$ from the set of propositional symbols to $\mathfrak{R}$. Note that each of these interpretations can be uniquely extended via the adjoint condition to the whole set of formulas, in this case it is denoted $\hat{I}$. The ordering $\preceq$ on the underlying lattice can also be easily extended to the set of interpretations, inheriting a structure of complete lattice.

[1] Centro Inteligência Artificial, U. Nova de Lisboa (cd@di.fct.unl.pt)
[2] Dept. Matemática Aplicada, U. de Málaga (jmedina@ctima.uma.es)
[3] Dept. Matemática Aplicada, U. de Málaga (aciego@ctima.uma.es)

**Definition:**

1. An interpretation $I$ *satisfies* a weighted rule $A \overset{\vartheta}{\longleftarrow} \mathcal{B}$ if and only if $\vartheta \otimes \hat{I}\,(\mathcal{B}) \preceq I(A)$, i.e. $\vartheta \preceq \hat{I}\,(A \longleftarrow \mathcal{B})$.
2. An interpretation $I$ is a *model* of a residuated logic program $\mathbb{P}$ iff all weighted rules in $\mathbb{P}$ are satisfied by $I$.
3. An element $\lambda \in \mathfrak{R}$ is a *correct answer* for a program $\mathbb{P}$ and a query $?A$ if for any interpretation $I$ which is a model of $\mathbb{P}$ we have $\lambda \preceq I(A)$.

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

**Definition:** Let $\mathbb{P}$ be a residuated program over a complete lattice $\mathfrak{R}$. The *immediate consequences operator* $T_{\mathbb{P}}^{\mathfrak{R}}$ maps interpretations to interpretations, and is defined by

$$T_{\mathbb{P}}^{\mathfrak{R}}(I)(A) = \bigsqcup_{\mathfrak{R}} \{\vartheta \otimes \hat{I}(\mathcal{B}) \mid A \overset{\vartheta}{\longleftarrow} \mathcal{B} \in \mathbb{P}\}$$

The semantics of a residuated logic program can be characterised, as usual, by the post-fixpoints of $T_{\mathbb{P}}^{\mathfrak{R}}$; that is, an interpretation $I$ is a model of a residuated logic program $\mathbb{P}$ iff $T_{\mathbb{P}}^{\mathfrak{R}}(I)(A) \preceq I(A)$ for all propositional symbol $A$. The $T_{\mathbb{P}}^{\mathfrak{R}}$ operator is proved to be monotonic and continuous under very general hypotheses, see [16, 17], 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 $T_{\mathbb{P}}^{\mathfrak{R}}$ on the least interpretation $\triangle$, which maps every propositional symbol to the least element of $\mathfrak{R}$.

## 3 DESCRIPTION OF THE PROCEDURE

The issue now relies in the definition of an appropriate query procedure for residuated logic programs, although it can be readily adapted for the general case of multi-adjoint lattices [16]. There are two major problems to address: termination and efficiency. On the one hand, the $T_{\mathbb{P}}^{\mathfrak{R}}$ operator is bottom-up but not goal-oriented. Furthermore, in every step the bodies of rules are all recomputed. On the other hand, the usual SLD based implementations of Fuzzy Logic Programming languages (e.g. [25]) are goal-oriented, but inherit the problems of non-termination and recomputation of goals. For tackling these issues, the tabulation implementation technique has been proposed in the deductive databases and logic programming communities [3, 5, 22]. More recently, it is proposed in [12, 21] an extension of SLD for implementing generalized annotated logic programs [12] that will be used to implement the here defined tabling procedure. Other implementation techniques have been proposed for dealing with uncertainty in logic programming, for instance translation into Disjunctive Stable Models [14], but rely on the properties of specific truth-value domains.

