Available online at www.sciencedirect.com

ScienceDirect

Fuzzy Sets and Systems ••• (••••) •••–•••

FUZZY sets and systems

www.elsevier.com/locate/fss

Fuzzy logic programs as hypergraphs. Termination results ☆

Juan Carlos Díaz-Moreno [a], Jesús Medina [a,*], José R. Portillo [b]

[a] *Department of Mathematics, University of Cádiz, Spain*

[b] *Departamento de Matemática Aplicada I and Instituto Universitario de Investigación de Matemáticas de la Universidad de Sevilla (IMUS), Universidad de Sevilla, Spain*

Received 8 February 2021; received in revised form 19 January 2022; accepted 1 February 2022

Abstract

Graph theory has been a useful tool for logic programming in many aspects. In this paper, we propose an equivalent representation of multi-adjoint logic programs using hypergraphs, which are a generalization of classical graphs that allows the use of hypergraph theory in logic programming. Specifically, this representation has been considered in this paper to increase the level and flexibility of different termination results of the computation of the least model of fuzzy logic programs via the immediate consequence operator. Consequently, the least model of more general and versatile fuzzy logic programs can be obtained after finitely many iterations, although infinite programs or programs with loops and general aggregators will be considered.

Keywords: Logic programming; Fuzzy sets; Termination; Hypergraphs

1. Introduction

Logic programming [45] is an important framework to obtain information from a dataset, which has been modeled by a set of logic rules that are called either 'logic program' or simply 'program.' Therefore, logic programming is a core part of decision and recommendation systems, among others.

Many researchers have studied logic programming from a theoretical point of view and it has been applied to many frameworks [8,9,12,13,15,35,40,43,55]. Several fuzzy extensions have been introduced to handle imprecise, incomplete or imperfect data [21,36,44,23,48–50]. Multi-adjoint logic programming arose as a general logic-programming framework, which embeds residuated and monotonic logic programming, fuzzy logic programming, probabilistic logic programming, and so on. The main idea behind this general framework is to consider the most general mathematical setting from which the most useful results can be proved. Consequently, the semantics is based on a complete

☆ Partially supported by the 2014-2020 ERDF Operational Programme in collaboration with the State Research Agency (AEI) in project PID2019-108991GB-I00, and with the Department of Economy, Knowledge, Business and University of the Regional Government of Andalusia in project FEDER-UCA18-108612, and by the European Cooperation in Science & Technology (COST) Action CA17124.

* Corresponding author.
E-mail address: jesus.medina@uca.es (J. Medina).

https://doi.org/10.1016/j.fss.2022.02.001

lattice in which (for example) the operators can be neither commutative nor associative, and different implications can be taken into account in the same logic program. These general structures allow the consideration of a more flexible and suitable set of rules to better simulate the behavior of a given knowledge system.

An important issue in multi-adjoint logic programming is to compute the consequences (least model) that are associated with the dataset modeled by the program (computed set of rules); that is, the values from which we can offer predictions and recommendations to the user. These values can be obtained in this and in many other frameworks by the immediate consequence operator $T_{\mathbb{P}}$ for a program $\mathbb{P}$ [27,59]. Specifically, the least model is obtained by iterating the $T_{\mathbb{P}}$ from the least interpretation (which associates the bottom of the lattice with every atom) until a fixed point is obtained. We must take into account that in many cases it is difficult to know whether the computation of the least fixed point of $T_{\mathbb{P}}$ finishes for every query (e.g., when the considered program has infinite rules and/or has loops). Therefore, an important goal in these (fuzzy) logic programs is to know a priori when the computation of the least model will terminate in a finite number of iterations; that is, if $T_{\mathbb{P}}$ terminates, which has been studied in many papers [18,22,25,44,23,52]. In [20], the authors introduced several interesting termination theorems that are based on the notion of dependency graph, together with useful applications to probabilistic programs, such as ordinary probabilistic logic programs, probabilistic deductive databases, and hybrid probabilistic logic programs.

This paper considers the theory of hypergraphs [10] to provide an efficient representation of a given logic program and to increase the level and flexibility of the termination theorems given in [20]. In particular, we use directed hypergraphs [6,30], which appear in a lot of different contexts, such as propositional logic [5,24,25,31,32,53], artificial intelligence [51], technological processing of product assembly [47], relational databases [7,62], probabilistic parsing [42], chemical reaction mechanisms [63], operations research [30], transportation planning [54], Petri nets [1], and so on. Alongside this work, we will focus on a special kind of directed hypergraphs, namely B-graphs, which have been used as a tool to analyze deductive databases (see [3,6,30,61]) and to solve Horn formulae [11,26,56,57]. B-graphs are also used to study Leontiev substitution matrices and flow problems [34], and to plan urban transportation [54].

Several relationships among graphs, hypergraphs and logic programs have been studied. Consequently, it has been possible to translate the information given in a program to a hypergraph, and vice versa. In fact, the proposed computation of a labeled B-graph from a program keeps all of the information provided by the program, while the dependency graph, even when labeled, loses some information. Obviously, this loss can be avoided by relabeling the edges of the dependency graph. However, in that case the obtained structure is equivalent to a hypergraph but represented in a way that is more complex and computationally less manageable than this one. Hence, hypergraphs provide a simpler and more tractable representation of logic programs (i.e., there is only one labeled hyperarc for each rule instead of multiple edges with the same label). This representation facilitates visualizing, and gives additional insight into, the structure of multi-adjoint logic programs. This allows us to demonstrate new results and open up promising new lines of investigation.

The first termination theorem is proved based on the obtained hypergraph and other intermediate results. A variant of this hypergraph has been analyzed to prove the second termination theorem, which significantly increases the number of operators that can be used in the program. Both of these theorems generalize the main results presented in [20] and provide a broader range of applications, from which we can know in advance if the computation of the least model terminates in a finite number of iterations. Note that it is not possible to directly apply the result in [20] only considering dependency graphs, they must be adapted and extra results are required. However, the underlying mathematical proofs are similar to those proved in this paper. Hence, hypergraphs have been considered not only because they improve the representation of the program but they also provide a better understanding of the hypotheses and (aesthetically) simplification of different parts of the proofs.

The rest of this paper is structured as follows. Section 2 includes the preliminary notions to be used throughout the paper. Section 3 presents the results relating to graphs and hypergraphs, and the representation of programs by hypergraphs, which have been applied to the termination results given in Section 4. In the final section, this paper draws the conclusions and looks to the prospects for future work.

2. Preliminaries

The main notions in hypergraph theory and multi-adjoint logic programming considered in this paper are recalled next.

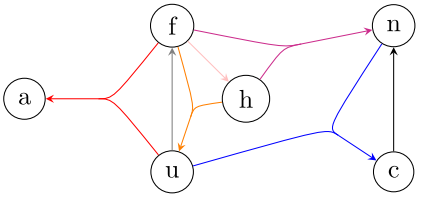

Fig. 1. Example of B-graph $\mathcal{H} = (V, E)$.

2.1. Basic definitions of hypergraphs

This section recalls the notions related to hypergraphs that we will need throughout this paper. For the basic notions of (hyper)graph theory, see [10].

A graph is a pair of sets (V, E), where V is the set of *vertices* or *nodes* and E is a subset of unordered pairs of vertices, which are called *edges*. A hypergraph is a generalization of the notion of a graph in which an edge can join any number of vertices. The generalization is given through the notion of hyperedge. Specifically, a hypergraph is defined as a pair of sets (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 [10] for more details). Note that a graph is a kind of hypergraph verifying that the cardinal of all hyperedges is two.

A *directed graph* or *digraph* is a graph whose edges are ordered pairs of vertices. The corresponding generalization of a directed graph is called a *directed hypergraph* [6,30]. Formally, a *directed hypergraph* is a pair of sets (V, E), where the elements of E are called *directed hyperedges* or *hyperarcs* and each of them is an ordered pair, $e = (X, Y)$, of disjoint subsets of vertices. We denote X as the *tail* of e and Y as its *head*. Hereinafter, the tail and the head of a hyperarc e will be denoted by $T(e)$ and $H(e)$, respectively. Hence, a directed hypergraph is a hypergraph with directed hyperedges. In the following, when no confusion arises, directed hypergraphs will simply be called hypergraphs.

Given a graph $G = (V, E)$, a *vertex labeling* is a function of V to a set of labels. A *vertex-labeled* graph is a graph with one or more associated vertex labeling functions. Likewise, an *edge labeling* is a function from E to a set of labels and we say that a graph with an edge labeling is an *edge-labeled graph*. The definition of labeled graphs is extended to digraphs and (directed) hypergraphs, in a natural way.

A *backward hyperarc*, or simply *B-arc*, is a hyperarc e, where the head $H(e)$ has exactly one vertex; that is, it is a singleton. In a similar way, *forward hyperarc*, or simply *F-arc*, is a hyperarc f where the tail $T(e)$ exactly has one vertex. When all of the hyperarcs of a hypergraph are B-arcs (resp. F-arcs), then the hypergraph is called *B-graph* (resp. *F-graph*) [30]. Note that a digraph is simultaneously a particular case of a B-graph and an F-graph. For example, the hypergraph $\mathcal{H} = (V, E)$ introduced in Fig. 1, where $V = \{\mathrm{a}, \mathrm{c}, \mathrm{f}, \mathrm{h}, \mathrm{n}, \mathrm{u}\}$ and $E = \{(\{\mathrm{c}\}, \{\mathrm{n}\}), (\{\mathrm{f}\}, \{\mathrm{h}\}), (\{\mathrm{f}, \mathrm{h}\}, \{\mathrm{n}\}), (\{\mathrm{f}, \mathrm{h}\}, \{\mathrm{u}\}), (\{\mathrm{f}, \mathrm{u}\}, \{\mathrm{a}\}), (\{\mathrm{n}, \mathrm{u}\}, \{\mathrm{c}\}), (\{\mathrm{u}\}, \{\mathrm{f}\})\}$ is a B-graph with six vertices and seven B-arcs. This paper will only consider this kind of directed hypergraph. Because hyperarcs provide a natural representation of Horn formulae and databases, B-graphs and F-graphs are useful for many applications [5,6,30–32,34,61]. Indeed, they have been presented many times with different names in the literature. For example, the B-graphs are called "labeled graphs" (not to be confused with the two labeled graphs that were defined previously) in [26,32,57] to represent the Horn formulae. Torres and Araoz [61] studied hypergraphs and B-graphs, called "rule hypergraphs", to represent deduction properties in databases as paths in hypergraphs. Ausellio and Italiano [5] also used this kind of directed hypergraph, although they called them "FD-graphs", for construct polynomial algorithms to solve satisfiability problems. In this work, we will use labeled B-arcs as a natural representation of rules in a multi-adjoint logic program.

The notion of a subgraph is extended to directed hypergraphs in [60].

Definition 1. Given a directed hypergraph $\mathcal{H} = (V, E)$, we say that $\mathcal{H}' = (V', E')$ is a *subhypergraph* of $\mathcal{H}$ if V' is a subset of V and E' is a subset of $E \cap (2^{V'} \times 2^{V'})$.

The definition of *hypergraph induced by the set of vertices* V' is adapted from [58] to directed hypergraphs and it is the subhypergraph $\mathcal{H}' = (V', E')$, where $E' = E \cap (2^{V'} \times 2^{V'})$.

The following definition has been adapted from [30,60].

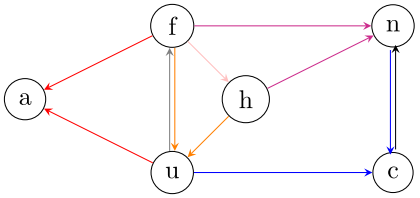

Fig. 2. Directed graph subjacent to the directed hypergraph in Fig. 1.

