A Tabulation Proof Procedure for First-Order Residuated Logic Programs: Soundness, Completeness and Optimisations

C.V. Damásio, J. Medina, and M. Ojeda-Aciego

***Abstract*— Residuated logic programs have shown to be a generalisation of a number of approaches to logic programming under uncertain or vague information, including fuzzy or annotated or probabilistic or similarity-based logic programming frameworks. Various computational approaches have been developed for propositional residuated logic programs: on the one hand, there exists a bottom-up neural-like implementation of the fixed-point semantics which calculates the successive iterations of the immediate consequences operator; on the other hand, a goal-oriented top-down tabulation procedure has recently been introduced. In this paper, we introduce a sound and complete tabulation-based proof procedure for the first-order extension of residuated logic programs.**

I. Introduction

The development of logics for dealing with imperfect information (ie uncertain, vague, imprecise) has been considered an interesting research topic in the recent years. Regarding extension of logic programming in this context, several different approaches to the so-called inexact or fuzzy or approximate reasoning have been proposed. In the specific topic of logic programming under uncertainty, we can find approaches involving either fuzzy or annotated or probabilistic or similarity-based logic programming [1], [2], [9]–[11], [17]–[19], [21], [22].

Residuated and monotonic logic programs [5] and multi-adjoint logic programs [14] were introduced as general frameworks which abstract out the particular details of the different approaches cited above, and focus only on the computational mechanism of inference. This higher level of abstraction makes possible the development of general results about the behaviour of several of the previously cited approaches.

We will focus here on the particular framework of residuated logic programming. The semantics of monotone and residuated logic 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 [14]. Following traditional techniques of logic programming, a procedural semantics was given in [15], in which non-determinism was discarded by using reductants. Various computational approaches have been developed for propositional residuated logic programs: on the one hand, there exists a bottom-up neural-like implementation of the fixed-point semantics which calculates the successive iterations of the immediate consequences operator [13]; on the other hand, a goal-oriented top-down approach tabulation procedure has been presented in [7], [8]. In this paper we maintain our interest on the use of tabulation (tabling, or memoizing) methods.

Tabulation is a technique which is receiving increasing attention in the logic programming and deductive database communities [3], [4], [19], [20]. 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 for first-order monotone and residuated logic programs, the main contribution being the proofs of soundness and completeness.

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 tabling procedure appear in Section 4. Then, an example illustrating the procedure is extensively discussed in Section 5. The paper finishes with some conclusions and pointers to future work.

II. Syntax and Semantics

In this section the essentials of first order residuated logic programming are reviewed. The reader might consult [5] for the propositional version or [16] 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.

C.V. Damásio is with the Centro Inteligência Artificial, Univ. Nova de Lisboa, Portugal. (email: cd@di.fct.unl.pt).

J. Medina is with the Dept. Matemática Aplicada, Univ. de Málaga, Spain. (email: jmedina@ctima.uma.es).

M. Ojeda-Aciego is with the Dept. Matemática Aplicada, Univ. de Málaga, Spain. (email: aciego@ctima.uma.es).

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

$$x \preceq (y \leftarrow z) \quad \text{if and only if} \quad (x \,\&\, z) \preceq y \quad (1)$$

The first two conditions for adjoint pairs specify the usual properties of "conjunction" and "implication". The adjoint condition (1) is more interesting and allows us to use many-valued versions of *modus ponens*. The value x can be understood as the weight associated to the rules, and therefore condition (1) expresses that in order to satisfy the rule the value of the consequent (head) must be larger than or equal to the value of the rule weight conjoined to the value of the body. Dropping any of the sides of the equivalence in condition (1) destroys the expected properties of models of our programs (see [6]). This is the basic inference rule used in residuated logic programs.

Now, we can introduce residuated logic programs as those constructed from a signature of monotone operators and interpreted on a complete residuated lattice:

Definition 2: A *residuated 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 a propositional symbol intended as a question $?A$ prompting the system.

An *interpretation* is a mapping I from the Herbrand base of the program to L. 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.