In this section we present a general tabulation procedure for propositional residuated logic programs. The datatype we will use for the description of the method is that of a *forest*, that is, a finite set of trees. Each one of these trees has a root labeled with a propositional symbol together with a truth-value from the underlying lattice (called the *current value* for the *tabulated* symbol); the rest of the nodes of each of these trees are labeled with an “extended” formula in which some of the propositional symbols have been substituted by its corresponding value. For the description of the adaptation of the tabulation procedure to the framework of residuated logic programming, we will assume a program $\mathbb{P}$ consisting of a finite number of weighted rules having the form $A \overset{\vartheta}{\longleftarrow} \mathcal{B}$ together with a query $?A$. The purpose of the computational procedure is to give (if possible) the greatest truth-value for $A$ that can be inferred from the information in the program $\mathbb{P}$.

### 3.1 Operations for Tabulation

For the sake of clarity in the presentation, we will introduce the following notation: Given a propositional symbol $A$, we will denote by $\mathbb{P}(A)$ the set of rules in $\mathbb{P}$ which have head $A$. The tabulation procedure requires four basic operations: Create New Tree, New Subgoal, Value Update, and Answer Return. The first operation creates a tree for the first invocation of a given goal. New Subgoal is applied whenever a propositional variable in the body of rule is found without a corresponding tree in the forest, and resorts to the previous operation. Value update is used to propagate the truth-values of answers to the root of the corresponding tree. Finally, answer return substitutes a propositional variable by the current truth-value in the corresponding tree. We now describe formally the operations:

#### 3.1.1 *Rule 1: Create New Tree.*

Given a propositional symbol $A$, assume $\mathbb{P}(A) = \{A \overset{\vartheta_j}{\longleftarrow} \mathcal{B}_j \mid j = 1, \dots, m\}$ and construct the tree below, and append it to the current forest. If the forest did not exist, then generate a singleton list with the tree.

$$\begin{tikzcd}
 & & A\colon \bot \arrow[dll, dash] \arrow[dl, dash] \arrow[dr, dash] & \\
\vartheta_1 \otimes \mathcal{B}_1 & \vartheta_2 \otimes \mathcal{B}_2 & \dots & \vartheta_m \otimes \mathcal{B}_m
\end{tikzcd}$$

#### 3.1.2 *Rule 2: New Subgoal.*

Select a non-tabulated propositional symbol $C$ occurring in a leaf of some tree (this means that there is no tree in the forest with the root node labeled with $C$), then create a new tree as indicated in Rule 1, and append it to the forest.

#### 3.1.3 *Rule 3: Value Update.*

If there are no propositional symbols in a leaf, then evaluate the corresponding arithmetic formula (assume that its value is, say, $s$) and then update the current value (say $r$) of the propositional symbol at the root of the tree by the value of $\mathrm{lub}_{\mathfrak{R}}(r, s)$.

#### 3.1.4 *Rule 4: Answer Return.*

Select in any non-root node a propositional symbol $C$ which is tabulated, and consider that the current value of $C$ is $r$.

- If the propositional symbol has been selected in a leaf $\mathcal{B}[\dots, C, \dots]$, then extend the branch with the node shown in the figure below.

$$\begin{tikzcd}
\mathcal{B}[\dots, C, \dots] \arrow[d, dash] \\
\mathcal{B}[\dots, r, \dots]
\end{tikzcd}$$

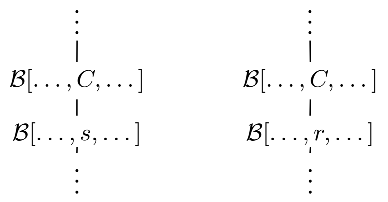
$\mathcal{B}[\ldots, C, \ldots]$
$\mathcal{B}[\ldots, s, \ldots]$
$\mathcal{B}[\ldots, C, \ldots]$
$\mathcal{B}[\ldots, r, \ldots]$

**Figure 1.** Answer Return operation

- Otherwise, if the propositional symbol has been selected in a non-leaf node $\mathcal{B}[\ldots, C, \ldots]$ such as that in the left of Fig. 1 then, if $s \preceq_{\mathfrak{R}} r$, then update the whole branch substituting the constant $s$ by $r$, as in the right of Fig. 1.

It is worth to interpret the execution of each of the previous rules in terms of the better known fixpoint semantics.

