Towards the Use of Hypergraphs in Multi-adjoint Logic Programming

Juan Carlos Díaz-Moreno, Jesús Medina and José R. Portillo

Abstract The representation of a logic program by a graph is a useful procedure in order to obtain interesting properties of the program and in the computation of the least model, when it exists. In this paper, we consider hypergraphs for representing multi-adjoint logic programs and, based on this representation, the hypotheses of an interesting termination result have been weakened.

1 Introduction

One of the most important problems in logic programming with non-decreasing operators is the computation of the least model of a given program. In order to obtain such a model the fix-point semantics is usually considered. This semantics is based on the iteration of the immediate consequence operator from the least interpretation. Since this iteration can be infinite, one important goal is to get termination properties of this iteration. In [3, 4], different termination theorems were introduced in the multi-adjoint logic programming. This logic programming framework was introduced in [10] as a generalization of different non-classical logic programming frameworks, such as the residuated logic programming [5] and the fuzzy logic programming framework presented in [11].

This paper considers directed hypergraphs [1, 8] in order to represent a multi-adjoint logic program and, based on this representation, introduce a termination result, which generalizes one of the most important termination theorems given in [4].

J. C. Díaz-Moreno · J. Medina
Department of Mathematics, University of Cádiz, Cádiz, Spain
e-mail: juancarlos.diaz@uca.es

J. Medina
e-mail: jesus.medina@uca.es

J. R. Portillo (✉)
Department of Applied Mathematics I, University of Sevilla, Sevilla, Spain
e-mail: josera@us.es

2 Basic Definitions on Hypergraphs

This section recalls the notions we will need throughout the paper related to hypergraphs. For basic notions of graph theory see [2].

A *graph* is a pair of sets (V, E). V is the set of *vertices* or *nodes*. E is a set of 2-element subsets of V, named *edges*. The edges may be directed or undirected (the pairs are ordered or not). Directed edges are called *arcs*. A *cycle* in a graph is a path of edges and vertices wherein a vertex is reachable from itself. I.e., a ordered set of vertices $\{u_1, \dots, u_i, \dots u_p\}$ such that $u_i u_{i+1}$ is an edge of the graph and $u_1 = u_p$.

The first notion for *hypergraph*s is the definition, which is a generalization of a graph in which an edge is a non-empty subset of vertices. Specifically, a hypergraph $\mathcal{H}$ is a pair $\mathcal{H} = (V, E)$ where V is a set of elements called *nodes* or *vertices*, and E is a set of non-empty subsets of V called *hyperedges* or edges, see [2] for more details. Therefore, E is a subset of $\mathcal{P}(V) \setminus \{\emptyset\}$, where $\mathcal{P}(V)$ is the power set of V. Note that, when the cardinal of all hyperedges is 2, the hypergraph is a standard graph.

The generalization of a directed graph is called directed hypergraph and contains directed hyperedges. A *directed hyperedge* or *hyperarc* is an ordered pair, $e = (X, Y)$, of (possibly empty) disjoint subsets of vertices; X is called the *tail* of e and Y is its *head*. From now on, the tail and the head of an hyperarc e will be denoted by $T(e)$ and $H(e)$, respectively.

Hence, a directed hypergraph is a hypergraph with directed hyperedges [1, 8]. A *backward hyperarc*, or simply *B-arc*, is a hyperarc $e = (T(e), H(e))$, where the head exactly has one vertex. When all the hyperarcs of a hypergraph are B-arcs, then the hypergraph is called *B-graph* (or *B-hypergraph*) [8]. For example, the hypergraph $\mathcal{H} = (V, E)$ introduced on the left of Fig. 1, where $V = \{a, b, c, d\}$ and $E = \{(\{a\}, \{b\}), (\{b, c\}, \{a\}), (\{c, d\}, \{a\})\}$ is a B-graph. This paper will only consider this kind of hypergraphs.

B-graphs (and the dually defined F-graphs) are a useful tool in different applications [1, 8, 9]. As a consequence, they have been introduced many times in the literature with various names. For example, the labelled graphs, used in [6, 7] to represent Horn formulae, are B-graphs.

