On Termination of a Tabulation Procedure for Residuated Logic Programming ⋆

C.V. Damásio[1] and M. Ojeda-Aciego[2]

[1] Centro Inteligência Artificial. Univ. Nova de Lisboa. Portugal (cd@di.fct.unl.pt)
[2] Dept. Matemática Aplicada. Univ. de Málaga. Spain (aciego@ctima.uma.es)

Abstract. Residuated Logic Programs allow to capture a spate of different semantics dealing with uncertainty and vagueness. A first result states that for any *definite* residuated logic program the sequence of iterations of the immediate consequences operator reaches the least fixpoint after only finitely many steps. Then, a tabulation query procedure is introduced, and it is shown that the procedure terminates every definite residuated logic program.

In this work we focus on the framework of residuated logic programming, which generalizes several approaches to the extension of logic programming techniques to the fuzzy case. Our aim here is in the use of tabling (or memoizing) methods to increase the efficiency of the previously proposed semantics. For the essentials of residuated logic programming the reader is referred to [3]. The semantics of a residuated logic program is characterised, by the post-fixpoints of the immediate consequences operator $T_{\mathbb{P}}$, whose definition can be easily generalised to the framework of residuated logic programs. Moreover, it can be shown that $T_{\mathbb{P}}$ is always increasing.

Residuated logic programs allow arbitrary combinations of operators in the body of rules, however the most frequently used fuzzy rule systems employ a single type of conjunctor, usually a *t-norm* (an associative, commutative operator on the unit interval $[0,1]$, with 1 as neutral element). For instance, in [2] we illustrated how to encode approximate deductions and fuzzy rules in control systems into residuated logic programming. The bodies of weighted rules obtained by the encoding have the simple form: $\langle A;\vartheta\rangle$ or $\langle A \longleftarrow B_1 \otimes \ldots \otimes B_n;\vartheta\rangle$, where A and B_i's are propositional variables and ϑ is a truth-value in $[0,1]$. Programs having this form are called *definite* residuated logic programs, i.e. where the body of rules is constructed by conjoining together propositional variables with a unique t-norm operator.

Our first theorem concerning termination in this context is the following:

Theorem 1. *Consider a definite residuated logic program over a continuous t-norm. Then, the $T_{\mathbb{P}}$ operator reaches its least fixpoint after finitely many steps.*

Proof (Sketch): It is well-known that every continuous t-norm is either minimum, Archimedean, or the ordinal sum of a family of continuous Archimedean t-norms. Therefore the proof can be split into two cases: the minimum t-norm and any Archimedean t-norm.

⋆ Partially supported by Integrated Action HP2001-0078 and E-42/02.

For the case of minimum the result is obvious because of the monotonicity of $T_{\mathbb{P}}$ and the fact that we have a finite program.

For the case of Archimedean t-norms the result follows from the fact that any computed value by the semantics has the general form ${\vartheta_1}^{n_1} \otimes \cdots \otimes {\vartheta_m}^{n_m}$ and such an element cannot have infinitely many strict upper bounds of that form. □

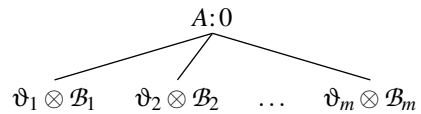
The issue now relies in the definition of an appropriate query procedure for residuated logic programs. In the one hand, the $T_{\mathbb{P}}$ operator is bottom-up but not goal-oriented and in every step the bodies of the rules are recomputed. On the other hand, the usual SLD based implementations of Fuzzy Logic Programming languages 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 [1].

We introduce a general tabulation procedure for 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 of these trees has a root labeled with a propositional symbol together with a truth-value from the real unit interval (called the *current value* for the *tabulated* symbol); the rest of the nodes of each of these trees are labeled with an "extended" formula (defined in [5]).

In the description of the tabulation procedure for residuated logic programming, we will assume a program $\mathbb{P}$ consisting of a finite number of weighted propositional formulas of the form $\langle A \longleftarrow \mathcal{B}, \vartheta \rangle$ together with a query $?Q$. The body is an arbitrary combination of computable monotonic functions applied to propositional variables. 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 $\mathbb{P}$.

Operations for Tabulation. Consider 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:

Create New Tree (CNT). Given a propositional symbol A, construct the tree below, where we assume that $\mathbb{P}(A) = \{\langle A \longleftarrow \mathcal{B}_j, \vartheta_j \rangle \mid j = 1, \ldots, m\}$,



and append it to the current forest (finite list of trees). In the case that the forest did not exist, then generate a singleton list with the tree above.

New Subgoal (NS). 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 CNT, and append it to the forest.

Value Update (VU). If there are no propositional symbols in a leaf, then evaluate the corresponding arithmetic formula, assume that its value is, say, s and update the current value r of the propositional symbol at the root of the tree by the value of $\sup(r,s)$.

Answer Return (AR). 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 $\mathcal{B}[\ldots,C,\ldots]$, then extend the branch with $\mathcal{B}[\ldots,r,\ldots]$

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

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

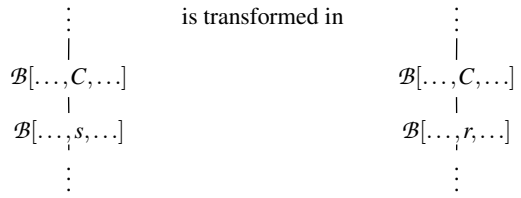


Fig. 1. Updating a branch with a current value.

Note that the only operation which changes the values of the roots of the trees in the forest is VU. Note also that the only nodes with several immediate successors are the root nodes. From there downwards, the extension is done by AR, which either updates the nodes of an existing branch or extends the branch with one new node.

A non-deterministic procedure for tabling. Now, we can state the general non-deterministic procedure for calculating the answer to a given query by using a tabling technique in terms of the previous rules.

Initial step Create the initial forest applying CNT to the query.
Next steps Non-deterministically select a propositional symbol and apply one of the operations NS, VU, and AR.

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 prove two propositions, which provide as a consequence the independence of the ordering of applications of steps in the tabulation proof procedure, as well as soundness and completeness.

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 $T_{\mathbb{P}}$.*
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 $T_{\mathbb{P}}$ operator such that $r_j \leq T_{\mathbb{P}}^k(C_j)$.*

As an easy consequence of the previous proposition we obtain:

Theorem 2.

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

Since in Theorem 1 we showed that for every definite residuated logic program the minimal model is reached after finitely many iterations of the $T_{\mathbb{P}}$ operator, we conclude:

Corollary 1. *The tabulation procedure is terminating for the class of definite residuated logic programs.*

Concluding remarks. A non-deterministic tabulation goal-oriented query procedure has been introduced. We have also shown that the procedure terminates for the class of definite residuated logic programs. As future work, on the one hand, we would like to study further conditions of termination for the general lattice-valued framework of residuated logic programs; on the other hand, we would also study the generalized case of multi-adjoint logic programs [4]. Another interesting research line is to attempt the extension of this technique to the case of first order definite residuated programs.

References

1. 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.
2. C. V. Damásio and M. Ojeda-Aciego. A tabulation proof procedure for residuated logic programming. Submitted, 2003.
3. 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. Lect. Notes in Artificial Intelligence 2143, 2001.
4. 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. Lect. Notes in Artificial Intelligence 2173, 2001.
5. 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. Lect. Notes in Artificial Intelligence 2258, 2001.