For instance, the only rule which changes the values of the roots of the trees in the forest is Rule 3. Note that, the only nodes with several immediate successors are the root nodes, these successors correspond to the different rules with head identical the label of the root node. From there downwards, the extension is done by Rule 4, which either updates the nodes of an existing branch or extends the branch with one new node.

**Remark:** It is convenient to note that in the leaf of each branch there is a conjunction of the truth value of the rule which determined the branch with an instantiation of the body of the rule.

## 3.2 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 with the *create new tree* rule, applied to the query.

**Next steps** Non-deterministically select a propositional symbol and apply one of the rules 2, 3, or 4.

As we shall show, the order of application of the rules is irrelevant. There are other improvements that can be made to the basic tabulation proof procedure. In particular, all nodes whose value of the body cannot surpass the current value of the root node can be safely removed. A sound rule for determining the maximum value the body can achieve consists in substituting all the propositional variables occurring in the node by $\top$. This rule can reduce the search space further more. This pruning rule can be further enhanced if there is information available about completed tables in the forest, i.e. the ones which reached the fixpoint.

## 4 SOUNDNESS AND COMPLETENESS

As in any non-deterministic procedure, it is necessary to show that the obtained result is independent from the different choices made during the execution of the algorithm. With this aim, we state two propositions, which will provide, as a consequence, the independence of the ordering of applications of steps in the tabulation proof procedure as well as soundness and completeness.

**Definition:** Given a residuated logic program $\mathbb{P}$ and a query $?A$. We say that the tabling procedure has constructed a *terminated* forest for $\mathbb{P}$ and $?A$ when no rules of the tabling proof procedure can be applied.

**Proposition 1** *1. The current values of a terminated forest generate a model of $\mathbb{P}$. That is, the current values are greater or equal than those given by the least fixpoint of the immediate consequences operator $T_{\mathbb{P}}^{\mathfrak{R}}$.*

*2. Given a forest (terminated or not), then for all roots $C_j : r_j$ we have that there exists an iteration $k$ of the $T_{\mathbb{P}}^{\mathfrak{R}}$ operator such that $r_j \le T_{\mathbb{P}}^{\mathfrak{R}} \uparrow^k (C_j)$.*

As an easy consequence of the previous proposition we obtain the following result:

**Theorem 1**

1. *Every terminated forest calculates exactly the minimal model for the program.*
2. *The tabulation procedure terminates for all queries if and only if the minimal model is reached by iterating the $T_{\mathbb{P}}^{\mathfrak{R}}$ operator a finite number of times.*

Residuated Logic Programs can be built from arbitrary monotonic operators. A sufficient condition to guarantee termination is described in the following theorem:

**Theorem 2** *Let $\mathbb{P}$ be a residuated logic program over a totally ordered complete lattice $\mathfrak{R}$, in which all the operators $f$ but implication in the language obey to the boundary condition:*

$$\begin{array}{rcl} f(v, \top, \ldots, \top) & \preceq & v \\ f(\top, v, \top, \ldots, \top) & \preceq & v \\ & \vdots & \\ f(\top, \ldots, \top, v) & \preceq & v \end{array}$$

*where $\top$ is the top element of $\mathfrak{R}$, and $v$ an arbitrary element of $\mathfrak{R}$. In particular, if $f$ is a unary function symbol then $f(v) \preceq v$. If $A$ is a propositional symbol and $T_{\mathbb{P}}^{\mathfrak{R}} \uparrow^{n+1} (\triangle)(A) > T_{\mathbb{P}}^{\mathfrak{R}} \uparrow^{n} (\triangle)(A)$, then at least $n+1$ different rules were used on the calculation of $T_{\mathbb{P}}^{\mathfrak{R}} \uparrow^{n+1} (\triangle)(A)$.*

Several proposals in the literature use as underlying complete lattice the closed unit interval $[0, 1]$ with operators obeying to the conditions of Theorem 2. These include van Emden's Quantitative Deduction [23], Possibilistic Logic Programming [9], Non-classical SLD resolution [25], and Ordinary Probabilistic Logic Programs [15].