Definition 2. Given a directed hypergraph $\mathcal{H} = (V, E)$, two vertices $s, t \in V$, and a natural number q, a *path* in $\mathcal{H}$ from s to t of length q, denoted as P_{st}, is a sequence of nodes and hyperarcs: $P_{st} = \langle v_1, E_1, v_2, E_2, \ldots, v_q, E_q, v_{q+1}\rangle$ where: $v_1 = s$, $v_{q+1} = t$, $s \in T(E_1)$, $t \in H(E_q)$, and $v_i \in H(E_{i-1}) \cap T(E_i)$, for all $i \in \{2, ..., q\}$.

The trivial sequence $\langle v\rangle$, with $v \in V$, will be a path of length 0.

Nodes s and t are the *origin* (*source*) and the *destination* (*sink*) of P_{st}, respectively, and we say that t *is connected* to s or that t *is weakly reachable* from s [29]. 'Weakly' is used because some applications of directed hypergraphs (e.g., assembly or databases) need a different and strong notion of reachability [7]. If $t \in T(E_1)$, then P_{st} is said to be a *cycle* [30]. In particular, this is true when $t = s$. Note that if there is a cycle from s to t, then there is also a cycle from t to t.

There are different definitions for *paths* in directed hyperarcs in the literature [2–4,6,10,30,60]. We have considered the definition introduced in [30] and [60], which is better suited to the aims of this paper and is less restricted than the definitions given in other papers.

Given that we are considering B-graphs, a path $P_{st} = \langle s, E_1, v_2, E_2, \ldots, E_q, t\rangle$ is univocally determined by its B-arcs and its origin because v_i is the only vertex in the head of E_{i-1}. In this case, we can denote the path as $P_{st} = \langle s, E_1, E_2, \ldots, E_q, t\rangle$. For example, in the B-graph shown in Fig. 1, ⟨f, ({f}, {h}), ({f, h}, {u}), ({u}, {f}), f⟩ and ⟨c, ({c}, {n}), ({n, u}, {c}), c⟩ are cycles.

Every directed hypergraph is associated with a digraph.

Definition 3. Given a directed hypergraph $\mathcal{H} = (V, E)$, the *primal digraph* (or *subjacent digraph*) $D(\mathcal{H}) = (V, A)$ of the hypergraph $\mathcal{H}$ has the same nodes that the hypergraph and an arc exists in A 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. 2 shows the subjacent digraph to the B-graph in Fig. 1. The cycles ⟨f, ({f}, {h}), ({f, h}, {u}), ({u}, {f}), f⟩ and ⟨c, ({c}, {n}), ({n, u}, {c}), c⟩ in the B-graph correspond to the cycles ⟨f, h, u, f⟩ and ⟨c, n, c⟩ in the primal graph.

Note that the correspondence is not injective because two directed hypergraphs can be associated with the same digraph.

2.2. Multi-adjoint logic programming

In this section, we recall the notion of a multi-adjoint logic program and the most interesting termination theorems that were introduced in [20]. We will first present the so-called adjoint pairs. These are the basic operators considered in this framework. They generalize left-continuous t-norms and their residuated implications.

Definition 4. Given a partially ordered set $(P, \leq)$, the pair $(\&, \leftarrow)$ is an *adjoint pair* with respect to $(P, \leq)$ if the mappings $\&, \leftarrow\colon P \times P \to P$ 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 $x \leq z \leftarrow y$ if and only if $x \,\&\, y \leq z$ holds, for all $x, y, z \in P$.

Note that the pairs composed of the product, Gödel and Lukasiewicz t-norms and its corresponding residuated implication, which will be denoted as $(\&_{\mathrm{P}}, \leftarrow_{\mathrm{P}})$, $(\&_{\mathrm{G}}, \leftarrow_{\mathrm{G}})$, $(\&_{Ł}, \leftarrow_{Ł})$, and are defined, for each $x, y, z \in [0,1]$, as

$$x \,\&_{\mathrm{P}}\, y = x \cdot y \qquad z \leftarrow_{\mathrm{P}} y = \begin{cases} 1 & \text{if } y \leq z \\ \dfrac{z}{y} & \text{if } y > z \end{cases}$$

$$x \,\&_{\mathrm{G}}\, y = \min\{x, y\} \qquad z \leftarrow_{\mathrm{G}} y = \begin{cases} 1 & \text{if } y \leq z \\ z & \text{if } y > z \end{cases}$$

$$x \,\&_{Ł}\, y = \max\{0, x + y - 1\} \qquad z \leftarrow_{Ł} y = \min\{1, 1 - y + z\}$$

are adjoint pairs. Clearly, these operators have infinite range (i.e., the image set is infinite). The following examples show particular cases of adjoint pairs, which correspond to operators with finite range.

Example 5. Let $[0,1]_m$ be a regular partition of $[0,1]$ into m pieces; for example, $[0,1]_4 = \{0, 0.25, 0.5, 0, 75, 1\}$ divides the unit interval into four pieces. A discretization of a t-norm $\&\colon [0,1] \times [0,1] \to [0,1]$ is the operator $\&^*\colon [0,1]_m \times [0,1]_n \to [0,1]_k$, where $n, m, k \in \mathbb{N}$, defined for each $x \in [0,1]_m$ and $y \in [0,1]_n$ as:

$$x \,\&^*\, y = \frac{\lceil k \cdot (x \,\&\, y) \rceil}{k}$$

where $\lceil \_ \rceil$ is the ceiling function.

The discretization of the corresponding residuated implication $\leftarrow_*\colon [0,1]_k \times [0,1]_n \to [0,1]_m$ is defined as:

$$z \leftarrow^* y = \frac{\lfloor m \cdot (z \leftarrow y) \rfloor}{m}$$

where $\lfloor \_ \rfloor$ is the floor function and $\leftarrow$ is the residuated implication of the t-norm $\&$. We have that $(\&^*, \leftarrow^*)$ is an adjoint pair in the unit interval [16,48], and $\&^*$, $\leftarrow^*$ have finite range. □

The considered algebraic structure contains this general kind of operators and is called a multi-adjoint lattice.

Definition 6. A *multi-adjoint lattice* is a tuple $(L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$ 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 $i \in \{1, \ldots, \mathrm{n}\}$;
3. $\top \,\&_i\, \vartheta = \vartheta \,\&_i\, \top = \vartheta$, for all $\vartheta \in L$ and $i \in \{1, \ldots, \mathrm{n}\}$.

Multi-adjoint lattices were fundamental in [20, Definition 26] to define the general algebraic structure taken into account in the aforementioned paper, which is called local sorted multi-adjoint Σ-algebra. To simplify the notation, in this paper only one sort will be considered.

Definition 7. A *local multi-adjoint* Σ*-algebra* $\mathfrak{L}$ is a multi-adjoint lattice

$$(L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$$

on which other operator is defined, such as conjunctors $\wedge_1, \ldots, \wedge_k$, disjunctors $\vee_1, \ldots, \vee_l$ and general aggregators $@_1, \ldots, @_h$. The set of those monotonic operators (aggregator operators, in particular) in the Σ-algebra will be denoted as $\mathfrak{A}$; that is,

$$\mathfrak{A} = \{\&_1, \ldots, \&_n, \wedge_1, \ldots, \wedge_k, \vee_1, \ldots, \vee_l, @_1, \ldots, @_h\}$$

and each operator $@\colon L^m \to L$ in $\mathfrak{A}$ satisfies the boundary condition with the top element:

$$@(\underbrace{\top, \ldots, \top}_{s}, x, \underbrace{\top, \ldots, \top}_{m-s-1}) \preceq x \tag{1}$$

for all $x \in L$.

From now on, a local multi-adjoint Σ-algebra $\mathfrak{L}$, a set of propositional symbols Π and a language denoted as $\mathfrak{F}$ will be fixed. From these notions, we can introduced the definition of multi-adjoint logic program (set of rules).

Definition 8. A *multi-adjoint logic program* 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 $B_1, \ldots, B_n$ ($n \geq 0$) by the use of conjunctors $\&_1, \ldots, \&_r$ and $\wedge_1, \ldots, \wedge_k$, disjunctors $\vee_1, \ldots, \vee_l$, aggregators $@_1, \ldots, @_m$ and elements of L.
5. *Facts* are rules with body $\top$.

For a body formula $\mathcal{B}$, as in point 4 of this definition, we can express it by $@[B_1, \ldots, B_n]$, where $B_1, \ldots, B_n$ are all propositional symbols in $\mathcal{B}$ and $@$ the composition of the operators in $\mathcal{B}$. For example, if $\mathcal{B} = (B_1 \,\&\, B_2) \wedge B_3$, then it can be written as $@[B_1, B_2, B_3]$; that is, $@[B_1, B_2, B_3] = (B_1 \,\&\, B_2) \wedge B_3 = \mathcal{B}$. The following example introduces a particular multi-adjoint logic program.

Example 9. We will consider the local multi-adjoint Σ-algebra $\mathfrak{F}$ composed of the unit interval as a complete lattice, the adjoint pairs corresponding to the product, Gödel and Lukasiewicz t-norms, $(\&_{\mathrm{P}}, \leftarrow_{\mathrm{P}})$, $(\&_{\mathrm{G}}, \leftarrow_{\mathrm{G}})$, $(\&_{Ł}, \leftarrow_{Ł})$, and the weighted sums $@_{(3,1)}$ and $@_{(1,2)}$ defined as $@_{(3,1)}(x, y) = \dfrac{3x + y}{4}$ and $@_{(1,2)}(x, y) = \dfrac{x + 2y}{3}$, for every $(x, y) \in [0, 1]^2$. On this Σ-algebra $\mathfrak{F}$, the following program $\mathbb{P}$ is defined, which simulates the behavior related to flu symptoms:

$\langle \mathrm{c} \leftarrow_{\mathrm{P}} \mathrm{n} \,\&_{\mathrm{P}}\, \mathrm{u}, 0.8 \rangle$
$\langle \mathrm{n} \leftarrow_{\mathrm{P}} \mathrm{c}, 0.8 \rangle$
$\langle \mathrm{n} \leftarrow_{\mathrm{P}} @_{(1,2)}(\mathrm{h}, \mathrm{f}), 0.6 \rangle$
$\langle \mathrm{h} \leftarrow_{\mathrm{P}} \mathrm{f}, 0.7 \rangle$
$\langle \mathrm{u} \leftarrow_{\mathrm{G}} \mathrm{h} \,\&_{Ł}\, \mathrm{f}, 0.7 \rangle$
$\langle \mathrm{f} \leftarrow_{\mathrm{P}} \mathrm{u}, 0.9 \rangle$
$\langle \mathrm{a} \leftarrow_{\mathrm{P}} @_{(3,1)}(\mathrm{u}, \mathrm{f}), 1.0 \rangle$
$\langle \mathrm{f} \leftarrow_{\mathrm{P}} 1.0, 0.8 \rangle$
$\langle \mathrm{n} \leftarrow_{\mathrm{P}} 1.0, 0.5 \rangle$

where c, n, h, f, u and a correspond to cough, nasal ache, headache, fever, flu and muscle ache. □

As we reported in the introduction, this paper is focused on the most important theorems introduced in [20]. Before recalling these results, we need several definitions.

Definition 10. An *interpretation* is a mapping $I : \Pi \to L$. The set of all interpretations is denoted as $\mathcal{I}_L$.

Notice that each of these interpretations, by the unique homomorphic extension theorem, can be uniquely extended to the set of formulae $\mathfrak{F}$. The function obtained in this way from an interpretation I is denoted by $\hat{I}$.

