MATHEMATICS AND COMPUTATIONAL SCIENCES

Vol 6(1), 2025, pp:67-88

DOI:10.30511/mcs.2025.2034747.1201

ISSN:2717-2708

http://mcs.qut.ac.ir/

Qom University of Technology

Algorithms for determining the type of algebraic hyperstructures and morphisms

Aboutorab Pourhaghani [1], **Seid Mohammad Anvariyeh** [2], **Bijan Davvaz** [3]

[1,2,3]Department of Mathematical Sciences, Yazd University, Yazd, Iran

Abstract: In this paper, we present some primary methods to define a hypergroupoid by algorithm. Then, we present algorithms for checking if it is closed under $\circ$, associativity, weak associativity, commutativity, weak commutativity, establishing the reproduction axiom, determining the type of a hypergroupoid $(H, \circ)$ and the type of a morphism f in a hypergroupoid $(H, \circ)$. The goal of this paper is to provide algorithms for checking the basic features and morphisms in algebraic hyperstructures. Our attention is on algorithms for algebraic hyperstructures with one hyperoperation (i.e. hypergroupoids). The algorithms can be developed for other algebraic hyperstructures.

Keywords: Algorithm; Hypergroupoid; Hypergroup; H_v-group; Algebraic Hyperstructure; Homomorphism
2020 Mathematics Subject Classification: 68W01; 68W30; 15B34; 20N99
Receive: 07 July 2024, **Accepted:** 08 January 2025

1 Introduction

The concept of algebraic hyperstructures was introduced in 1934 by Marty [9], at the 8th Congress of Scandinavian Mathematicians and has been studied in the following decades and nowadays by many mathematicians. In a classical algebraic structure, the composition of two elements is an element, while in an algebraic hyperstructure, the composition of two elements results in a set of elements [5, p. 1]. The development of algebraic hyperstructures has made significant theoretical progress in recent years. These developments are more focused on pure topics and less attention has been given to presenting algorithm.

For example, although checking the basic features in classical algebraic structures can be done by software, e.g. by "GAP[8]", but checking them in algebraic hyperstructures has not been provided by any software. The reason for this is the lack of the necessary algorithms in this field. So far, few algorithms have been presented in algebraic hyperstructures. For example, we can refer to [1, 13, 11, 10, 2, 15, 16, 14, 12, 6]. Most of these algorithms only enumerate some specific algebraic hyperstructures (such as Rosenberg hypergroups). But none of them have provided algorithms to check the basic features and morphisms in algebraic hyperstructures.

This paper is an attempt to solve this deficiency by presenting basic algorithms. Algorithms will be presented in such a way that they can be easily implemented in mathematical software and programming languages. Also, the Maple codes for the algorithms are provided at the end of each part.

There are several important questions about these issues that we try to answer them in this paper:

- How to define a hypergroupoid $(H, \circ)$ by algorithm?

[1]pourhaghani@stu.yazd.ac.ir
[2]Corresponding author: anvariyeh@yazd.ac.ir
[3]davvaz@yazd.ac.ir

- How can we check the basic features, such as associativity and commutativity, in a hypergroupoid $(H, \circ)$ by algorithm?
- How to determine the type of a homomorphism f?

We investigate about the following issues for hypergroupoids:

a) Defining a hypergroupoid $(H, \circ)$, which led to Algorithms 1 and 2 and additional Algorithm 3;

b) Checking the basic features and determining the type of a hypergroupoid $(H, \circ)$ by algorithm, which led to Algorithms 4, 5, 6, 7, 8, 9, 10 and 11;

c) Determining the type of a homomorphism f, which led to Algorithm 12.

In issue a), using the mentioned algorithms, we can easily define a hypergroupoid $(H, \circ)$. In example 3.2 and the paragraph after that, these algorithms are stated and how they are used to define a hypergroupoid $(H, \circ)$ is presented. It should be noted that the mentioned methods for defining hypergroupoid $(H, \circ)$ are primary, and dear reader can use more complex methods, e.g. using generators, in defining hypergroupoid $(H, \circ)$. The only thing that should be observed is the number and type of inputs and outputs, which we have discussed in its place.

In issue b), using the mentioned algorithms, we can confirm the basic features by algorithms. These features are closedness, associativity, weak associativity, commutativity, weak commutativity and reproduction axiom. (A weak relation means that instead of equality, it is a non-empty intersection and reproduction axiom means $xH = H$ for any $x \in H$). Other features can be check similarly by algorithms. Also, Algorithm 11 is presented to determine the type of a hypergroupoid $(H, \circ)$ based on these features, such as semihypergroup, H_v-group and so on.

