A Static Preprocess for Improving Fuzzy Thresholded Tabulation[⋆]

P. Julián[1], J. Medina[2], P. J. Morcillo[3], G. Moreno[3], and M. Ojeda-Aciego[4]

[1] Dept. of Information Technologies and Systems. University of Castilla-La Mancha
Email: `pascual.julian@uclm.es`
[2] Department of Mathematics. University of Cadiz.
Email: `jesus.medina@uca.es`
[3] Department of Computing Systems. University of Castilla-La Mancha.
Email: `gines.moreno@uclm.es,pmorcillo@dsi.uclm.es`
[4] Department of Applied Mathematics. University of Málaga.
Email: `aciego@uma.es`

Abstract. Tabulation has been widely used in most (crisp) declarative paradigms for efficiently running programs without the redundant evaluation of goals. More recently, we have reinforced the original method in a fuzzy setting, by the dynamic generation of thresholds which avoid many useless computations leading to insignificant solutions. In this paper, we draw a static technique for generating such filters without requiring the consumption of extra computational resources at execution time.

Key words: Fuzzy Logic Programming, Tabulation, Thresholding, Unfolding

1 Introduction

Fuzzy logic programming represents a flexible and powerful declarative paradigm amalgamating fuzzy logic and logic programming, for which there exists different promising approaches described in the literature [5, 9, 2, 11]. One step beyond of [6], in this work we refine an improved fuzzy query answering procedure for the so-called MALP (*Multi-Adjoint Logic Programming*) approach [10, 11], which avoids the re-evaluation of goals and the generation of useless computations thanks to the combined use of tabulation [13, 4] with thresholding techniques [8], respectively. As shown in Section 2, the general idea is that, when trying to perform a computation step by using a given program rule R, we firstly analyze if such step might contribute to reach further significant solutions (not tabulated - saved, stored- yet). When it is the case, it is possible to avoid useless computation steps via rule R by using thresholds/filters based on the truth degree of R, as well as a safe, accurate and dynamic estimation of the maximum truth degree associated to its body. Moreover, in Section 3, we propose too a static preprocess

⋆ Work supported by the Spanish MICINN Ministry projects TIN2009-14562-C05-01, TIN2009-14562-C05-03 and TIN 2007-65749 and by the Andalucía and Castilla-La Mancha Administrations under grants P09-FQM-5233 and PII1I09-0117-4481.

with links to well-known unfolding techniques [3, 14, 1, 7] in order to build and manage a powerful kind of filters which largely enhances the benefits achieved by thresholding when combined with fuzzy tabulation.

The MALP approach (see the original formal definition in [10, 11] a real implementation in [12]) considers a language, $\mathcal{L}$, containing propositional variables, constants, and a set of logical connectives. In our fuzzy setting, we use implication connectives ($\leftarrow_1, \leftarrow_2, \ldots, \leftarrow_m$) together with a number of aggregators. They will be used to combine/propagate truth values through the rules. The general definition of aggregation operators subsumes conjunctive operators (denoted by $\&_1, \&_2, \ldots, \&_k$), disjunctive operators ($\vee_1, \vee_2, \ldots, \vee_l$), and average and hybrid operators (usually denoted by $@_1, @_2, \ldots, @_n$). Aggregators are useful to describe/specify user preferences: when interpreted as a truth function they may be considered, for instance, as an arithmetic mean or a weighted sum. The language $\mathcal{L}$ will be interpreted on a *multi-adjoint lattice*, $\langle L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n \rangle$, which is a complete lattice equipped with a collection of adjoint pairs $\langle \leftarrow_i, \&_i \rangle$, where each $\&_i$ is a conjunctor[5] intended to provide a *modus ponens*-rule w.r.t. $\leftarrow_i$. In general, the set of truth values L may be the carrier of any complete bounded lattice but, for simplicity, in the examples of this work we shall select L as the set of real numbers in the interval $[0, 1]$.