The definition of satisfiability, as stated above, relies heavily in the adjoint condition, and is the following:

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 $\mathbb{P}$ if and only if $T_{\mathbb{P}}(I)(A) \preceq I(A)$ for all ground atom A.

The $T_{\mathbb{P}}$ operator is proved to be monotonic and continuous under very general hypotheses, see [14], 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 (this requirement ensures that there is at most finitely many rules matching a goal, hence generating finitely many branches after the applications of rules R1 and R2 below).

III. The Tabling Procedure

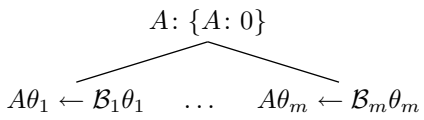
In this section we describe a simple version of the first-order tabling proof procedure for residuated logic programs which allows to obtain more directly the proofs of soundness and completeness.

The procedure generates a set of trees, each one computing answers for a given subgoal. The answers computed for any subgoal are stored in a list (the *answer list*), which is denoted pictorially as a label of the root node of each tree.

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 mgu θ_j satisfying $C_j\theta_j = A\theta_j$, where $j = 1, \ldots, m$.

Construct the following tree with root A



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 $C'\theta' : r$ be an element of the answer list of C', which unifies with C, and was not consumed before. Let $C\Theta = C'\theta'\Theta$ be their most general unifier. Then, add a new successor node

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

R4: Value Update.

Consider 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 is 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 $C\theta : s$.

R5: Answer merging.

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

Remarks:

1) 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 $\mathfrak{AL}(C)$.
2) 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 with the application of 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 of the basic procedure.

IV. Soundness and Completeness

We start by considering the soundness proof of the tabling 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 $A\theta : \vartheta$ 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.

An equivalent definition of correct answer in terms of the $T_{\mathbb{P}}$ operator can be given as follows:

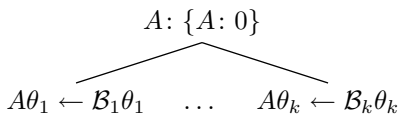
2') (θ, ϑ) is a correct answer for A in $\mathbb{P}$ if for every grounding substitution η we have that

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

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

Proof: Let (θ, ϑ) be a computed answer for A. The result is shown by induction in the number of rules applied by the tabling procedure.

Induction Base: If it is applied only one rule, this must be Rule 1. The following tree is constructed :



Since the answer list is the singleton, $\{A : 0\}$, then we have $0 \leq M(A\eta)$ for any Herbrand model M, and we are done.

Induction Step: Assume that the result holds after the application of n tabling rules.

If we apply Rule 2, the proof is like in the previous case.

Rule 3 does not modify the answer list, hence there is nothing to prove.

For Rule 4, assume we have a leaf for a tree with root A:

$$A\sigma \leftarrow \mathcal{B}[s_1, \ldots, s_n]\sigma$$

In this case we have a leaf where in $\mathcal{B}[s_1, \ldots, s_n]$ there are no atoms, and the answer substitution is σ. Evaluate the corresponding arithmetic formula, assume that its value is, say, s. We have two possibilities:

1) If $A\sigma : s$ is a variant of an answer in the $\mathfrak{AL}(A)$, then we do not change the answer list. Therefore, we finish by the induction hypothesis.
2) If $A\sigma : s$ is not a variant then it is added to the answer list.

It remains to be shown that, given a grounding substitution η, the inequality below holds

$$s \leq T_{\mathbb{P}} \uparrow^{\omega}(A\sigma\eta) \tag{2}$$

Without loss of generality, we assume that there are just two atoms in the body $\mathcal{B}$ and we have $B_1\rho_1 : s_1$ and $B_2\rho_2 : s_2$ in the corresponding answer lists, which satisfy

$$s_i \leq T_{\mathbb{P}} \uparrow^{\omega}(B_i\rho_i\eta) \quad \text{for } i = 1, 2 \tag{3}$$

for all grounding substitution η.