The ordering $\preceq$ on the truth-values lattice L can be extended to the set of interpretations $\mathcal{I}_L$.

Proposition 11. *Let $\sqsubseteq$ be the ordering defined on $\mathcal{I}_L$ as, $I_1 \sqsubseteq I_2$ iff $I_1(p) \preceq I_2(p)$, for $I_1, I_2 \in \mathcal{I}_L$ and $p \in \Pi$. The pair $(\mathcal{I}_L, \sqsubseteq)$ is a bounded lattice. The least interpretation $\triangle$ maps every propositional symbol to the least element $\bot \in L$.*

The semantic of multi-adjoint logic programming is based on the following notions of satisfiability and model.

Definition 12. Given an interpretation $I \in \mathcal{I}_L$, a weighted rule $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ is *satisfied* by I, if $\vartheta \preceq \hat{I}(A \leftarrow_i \mathcal{B})$. An interpretation $I \in \mathcal{I}_L$ is a *model* of a multi-adjoint logic program $\mathbb{P}$ if all weighted rules in $\mathbb{P}$ are satisfied by I.

The immediate consequences operator, which was given by van Emden and Kowalski [27], is defined in this framework as follows.

Definition 13. 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 as

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

The main feature of $T_{\mathbb{P}}$ is that its least fixed-point, denoted by $\mathrm{lfp}(T_{\mathbb{P}})$, coincides with the least model of the program $\mathbb{P}$ [49] and is obtained iterating the $T_{\mathbb{P}}$ operator from the least interpretation $\triangle$. As we also highlighted in the introduction, it is important to know when this iteration finishes in a finite number of steps.

Definition 14. 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 *culprit set* for A with respect to I is the set of rules $\langle A \leftarrow_i \mathcal{B}, \vartheta\rangle$ of $\mathbb{P}$ such that $\vartheta \,\&_i\, \hat{I}(\mathcal{B})$ belongs to $R^I_{\mathbb{P}}(A)$. Rules in a culprit set are called *culprits*. The *culprit collection* for $T^n_{\mathbb{P}}(\triangle)(A)$ is defined as the set of culprits used in the recursive computation of $T^n_{\mathbb{P}}(\triangle)(A)$.

The notion of termination used in [20], which we will also consider in this paper, is the definition of fixpoint-reachability of Kifer and Subrahmanian [41].

Definition 15. Let $\mathbb{P}$ be a multi-adjoint logic program with respect to a multi-adjoint Σ-algebra $\mathfrak{L}$ and a set of propositional symbols Π. We say that $T_{\mathbb{P}}$ *terminates for every query* if for every propositional symbol A, there is a finite n such that $T^n_{\mathbb{P}}(\triangle)(A)$ is identical to $\mathrm{lfp}(T_{\mathbb{P}})(A)$.

An interesting termination theorem given in [20] was based on the notions of dependency graph and finite dependences.

Definition 16. The *dependency graph* of a multi-adjoint logic program $\mathbb{P}$ is a digraph where the vertices are the propositional symbols in Π, and there is an arc from a propositional symbol A to a propositional symbol B for each rule with head A and the body containing an occurrence of B. The dependency graph for a propositional symbol A is the subgraph of the dependency graph containing all of the vertices connected to A and corresponding edges.

We say that a multi-adjoint logic program $\mathbb{P}$ has *finite dependences* if the number of edges in the dependency graph for A is finite for every propositional symbol A.

Notice that a program can have finite dependences, even though the program has an infinite set of rules. A particular case was shown in Example 23 of [20], in which a countable infinite program with finite dependences was presented. Based on this definition, the following theorem was introduced in [20], which is one of the most important termination results given in the aforementioned paper.

Theorem 17 *([20]). Let $\mathbb{P}$ be a multi-adjoint logic program with respect to a local multi-adjoint Σ-algebra $\mathfrak{L}$ and the set of propositional symbols Π, and having finite dependences. 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.*

Because we assume finite dependences, when the unit interval or any linear lattice is considered, for every iteration n, $T^n_{\mathbb{P}}(\triangle)(A)$ is a maximum for all propositional symbol A and so, the previous theorem can be applied.

Corollary 18 *([20]). Given a local multi-adjoint Σ-algebra $\mathfrak{L}$, where the lattice is the unit interval $[0, 1]$, and a multi-adjoint logic program $\mathbb{P}$, having finite dependences, we have that $T_{\mathbb{P}}$ terminates for every query.*

In addition, if the theorem cannot be applied to show termination results (e.g., in the case of Hybrid Probabilistic Logic Programs (HPLPs) appearing in [23], because operators employed to capture disjunctive probabilistic strategies do not satisfy the boundary conditions), then we have the following result, which is based on the notion of *range dependency graph*.

Definition 19. The *range dependency graph* of a multi-adjoint logic program $\mathbb{P}$ has a vertex for each propositional symbol in Π. There is an arc from a propositional symbol A to a propositional symbol B if A is the head of a rule with body containing an occurrence of B, which does not appear in a sub-term rooted by a function symbol having finite range.

This definition reduces the dependency graph of a multi-adjoint logic program $\mathbb{P}$ removing arcs associated with operators with finite range. Consequently, arcs involved in infinite loops can be removed, and so the loops can be broken. For example, $\&_{\text{Ł}}^{*}\colon [0,1]_{10} \times [0,1]_{10} \to [0,1]_{10}$ has a finite range because at most only 11 values can be obtained when this operator is used. Hence, if we have the rule $A \leftarrow \&_{\text{Ł}}^{*}(\&_P(A,B),B) \otimes \&_P(B,D)$, then the range dependency graph will not consider the subterm $\&_{\text{Ł}}^{*}(\&_P(A,B),B)$ and so the representation of the previous rule in the range dependency graph will be equivalent to the representation of the rule: $A \leftarrow \&_P(B,D)$.

We will say that a multi-adjoint program $\mathbb{P}$ has *range finite dependences*, if for every propositional symbol A, the number of edges in the associated range dependency graph for A is finite. On this notion, we have the following result.

Theorem 20 *([20]). If $\mathbb{P}$ is a multi-adjoint logic program with an acyclic range dependency graph having range finite dependences, then $T_{\mathbb{P}}$ terminates for every query.*

Finally, the last result merges Theorems 17 and 20 to allow more flexible logic programs.

Theorem 21 *([20]). Let $\mathbb{P}$ be a multi-adjoint logic program with respect to a local multi-adjoint Σ-algebra with finite operators $\mathfrak{L}$ and the set of propositional symbols Π, and having range finite dependences.*

If 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.

Although these results are useful and general, as we will show next, they do not allow the use of aggregator operators (e.g., weighted sums or disjunctive operators) in general, which is very suitable in many applications.

Example 22. Given the program introduced in Example 9, the least model of the program through the immediate consequence operator $T_{\mathbb{P}}$ will be computed next.

A simple implementation of the $T_{\mathbb{P}}$ operator can be done (e.g., in Python):

```
c=n=h=f=u=a=0;
c1=n1=h1=f1=u1=a1=0;
count=0;
while [c,n,h,u,f,a] < > [c1,n1,h1,u1,f1,a1] or count==0:
    c=c1;n=n1;h=h1;u=u1;f=f1;a=a1;
    c1=n*u*0.8;
    n1=max(c*0.8,(h+2*f)*0.6/3,0.5);
    h1=f*0.7;
    u1=min(max(0,(h+f-1)),0.7);
    f1=max(0.8,0.9*u)
    a1=(3*u+f)/4
    print count,' ',f,' ',h,' ',u,' ',c,' ',n,' ',a
    count += 1;
print count,' ',f,' ',h,' ',u,' ',c,' ',n,' ',a
```

The results of the iterations of this program are depicted in Table 1.

Therefore, the least model of the program is obtained after a finite number of iterations. However, the program does not satisfy the hypotheses of Theorems 17 and 21. Specifically, the weighted sums $@_{(3,1)}$ and $@_{(1,2)}$ do not verify the boundary condition with the top element. For example, we have $@_{(3,1)}(1,0.5) = 0.875 \not\leq 0.5$ and $@_{(1,2)}(0.4,1) = 0.8 \not\leq 0.4$. □

Table 1
Iteration of $T_{\mathbb{P}}$ operator from the least interpretation $\triangle$.

	f	h	u	c	n	a
$\triangle$	0	0	0	0	0	0
$T_{\mathbb{P}}(\triangle)$	0.8	0	0	0	0.5	0
$T^2_{\mathbb{P}}(\triangle)$	0.8	0.56	0	0	0.5	0.2
$T^3_{\mathbb{P}}(\triangle)$	0.8	0.56	0.36	0	0.5	0.2
$T^4_{\mathbb{P}}(\triangle)$	0.8	0.56	0.36	0.144	0.5	0.47
$T^5_{\mathbb{P}}(\triangle)$	0.8	0.56	0.36	0.144	0.5	0.47

Hence, Theorems 17 and 21 present interesting and useful termination results, but there are some cases in which the hypotheses are not satisfied, even though $T_{\mathbb{P}}$ terminates for every query. Therefore, one important goal is to weaken the hypotheses or introduce new termination results that can be applied to a bigger range of logic programs. Therefore, hypergraphs instead of graphs will be taken into account to reformulate these theorems. The use of hypergraphs for representing logic programs is studied in the following section.

3. Representing programs by labeled directed hypergraphs

The dependency graph of a program has been a useful tool for the termination results recalled above [20]. However, important information in the program is missing in the dependency graph. This is one of the most important reasons why hypergraphs are useful in this framework. Specifically, in this section we will use labeled directed hyperarcs instead of edges to capture more information about the relationships among the variables in the body of a rule, which are lost when only edges are considered. Consequently, we will show that the consideration of hyperarcs, and so hypergraphs, will provide a richer representation of the program than the dependency graph.

3.1. Edge-labeled B-graphs for representing multi-adjoint logic programs

The notion of associated B-graph of a multi-adjoint logic program $\mathbb{P}$ is first introduced. We then present the natural steps for computing such a hypergraph on an example.

Definition 23. Given a multi-adjoint logic program $\mathbb{P}$, the *associated B-graph* of $\mathbb{P}$ is the edge-labeled B-graph

$$\mathcal{H}_{\mathbb{P}} = (\Pi, \{((\{B_1, \ldots, B_n\}, \{A\}), @) \mid \langle A \leftarrow_i @[B_1, \ldots, B_n], \vartheta \rangle \in \mathbb{P}\})$$

where $((\{B_1, \ldots, B_n\}, \{A\}), @)$ denotes the hyperarc $(\{B_1, \ldots, B_n\}, \{A\})$ with the label $@$.

The B-graph $\mathcal{H}_{\mathbb{P}}$ associated with a given program $\mathbb{P}$, which is obtained from Definition 23, is constructed as follows:

1. The vertex set of the hypergraph is the propositional symbol set Π of the program. For instance, in Example 9, we have $V(\mathcal{H}_{\mathbb{P}}) = \{\mathrm{a}, \mathrm{c}, \mathrm{f}, \mathrm{h}, \mathrm{n}, \mathrm{u}\}$.
2. Each program rule provides a hyperarc e, as follows: the propositional symbols of the body of the rule will be the tail $T(e)$ of the hyperarc, and the propositional symbol of the head of the rule will be the only element of the head $H(e)$. This hyperarc is labeled with the composition operator in the body of the rule. When no operator appears in the body of the rule, we will consider the identity mapping and we will avoid including the symbol in the representation of the hypergraph. For example, from the rule $\langle \mathrm{u} \leftarrow_P \mathrm{h} \,\&_L\, \mathrm{f}, 0.7 \rangle$ in Example 9, we obtain the hyperarc $(\{\mathrm{f}, \mathrm{h}\}, \{\mathrm{u}\})$ with label $\&_L$.