**Corollary 1** *Let $\mathbb{P}$ be a residuated logic program containing exactly $n$ different rules over the unit interval in which the operators satisfy the conditions of Theorem 2. Then $T_{\mathbb{P}}$ reaches the least fixpoint in at most $n$ iterations.*

We conclude immediately from Theorem 1 and Corollary 1 that the tabulation procedure also terminates for the important class of residuated logic programs, whenever the operators obeys to the above boundary condition in totally ordered complete lattices. All continuous t-norms satisfy, in particular, the boundary condition; therefore the presented results generalize those in [18]. Furthermore, if the usual ordering in the unit interval is reversed then dual conditions can be specified for t-conorms and the termination results still hold (in this case l.u.b. corresponds to minimum, not maximum).

The limitation to totally ordered lattices in Theorem 2 is a rather strong one, preventing the immediate application to more complex truth-value spaces. However, the following result allows us to lift the previous results to important partially ordered domains:

**Theorem 3** *Consider a fixed language (signature $\Sigma$) and several interpretations ($\Sigma$-algebras) of the language over complete residuated lattices $\mathfrak{R}_1, \ldots, \mathfrak{R}_n$. Assume that $T_{\mathbb{P}}^{\mathfrak{R}_i}$ terminates for every residuated logic program over $\mathfrak{R}_i$, with $1 \leq i \leq n$. Then, $T_{\mathbb{P}}^{\mathfrak{R}}$ terminates for the complete product residuated lattice $\mathfrak{R} = \mathfrak{R}_1 \times \ldots \times \mathfrak{R}_n$ where the partial ordering and interpretation of functions are the usual coordinate-wise ones.*

An application of the previous theorem is the proof of termination of the immediate consequences operator of Probabilistic Deductive Databases (PDDs) [13], whenever it is used positive correlation as disjunctive mode for combining several rules in the program, and arbitrary conjunctive modes. The truth-values of PDDs are confidence levels of the form $\langle [\alpha, \beta], [\gamma, \delta] \rangle$, where $\alpha, \beta, \gamma$, and $\delta$ are real numbers in the unit interval[4]. The values $\alpha$ and $\beta$ are, respectively, the expert's lower and upper bounds of belief, while $\gamma$ and $\delta$ are the bounds for the expert's doubt. The fixpoint semantics of PDDs relies on truth-ordering of confidence levels. Suppose $c_1 = \langle [\alpha_1, \beta_1], [\gamma_1, \delta_1] \rangle$ and $c_2 = \langle [\alpha_2, \beta_2], [\gamma_2, \delta_2] \rangle$ are confidence levels, then we say that:

$$c_1 \leq_t c_2 \text{ iff } \alpha_1 \leq \alpha_2, \beta_1 \leq \beta_2 \text{ and } \gamma_1 \geq \gamma_2, \delta_1 \geq \delta_2,$$

with corresponding least upper bound operation $c_1 \oplus_t c_2$ defined as

$$\langle [\max\{\alpha_1, \alpha_2\}, \max\{\beta_1, \beta_2\}], [\min\{\gamma_1, \gamma_2\}, \min\{\delta_1, \delta_2\}] \rangle$$

and greatest lower bound $c_1 \otimes_t c_2$ as:

$$\langle [\min\{\alpha_1, \alpha_2\}, \min\{\beta_1, \beta_2\}], [\max\{\gamma_1, \gamma_2\}, \max\{\delta_1, \delta_2\}] \rangle$$

The least upper bound of truth-ordering corresponds to the disjunctive mode designated "positive correlation", which is used to combine the contributions from several rules for a given propositional symbol. We restrict attention to this disjunctive mode, since the termination results presented in [13] assume all the rules adopt this mode. Conjunctive modes are used to combine propositional symbols in the body, and $\otimes_t$ corresponds to the *positive correlation* conjunctive mode. Another conjunctive mode is *independence* with $c_1 \wedge_{ind} c_2$ defined as

$$\begin{aligned} &\langle\, [\alpha_1 \times \alpha_2, \beta_1 \times \beta_2], \\ &\quad [1 - (1 - \gamma_1) \times (1 - \gamma_2), 1 - (1 - \delta_1) \times (1 - \delta_2)] \rangle \end{aligned}$$