In issue c), using the mentioned algorithm, we can determine the type of a mapping f. This algorithm can be extended for other algebraic hyperstructures. This algorithm can be easily developed to check other features and return the appropriate output.

2 Preliminaries

In this section, we recall some required definitions in algebraic hyperstructures. The content of this section is mostly adopted from [5, 7, 3].

Let H be a non-empty set. A mapping $\circ : H \times H \to \mathcal{P}^*(H)$, where $\mathcal{P}^*(H)$ denotes the family of all non-empty subsets of H, is called a *hyperoperation* on H. The couple $(H, \circ)$ is called a *hypergroupoid*. A hypergroupoid $(H, \circ)$ is called *finite* if H has only a finitely many elements.

Let A and B are two non-empty subsets of H and $a, b \in H$, then we denote

$$A \circ B = \bigcup_{\substack{x \in A \\ y \in B}} x \circ y, \quad a \circ B = \{a\} \circ B = \bigcup_{y \in B} a \circ y, \quad A \circ b = A \circ \{b\} = \bigcup_{x \in A} x \circ b.$$

Other required definitions will come in place.

Example 2.1. *[4] Let $H = \{O, A, B, AB\}$. Define the hyperoperation $\circ$ on H by the following table:*

$\circ$	O	A	B	AB
O	$\{O\}$	$\{O, A\}$	$\{O, B\}$	$\{A, B\}$
A	$\{O, A\}$	$\{O, A\}$	$\{AB, A, B, O\}$	$\{AB, A, B\}$
B	$\{O, B\}$	$\{AB, A, B, O\}$	$\{O, B\}$	$\{AB, A, B\}$
O	$\{A, B\}$	$\{AB, A, B\}$	$\{AB, A, B\}$	$\{AB, A, B\}$

This is ABO blood table. The $(H, \circ)$ is a hypergroupoid. This hypergroupoid is used for future algorithms as an example.

3 Algorithms

In this section, we first present methods for defining hypergroupoid $(H, \circ)$ (subsection 3.1). Then, in subsection 3.2, we examine the basic features of hypergroupoids. Finally, in subsection 3.3, we present algorithms for checking morphims.

In writing algorithms, modularity has been observed, so that minimal changes are required when changing or updating algorithms.

3.1 Defining algebraic hyperstructures by algorithm

In this subsection, we present two methods for defining a finite hypergroupoid $(H, \circ)$. First, we need to define the set H and hyperoperation $\circ$ on H appropriately for using in next algorithms. If the set H is infinite, the algorithms are logically correct but may not terminate. Therefore, we suppose that the set H is finite.

The set H can be easily implemented by tools available in programming languages or mathematical software. In these languages and software, there is often a tool called "set" to define the set. We can also use "linked list" or "dynamic array" to define the set. We can implement hyperoperation $\circ$ as a "procedure" or "function" that takes two inputs, such as $a, b \in H$, and returns the desired output, i.e. $a \circ b \subseteq H$. This method of implementation of hyperoperation $\circ$ in programming languages or mathematical software frees us from the details of defining hyperoperation. We may define hyperoperation using a table of hyperoperation, or we may define hyperoperation using generator members, or in any desired method. The details of implementation of hyperoperation $\circ$ are not important, the only thing that matters in implementation of hyperoperation $\circ$ is that it is defined as a procedure or function that takes two inputs, say $a, b \in H$, and returns the corresponding set, say $a \circ b \subseteq H$. In example 3.2, we have shown an algorithm for defining hyperoperation $\circ$ using the table of hyperoperation. Although the table of hyperoperation is the simplest method to define hyperoperation, it is not an optimal method in programming. Example 3.2 is just a simple example for implementation of hyperoperation using a table of hyperoperation.

Also, we need an array to specify the order of members of set H. See Example 3.1.

Example 3.1. *Let $H = \{b, c, a, d\}$. So, we have $|H| = 4$. Suppose the order of members of set H is equal to array $\mathbf{A}_H = [a\ b\ c\ d]$. We call the array $\mathbf{A}_H$, the array of order of elements of H. In example 2.1, we have $\mathbf{A}_H = [O\ A\ B\ AB]$. This array is equal to the first row or first column of the table of hyperoperation $\circ$.*

We denote the index of an element $a \in H$ by $a_{\mathbf{index}}$ in terms of array $\mathbf{A}_H$ (with default name "$aindex$" in Maple codes). We need a procedure, named "$WhatIndex$", for this. In Maple, the procedure "Search" finds the index of an entry in an array. We can define procedure $WhatIndex$ based on this procedure, or define by a loop "for". For using it, we simply write $a_{\mathbf{index}} := WhatIndex(a, \mathbf{A}_H)$.