Again w.l.o.g. the "value" $\mathcal{B}[s_1, s_2]$ has been generated from (an instance of) a rule[1]

$$A \leftarrow \mathcal{B}[B_1, B_2]$$

[1]Note that, for the argument, it is irrelevant whether it is an instance of a rule in $\mathbb{P}$ or not.

by successive applications of Rule 3:

$$\begin{aligned} A &\leftarrow \mathcal{B}[B_1, B_2] \\ A\Theta_1 &\leftarrow \mathcal{B}[s_1, B_2]\Theta_1 \\ A\Theta_1\Theta_2 &\leftarrow \mathcal{B}[s_1, s_2]\Theta_1\Theta_2 \end{aligned}$$

where $B_1\Theta_1 = B_1\rho_1\Theta_1$ and $B_2\Theta_1\Theta_2 = B_2\rho_2\Theta_1\Theta_2$. In this case, we would have $\sigma = \Theta_1\Theta_2$ in Eq (2).

Now,

$$\begin{aligned} T_{\mathbb{P}}{\uparrow}^{\omega}(A\sigma\eta) &\geq T_{\mathbb{P}}{\uparrow}^{\omega}(\mathcal{B}[C_1, C_2]\sigma\eta) \\ &= T_{\mathbb{P}}{\uparrow}^{\omega}(\mathcal{B}[C_1\sigma\eta, C_2\sigma\eta]) \\ &= \mathcal{B}[T_{\mathbb{P}}{\uparrow}^{\omega}(B_1\rho_1\eta'), T_{\mathbb{P}}{\uparrow}^{\omega}(B_2\rho_2\eta'')] \\ &\geq \mathcal{B}[s_1, s_2] = s \end{aligned}$$

Finally, for Rule 5, assume $A_1 : s_1$ and $A_2 : s_2$ in an answer list, which unify under mgu θ. The rule adds the answer $A_1\theta : \sup\{s_1, s_2\}$. By induction, for $i = 1, 2$ we have $s_i \leq T_{\mathbb{P}}{\uparrow}^{\omega}(A\sigma_m\eta)$, thus trivially we obtain that

$$\sup\{s_1, s_2\} \leq T_{\mathbb{P}}{\uparrow}^{\omega}(A\sigma_m\eta)$$

■

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* and some other technical concepts.

Definition 6:

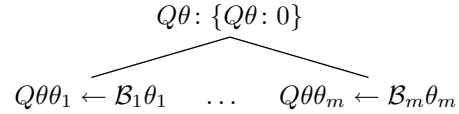
- We say that $\mathfrak{F}'$ is a *super-forest* of a given forest $\mathfrak{F}$ if every tree in $\mathfrak{F}$ is subsumed by another tree in $\mathfrak{F}'$ which, moreover, whose nodes are labelled by formulas more general than those in $\mathfrak{F}$.
- In addition, we say that computed values are *preserved by the super-forest* if for every element $Q\theta\eta : s$ in the answer list of $Q\theta$ in $\mathfrak{F}$ there exists a substitution η' such that $Q\eta' : s$ 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 tabling forest $\mathfrak{F}$ for $Q\theta$, there exists a tabling super-forest for Q which preserves computed answers in $\mathfrak{F}$.

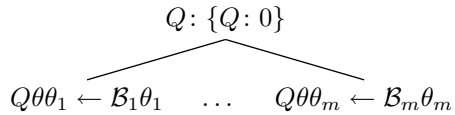
Proof: The idea is to show that it is possible to mimic the construction of a forest for $Q\theta$ by a forest for Q such that the computed values of the atoms in the answer lists are preserved.

The proof is by structural induction, comparing the generation of tabling forests for $Q\theta$ and Q.

In the initial case, $\mathfrak{F}$ is started by an application of R1 to the atom $Q\theta$, we would obtain



whereas for $\mathfrak{F}'$ we apply R1 to the atom Q, obtaining at least the following branches[2]



Note that R1 might introduce more branches since Q could be unifiable with the heads of more rules than $Q\theta$, and we would have a proper super-forest $\mathfrak{F}'$.

For the inductive case, assume that we have a forest $\mathfrak{F}$ for which the statement of the lemma holds; and consider the application of a further rule.