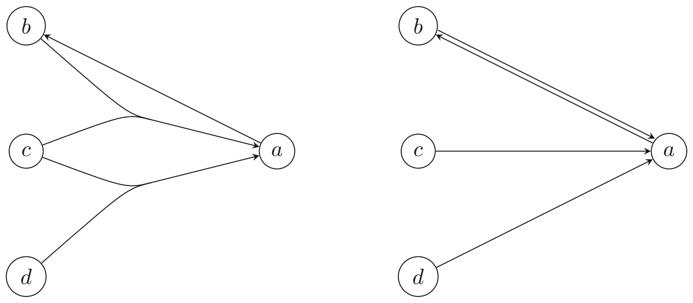

Fig. 1 Left: Example of B-graph $\mathcal{H} = (V, E)$. Right: Directed graph subjacent to $\mathcal{H}$

In contrast with ordinary graphs for which there is a single natural notion of cycles and acyclic graphs, there are multiple natural non-equivalent definitions of acyclicity for hypergraphs which collapse to ordinary graph acyclicity for the special case of ordinary graphs.

In this paper, we only need to consider the cycles on the subjacent directed graph of a B-graph. Given any directed hypergraph $\mathcal{H} = (V, E)$, the subjacent directed graph $G(\mathcal{H}) = (V(G), E(G))$ has the same nodes that $\mathcal{H}$, i.e. $V(G) = V$ and an arc exists in $E(G)$ from the node u to the node v if and only if it exists a hyperedge $e \in E$ such that $u \in T(e)$ and $v \in H(e)$. For example, Fig. 1 shows on the right the subjacent graph to the hypergraph on the left. The vertices a and b form a cycle in that graph, but the hypergraph has not hypercycles under the common definitions [2].

3 Multi-adjoint Logic Programming

This section recalls the algebraic structure considered in this framework, the notion of multi-adjoint logic program, and one of the most interesting termination theorems introduced in [4]. The basic operators considered in this framework are adjoint pairs.

Definition 1 Given a partially ordered set $(P, \leq)$, the pair $(\&, \leftarrow)$ is an *adjoint pair* with respect to $(P, \leq)$ if the mappings $\&, \leftarrow : P \times P \to P$ satisfy that:

1. $\&$ is order-preserving in both arguments.
2. $\leftarrow$ is order-preserving in the first argument (the consequent) and order-reversing in the second argument (the antecedent).
3. The equivalence $x \leq y \leftarrow z$ if and only if $x \& z \leq y$ holds, for all $x, y, z \in P$.

The algebraic structure considered in this logic programming framework is called multi-adjoint lattice.

Definition 2 A *multi-adjoint lattice* is a tuple $(L, \preceq, \leftarrow_1, \&_1, \dots, \leftarrow_n, \&_n)$ verifying the following properties:

1. $(L, \preceq)$ is bounded lattice, i.e. it has bottom $(\bot)$ and top $(\top)$ elements;
2. $(\&_i, \leftarrow_i)$ is an adjoint pair in $(L, \preceq)$, for all $i \in \{1, \dots, n\}$;
3. $\top \&_i \vartheta = \vartheta \&_i \top = \vartheta$, for all $\vartheta \in L$ and for all $i \in \{1, \dots, n\}$.

Given a multi-adjoint lattice, a set of propositional symbols Π, a given language denoted as $\mathfrak{F}$ and different monotonic operators defined on L, the notion of program (set of rules) is introduced in this framework.

Definition 3 Given a multi-adjoint lattice $(L, \preceq, \leftarrow_1, \&_1, \dots, \leftarrow_n, \&_n)$. A *multi-adjoint logic program* $\mathbb{P}$ is a set of rules of the form $\langle (A \leftarrow_i \mathcal{B}), \vartheta \rangle$ such that:

1. The *rule* $(A \leftarrow_i \mathcal{B})$ is a formula of $\mathfrak{F}$;
2. The *confidence factor* ϑ is an element (a truth-value) of L;
3. The *head* of the rule A is a propositional symbol of Π.
4. The *body* formula $\mathcal{B}$ is a formula of $\mathfrak{F}$ built from propositional symbols $B_1, \dots, B_n$ $(n \geq 0)$ by the use of conjunctors $\&_1, \dots, \&_n$ and $\wedge_1, \dots, \wedge_k$, disjunctors $\vee_1, \dots, \vee_l$, aggregators $@_1, \dots, @_m$ and elements of L.
5. *Facts* are rules with body $\top$.

This paper will be focused on one of the most important theorems introduced in [4]. Before recalling this result we need different definitions.

Definition 4 Let $\mathbb{P}$ be a multi-adjoint program, and $A \in \Pi$. The set $R^I_{\mathbb{P}}(A)$ of *relevant values* for A with respect to an interpretation I is the set of maximal values of the set $\{\vartheta \,\&_i\, \hat{I}(\mathcal{B}) \mid \langle A \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}\}$.