Due to the considered mechanism, the obtained hypergraph is always an edge-labeled B-graph. Moreover, it is possible to obtain the original program from the hypergraph. Hence, both representations are equivalent. However, this fact does not hold for dependency graphs. Fig. 3 shows the B-graph $\mathcal{H}_{\mathbb{P}}$ associated with the program $\mathbb{P}$ given in Example 9. Notice that $\mathcal{H}_{\mathbb{P}}$ is an edge-labeled hypergraph of the one associated with the example given in page 3.

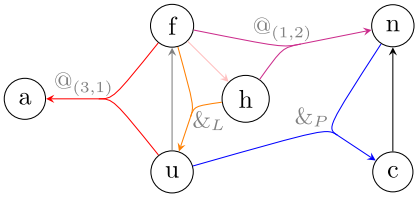

Fig. 3. Edge-labeled B-graph associated with the program given in Example 9.

Moreover, in this particular labeled hypergraph, the hyperarc from f to h has been labeled by the identity mapping (which is not depicted) and the other hyperarcs have the operators in the body of the rules as labels, specifically the symbols $@_{3,1}$, $@_{1,2}$, $\&_{\text{Ł}}$ and $\&_{\text{P}}$.

As a consequence of the representation of programs by hypergraphs, the manifold results and algorithms development in directed hypergraphs can be applied to study fuzzy logic programs. For example, it will be used to improve different termination theorems. These theorems do not consider the whole graph associated with a program but only the local notion of dependency graph of a particular propositional symbol. Indeed, infinite programs can be considered with infinite propositional symbols, such as shown by Example 23 in [20]. The next section will introduce an extension of the dependency graph of a propositional symbol to hypergraphs.

3.2. Antecedent sets and associated hypergraphs

The first notion that we need to present collects all of the vertices 'affecting' a given vertex, which has been adapted from [33].

Definition 24. Given a vertex v of a directed hypergraph $\mathcal{H} = (V, E)$, the *antecedent set* of v (denoted as $\text{AntSet}_{\mathcal{H}}(v)$) is the set of all vertices $u \in V$ verifying that there exists a path from u to v.

For example, in the hypergraph in Fig. 1, we have

$$\text{AntSet}_{\mathcal{H}}(\text{a}) = \{\text{a}, \text{f}, \text{h}, \text{u}\}$$
$$\text{AntSet}_{\mathcal{H}}(\text{c}) = \{\text{c}, \text{f}, \text{h}, \text{n}, \text{u}\}$$
$$\text{AntSet}_{\mathcal{H}}(\text{f}) = \{\text{f}, \text{h}, \text{u}\}$$
$$\text{AntSet}_{\mathcal{H}}(\text{h}) = \{\text{f}, \text{h}, \text{u}\}$$
$$\text{AntSet}_{\mathcal{H}}(\text{n}) = \{\text{c}, \text{f}, \text{h}, \text{n}, \text{u}\}$$
$$\text{AntSet}_{\mathcal{H}}(\text{u}) = \{\text{f}, \text{h}, \text{u}\}$$

The left-hand side of Fig. 4 displays the hypergraph induced by the vertices in the antecedent set of vertex n, $\text{AntSet}_{\mathcal{H}}(\text{n})$, of the hypergraph in Fig. 1. The right-hand side of this figure displays the hypergraph induced by the vertices in $\text{AntSet}_{\mathcal{H}}(\text{a})$. The vertices and hyperarcs not included in both induced subhypergraphs are painted in dotted lines. Clearly, the $\text{AntSet}_{\mathcal{H}_{\mathbb{P}}}(A)$ is the set of vertices in the dependency graph for the propositional symbol A in a program $\mathbb{P}$.

Given a multi-adjoint logic program $\mathbb{P}$ and a propositional symbol $A \in \Pi$, we will denote by $\mathbb{P}_A$ the subprogram of $\mathbb{P}$ containing the rules with head in $Ant\,Set_{\mathcal{H}_{\mathbb{P}}}(A)$; that is,

$$\mathbb{P}_A = \{\langle E \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P} \mid E \in \text{AntSet}_{\mathcal{H}_{\mathbb{P}}}(A)\}$$

This subset allows the local computation of the least fixed-point of a specific propositional symbol.

Proposition 25. *Let $\mathbb{P}$ be a multi-adjoint logic program and $A \in \Pi$ a propositional symbol. The value of the least fixed point of $T_{\mathbb{P}}$ on A is equal to the obtained value considering only the rules in $\mathbb{P}_A$; that is, $lfp(T_{\mathbb{P}})(A) = lfp(T_{\mathbb{P}_A})(A)$.*

Proof. The proof holds because $\mathbb{P}_A$ includes all of the rules in $\mathbb{P}$ that are required to compute $\text{lfp}(T_{\mathbb{P}})(A)$.

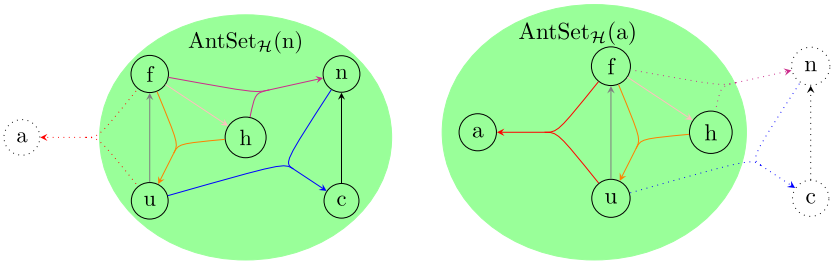

Fig. 4. Subhypergraphs induced by AntSet$_{\mathcal{H}}$(n) (left-hand) and AntSet$_{\mathcal{H}}$(a) (right-hand) from the hypergraph depicted in Fig. 1.

The fundamental hypothesis of "finite dependences" has a direct translation in terms of $\mathbb{P}_A$. Specifically, from this program the associated B-graph $\mathcal{H}_{\mathbb{P}_A}$ can be defined, obtaining the following equivalence.

Proposition 26. *A multi-adjoint logic program $\mathbb{P}$ has finite dependences if and only if for every propositional symbol A, the hypergraph $\mathcal{H}_{\mathbb{P}_A}$ is finite.*

Proof. The proof follows from the fact that $E \in \mathrm{AntSet}_{\mathcal{H}_{\mathbb{P}}}(A)$ if and only if E is a vertex of the dependency graph for A. Because the number of propositional symbols in the body of each rule is finite, if $\mathcal{H}_{\mathbb{P}_A}$ is finite, then the number of vertices and edges in the dependency graph for A is also finite, and so $\mathbb{P}$ has finite dependences. The other implication holds analogously. □

Clearly $\mathcal{H}_{\mathbb{P}_A}$ is a subhypergraph of $\mathcal{H}_{\mathbb{P}}$. Indeed, it is the hypergraph induced by $\mathrm{AntSet}_{\mathcal{H}_{\mathbb{P}}}(A)$. From Proposition 26, we also have that the computation of $\mathcal{H}_{\mathbb{P}_A}$ is only needed to check whether $\mathbb{P}$ has finite dependences. Therefore, the dependency graph of a program is not required to be computed but only $\mathcal{H}_{\mathbb{P}_A}$. In the following, based on this equivalence, we will preserve the terminology "finite dependences" to help the reading and use of the termination theorems given in [20].

Once the definition of dependency graph has been translated to a hypergraph, we will study the obtained associated B-graph to reach new properties for weakening the hypotheses in the theorems recalled in Section 2.2.

3.3. Strongly path-connected components

The notion of strongly path-connected component in hypergraphs allows us to obtain a partition of the set of vertices, which will be considered for defining two fundamental definitions in this paper. We first need to recall a similar notion in digraph.

Given a digraph $D = (W, A)$, we say that a vertex u *reaches* another vertex v, if there is a directed path from u to v. If they are mutually reachable, then we have that u and v are *strongly connected*. As connectivity in undirected graphs, strong connectivity in digraphs induces an equivalence relation on the set of vertices. Therefore, strong connectivity provides a partition in the set of vertices into equivalence classes. In particular, these classes are maximal subsets of vertices that are strongly connected to each other and each vertex is in exactly one subset. They are translated to hypergraph theory using the notion of weakly reachable vertices given in page 4, as follows.

Definition 27. A pair of vertices u and v of a directed hypergraph are said to be *strongly path-connected* if u is weakly reachable from v and v is weakly reachable from u. The binary relation R, which is defined as

$$R = \{(u, v) \in V \times V \mid u \text{ and } v \text{ are strongly path-connected in } \mathcal{H}\},$$

is an equivalence relation. The equivalence classes associated with R are called *strongly path-connected components*, *spc-components* in short.

It is easy to see that the vertices in each cycle P_{st}, where $s = t$, in a hypergraph belong to the same strongly path-connected component. Strongly path connected components can only have one vertex. Notice that the hypergraph in

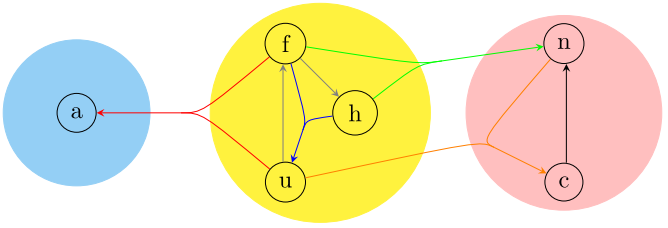

Fig. 5. Strongly path-connected components of the hypergraph in Fig. 1.

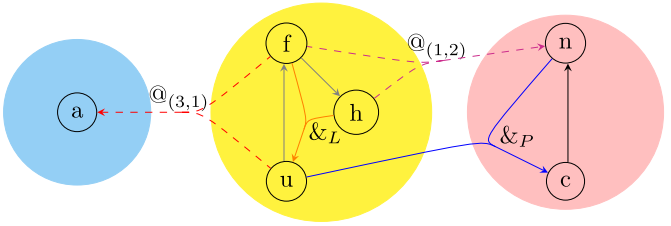

Fig. 6. Aggregator operators $@_{(3,1)}$ and $@_{(1,2)}$ label d-hyperarcs, shown with dashed lines. Operators $\&_P$ and $\&_L$ and non-labeled rules are associated with s-hyperarcs.

Fig. 1 has three spc-components, which are depicted in Fig. 5. One of them is the singleton {a}, and the other two are {f, h, u} and {c, n}.

We can also observe that if an arbitrary vertex v belongs to the antecedent set of another vertex w—that is, $v \in \mathrm{AntSet}_{\mathcal{H}}(w)$—, then all vertices in the same spc-component of v also belong to $\mathrm{AntSet}_{\mathcal{H}}(w)$. Moreover, if v and w belong to the same spc-component, then $\mathrm{AntSet}_{\mathcal{H}}(v) = \mathrm{AntSet}_{\mathcal{H}}(w)$. From these comments, we obtain the following result.

Proposition 28. *Given a directed hypergraph $\mathcal{H} = (V, E)$ and a vertex $v \in V$, we have that $\mathrm{AntSet}_{\mathcal{H}}(v)$ is a spc-component of $\mathcal{H}$ or the union of spc-components of $\mathcal{H}$.*

Proof. The proof straightforwardly follows from the previous statements. □

From the notion of spc-component, we can also split the hyperarcs of a B-graph into two types:

d-hyperarcs: every vertex of its tail belongs to a **d**ifferent spc-component from the one containing the vertex of its head; that is, no vertex of its tail belongs to the spc-component of its head. These hyperarcs are not involved in any cycle P_{st}, where $s = t$.

s-hyperarcs: some vertex of its tail belongs to the **s**ame spc-component as the vertex of its head. These hyperarcs are involved in one or more cycles of the form P_{st}, where $s = t$. (corresponding to cycles of the primal graph).