A *rule* is a formula $A \leftarrow_i \mathcal{B}$, where the *head* A is an propositional symbol and the *body* $\mathcal{B}$ is a formula built from propositional symbols $B_1, \ldots, B_n$ ($n \geq 0$), truth values of L and conjunctions, disjunctions and aggregations. Rules with an empty body are called *facts*. A *goal* is a body submitted as a query to the system. Roughly speaking, a MALP program is a set of pairs $\langle R; \alpha \rangle$, where R is a rule and α is a value of L, which might express the confidence which the user of the system has in the truth of the rule R (note that the truth degrees in a given program are expected to be assigned by an expert). In contrast with the fuzzy extension of SLD-resolution described for MALP programs in [10, 11], in what follows we recast from [6] the much more efficient procedural principle based on thresholded tabulation for efficiently executing MALP programs.

2 The Fuzzy Thresholded Tabulation Procedure

Tabulation arises as a technique to solve two important problems in deductive databases and logic programming: termination and efficiency. The idea of tabulation (or tabling) is simply to create a table for collecting all the answers to a given goal without repetitions. Every time a goal is invoked it is checked whether there is already a table for that goal. If so, the caller becomes a consumer of the tree, otherwise the construction of a new table is started. All answers produced are kept in the table without repetitions, and are propagated to the pending consumers.

For the description of the procedure to the framework of multi-adjoint logic programming, we will assume a program $\mathbb{P}$ consisting of a finite number of

[5] An increasing operator satisfying boundary conditions with the top element.

weighted rules 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 $\mathbb{P}$.

The datatype we will use for the description of the proposed 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.

The idea is to unfold goals, as much as possible, using the notion of unfolding rule developed in [7] for multi-adjoint logic programs, in order to obtain an optimized version of the original program. In [8], the construction of such "unfolding trees" was improved by pruning some useless branches or, more exactly, by avoiding the use (during unfolding) of those program rules whose weights do not surpass a given "threshold" value. The following descriptions will be considered, in order to introduce this idea in the procedure

– Let $\mathcal{R} = \langle A \leftarrow_i \mathcal{B}; \vartheta \rangle$ be a program rule.
– Let $\mathcal{B}'$ be an expression with no atoms, obtained from body $\mathcal{B}$ by replacing each occurrence of a propositional symbol by $\top$.
– Let $v \in L$ be the result of interpreting $\mathcal{B}'$ under a given lattice.
– Then, $Up\_body(\mathcal{R}) = v$.

Apart from the truth degree ϑ of a program rule $\mathcal{R} = \langle A \leftarrow_i \mathcal{B}; \vartheta \rangle$ and the maximum truth degree of its body $Up\_body(\mathcal{R})$, in the multi-adjoint logic setting, we can consider a third kind of filter for reinforcing thresholding. The idea is to combine the two previous measures by means of the adjoint conjunction $\&_i$ of the implication $\leftarrow_i$ in rule $\mathcal{R}$. Now, we define the *maximum truth degree of a program rule*, symbolized by function Up_rule, as: $Up\_rule(\mathcal{R}) = \vartheta \&_i (Up\_body(\mathcal{R}))$.

These considerations will be in the first rule: root expansion.

Operations for tabulation with thresholding

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: Root Expansion, New Subgoal/Tree, Value Update, and Answer Return.

In the first operation, the *filters* for thresholding argued previously are implemented, from which, the number of nodes in trees can be drastically diminished. Note that by avoiding the generation of a single node, the method implicitly avoids the generation of all its possible descendants as well. On the other hand, the time required to properly evaluate the filters is largely compensated. Anyway, in order to perform an efficient evaluation of filters, it must be taken into account that a condition only is checked if none of the previous ones fails. In particular, the only situation in which the three filters are completely evaluated appears only when the first two ones do not fail.

New Subgoal is applied whenever a propositional variable is found without a corresponding tree in the forest. 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:

Rule 1: Root Expansion. Given a tree with root $A\colon r$ in the forest, if there is at least a program rule $\mathcal{R} = \langle A \leftarrow_i \mathcal{B}; \vartheta \rangle$ not consumed before and verifying the three conditions below, append the new child $\vartheta \&_i \mathcal{B}$ to the root of the tree.