The immediate consequences operator, given by van Emden and Kowalski, is defined in this framework as follows.

Definition 5 Given a multi-adjoint logic program $\mathbb{P}$. The *immediate consequences operator* $T_{\mathbb{P}}$ maps interpretations to interpretations, and for an interpretation I and an arbitrary propositional symbol A is defined by

$$T_{\mathbb{P}}(I)(A) = \sup\{\vartheta \,\&_i\, \hat{I}(\mathcal{B}) \mid \langle A \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}\}$$

The main feature of $T_{\mathbb{P}}$ is that its least fix-point coincides with the least model of the program $\mathbb{P}$ [10]. Since the least fix-point is computed iterating the $T_{\mathbb{P}}$ operator from the least interpretation, Δ, it is important to know when this iteration finishes in a finite number of steps.

The termination theorem in [4] was introduced for sorted and local multi-adjoint logic programs. In order to simplify the notation, we have adapted it for (uni-sorted) multi-adjoint logic programs.

Theorem 1 *Given a multi-adjoint logic program $\mathbb{P}$ with finite dependences and where the operators $@ \colon L^m \to L$ in the body of the rules satisfy the boundary condition with the $\top$ element, that is,*

$$@(\underbrace{\top, \dots, \top}_{k}, x, \underbrace{\top, \dots, \top}_{m-k-1}) \preceq x$$

for all $x \in L$. If for every iteration n and propositional symbol A the set of relevant values for A with respect to $T^n_{\mathbb{P}}(\triangle)$ is a singleton, then $T_{\mathbb{P}}$ terminates for every query.

This result will be weakened in the following section.

4 Representing Programs by Hypergraphs

This section presents a simple example which does not satisfy the hypotheses of Theorem 1, but the least model of the program is obtained after finitely many iterations. Then, in order to extend this result to a bigger number of programs, a straightforward mechanism for representing a logic program by a B-graph is introduced. Finally, based on this representation, the hypotheses of Theorem 1 will be weakened.

Example 1 Consider the following program $\mathbb{P}$:

$$\begin{array}{ll} \langle a \leftarrow_P b \,\&_G\, c, 0.8\rangle & \langle b \leftarrow_P a, 0.7\rangle \\ \langle a \leftarrow_P @(d, c), 1.0\rangle & \langle c \leftarrow_P 1.0, 1.0\rangle \\ & \langle a \leftarrow_P 1.0, 0.5\rangle \end{array}$$

where the aggregator $@\colon [0, 1] \times [0, 1] \to [0, 1]$ is the weighted sum defined as $@(x, y) = (x + 3y)/4$, for all $x, y \in [0, 1]$.

Although the minimum operator $\&_G$ satisfies the boundary condition with the 1 element (hypothesis in Theorem 1), the aggregator @ does not verify it and so, we cannot apply this theorem in order to know whether the computation of the least model terminates in a finite number of iterations. However, in this case, only 3 iterations are needed, as we show below:

	a	b	c	d
$T_{\mathbb{P}}{}^0 =$	0.0	0.0	0.0	0.0
$T_{\mathbb{P}}{}^1 =$	0.75	0.0	1.0	0.0
$T_{\mathbb{P}}{}^2 =$	0.75	0.525	1.0	0.0
$T_{\mathbb{P}}{}^3 =$	0.75	0.525	1.0	0.0

This example shows that the hypotheses in Theorem 1 should be weakened. For that, we will represent a program by a B-graph and we will relate the termination of the iterations to the existence of cycles in the subjacent directed graph and whether aggregator operators, which do not satisfy the hypotheses of the theorem, are involved in these cycles.

Note that, it has been possible to compute the least model in a finite number of iterations because the aggregator operator @ is not in a cycle of the subjacent digraph of the associated B-graph.