For example, ({f, h}, {n}) and ({f, u}, {a}) are d-hyperarcs of the B-graph in Fig. 5, and ({f}, {h}) ({f, h}, {u}), ({n, u}, {c}), and ({u}, {f}) are s-hyperarcs. Fig. 6 depicts the edge-labeled B-graph that is associated with the multi-adjoint logic program $\mathbb{P}$ in Example 9, in which the spc-components are highlighted: the d-hyperarcs are shown with dashed lines, and the s-hyperarcs are represented in continuous lines. Moreover, the labels of each hyperarc have been highlighted. Specifically, we have that the d-hyperarcs are labeled by the symbols of the aggregator operators $@_{3,1}$ and $@_{1,2}$. We must take into account that these operators do not satisfy the boundary condition of Equation (1).

As was remarked in Example 22, although the hypotheses in Theorem 17 are not satisfied, the iteration in the computation of the least fixed point of the immediate consequence operator terminates. The next section will show that this fact holds because the aggregator operators $@_{3,1}$ and $@_{1,2}$ are associated with d-hyperarcs, and so they are not involved in any "cycle" that can produce an incremental loop in the values of some propositional symbol.

4. Termination results

This section will introduce the main results of this paper, using directed hypergraphs to enhance the termination theorems given in [20]. Consequently, we will increase the number of logic programs in which the termination theorems can be applied.

The following result modifies Theorem 17 considering any interpretation in the first step of the iteration of the $T_{\mathbb{P}}$ operator instead of the bottom interpretation $\triangle$.

Theorem 29. *Given a local multi-adjoint Σ-algebra $\mathfrak{L}$, a multi-adjoint logic program $\mathbb{P}$ with finite dependences and an interpretation I. If for every iteration n and propositional symbol A, the set of relevant values for A with respect to $T^n_{\mathbb{P}}(I)$ is a singleton, then there is a finite n such that $T^n_{\mathbb{P}}(I)$ is a fixed point of $T_{\mathbb{P}}$.*

Proof. We will first prove that for every $A \in \Pi$, such that $T^n_{\mathbb{P}}(I)(A) \prec T^{n+1}_{\mathbb{P}}(I)(A)$ holds, we have that the culprit collection for $T^{n+1}_{\mathbb{P}}(I)(A)$ has cardinality at least $n+1$. We will proceed by induction in the cardinality of the culprit collection for $T^n_{\mathbb{P}}(I)(A)$.

For $n = 0$, given $A \in \Pi$, such that $T^0_{\mathbb{P}}(I)(A) \prec T^1_{\mathbb{P}}(I)(A)$, if the culprit collection for $T^1_{\mathbb{P}}(I)(A)$ is empty, then the computation of the $T_{\mathbb{P}}(I)(A)$ is the supremum of the empty set, which is equal to the bottom element of the complete lattice $(L, \preceq)$. Therefore, because $\bot \preceq I(A) = T^0_{\mathbb{P}}(I)(A)$, we obtain a contradiction with $\bot \preceq I(A) \prec \bot$ and we can ensure that the culprit collection for $T^1_{\mathbb{P}}(I)(A)$ contains at least one element.

Now, we assume that the property holds for n and we need to prove it for $n+1$. Hence, we have that given $B \in \Pi$ with $T^{n-1}_{\mathbb{P}}(I)(B) \prec T^n_{\mathbb{P}}(I)(B)$, then the culprit collection for $T^n_{\mathbb{P}}(I)(B)$ has at least n different rules.

Given $A \in \Pi$ with $T^n_{\mathbb{P}}(I)(A) \prec T^{n+1}_{\mathbb{P}}(I)(A)$, we will prove that the culprit collection for $T^{n+1}_{\mathbb{P}}(I)(A)$ has at least $n+1$ different rules. Because, by hypothesis, $T^{n+1}_{\mathbb{P}}(I)(A)$ is a singleton, there is at least one rule in the program, $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$, such that

$$T^{n+1}_{\mathbb{P}}(I)(A) = \vartheta \,\&_i\, \widehat{T^n_{\mathbb{P}}(I)}(\mathcal{B})$$

Consequently, we have that

$$\vartheta \,\&_i\, \widehat{T^{n-1}_{\mathbb{P}}(I)}(\mathcal{B}) \overset{(*)}{\preceq} T^n_{\mathbb{P}}(I)(A) \prec T^{n+1}_{\mathbb{P}}(I)(A) = \vartheta \,\&_i\, \widehat{T^n_{\mathbb{P}}(I)}(\mathcal{B})$$

where $(*)$ holds because $T^n_{\mathbb{P}}(I)(A)$ is the supremum of all contributions of the rules in the program and $\vartheta \,\&_i\, \widehat{T^{n-1}_{\mathbb{P}}(I)}(\mathcal{B})$ is the one associated with the rule $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$.

Because the operators $T_{\mathbb{P}}$ and $\&_i$ are monotonic, we have that at least one propositional symbol $C_j \in \Pi$ occurring in the body $\mathcal{B} = @[C_1, \ldots, C_s]$ exists, such that $T^{n-1}_{\mathbb{P}}(I)(C_j) \prec T^n_{\mathbb{P}}(I)(C_j)$.

Therefore, by the induction hypothesis, at least n different rules are in the culprit collection of $T^n_{\mathbb{P}}(I)(C_j)$, and correspond to hyperarcs of the associated subhypergraph of $\mathbb{P}_A$, $\mathcal{H}_{\mathbb{P}_A}$ because C occurs in the body of a rule with head A. We will prove by reduction at absurdum that $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ is not in that culprit collection.

Hence, we suppose that $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ is in the culprit collection of $T^n_{\mathbb{P}}(I)(C_j)$. Therefore, this rule is applied, and so C_j must be in the antecedent set of A. Consequently, we have that the term $T^m_{\mathbb{P}}(I)(A)$ must appear in the computation of $T^n_{\mathbb{P}}(I)(C_j)$; that is, $T^n_{\mathbb{P}}(I)(C_j) = f[T^m_{\mathbb{P}}(I)(A)]$, where f is a mapping composed of all operators and values recursively appearing in the iterations of $T_{\mathbb{P}}$ on I and C_j, which also satisfies the boundary condition with the top (Equation (1)).

Thus, we have that

$$\begin{aligned} T^{n+1}_{\mathbb{P}}(I)(A) &= \vartheta \,\&_i\, @[T^n_{\mathbb{P}}(I)(C_1), \ldots, T^n_{\mathbb{P}}(I)(C_j), \ldots, T^n_{\mathbb{P}}(I)(C_s)] \\ &= \vartheta \,\&_i\, @[T^n_{\mathbb{P}}(I)(C_1), \ldots, f[T^m_{\mathbb{P}}(I)(A)], \ldots, T^n_{\mathbb{P}}(I)(C_s)] \\ &\preceq T^m_{\mathbb{P}}(I)(A) \end{aligned}$$

where the last inequality is given because all of the operators satisfy Equation (1). Consequently, by the monotonicity of $T_{\mathbb{P}}$, the assumed strict inequality and the inequality above, we obtain that

$$T_{\mathbb{P}}^{m}(I)(A) \preceq T_{\mathbb{P}}^{n}(I)(A) \prec T_{\mathbb{P}}^{n+1}(I)(A) \preceq T_{\mathbb{P}}^{m}(I)(A)$$

which leads us to a contradiction.

Therefore, we have that the culprit collection for $T_{\mathbb{P}}^{n+1}(I)(A)$ has at least $n+1$ rules. Consequently, the number of considered rules in each iteration of the $T_{\mathbb{P}}$ increases. We can ensure that the iteration of the $T_{\mathbb{P}}$ terminates for A because $\mathbb{P}$ has finite dependences, which proves the result. □

The following result introduces an useful termination theorem, which generalizes Theorem 17 and greatly increases the spectrum of applications.

Theorem 30. *Given a multi-adjoint Σ-algebra $\mathfrak{L}$ and a multi-adjoint logic program $\mathbb{P}$ with finite dependences, where the s-hyperarcs of the associated B-graph correspond to rules with an aggregator in the body satisfying Equation (1). If 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.*

Proof. We will proceed by strong induction in the number n of strongly path-connected components of the B-graph associated with $\mathbb{P}_A$; that is, of $\mathcal{H}_{\mathbb{P}_A}$.

We first assume that $n = 1$; that is, $\mathcal{H}_{\mathbb{P}_A}$ has only one spc-component. Hence, in this case, no d-hyperarc exists, and so all operators satisfy Equation (1) and we can apply Theorem 17, obtaining that $T_{\mathbb{P}}$ terminates for A.

Let us assume that the hypothesis when the number of components is $k \in \mathbb{N}$, with $1 \leq k \leq n$, is true and we need to prove the case $n+1$. Therefore, we consider a propositional symbol A such that $\mathcal{H}_{\mathbb{P}_A}$ has $n+1$ spc-components.

We can write each rule with head A in $\mathbb{P}$, as follows:

$$r = \langle A \leftarrow_i @[B_1, \ldots, B_n, C_1, \ldots, C_m], \vartheta \rangle$$

where $B_1, \ldots, B_n$ are the propositional symbols in the body of the rule, which are in the same spc-component than A and $C_1, \ldots, C_m$ are in other components.

Therefore, we can ensure that the number of spc-components of $\mathcal{H}_{\mathbb{P}_{C_j}}$ for each propositional symbol C_j (which is a subhypergraph of $\mathcal{H}_{\mathbb{P}_A}$), with $j \in \{1, \ldots, m\}$, has at most n components because $B_1, \ldots, B_n$ and A are not in the antecedent set of C_j. Otherwise, C_j will be in the same spc-component of A, as we will show next. Because for every $i \in \{1, \ldots, n\}$ we have that B_i and A are in the same spc-component, then there exist two paths P_{B_iA} and P_{AB_i}, from B_i to A, and vice versa, respectively. Now, if $i \in \{1, \ldots, n\}$ exists such that B_i is in the antecedent set of C_j, then there exists one path $P_{B_iC_j}$ and so by connecting P_{AB_i} with $P_{B_iC_j}$, we obtain a path from A to C_j, P_{AC_j}. Meanwhile, this rule provides another path from C_j to A, P_{C_jA}, which implies that A and C_j are in the same spc-component, by definition. This fact contradicts the definition of C_j because this propositional symbol is in a different spc-component from A. Finally, if we assume that A is in the antecedent set of C_j, then clearly A and C_j are in the same spc-component.

Consequently, we can apply the induction hypothesis and we have that $T_{\mathbb{P}}$ terminates for every C_j, with $j \in \{1, \ldots, m\}$. Let M_r be the maximum iteration of all $T_{\mathbb{P}}(\triangle)(C_j)$ in which the least fixed point is obtained; that is, $T_{\mathbb{P}}^{M_r}(\triangle)(C_j) = \mathrm{lfp}(T_{\mathbb{P}})(C_j)$, for all $j \in \{1, \ldots, m\}$. Hence, we can write the rule:

$$\langle A \leftarrow_i @'[B_1, \ldots, B_n], \vartheta \rangle$$

where $@'$ is the aggregator operator defined on $[0,1]^n$ as

$$@'[x_1, \ldots, x_n] = @[x_1, \ldots, x_n, \mathrm{lfp}(T_{\mathbb{P}})(C_1), \ldots, \mathrm{lfp}(T_{\mathbb{P}})(C_m)]$$

for all $x_1, \ldots, x_n \in [0,1]$, and the value associated with each propositional symbol B_i in the body, at iteration M_r, is $T_{\mathbb{P}}^{M_r}(\triangle)(B_i)$, for all $i \in \{1, \ldots, n\}$.