- Condition 1. $\vartheta \nleq r$.
- Condition 2. $\mathit{Up\_body}(\mathcal{R}) \nleq r$.
- Condition 3. $\mathit{Up\_rule}(\mathcal{R}) \nleq r$.

Rule 2: New Subgoal/Tree. 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 with a single node, the root $C\colon \bot$, and append it to the forest.

Rule 3: Value Update. If a tree, rooted at $C\colon r$, has a leaf $\mathcal{B}$ with no propositional symbols, and $\mathcal{B} {\rightarrow_{IS}}^* s$, where $s \in L$, then update the current value of the propositional symbol C by the value of $\sup_L\{r, s\}$.

Furthermore, once the tabulated truth-value of the tree rooted by C has been modified, for all the occurrences of C in a non-leaf node $\mathcal{B}[C]$ such as the one in the left of the figure below then, update the whole branch substituting the constant u by $\sup_L\{u, t\}$ (where t is the last tabulated truth-value for C, i.e. $\sup_L\{r, s\}$) as in the right of the figure.

$$\begin{tikzcd}[row sep=small]
\vdots \arrow[d, no head] & \vdots \arrow[d, no head] \\
\mathcal{B}[C] \arrow[d, no head] & \mathcal{B}[C] \arrow[d, no head] \\
\mathcal{B}[C/u] \arrow[d, no head, dotted] & \mathcal{B}[C/\sup_L\{u, t\}] \arrow[d, no head, dotted] \\
\vdots & \vdots
\end{tikzcd}$$

Rule 4: Answer Return. Select in any leaf a propositional symbol C which is tabulated, and assume that its current value is r; then add a new successor node as shown below:

$$\begin{tikzcd}[row sep=small]
\mathcal{B}[C] \arrow[d, no head] \\
\mathcal{B}[C/r]
\end{tikzcd}$$

Once we have presented the rules to be applied in the tabulation procedure, it is worth to recall some facts:

1. The only nodes with several immediate successors are root nodes; the successors correspond to the different rules whose head matches the label of the root node.
2. The leaf of each branch is a conjunction of the truth value of the rule which determined the branch with an instantiation of the body of the rule.
3. The extension of a tree is done only by Rule 4, which applies only to leaves and extends the branch with one new node.
4. The only rule which changes the values of the roots of the trees in the forest is Rule 3 which, moreover, might update the nodes of existing branches.

The non-deterministic thresholded tabulation procedure is as follows:

Initial step: Create an initial tree by using the rule *new subgoal/tree* on the query.

Next steps: Non-deterministically select a propositional symbol and apply one of the rules 1, 2, 3, or 4.

The correctness of the thresholded tabulation procedure was proved in [6]. Furthermore, a deterministic procedure was presented using the four basic operations above.

3 Improvements based on Static Preprocess

Before illustrating the fast execution method explained in the previous section, we would like to enhance a particularity of the first "**Root Expansion Rule**". Note that its application requires (in the worst case) the dynamic generation of three filters aiming to brake, when possible, the expansion of degenerated branches on trees. Such filters can be safely compiled on program rules after applying an easy static preprocess whose benefits will be largely be redeemed on further executions of the program. So, for any given MALP program $\mathbb{P}$, we can obtain its extended version $\mathbb{P}+$ (for being used during the "query answering" process) by adding to its program rules their *proper threshold* $\mathrm{Up\_rule}(\mathcal{R})$ as follows: $\mathbb{P}+ = \{\langle A \leftarrow_i \mathcal{B}; \vartheta;\ \mathit{Up\_rule}(\mathcal{R})\rangle \ \mid\ \mathcal{R} = \langle A \leftarrow_i \mathcal{B}; \vartheta\rangle \in \mathbb{P}\}$.