The attentive reader will surely notice that all these operations work independently in each component of the confidence level. Furthermore, the *independence* conjunctive mode combines the $\alpha$'s and $\beta$'s with a t-norm (product), and the $\gamma$ and $\delta$ parts are combined with a t-conorm. This is a property enjoyed by all conjunctive modes specified in [13]. The confidence levels can be seen as a pair of pairs, and by application of Theorems 1 and 3, and Corollary 1 we immediately conclude that the least fixpoint of residuated logic programs over the language containing as operators the conjunctive modes (and implication) always finitely terminates. This is a result shown based solely on general properties of the underlying lattices, not resorting to specific procedural concepts as in [13]. Furthermore, since the grounding of PDDs results in a finite program, there is no lack of generality by assuming finite propositional residuated logic programs.

[4] Even though the authors say that they usually assume that $\alpha \leq \beta$ and $\gamma \leq \delta$, this cannot be enforced otherwise they cannot specify properly the notion of trilattice. So, we also will not assume these constraints.

## 5 EXEMPLIFICATION OF THE PROCEDURE

We now illustrate the tabulation procedure at work, showing how our tabulation proof procedure handles mutual recursions.

**Example:** Consider the following residuated logic program:

$$\begin{aligned}
a &\xleftarrow{\top_t} \langle [0.8, 0.9], [0.0, 0.1] \rangle \wedge_{ind} b \wedge_{ind} c \\
a &\xleftarrow{\top_t} \langle [0.1, 0.3], [0.4, 0.6] \rangle \\
b &\xleftarrow{\top_t} \langle [0.9, 1.0], [0.0, 0.0] \rangle \\
c &\xleftarrow{\top_t} \langle [0.7, 0.8], [0.0, 1.0] \rangle \wedge_{ind} a \\
c &\xleftarrow{\top_t} \langle [0.3, 0.6], [0.2, 0.7] \rangle
\end{aligned}$$

The underlying complete lattice is the lattice of confidence levels of Probabilistic Deductive Databases, under the previous truth-ordering. Notice that all rules have confidence level $\top_t = \langle [1, 1], [0, 0] \rangle$, meaning that the rule is satisfied iff the value of the body is $\leq_t$ than the head. Furthermore, the conjunctor associated with the implication symbol is the greatest lower bound in truth-ordering, i.e. positive correlation conjunctive mode, and not $\wedge_{ind}$. Since it is not essential to provide an explicit definition of implication, we leave the details to the reader. Mark that the above program corresponds to the following probabilistic program of Lakshmanan and Sadri:

$$\begin{aligned}
&\left(a \xleftarrow{\langle [0.8,0.9],[0.0,0.1] \rangle} b, c \;;\; ind, pc\right) \\
&\left(a \xleftarrow{\langle [0.1,0.3],[0.4,0.6] \rangle} \quad\;\;\; ;\; ind, pc\right) \\
&\left(b \xleftarrow{\langle [0.9,1.0],[0.0,0.0] \rangle} \quad\;\;\; ;\; ind, pc\right) \\
&\left(c \xleftarrow{\langle [0.7,0.8],[0.0,1.0] \rangle} a \quad ;\; ind, pc\right) \\
&\left(c \xleftarrow{\langle [0.3,0.6],[0.2,0.7] \rangle} \quad\;\;\; ;\; ind, pc\right)
\end{aligned}$$

Suppose it is intended to determine the truth-degree of proposition $a$. The computation is started by applying rule 1 to $a$ and a possible forest generated by the algorithm is presented in Figure 2. All the nodes are annotated by a possible order of creation, and the selected nodes by rule 2 are underlined. Since, $\top_t \otimes_t v = v$, we omit these expressions in the Figure (introduced by rule 1). Other executions exist, but the computations will terminate in any case and generate the same truth-degrees for all propositional symbols.