This procedure can be applied to every rule r with head A or propositional symbol B in the same spc-component of A in $\mathcal{H}_{\mathbb{P}_A}$. Let M be the maximum natural number of iterations M_r associated with such rules. Consequently, a new set of rules is obtained, denoted as $\mathbb{P}_A^1$, which also has finite dependences and verifies that the propositional symbols appearing in the body of the rules are in the same spc-component of A. This last fact implies that there is no d-hyperarc in $\mathcal{H}_{\mathbb{P}_A^1}$ and so all operators satisfy Equation (1). Therefore, we can apply Theorem 29 to this program and the interpretation $I_A \colon \Pi \to L$ defined for all $D \in \Pi$ as:

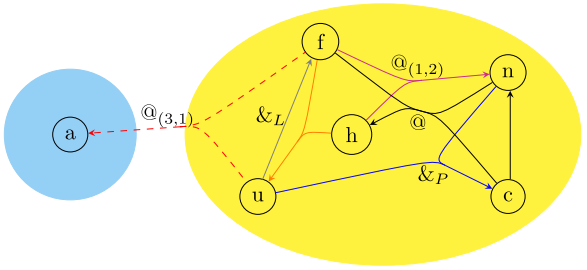

Fig. 7. Edge-labeled B-graph associated with the program given in Example 32.

$$I_A(D) = \begin{cases} T_{\mathbb{P}}^M(\triangle)(D) & \text{if } D \text{ and } A \text{ are in the same component} \\ \text{lfp}(T_{\mathbb{P}})(D) & \text{otherwise} \end{cases}$$

obtaining that $T_{\mathbb{P}_A^1}$ terminates for A.

Because the computation of $T_{\mathbb{P}_A^1}(I_A)(B)$ is equivalent to $T_{\mathbb{P}}^{M+1}(\triangle)(B)$, for every propositional symbol B in the subprogram $\mathbb{P}_A^1$ we obtain that $\text{lfp}(T_{\mathbb{P}_A^1})(A) = \text{lfp}(T_{\mathbb{P}})(A)$ and so $T_{\mathbb{P}}$ terminates for A. □

The assumption that the set of relevant values for A with respect to $T_{\mathbb{P}}^n(I)$ be a singleton is trivially fulfilled in the unit interval.

Corollary 31. *Given a multi-adjoint* Σ*-algebra* $\mathfrak{L}$*, on the unit interval* $[0, 1]$*, and a multi-adjoint logic program* $\mathbb{P}$ *with finite dependences, where the s-hyperarcs of the associated* B*-graph correspond to rules, with an aggregator in the body satisfying Equation* (1)*. We have that* $T_{\mathbb{P}}$ *terminates for every query.*

These results can be applied to Example 22 because the operators not satisfying Equation (1); that is, aggregator operators $@_{(3,1)}$ and $@_{(1,2)}$, are associated with d-hyperarcs, as Fig. 6 shows. Consequently, we can ensure from these results that $T_{\mathbb{P}}$ terminates for every query. Clearly, these results are more useful in larger programs in which the relationships between the variables are not very clear and the possible loops are not visually recognized.

The previous results will be complemented with the consideration of finite range operators, which remarkably increases the number of applications because any operator with finite range can be considered in the bodies of the rules, preserving the termination character of the immediate consequences operator. The next example presents a program with a new finite operator.

Example 32. We will consider the local multi-adjoint Σ-algebra $\mathfrak{F}$ as in Example 9 together with the discretization of the Łukasiewicz disjunction $\vee_{Ł}^*\colon [0, 1] \times [0, 1] \to [0, 1]_{100}$ defined as

$$\vee_{Ł}^*(x, y) = \frac{\lfloor \min\{x + y, 1\} \cdot 100 \rfloor}{100}$$

where $\lfloor \_ \rfloor$ is the floor function. On this Σ-algebra $\mathfrak{F}$, the following program $\mathbb{P}$ is defined:

$\langle \text{c} \leftarrow_{\text{P}} \text{n} \,\&_{\text{P}}\, \text{u}, 0.8\rangle$
$\langle \text{n} \leftarrow_{\text{P}} \text{c}, 0.8\rangle$
$\langle \text{n} \leftarrow_{\text{P}} @_{(1,2)}(\text{h}, \text{f}), 0.6\rangle$
$\langle \text{h} \leftarrow_{\text{P}} \text{f} \,\&_{\text{G}}\, \vee_{Ł}^* (\text{n}, \text{c}), 0.7\rangle$
$\langle \text{u} \leftarrow_{\text{G}} \text{h} \,\&_{\text{L}}\, \text{f}, 0.7\rangle$

$\langle \text{f} \leftarrow_{\text{P}} \text{u}, 0.9\rangle$
$\langle \text{a} \leftarrow_{\text{P}} @_{(3,1)}(\text{u}, \text{f}), 1.0\rangle$
$\langle \text{f} \leftarrow_{\text{P}} 1.0, 0.8\rangle$
$\langle \text{n} \leftarrow_{\text{P}} 1.0, 0.5\rangle$

where c, n, h, f, u and a are as in Example 9.

We can see in Fig. 7 that operators @ and $@_{(1,2)}$ are associated with s-hyperarcs, where @ represents the composition of $\&_G$ and $\vee_{Ł}^*$. The previous results cannot be applied to this logic program because these operators do not satisfy Equation (1). However, the iteration of the immediate consequence operator from the least interpretation terminates, as Table 2 shows. Thus, a new termination theorem that allows the use of finite range operators in the logic programs must be studied. □

Table 2
Computation of the lfp($T_{\mathbb{P}}$) of program $\mathbb{P}$ in Example 32.

	f	h	u	c	n	a
$\triangle$	0	0	0	0	0	0
$T^1_{\mathbb{P}}(\triangle)$	0.8	0	0	0	0.5	0
$T^2_{\mathbb{P}}(\triangle)$	0.8	0.35	0	0	0.5	0.2
$T^3_{\mathbb{P}}(\triangle)$	0.8	0.35	0.15	0	0.5	0.2
$T^4_{\mathbb{P}}(\triangle)$	0.8	0.35	0.15	0.06	0.5	0.3125
$T^5_{\mathbb{P}}(\triangle)$	0.8	0.385	0.15	0.06	0.5	0.3125
$T^6_{\mathbb{P}}(\triangle)$	0.8	0.385	0.185	0.06	0.5	0.3125
$T^7_{\mathbb{P}}(\triangle)$	0.8	0.385	0.185	0.074	0.5	0.33875
$T^8_{\mathbb{P}}(\triangle)$	0.8	0.399	0.185	0.074	0.5	0.33875
$T^9_{\mathbb{P}}(\triangle)$	0.8	0.399	0.199	0.074	0.5	0.33875
$T^{10}_{\mathbb{P}}(\triangle)$	0.8	0.399	0.199	0.0796	0.5	0.34925
$T^{11}_{\mathbb{P}}(\triangle)$	0.8	0.399	0.199	0.0796	0.5	0.34925

Before introducing the aforementioned theorem, different definitions from [20] will be adapted into our framework and a technical lemma will be proved.

From now on, we will consider a multi-adjoint logic program $\mathbb{P}$ with finite dependences, which are defined on a Σ-algebra $\mathfrak{L}$ containing finite range operators, or simply finite operators. Hence, the bodies of the rules $\langle A \leftarrow_i \mathcal{B}_j, \vartheta\rangle$ in the program can be written as follows

$$\mathcal{B}_j = @_j[g^j_1(\mathcal{D}^j_1), \ldots, g^j_{s_j}(\mathcal{D}^j_{s_j}), C^i_1, \ldots, C^j_{m_j}]$$

where for each $l \in \{1, \ldots, s_j\}$, the formula $\mathcal{D}^j_l$ is a subterm of the body $\mathcal{B}_j$, the argument of the outermost occurrence of a finite operator, which is denoted as g^j_l. The elements C^j_l are the propositional symbols that are not influenced by finite operators, and $@_j$ is the aggregator operator given by the composition of all the operators in the body not in the scope of any finite operator. For each propositional symbol A, the set of all finite operators considered in $\mathbb{P}_A$ will be denoted as $G(\mathbb{P}_A)$. For example, from the rule above, we have that $g^j_1, \ldots, g^j_{s_j} \in G(\mathbb{P}_A)$.

Because the finite operators can only increase a finite number of times, controlling this number in the iteration of $T_{\mathbb{P}}$ will be fundamental for proving one of the most important termination theorems presented in this paper. The following definition has been adapted from the one introduced in [20].

Definition 33. The *counting sets for* $\mathbb{P}$ *and* A for all $n \in \mathbb{N}$, denoted as S^A_n, are defined as follows:

$$S^A_n = \{k < n \mid \text{ there is } g^j_l \in G(\mathbb{P}_A) \text{ s.t. } g^j_l(\widehat{T^{k-1}_{\mathbb{P}}(\triangle)}(\mathcal{D}^j_l)) < g^j_l(\widehat{T^k_{\mathbb{P}}(\triangle)}(\mathcal{D}^j_l))\}$$

The hypergraph to be considered in the termination theorem will be a softer notion of the associated B-graph introduced in Definition 23, which accommodates Definition 19 to the hypergraph framework.

Definition 34. Given a multi-adjoint logic program $\mathbb{P}$, the *associated range B-graph* of $\mathbb{P}$ is the labeled hypergraph $\mathcal{H}^r_{\mathbb{P}}$, where the vertices are the propositional symbols used in $\mathbb{P}$ and the labeled hyperarcs are the pairs $((\{C^i_1, \ldots, C^j_{m_j}\}, \{A\}), @_j)$ that are associated with each rule of the program

$$\langle A \leftarrow_i @_j[g^j_1(\mathcal{D}^j_1), \ldots, g^j_{s_j}(\mathcal{D}^j_{s_j}), C^i_1, \ldots, C^j_{m_j}], \vartheta\rangle$$

Notice that the associated range B-graph of a program, $\mathcal{H}^r_{\mathbb{P}}$, is a subhypergraph of $\mathcal{H}_{\mathbb{P}}$. The associated range B-graph of the program in Example 32 is presented next.

Example 35. The program in Example 32 only considers a finite range operator, which is used in the rule

$$\langle \mathrm{h} \leftarrow_{\mathrm{P}} \mathrm{f} \,\&_{\mathrm{G}}\, \vee^*_{\mathrm{L}}(\mathrm{n}, \mathrm{c}), 0.7\rangle$$

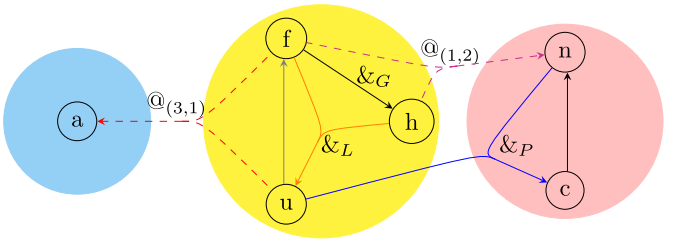

Fig. 8. Associated range B-graph of program $\mathbb{P}$ in Example 32.

Hence, the associated range B-graph does not take into consideration the subterm $\vee^*_{\text{Ł}}(\text{n}, \text{c})$ because it is influenced by the finite range operator $\vee^*_{\text{Ł}}$ and the labeled hyperarc associated with this rule is $((\{\text{f}\}, \{\text{h}\}), \&_{\text{G}})$. The other hyperarcs are obtained as in page 9. Fig. 8 depicts the associated range B-graph of $\mathbb{P}$.