Assuming the extended program $\mathbb{P}+$, we consider the new Rule 1:
Rule 1: Root Expansion. Given a tree with root $A\colon r$ in the forest, if there is at least a program rule $\mathcal{R} = \langle A \leftarrow_i \mathcal{B}; \vartheta;\ \mathit{Up\_rule}(\mathcal{R})\rangle$ not consumed before and verifying $\mathit{Up\_rule}(\mathcal{R}) \nleq r$, append the new child $\vartheta \&_i \mathcal{B}$ to the root of the tree.

Consider for instance the following extended program $\mathbb{P}+$ with mutual recursion and query $?p$, on the unit interval of real numbers $([0,1], \leq)$ (where the labels `P`, `G` and `L` stand for *Product*, *Gödel* and *Łukasiewicz* connectives):

$\mathcal{R}_1 : \langle\ p\ \leftarrow_{\mathtt{P}}\ q\ ; 0.6\ ; 0.6\ \rangle$
$\mathcal{R}_2 : \langle\ p\ \leftarrow_{\mathtt{P}}\ r\ ; 0.5\ ; 0.5\ \rangle$
$\mathcal{R}_3 : \langle\ q\ \leftarrow\ \ ; 0.9\ ; 0.9\ \rangle$
$\mathcal{R}_4 : \langle\ r\ \leftarrow\ \ ; 0.8\ ; 0.8\ \rangle$
$\mathcal{R}_5 : \langle\ r\ \leftarrow_{\mathtt{L}}\ p\ ; 0.9\ ; 0.9\ \rangle$

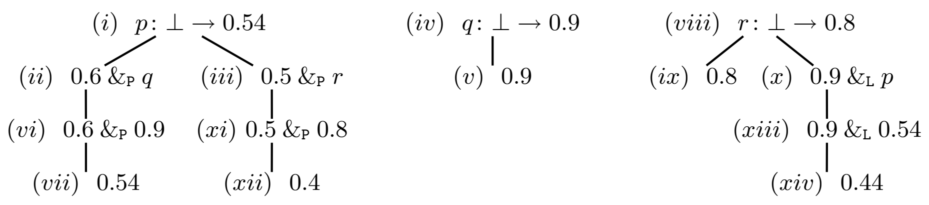

Fig. 1. Example forest for query $?p$.

Firstly, the initial tree consisting of nodes $(i), (ii), (iii)$ is generated, see Figure 1. Then *New Subgoal* is applied on q, a new tree is generated with nodes (iv) and (v), and its current value is directly updated to 0.9.

By using this value, *Answer Return* extends the initial tree with node (vi). Now *Value Update* generates node (vii) and updates the current value of p to 0.54.

Then, *New Subgoal* is applied on r, and a new tree is generated with nodes $(viii), (ix)$ and (x). *Value Update* increases the current value to 0.8.

By using this value, *Answer Return* extends the initial tree with node (xi). Now *Value Update* generates node (xii). The current value is not updated since its value is greater than the newly computed one.

Finally, *Answer Return* can be applied again on propositional symbol p in node (x), generating node $(xiii)$. A further application of *Value Update* generates node (xiv) and the forest is terminated, as no rule performs any modification.

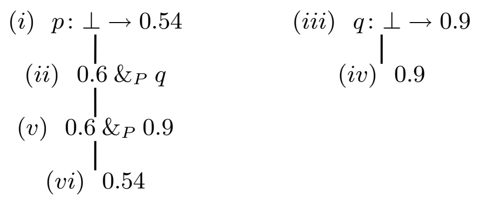

Fig. 2. Example threshold forest for p

In order to illustrate the advantages of our improved method, consider that in our extended program we replace the second program rule by:

$$\mathcal{R}'_2 : \langle p \ \leftarrow_{\mathrm{P}} \ (r \&_{\mathrm{P}} 0.9) \ \ ; \ \ 0.55; \ \ 0.495 \ \ \rangle$$

It is important to note now that even when the truth degree of the rule is 0.55, its threshold decreases to $Up\_rule(\mathcal{R}'_2) = 0.55 * 0.9 = 0.495 < 0.54$, which avoids extra expansions of the tree as Figure 2 shows.