The associated B-graph $\mathcal{H}_{\mathbb{P}}$ associated with a program $\mathbb{P}$ is constructed as follows: the vertex set of the hypergraph is the propositional symbol set Π of the program. Hence, for the program $\mathbb{P}$ in Example 1, we have $V(\mathcal{H}_{\mathbb{P}}) = \{a, b, c, d\}$. One hyperarc will be obtained from each rule as follows: Given a rule, the propositional symbols of the antecedent of the rule will be the tail $T(e)$ of the associated hyperarc and the propositional symbol of the head of the rule will be the only element of the head $H(e)$ of the associated hyperarc. This hyperarc is labeled with the aggregator in the body of the rule. When no aggregator operator appears in the body of the rule, we will

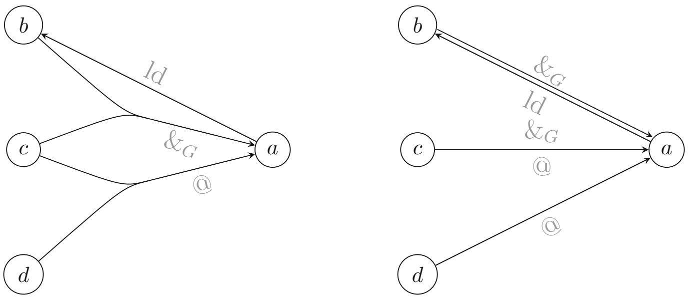

Fig. 2 Left: (Labelled) B-graph associated with the program given in Example 1. Right: (Labelled) Subjacent directed graph from the B-graph on the left

consider the identity mapping. For example, from the rule $\langle a \leftarrow_P b \,\&_G\, c, 0.8\rangle$ for the program $\mathbb{P}$ in Example 1, we obtain the hyperarc $(\{b, c\}, \{a\})$ with the label $\&_G$. Due to the considered mechanism, the hypergraph resultant is always a (labelled) B-graph. Figure 2 (left) shows the associated B-graph of the program $\mathbb{P}$ given in Example 1.

Finally, a weak version of Theorem 1 is introduced:

Theorem 2 *Given a multi-adjoint logic program $\mathbb{P}$ with finite dependences and the B-graph $\mathcal{H}_\mathbb{P}$ associated with $\mathbb{P}$. If the operators involved in the cycles of the subjacent directed graph of $\mathcal{H}_\mathbb{P}$ satisfy the boundary condition with the $\top$ element and, for every iteration n and propositional symbol A, the set of relevant values for A with respect to $T_\mathbb{P}^n(\triangle)$ is a singleton, then $T_\mathbb{P}$ terminates for every query.*

Note that this result only needs that the aggregators operators involved in the cycles of the subjacent directed graph of the B-graph associated with the program satisfy the boundary condition. Therefore, this result is notably more general than Theorem 1 and can be applied, for example, to the program given in Example 1.

5 Conclusions and Future Work

This paper has presented a procedure in order to represent a multi-adjoint logic program as a hypergraph. As a first consequence, we have generalized one of the most important termination results introduced in [4]. This representation will provide more interesting properties in the future. We will study other efficient termination results and we will analyze analogies between different notions in logic programming and in graph theory in order to create synergies between both theories. The obtained results will also be compared with the existent ones in the literature.

References

1. Ausiello, G., Laura, L.: Directed hypergraphs: introduction and fundamental algorithms—a survey. Theor. Comput. Sci. **658**, 293–306 (2017)
2. Berge, C.: Graphs and Hypergraphs. Elsevier Science Ltd. (1985)
3. Damásio, C., Medina, J., Ojeda-Aciego, M.: Sorted multi-adjoint logic programs: termination results and applications. In: Lecture Notes in Artificial Intelligence, vol. 3229, pp. 252–265 (2004)
4. Damásio, C., Medina, J., Ojeda-Aciego, M.: Termination of logic programs with imperfect information: applications and query procedure. J. Appl. Log. **5**, 435–458 (2007)
5. Damásio, C.V., Pereira, L.M.: Monotonic and residuated logic programs. In: Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2001. Lecture Notes in Artificial Intelligence, vol. 2143, pp. 748–759 (2001)
6. Dowling, W.F., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. J. Log. Program. **1**(3), 267–284 (1984)
7. Gallo, G., Urbani, G.: Algorithms for testing the satisfiability of propositional formulae. J. Log. Program. **7**(1), 45–61 (1989)
8. Gallo, G., Longo, G., Pallottino, S., Nguyen, S.: Directed hypergraphs and applications. Discrete Appl. Math. **42**(2–3), 177–201 (1993)
9. Jeroslow, R.G., Martin, R.K., Rardin, R.L., Wang, J.: Gainfree Leontief substitution flow problems. Math. Program. **57**, 375–414 (1992)
10. Medina, J., Ojeda-Aciego, M., Vojtáš, P.: Multi-adjoint logic programming with continuous semantics. In: Lecture Notes in Artificial Intelligence, vol. 2173, pp. 351–364 (2001)
11. Vojtáš, P.: Fuzzy logic programming. Fuzzy Sets Syst. **124**(3), 361–370 (2001)