The previous definitions will be taken into account to prove the following technical lemma. Notice that this result generalizes Lemma 39 in [20] and clarifies the proof in the last step because the application of the inductive hypothesis in the aforementioned result required some extra explanation.

Lemma 36. *Let $\mathbb{P}$ be a program on the Σ-algebra $\mathfrak{L}$, where the s-hyperarcs of the associated range B-graph correspond to rules with an aggregator in the body satisfy Equation (1), and such that the set of relevant values for A with respect to $T^n_{\mathbb{P}}(\triangle)$ is a singleton for every propositional symbol A and iteration n. If $T^n_{\mathbb{P}}(\triangle)(A) < T^{n+1}_{\mathbb{P}}(\triangle)(A)$, then either $|S^A_n| < |S^A_{n+1}|$ or the culprit collection for $T^{n+1}_{\mathbb{P}}(\triangle)(A)$ is greater than that for $T^n_{\mathbb{P}}(\triangle)(A)$.*

Proof. The proof will be given by induction on n. For the base case (i.e., when $n = 0$), we have that for every propositional symbol A, if $\bot =_{\triangle} (A) < T_{\mathbb{P}}(\triangle)(A)$, then it is necessary that a new rule be considered.

Now, we prove the inductive case. Hence, we suppose that the result is true for any propositional symbol and $n = k$, and we need to prove the result for $k + 1$.

If $T^k_{\mathbb{P}}(\triangle)(A) < T^{k+1}_{\mathbb{P}}(\triangle)(A)$, then by the singleton hypothesis, there is a rule in $\mathbb{P}$

$$r_i = \langle A \leftarrow_i \mathcal{B}_j, \vartheta \rangle$$

such that $T^{k+1}_{\mathbb{P}}(\triangle)(A) = \vartheta \,\&\, \widehat{T^k_{\mathbb{P}}(\triangle)}(\mathcal{B}_j)$, where $\mathcal{B}_j = @_j[g^j_1(\mathcal{D}^j_1), \ldots, g^j_{s_j}(\mathcal{D}^j_{s_j}), C^j_1, \ldots, C^j_{m_j}]$. Concerning the iteration n because $T^k_{\mathbb{P}}(\triangle)(A)$ is the supremum of all contributions given by the rules with head A, by the supremum property, we have that

$$\vartheta \,\&_i\, \widehat{T^{k-1}_{\mathbb{P}}(\triangle)}(\mathcal{B}_j) \leq T^k_{\mathbb{P}}(\triangle)(A)$$

Therefore, by hypothesis, we obtain the following strict inequality

$$\vartheta \,\&_i\, \widehat{T^{k-1}_{\mathbb{P}}(\triangle)}(@_j[g^i_1(\mathcal{D}^i_1), \ldots, g^i_{s_i}(\mathcal{D}^i_{s_i}), C^i_1, \ldots, C^i_{m_i}]) < \\ < \vartheta \,\&_i\, \widehat{T^k_{\mathbb{P}}(\triangle)}(@_j[g^i_1(\mathcal{D}^i_1), \ldots, g^i_{s_i}(\mathcal{D}^i_{s_i}), C^i_1, \ldots, C^i_{m_i}])$$

Because all of the operators in this expression are monotonic, the strict inequality arises when at least one of the following two cases holds.

1. If $j \in \{1, \ldots, s_i\}$ exists such that

   $$g^i_j(\widehat{T^{k-1}_{\mathbb{P}}(\triangle)}(\mathcal{D}^i_j)) < g^i_j(\widehat{T^k_{\mathbb{P}}(\triangle)}(\mathcal{D}^i_j))$$

   In this case, we straightforwardly obtain that $|S^A_k| < |S^A_{k+1}|$.
2. Otherwise, $j \in \{1, \ldots, m_i\}$ must exist such that

   $$T^{k-1}_{\mathbb{P}}(\triangle)(C^i_j) < T^k_{\mathbb{P}}(\triangle)(C^i_j)$$

In this case, we can ensure that no finite operator is considered in the rules in $\mathbb{P}_{C_j^i}$. Therefore, by the induction hypothesis we have that the culprit collection for $T_{\mathbb{P}}^k(\triangle)(C_j^i)$ is greater than that for $T_{\mathbb{P}}^{k-1}(\triangle)(C_j^i)$. Because this fact happens for all $j \in \{1, \ldots, m_i\}$, if the rule r_i is not used in the culprit collection of some $T_{\mathbb{P}}^k(\triangle)(C_j^i)$, then we can assert that the culprit collection for $T_{\mathbb{P}}^{k+1}(\triangle)(A)$ will be greater than that for $T_{\mathbb{P}}^k(\triangle)(A)$. Thus, in this case, we obtain the result.
Hence, it only remains to consider the case when the rule r_i is in the culprit collection of some $T_{\mathbb{P}}^k(\triangle)(C_j^i)$, which will lead us to a contradiction.
Therefore, we suppose that $\langle A \leftarrow_i \mathcal{B}_j, \vartheta \rangle$ is in the culprit collection of $T_{\mathbb{P}}^k(\triangle)(C_j^i)$. Hence, $C_j^i \in \mathrm{AntSet}(A)$. Indeed, we can ensure that A and C_j^i are in the same spc-component of the associated range B-graph $\mathcal{H}_{\mathbb{P}}^r$. Consequently, we have that the term $T_{\mathbb{P}}^m(\triangle)(A)$ must appear in the computation of $T_{\mathbb{P}}^k(\triangle)(C_j)$; that is, $T_{\mathbb{P}}^k(\triangle)(C_j^i) = f[T_{\mathbb{P}}^k(\triangle)(A)]$, where f is the composition by all operators and values recursively appearing in the iterations of $T_{\mathbb{P}}$ on $\triangle$ and C_j^i, and it satisfies $f[x] \preceq x$, for all $x \in L$, because A and C_j^i are in the same spc-component and, by hypothesis, the operators of s-hyperparcs of the associated range B-graph satisfy Equation (1). Moreover, the aggregator $@_j$ also satisfies this equation because it is the label of a s-hyperarc of the associated range B-graph. Therefore, the expression

$$@_j[\widehat{T_{\mathbb{P}}^k(\triangle)}(g_1^i(\mathcal{D}_1^i)), \ldots, \widehat{T_{\mathbb{P}}^k(\triangle)}(g_{s_i}^i(\mathcal{D}_{s_i}^i)), T_{\mathbb{P}}^k(\triangle)(C_1^i), \ldots, T_{\mathbb{P}}^k(\triangle)(C_j^i), \ldots, T_{\mathbb{P}}^k(\triangle)(C_{m_i}^i)]$$

is equal to

$$@_j[\widehat{T_{\mathbb{P}}^k(\triangle)}(g_1^i(\mathcal{D}_1^i)), \ldots, \widehat{T_{\mathbb{P}}^k(\triangle)}(g_{s_i}^i(\mathcal{D}_{s_i}^i)), T_{\mathbb{P}}^k(\triangle)(C_1^i), \ldots, f[T_{\mathbb{P}}^m(\triangle)(A)], \ldots, T_{\mathbb{P}}^k(\triangle)(C_{m_i}^i)]$$

which implies that $T_{\mathbb{P}}^{k+1}(\triangle)(A) \preceq T_{\mathbb{P}}^m(\triangle)(A)$ because $\&_i$ and $@_j$ satisfy Equation (1). As a consequence of this inequality, the assumed strict inequality and the monotonicity of $T_{\mathbb{P}}$, we have the chain of inequalities

$$T_{\mathbb{P}}^m(\triangle)(A) \preceq T_{\mathbb{P}}^k(\triangle)(A) \prec T_{\mathbb{P}}^{k+1}(\triangle)(A) \preceq T_{\mathbb{P}}^m(\triangle)(A)$$

which is a contradiction and finishes the proof. □

Finally, the last termination theorem can be proven.

Theorem 37. *Given a multi-adjoint Σ-algebra $\mathfrak{L}$ and a multi-adjoint logic program $\mathbb{P}$ with finite dependences. If the s-hyperarcs of the associated range B-graph correspond to rules with an aggregator in the body satisfying Equation (1), 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.*

Proof. For every propositional symbol A, we have that the cardinals $|S_n^A|$ cannot infinitely increase, due to the finitude of the operators g_l^j in $G(\mathbb{P}_A)$. Moreover, the culprit collections for A in each iteration are finite because the program has finite dependences. Thus, by Lemma 36, the iteration of $T_{\mathbb{P}}$ must reach the least fixed point in a finite number of steps; that is, $T_{\mathbb{P}}$ terminates for every query. □

An immediate consequence of the last result is its consideration on the unit interval, which also highlights its usefulness.

Corollary 38. *If the Σ-algebra is defined on $[0, 1]$, and the s-hyperarcs of the associated range B-graph correspond to rules with an aggregator in the body satisfy Equation (1), then $T_{\mathbb{P}}$ terminates for every query.*

This last theorem makes the hypotheses of previous termination results more flexible, thus increasing the applicability scope. For example, it can be applied to the program in Example 32, obtaining that $T_{\mathbb{P}}$ terminates for every query, unlike Theorem 30.

5. Conclusions and future work

We have shown that labeled directed hypergraphs capture more information from a program than graphs, indeed the original program can be recovered from this hypergraph. Hence, this new representation allows the use of hypergraph theory in logic programming. Therefore, hypergraphs provide a valuable representation of multi-adjoint logic programs, which both facilitates visualizing and also contains all of the information associated with the structure of these symbolically rich programs. For example, it has offered the possibility of generalizing the most important termination theorems given in [20] because it is not possible to directly apply the result in the aforementioned paper. For instance, it would be necessary to prove in any case Theorem 29. Moreover, Theorems 28 and 37 in [20] require programs with respect to local multi-adjoint Σ-algebras. Hence, the rules of the original program must be adapted to new 'subprograms' that satisfy the properties and it must be proven that the new programs are semantically equivalent to the original, among other requirements. With consideration of hypergraphs, only the original program is required and the new results with more general hypotheses can be proven, which adapts the results in [20]. Specifically, the first termination theorem introduced in this paper allows the use of general aggregators in the programs when they are in s-hyperarcs of the associated B-graph. The second termination theorem supports the use of other family of operators with finite range. Consequently, the flexibility to model a database to know a priori the termination character of the obtained program is considerably improved, and so the number of applications in which the termination theorems can be used also is much broader. For example, they can be considered in many operational semantics given in fuzzy logic programming [19,37–39], due to the narrow relationship with the $T_{\mathbb{P}}$ operator. Hence, it is possible to know whether these computational procedures will finish for every query in advance. Moreover, the termination for every query can be studied independently by applying the termination theorems only to the subprogram $\mathbb{P}_A$ and the associated subhypergraph $\mathcal{H}_{\mathbb{P}_A}$. In the future, this application will be studied in more depth for the most interesting operational semantics.

Furthermore, the obtained results will be studied in other applications and more results will be analyzed. The consideration of directed hypergraphs has provided an extra level of representation of logic programs. We will use this relationship in other frameworks, such as for introducing new computational procedures for obtaining the least fixed point of a (multi-adjoint) logic program and to detect possible bottlenecks in the program. Other important goals will be to adapt the obtained termination results for handling negations in non-monotonic logic programming [13,14,46] and to be applied in first-order logic programming [17,19,28,36,39,50], which will allow us to increase the range of possible applications even further.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] P. Alimonti, E. Feuerstein, U. Nanni, Linear time algorithms for liveness and boundedness in conflict-free Petri nets, in: Proceedings of the 1st Latin American Symposium on Theoretical Informatics, LATIN '92, Springer-Verlag, Berlin, Heidelberg, 1992, pp. 1–14.