As revealed in the previous examples, the presence of truth degrees on the body of program rules, is always desirable for optimizing the power of thresholding at tabulation time. In [7], we show that it is possible to transform a program rule into a semantically equivalent set of rules with the intended shape. The key point is the use of classical unfolding techniques initially described for crisp (i.e. not fuzzy) settings in [3, 14, 1]), in order to optimize programs. The underlying idea is to "apply computational steps" on program rules, whose benefits remain compiled in their bodies. Now, we can give a new practical taste to these techniques in order to empower the benefits of thresholding when executing fuzzy programs in a tabulated way. For instance, given a MALP program like:

$$
\begin{array}{llllllll}
\mathcal{R}_1: & \langle & p & \leftarrow & & ; 0.4 & \rangle \\
\mathcal{R}_2: & \langle & p & \leftarrow_{\mathtt{P}} & q_1 & ; 0.9 & \rangle \\
\mathcal{R}_3: & \langle & q_1 & \leftarrow_{\mathtt{P}} & q_2 & ; 0.9 & \rangle \\
\mathcal{R}_4: & \langle & q_2 & \leftarrow_{\mathtt{P}} & q_3 & ; 0.9 & \rangle \\
\mathcal{R}_5: & \langle & q_3 & \leftarrow_{\mathtt{P}} & q_4 & ; 0.9 & \rangle \\
 & & \vdots & & & &
\end{array}
$$

... for which the tabulation procedure would never end if the program be infinite (regarding goal p), the simple application of 9 unfolding steps on the second rule could produce the following extended program:

$$
\begin{array}{llllllll}
\mathcal{R}'_1: & \langle & p & \leftarrow & & ; 0.4 & ; 0.4 & \rangle \\
\mathcal{R}'_2: & \langle & p & \leftarrow_{\mathtt{P}} & 0.9 \&_{\mathrm{p}} 0.9 \ldots \&_{\mathrm{p}} 0.9 \ \&_{\mathrm{p}} \ q_{10} & ; 0.9 & ; 0.3874204890 & \rangle \\
 & & \vdots & & & & &
\end{array}
$$

The reader may easily check that following our improved thresholded tabulation technique, the unique solution (0.4) for our initial query (p) could be easily found by simply applying a very few number of computation steps.

Another interesting example where the powerful of the static preprocess is showed arise if we consider a program with the following rules:

$$
\begin{array}{lllllll}
\mathcal{R}_1: & \langle & p & \leftarrow & & ; 0.4 & \rangle \\
\mathcal{R}_2: & \langle & p & \leftarrow_{\mathtt{L}} & (r \&_{\mathtt{L}} \ 0.5) & ; 0.8 & \rangle \\
\mathcal{R}_3: & \langle & p & \leftarrow_{\mathtt{L}} & (r \&_{\mathtt{G}} \ 0.8) & ; 0.6 & \rangle \\
\mathcal{R}_4: & \langle & p & \leftarrow_{\mathtt{P}} & (r \&_{\mathtt{G}} \ 0.7) & ; 0.6 & \rangle \\
\mathcal{R}_5: & \langle & p & \leftarrow_{\mathtt{P}} & (r \&_{\mathtt{P}} \ 0.9) & ; 0.55 & \rangle
\end{array}
$$

If the extended program is not assumed, then, surely, all the rules will be considered in the tabulation procedure. However, if we calculate the proper threshold, we can reorder the rules in order to improve the efficiency of the procedure.