The first nodes $(i)$ $(ii)$ and $(iii)$ were created by the *Create New Tree* operation (rule 1). Applying rule 3 to node $(ii)$ we update the truth-degree for $a$ from $\langle [0.0, 0.0], [1.0, 1.0] \rangle$ to $\langle [0.1, 0.3], [0.4, 0.6] \rangle$. The *New SubGoal* selects propositional variable $b$ at node $(iii)$ and creates the new tree with root $(iv)$. The computation proceeds and we get for $b$ the truth-degree $\langle [0.9, 1.0], [0.0, 0.0] \rangle$, by a simple application of *Value Update* to node $(v)$. The *Answer Return* applied to node $(iii)$ generates the new node $(vi)$. The procedure now determines the truth-degree of $c$, and the reader can easily follow the steps. Notice that, *New Subgoal* operation at node $(ix)$ does not create a new tree. The truth-value of $c$ is back propagated to node $(vi)$ and originating a new answer update for $a$. Notice that node $(xi)$ evaluates to $\langle [0.216, 0.54], [0.2, 0.73] \rangle$, and thus the value for $a$ is a mixture of the previous tabled value with the new one. This value is then consumed by node $(i)$ but the truth-value obtained at node $(xi)$ is smaller than the current root node value of $c$, and the computation terminates since no more operations are applicable. This is expected from the discussion in Section 4.

## 6 CONCLUSIONS AND FURTHER WORK

This paper specifies a general non-deterministic tabulation goal-oriented query procedure for residuated logic programs over com-

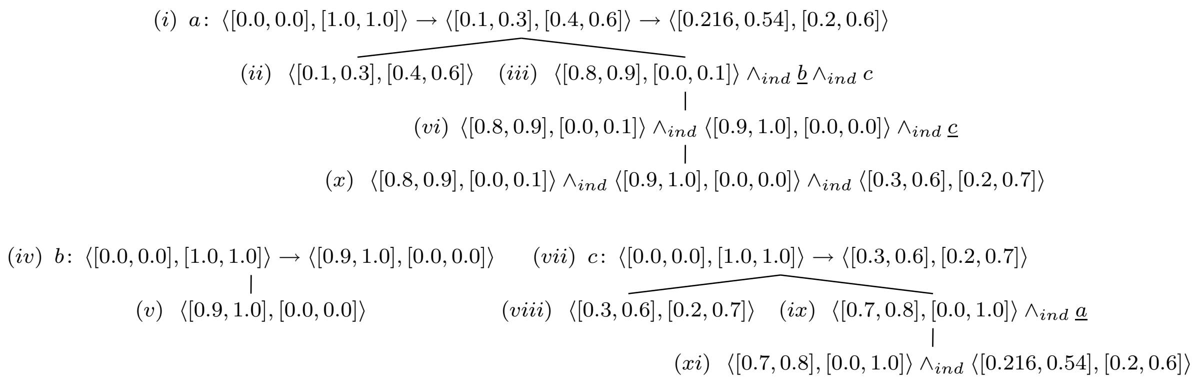
$(i)\ a\colon \langle[0.0,0.0],[1.0,1.0]\rangle \to \langle[0.1,0.3],[0.4,0.6]\rangle \to \langle[0.216,0.54],[0.2,0.6]\rangle$
$(ii)\ \langle[0.1,0.3],[0.4,0.6]\rangle$
$(iii)\ \langle[0.8,0.9],[0.0,0.1]\rangle \wedge_{ind} \underline{b} \wedge_{ind} c$
$(vi)\ \langle[0.8,0.9],[0.0,0.1]\rangle \wedge_{ind} \langle[0.9,1.0],[0.0,0.0]\rangle \wedge_{ind} \underline{c}$
$(x)\ \langle[0.8,0.9],[0.0,0.1]\rangle \wedge_{ind} \langle[0.9,1.0],[0.0,0.0]\rangle \wedge_{ind} \langle[0.3,0.6],[0.2,0.7]\rangle$
$(iv)\ b\colon \langle[0.0,0.0],[1.0,1.0]\rangle \to \langle[0.9,1.0],[0.0,0.0]\rangle$
$(v)\ \langle[0.9,1.0],[0.0,0.0]\rangle$
$(vii)\ c\colon \langle[0.0,0.0],[1.0,1.0]\rangle \to \langle[0.3,0.6],[0.2,0.7]\rangle$
$(viii)\ \langle[0.3,0.6],[0.2,0.7]\rangle$
$(ix)\ \langle[0.7,0.8],[0.0,1.0]\rangle \wedge_{ind} \underline{a}$
$(xi)\ \langle[0.7,0.8],[0.0,1.0]\rangle \wedge_{ind} \langle[0.216,0.54],[0.2,0.6]\rangle$