[2] X. Allamigeon, On the complexity of strongly connected components in directed hypergraphs, Algorithmica 69 (2) (Jun 2014) 335–369.

[3] G. Ausiello, A. D'Atri, D. Saccà, Minimal Representations of Directed Hypergraphs and Their Application to Database Design, Springer Vienna, Vienna, 1984, pp. 125–157.

[4] G. Ausiello, R. Giaccio, G.F. Italiano, U. Nanni, Optimal traversal of directed hypergraphs, Technical report International Computer Science Institute, UC Berkeley, 1992.

[5] G. Ausiello, G.F. Italiano, On-line algorithms for polynomially solvable satisfiability problems, J. Log. Program. 10 (1) (1991) 69–90.

[6] G. Ausiello, L. Laura, Directed hypergraphs: introduction and fundamental algorithms—a survey, Theor. Comput. Sci. 658 (PB) (2017) 293–306.

[7] G. Ausiello, M.G. Scutellà, Directed hypergraphs as a modelling paradigm, Riv. AMASES 21 (1–2) (1998) 97–123.

[8] D. Azzolini, F. Riguzzi, E. Lamma, Studying transaction fees in the bitcoin blockchain with probabilistic logic programming, Information 10 (11) (2019).

[9] A. Bădică, C. Bădică, M. Ivanović, D. Logofătu, Exploring the space of block structured scheduling processes using constraint logic programming, in: I. Kotenko, C. Badica, V. Desnitsky, D. El Baz, M. Ivanovic (Eds.), Intelligent Distributed Computing XIII, Springer International Publishing, Cham, 2020, pp. 149–159.

[10] C. Berge, Graphs and Hypergraphs, Elsevier Science Ltd, 1985.

[11] V. Chandru, C.R. Coullard, P.L. Hammer, M. Montañez, X. Sun, On renamable Horn and generalized Horn functions, Ann. Math. Artif. Intell. 1 (1990) 33–48.
[12] M.E. Cornejo, D. Lobo, J. Medina, Characterizing fuzzy y-models in multi-adjoint normal logic programming, in: J. Medina, M. Ojeda-Aciego, J.L. Verdegay, I. Perfilieva, B. Bouchon-Meunier, R.R. Yager (Eds.), Information Processing and Management of Uncertainty in Knowledge-Based Systems. Applications, Springer International Publishing, Cham, 2018, pp. 541–552.
[13] M.E. Cornejo, D. Lobo, J. Medina, Syntax and semantics of multi-adjoint normal logic programming, Fuzzy Sets Syst. 345 (2018) 41–62.
[14] M.E. Cornejo, D. Lobo, J. Medina, Relating multi-adjoint normal logic programs to core fuzzy answer set programs from a semantical approach, Mathematics 8 (6) (2020) 881.
[15] M.E. Cornejo, D. Lobo, J. Medina, Extended multi-adjoint logic programming, Fuzzy Sets Syst. 388 (2020) 124–145.
[16] M.E. Cornejo, J. Medina, E. Ramírez-Poussa, A comparative study of adjoint triples, Fuzzy Sets Syst. 211 (2013) 1–14.
[17] M.E. Cornejo, J. Medina Moreno, C. Rubio-Manzano, Towards a full fuzzy unification in the Bousi prolog system, in: 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2018, pp. 1–7.
[18] C. Damásio, J. Medina, M. Ojeda-Aciego, Sorted Multi-Adjoint Logic Programs: Termination Results and Applications, Lecture Notes in Artificial Intelligence, vol. 3229, 2004, pp. 252–265.
[19] C. Damásio, J. Medina, M. Ojeda-Aciego, A tabulation procedure for first-order residuated logic programs: soundness, completeness and optimisations, in: Proc. FUZZ-IEEE 2006, in: IEEE Congress on Computational Intelligence (section Fuzzy Systems), 2006, pp. 9576–9583.
[20] C. Damásio, J. Medina, M. Ojeda-Aciego, Termination of logic programs with imperfect information: applications and query procedure, J. Appl. Log. 5 (2007) 435–458.
[21] C.V. Damásio, L.M. Pereira, Monotonic and residuated logic programs, in: Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'01, in: Lecture Notes in Artificial Intelligence, vol. 2143, 2001, pp. 748–759.
[22] C.V. Damásio, L.M. Pereira, Termination results for sorted multi-adjoint logic programming, in: Information Processing and Management of Uncertainty for Knowledge-Based Systems, IPMU?04, 2004, pp. 1879–1886.
[23] M. Dekhtyar, A. Dekhtyar, V. Subrahmanian, Hybrid probabilistic programs: algorithms and complexity, in: Proc. of 1999 Conference on Uncertainty in AI, 1999.
[24] A. del Val, On 2-SAT and renamable Horn, in: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, AAAI Press/The MIT Press, 2000, pp. 279–284.
[25] J.C. Díaz-Moreno, J. Medina, J.R. Portillo, Towards the use of hypergraphs in multi-adjoint logic programming, Stud. Comput. Intell. 796 (2019) 53–59.
[26] W.F. Dowling, J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, J. Log. Program. 1 (3) (1984) 267–284.
[27] M.v. Emden, R. Kowalski, The semantics of predicate logic as a programming language, J. ACM 23 (4) (1976) 733–742.
[28] F. Formato, G. Gerla, M. Sessa, Similarity-based unification, Fundam. Inform. 41 (4) (2000) 393–414.
[29] A. Frank, T. Király, Z. Király, On the orientation of graphs and hypergraphs, in: Submodularity, Discrete Appl. Math. 131 (2) (2003) 385–400.
[30] G. Gallo, G. Longo, S. Pallottino, S. Nguyen, Directed hypergraphs and applications, Discrete Appl. Math. 42 (2–3) (apr 1993) 177–201.
[31] G. Gallo, D. Pretolani, A new algorithm for the propositional satisfiability problem, Discrete Appl. Math. 60 (1) (1995) 159–179.
[32] G. Gallo, G. Urbani, Algorithms for testing the satisfiability of propositional formulae, J. Log. Program. 7 (1) (1989) 45–61.
[33] F. Harary, Graph Theory, Addison-Wesley Series in Mathematics, Addison Wesley, 1969.
[34] R.G. Jeroslow, R.K. Martin, R.L. Rardin, J. Wang, Gainfree Leontief substitution flow problems, Math. Program. 57 (1992) 375–414.
[35] P. Julián-Iranzo, J. Medina, M. Ojeda-Aciego, On reductants in the framework of multi-adjoint logic programming, Fuzzy Sets Syst. 317 (2017) 27–43.
[36] P. Julián, G. Moreno, J. Penabad, On fuzzy unfolding: a multi-adjoint approach, Fuzzy Sets Syst. 154 (1) (2005) 16–33.
[37] P. Julián-Iranzo, J. Medina, P.J. Morcillo, G. Moreno, M. Ojeda-Aciego, An Unfolding-Based Preprocess for Reinforcing Thresholds in Fuzzy Tabulation, Lecture Notes in Computer Science, vol. 7902, 2013, pp. 647–655.
[38] P. Julián-Iranzo, J. Medina, G. Moreno, M. Ojeda-Aciego, Thresholded tabulation in a fuzzy logic setting, Electron. Notes Theor. Comput. Sci. 248 (2009) 115–130.
[39] P. Julián-Iranzo, G. Moreno, J. Penabad, Thresholded semantic framework for a fully integrated fuzzy logic language, J. Log. Algebraic Methods Program. 93 (2017) 42–67.
[40] H. Karimi, A. Kamandi, A learning-based ontology alignment approach using inductive logic programming, Expert Syst. Appl. 125 (2019) 412–424.
[41] M. Kifer, V.S. Subrahmanian, Theory of generalized annotated logic programming and its applications, J. Log. Program. 12 (1992) 335–367.
[42] D. Klein, C.D. Manning, Parsing and hypergraphs, in: Proceedings of the Seventh International Workshop on Parsing Technologies, Beijing, China, Oct. 2001, pp. 123–134.
[43] T. Kuhr, V. Vychodil, Fuzzy logic programming reduced to reasoning with attribute implications, Fuzzy Sets Syst. 262 (2015) 1–20, Theme: Logic and Computer Science.
[44] L.V.S. Lakshmanan, F. Sadri, On a theory of probabilistic deductive databases, Theory Pract. Log. Program. 1 (1) (2001) 5–42.
[45] J. Lloyd, Foundations of Logic Programming, Springer Verlag, 1987.
[46] N. Madrid, M. Ojeda-Aciego, On the existence and unicity of stable models in normal residuated logic programs, Int. J. Comput. Math. 89 (3) (2012) 310–324.
[47] S.L. Marcin Suszyński, Jan Žurek, Modelling of assembly sequences using hypergraph and directed graph, Tehn. Vjesnik 21 (2014) 1229–1233.
[48] J. Medina, M. Ojeda-Aciego, A. Valverde, P. Vojtáš, Towards Biresiduated Multi-Adjoint Logic Programming, Lecture Notes in Artificial Intelligence, vol. 3040, 2004, pp. 608–617.

[49] J. Medina, M. Ojeda-Aciego, P. Vojtáš, Multi-adjoint logic programming with continuous semantics, in: Logic Programming and Non-Monotonic Reasoning, LPNMR'01, in: Lecture Notes in Artificial Intelligence, vol. 2173, 2001, pp. 351–364.
[50] J. Medina, M. Ojeda-Aciego, P. Vojtáš, Similarity-based unification: a multi-adjoint approach, Fuzzy Sets Syst. 146 (2004) 43–62.
[51] N.J. Nilsson, Principles of Artificial Intelligence, Morgan Kaufmann, San Francisco (CA), 1980.
[52] L. Paulík, Best Possible Answer is Computable for SLD-Resolution, Lecture Notes in Logic, vol. 6, 1996, pp. 257–266.
[53] D. Pretolani, Satisfiability and Hypergraphs, PhD thesis, Università di Pisa, 1993.
[54] D. Pretolani, A directed hypergraph model for random time dependent shortest paths, Eur. J. Oper. Res. 123 (1998) 315–324.
[55] F. Sáenz-Pérez, Applying constraint logic programming to SQL semantic analysis, Theory Pract. Log. Program. 19 (5–6) (2019) 808–825.
[56] J. Schlipf, F. Annexstein, J. Franco, R. Swaminathan, On finding solutions for extended Horn formulas, Inf. Process. Lett. 54 (3) (1995) 133–137.
[57] M.G. Scutella, A note on Dowling and Gallier's top-down algorithm for propositional Horn satisfiability, J. Log. Program. 8 (3) (1990) 265–273.
[58] Y. Tadesse, Using edge-induced and vertex-induced subhypergraph polynomials, Math. Scand. 117 (2) (2015) 161–169.
[59] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, Pac. J. Math. 5 (2) (1955) 285–309.
[60] M. Thakur, R. Tripathi, Linear connectivity problems in directed hypergraphs, Theor. Comput. Sci. 410 (27) (2009) 2592–2618.
[61] A. Torres, J. Araoz, Combinatorial models for searching in knowledge bases, Acta Cient. Venez. 39 (1988) 387–394.
[62] J.D. Ullman, Principles of Database Systems, 2nd edition, W. H. Freeman & Co., USA, 1983.
[63] A.V. Zeigarnik, O.N. Temkin, D. Bonchev, Chemical Reaction Networks: A Graph-Theoretical Approach, CRC Press, 1996.