$$
\begin{array}{llllllll}
\mathcal{R}_5: & \langle & p & \leftarrow_{\mathtt{P}} & (r \&_{\mathtt{P}} \ 0.9) & ; 0.55; & 0.495 & \rangle \\
\mathcal{R}_4: & \langle & p & \leftarrow_{\mathtt{P}} & (r \&_{\mathtt{G}} \ 0.7) & ; 0.6; & 0.42 & \rangle \\
\mathcal{R}_1: & \langle & p & \leftarrow & & ; 0.4; & 0.4 & \rangle \\
\mathcal{R}_3: & \langle & p & \leftarrow_{\mathtt{L}} & (r \&_{\mathtt{G}} \ 0.8) & ; 0.6; & 0.4 & \rangle \\
\mathcal{R}_2: & \langle & p & \leftarrow_{\mathtt{L}} & (r \&_{\mathtt{L}} \ 0.5) & ; 0.8; & 0.3 & \rangle
\end{array}
$$

In this last case, if the fact $\mathcal{R}_6 : \langle r \leftarrow \ ; 1 \rangle$ is considered, only one of the five rules will be applied in the thresholded tabulation procedure.

4 Conclusions and Future Work

In this paper, we were concerned with same static improvements that can be easily achieved on the thresholded tabulation procedure we have recently designed in [6] for the fast execution of MALP programs. Before lifting our results to the first order case and implementing it into our FLOPER platform [12], for the near future we plan to formally define the unfolding process of the method drawn here, providing stopping criteria and guides for applying the unfolding operation to program rules in a satisfiable way.

References

1. M. Alpuente, M. Falaschi, G. Moreno, and G. Vidal. Rules + Strategies for Transforming Lazy Functional Logic Programs. *Theoretical Computer Science, Elsevier*, 311(1-3):479–525, Jan. 2004.
2. J. F. Baldwin, T. P. Martin, and B. W. Pilsworth. *Fril- Fuzzy and Evidential Reasoning in Artificial Intelligence*. John Wiley & Sons, Inc., 1995.
3. R.M. Burstall and J. Darlington. A Transformation System for Developing Recursive Programs. *Journal of the ACM*, 24(1):44–67, 1977.
4. C.V. Damásio, J. Medina, and M. Ojeda-Aciego. A tabulation proof procedure for residuated logic programming. *In Proc. of the European Conf. on Artificial Intelligence, Frontiers in Artificial Intelligence and Applications*, 110:808–812, 2004.
5. M. Ishizuka and N. Kanai. Prolog-ELF Incorporating Fuzzy Logic. In Aravind K. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85)*, pages 701–703. Morgan Kaufmann, 1985.
6. P. Julián, J. Medina, G. Moreno, and M. Ojeda. Efficient thresholded tabulation for fuzzy query answering. *Studies in Fuzziness and Soft Computing (Foundations of Reasoning under Uncertainty)*, 249:125–141, 2010.
7. P. Julián, G. Moreno, and J. Penabad. On Fuzzy Unfolding. A Multi-adjoint Approach. *Fuzzy Sets and Systems, Elsevier*, 154:16–33, 2005.
8. P. Julián, G. Moreno, and J. Penabad. Efficient reductants calculi using partial evaluation techniques with thresholding. *Electronic Notes in Theoretical Computer Science, Elsevier Science*, 188:77–90, 2007.
9. M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
10. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. *Proc of Logic Programming and Non-Monotonic Reasoning, LPNMR'01, Springer-Verlag, Lecture Notes in Artificial Intelligence*, 2173:351–364, 2001.
11. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Similarity-based Unification: a multi-adjoint approach. *Fuzzy Sets and Systems*, 146:43–62, 2004.
12. P. J. Morcillo and G. Moreno. Programming with fuzzy logic rules by using the FLOPER tool. In *Proc. of 2nd. Intl. Symposium on Rule Interchange and Applications (RuleML'08)*, volume 5321 of *LNCS*, pages 119–126. Springer, 2008.
13. T. Swift. Tabling for non-monotonic programming. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):201–240, 1999.
14. H. Tamaki and T. Sato. Unfold/Fold Transformations of Logic Programs. In S. Tärnlund, editor, *Proc. of Second Int'l Conf. on Logic Programming*, pages 127–139, 1984.