Note that we only need to focus on the application of R4, since for the rest of cases the result can be checked easily.

For R4, we have a leaf in the forest for $Q\theta$ of the form

$$A\sigma \leftarrow \mathcal{B}[s_1, \ldots, s_m]\sigma$$

We have two possibilities:

1) If $A\sigma : s$ or a variant is in the answer list, then there is nothing to prove, since the answer list is not modified.
2) If $A\sigma : s$ is not a variant then it is added to the answer list.

In this case, by the induction hypothesis, we have a more general leaf in the forest for Q

$$A\eta' \leftarrow \mathcal{B}[s_1, \ldots, s_m]\eta'$$

in which σ' is such that $\sigma \leq \sigma'$.

For this leaf, an application of R4 either a variant of the corresponding computed answer is already in the answer list or it is included, satisfying in any case what we want to prove. ■

Definition 7: 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

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

Note that, by repeated application of Rule 5 and assuming finite termination of the forest construction, there must exist some answer $A\theta : r$ 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

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

Proof: As A is ground, obviously we have $A = A\eta$ and the relative computed values do not depend on the grounding substitution η.

By induction on $n \in \mathbb{N}$.

Induction Base: It is straightforward because

$$T_{\mathbb{P}}{\uparrow}^{0}(A\eta) = 0 \quad \text{for all atom } A,$$

and by application of Rules 1 and 2.

Induction Step: Assume as induction hypothesis that

$$T_{\mathbb{P}}{\uparrow}^{n}(B) \leq r(B) \quad \text{for all ground atom } B.$$

We will show that given a terminated forest for A, we have that also $T_{\mathbb{P}}{\uparrow}^{n+1}(A) \leq r(A)$.

[2]Here we are taking advantage of the variables in the rules being renamed apart.

Recalling the definition of $T_{\mathbb{P}}$, consider any rule, say

$$(C_j \leftarrow \mathcal{B}_j[B_1, \ldots, B_m])\zeta,$$

in the grounding of $\mathbb{P}$ whose head is A.

It is clear that a corresponding more general

$$C_j \leftarrow \mathcal{B}_j[B_1, \ldots, B_m]\sigma$$

should appear in the tree for A. Rules 1 and 2, combined with Rule 4, guarantee that a tree for each $B_k\sigma$ $(1 \leq k \leq m)$ will be created in the forest. Thus, for every ground atom $B_k\zeta$ we will have a tree $B_k\sigma$ for it in the forest.

By application of induction hypothesis, we have that

$$T_{\mathbb{P}}\uparrow^n(B_k\zeta) \leq r(B_k\zeta)$$

and there is an answer in $B_k\sigma$ with computed value $r(B_k\zeta)$ (we will denote it s_k for brevity) which unifies with $B_k\zeta$. By consumption of these answers by Rule 3, we have a leaf in the tree for A with body $\mathcal{B}_j[s_1, \ldots, s_m]$ whose head unifies with A.

The choice of the rule in $\mathbb{P}$ for A was completely arbitrary, therefore for each ground rule $C_j \leftarrow \mathcal{B}_j[B_1, \ldots, B_m]\zeta$ we have

$$\begin{aligned}\mathcal{B}_j[\ldots, T_{\mathbb{P}}\uparrow^n(B_k\zeta), \ldots] \leq \\ \leq \quad & \mathcal{B}_j[\ldots, s_k, \ldots] \\ \leq \quad & r_\eta(A) \quad \text{for all } \eta\end{aligned}$$

Therefore $T_{\mathbb{P}} \uparrow^{n+1}(A) \leq r_\eta(A)$ by definition of $T_{\mathbb{P}}$, since $T_{\mathbb{P}}\uparrow^{n+1}(A)$ is the supremum of all the bodies

$$\mathcal{B}_j[\ldots, T_{\mathbb{P}}\uparrow^n(B_k\zeta), \ldots]$$

of every rule in the grounded version of $\mathbb{P}$ with head A. ■

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

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

Proof: Let r be the maximum computed value *in the tree for* $A\eta$, and let S be the set of values considered for calculating the computed value for A relative to the substitution η, this is:

$$S = \{s \mid A\theta\colon s \in \mathfrak{AL}(A),\ \theta\gamma = \eta\}$$

Applying the Lifting Lemma to the computed answer associated to r, there is a substitution σ more general than η such that $A\sigma\colon r \in \mathfrak{AL}(A)$, therefore $r \in S$ and we have

$$r \leq \sup S = r_\eta(A)$$

We finish the proof from the following chain of inequalities:

$$T_{\mathbb{P}}\uparrow^n(A\eta) \leq r \leq \sup S = r_\eta(A)$$

■

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.