**Figure 2.** Example forest for query $a$

plete lattices. We present soundness and completeness results for finite propositional residuated logic programs, as well as independence of the selection ordering. The class of proven terminating programs is general enough to include van Emden's Quantitative Deduction [23], Possibilistic Logic Programming [9], Non-classical SLD resolution [25], and Ordinary Probabilistic Logic Programs [15]. We also present a lifting termination theorem for products of residuated lattices, and use it to obtain the known termination results of Probabilistic Deductive Databases [13]. For all these situations, reasoning is polynomial in the size of the ground program.

As future work, on the one hand, a first goal is the attempt to extend this technique to the case of first order residuated logic programs; on the other hand, we are also interested in gaining a better understanding of Fuzzy Rule Systems to be translated into the framework of residuated logic programs. An implementation of the tabulation procedure is underway using the GAP package of XSB Prolog [21]. Last but not the least, theoretical and/or experimental comparison with existent approaches to the computation of minimal models for fuzzy logic programs [1, 2, 9, 12, 19, 20, 21, 23, 24] are needed. However, a major distinguishing feature of our tabulation proof-procedure is that it is defined for arbitrary combinations of operators in the body of programs.

## ACKNOWLEDGMENTS

C. V. Damásio has been partially supported by Acção Integrada Luso-Espanhola E-42/02, by FSE/FEDER project TARDE (POSI/EEI/12097/2001), 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). J. Medina and M. Ojeda-Aciego have been partially supported by Acción Integrada HP2001-0078, and Spanish projects BFM2000-1054- C02-02 and TIC2003-09001-C02-01.

## 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] T. Cao. Annotated fuzzy logic programs. *Fuzzy Sets and Systems*, 113(2):277–298, 2000.

[5] 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.

[6] C. V. Damásio and L. M. Pereira. Monotonic and residuated logic programs. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU-2001*, pages 748–759. LNAI 2143, 2001.

[7] C. V. Damásio and L. M. Pereira. Hybrid probabilistic logic programs as residuated logic programs. *Studia Logica*, 72(1):113–138, 2002.

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

[9] D. Dubois, J. Lang, and H. Prade. Towards possibilistic logic programming. In *Int. Conf. on Logic Progr. 1991*, pages 581–598. MIT Press, 1991.

[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. Jaeger. Automatic derivation of probabilistic inference rules. *International Journal of Approximate Reasoning*, 28(1):1–22, 2001.

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

[13] L. V. S. Lakshmanan and F. Sadri. On a theory of probabilistic deductive databases. *Theory and Practice of Logic Progr.*, 1(1):5–42, 2001.

[14] T. Lukasiewicz. Fixpoint Characterizations for Many-valued Disjunctive Logic Programs with Probabilistic Semantics In *Logic Progr. and Non-Monotonic Reasoning, LPNMR'01*, LNAI 2173:336–350, 2001.

[15] T. Lukasiewicz. Probabilistic logic programming with conditional constraints. *ACM Trans. Comput. Logic*, 2(3):289–339, 2001.

[16] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. In *Logic Programming and Non-Monotonic Reasoning, LPNMR'01*, pages 351–364. LNAI 2173, 2001.

[17] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A procedural semantics for multi-adjoint logic programming. In *Progress in Artificial Intelligence, EPIA'01*, pages 290–297. LNAI 2258, 2001.

[18] L. Paulík. Best possible answer is computable for SLD-resolution. *Lecture Notes in Logic*, 6:257–266, 1996.

[19] 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.

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

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

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

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

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

[25] P. Vojtás and L. Paulík. Soundness and completeness of non-classical extended SLD-resolution. In *Proc. of the Ws. on Extensions of Logic Programming (ELP'96)*. LNCS 1050:289–301, 1996.