Proof: On the one hand, from the definition of correct answer, we have that

$$\vartheta \leq T_{\mathbb{P}}\uparrow^\omega(A\eta) \tag{4}$$

On the other hand, the previous theorem states that

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

then:

$$T_{\mathbb{P}}\uparrow^\omega(A\eta) \leq r_\eta(A) \tag{5}$$

The result follows from (4) and (5) ■

V. A Worked Example

We now introduce an example of the procedure which illustrates how it handles mutual recursion.

Example 1: Consider the following residuated logic program:

$$\begin{aligned} P(x_1) &\leftarrow Q(x_2)\ \&_G\ R(x_1) \\ P(a) &\leftarrow 0.6 \\ Q(f(b)) &\leftarrow 0.7 \\ R(x_3) &\leftarrow P(x_3)\ \&_G\ 0.7 \\ R(a) &\leftarrow 0.7 \end{aligned}$$

where the underlying residuated complete lattice is the unit interval, the symbols a, b denote constants, x_1, x_2, x_3 are variables, f is a unitary function symbol, P, Q, R are predicate symbols and $\&_G$ is Gödel conjunction (the minimum).

Let us describe the execution of the non-deterministic tabling procedure for the initial query $P(y)$, a possible forest generated by the procedure is presented in Figure 1. All the nodes are annotated by a possible order of creation, and the selected nodes by R2 are underlined.

The forest is started by applying R1 to $A(y)$, and nodes (i), (ii) and (iii) are created with substitutions

$$\{y/a\} \quad \text{and} \quad \{y/x_1\}$$

An application of R4 to node (ii) results in adding $P(a)\colon 0.6$ to the answer list, since there is not a variant of $P(a)$ with value 0.6 (these updates of the answer lists are denoted in the picture by using the arrow $\rightsquigarrow$).

Now, R2 selects the atom $Q(x_2)$ at node (iii) and creates the new tree with root (iv) and leaf (v), with mgu $\{x_2/f(b)\}$. The computation proceeds and R4 adds $Q(f(b))\colon 0.7$ to the answer list for $Q(x_2)$.

The new value included in the answer lists enables an application of R3, which generates the new node (vi) using the mgu $\{x_2/f(b)\}$.

A new application of R2 selects atom $R(x_1)$ at node (vi) and generates a new tree in the forest consisting of nodes (vii), $(viii)$ and (ix). Note that node $(viii)$ can be selected for an application of R4, dealing to an update of the answer list, including $R(a)\colon 0.7$.

The new value inserted in the answer list for $R(x_1)$ can now be used by R3 to generate node (x), where the mgu is the substitution $\{x_1/a\}$. Now, a new application of R4 updates the answer list of $P(y)$ by adding the value $P(a)\colon 0.7$.

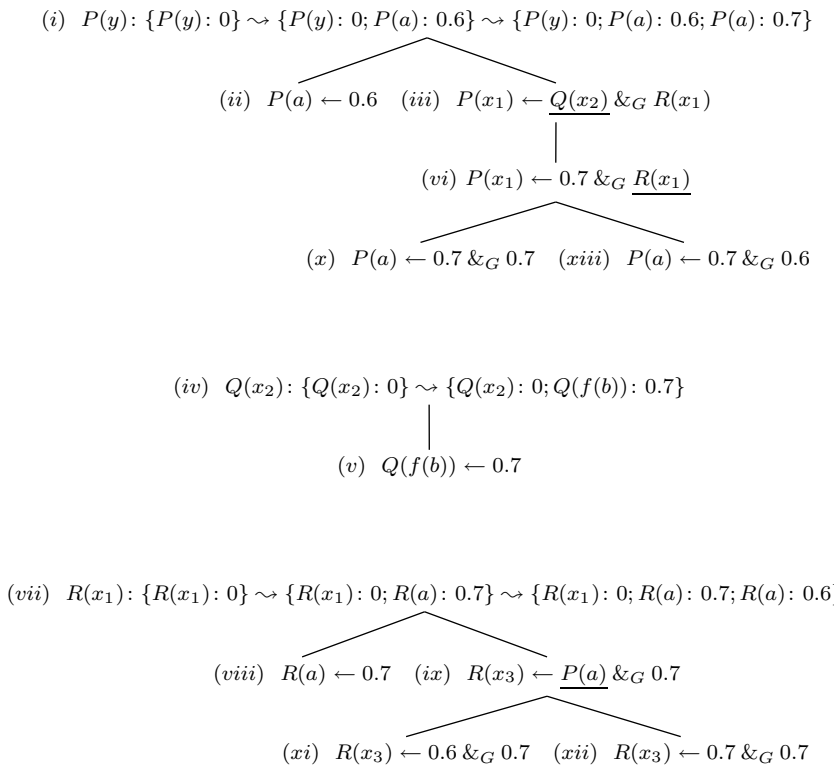


Fig. 1. A forest for Example 1.

Note that R5 does not apply here, since $P(a)\colon \sup\{0.7, 0.6\}$ is already in the answer list (essentially, this rule is relevant when the underlying truth-value set is not totally ordered).

The non-consumed answers $P(a)\colon 0.6$ and $P(a)\colon 0.7$ are used for two applications of R3 in order to generate nodes (xi) and (xii). Both nodes trigger R4, an application on (xi) updates the answer list by including $R(a)\colon 0.6$, whereas the application on (xii) does not modify the answer list, for $R(a)\colon 0.7$ is already in the list.

Finally, the recently obtained value $R(a)\colon 0.6$ is consumed by R3 on node (vi); as a result, node $(xiii)$ is generated, which triggers R4 with substitution $\{x_1/a\}$ but does not update the answer list of $P(y)$, since $P(a)\colon 0.6$ is already there.

The computation terminates since the forest does not get changed by any rule. □

Note that the answers labeled with 0 have not been consumed for the sake of simplicity of the picture. Moreover, note that the previous example also illustrates a number of superfluous computations which should be avoided in a practical implementation, but recall that our aim in this work was obtaining a tabulation procedure with a reasonable proof of completeness.

VI. Conclusions and Future Work

In this paper we have introduced a tabulation goal-oriented query procedure for first-order residuated logic programs, which generalises that given for the propositional case in [7]. After introducing a non-deterministic procedure for tabulation, we prove the corresponding soundness and completeness theorems.

It is worth to note that the basic tabling procedure proposed admits a number of possible improvements:

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 tabling procedure might not terminate. Sufficient conditions for termination of the procedure are our target for continuing research on these topics.

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. Sorted monotonic logic programs and their embeddings. In *Information Processing and Management of Uncertainty for Knowledge-Based Systems*, IPMU'04, pages 807–814, 2004.

[7] C.V. Damásio, J. Medina, and M. Ojeda-Aciego. A tabulation proof procedure for residuated logic programming. In *European Conference on Artificial Intelligence*, volume 110 of *Frontiers in Artificial Intelligence and Applications*, pages 808-812, 2004.

[8] C.V. Damásio, J. Medina and M. Ojeda-Aciego. Sorted multi-adjoint logic programs: termination results and applications. *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.W. Lloyd. Foundations of Logic Programming, *Springer Verlag*, 1987.

[13] J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. A neural implementation of multi-adjoint logic programs. *Journal of Applied Logic* 2(3):301–324, 2004.

[14] 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.

[15] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A procedural semantics for multi-adjoint logic programming. *Lect. Notes in Artificial Intelligence* 2258:290–297, 2001.

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

[17] 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.

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

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

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

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

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