Now, we define a hypergroupoid by algorithm in Example 3.2.

Example 3.2. *Consider Example 2.1. We have $H = \{O, A, B, AB\}$ and the hyperoperation $\circ$ on H by the following table:*

$\circ$	O	A	B	AB
O	$\{O\}$	$\{O, A\}$	$\{O, B\}$	$\{A, B\}$
A	$\{O, A\}$	$\{O, A\}$	$\{AB, A, B, O\}$	$\{AB, A, B\}$
B	$\{O, B\}$	$\{AB, A, B, O\}$	$\{O, B\}$	$\{AB, A, B\}$
O	$\{A, B\}$	$\{AB, A, B\}$	$\{AB, A, B\}$	$\{AB, A, B\}$

First we define and calculate three variables. For any hypergroupoid, we need these variables to define the procedure of hyperoperation $\circ$. It dœsn't matter by which method hyperoperation $\circ$ is defined.
$H = \{O, A, B, AB\}$;
$|H| = cardinal\ of\ H$;
$\mathbf{A}_H = [O\ A\ B\ AB]$.

By array $\mathbf{A}_H$, we calculate the index of each member in the set H according to the position of it in the array. For example, the index of B in the $\mathbf{A}_H$ is equal to 3 and we write $B_{\mathbf{index}} = 3$. This index can be find by procedure $WhatIndex$ (discussed before). Therefore, we can write $B_{\mathbf{index}} := WhatIndex(B, \mathbf{A}_H)$.

Now, according to the above table, we can define hyperoperation $\circ$.

First, we define a matrix with dimension $|H| \times |H|$, called $\mathbf{Table}_\circ$. The value of each entry in the matrix $\mathbf{Table}_\circ$ is determined according to the table of hyperoperation $\circ$ and array $\mathbf{A}_H$.

Therefore, we have:

$$\mathbf{Table}_\circ := \begin{bmatrix} \{O\} & \{A,O\} & \{B,O\} & \{A,B\} \\ \{A,O\} & \{A,O\} & \{A,AB,B,O\} & \{A,AB,B\} \\ \{B,O\} & \{A,AB,B,O\} & \{B,O\} & \{A,AB,B\} \\ \{A,B\} & \{A,AB,B\} & \{A,AB,B\} & \{A,AB,B\} \end{bmatrix}$$

Based on $\mathbf{Table}_\circ$, hyperoperation $\circ$ is presented in Algorithm 1 and procedure $\circ$ corresponds to hyperoperation $\circ$.

Algorithm 1: Defining hyperoperation $\circ$ based on matrix $\mathbf{Table}_\circ$

Input: *Matrix $\mathbf{Table}_\circ$, $a \in H$, $b \in H$*
Output: *Set $a \circ b$*
Procedure: $hyperoTable(\mathbf{Table}_\circ : matrix, a, b) : set$
return $\mathbf{Table}_\circ[WhatIndex(a, \mathbf{A}_H), WhatIndex(b, \mathbf{A}_H)]$

Input: $a \in H$, $b \in H$
Output: *Set $a \circ b$*
Procedure: $\circ(a, b)$
return $hyperoTable(\mathbf{Table}_\circ, a, b)$

Algorithm 1 returns the value of the set $a \circ b$ using the indices corresponding to a and b and the matrix $\mathbf{Table}_\circ$.

If the hyperoperation $\circ$ in hypergroupoid $(H, \circ)$ is defined by mapping $f : H \times H \to \mathcal{P}^*(H)$, instead of the table of hyperoperation $\circ$, we can easily define procedures $\circ$ as follows:

Algorithm 2: Defining hyperoperation for hypergroupoid $(H, \circ)$ by mapping $f : H \times H \to \mathcal{P}^*(H)$

Input: $a \in H$, $b \in H$
Output: Set $a \circ b$
Procedure: $\circ(a, b) : set$
return $f(a, b)$

For Example 3.2, the Maple codes for set H, array $\mathbf{A}_H$, $|H|$ and $\mathbf{Table}_\circ$ is as follow. In Maple, Boolean variables are initialized with "$true$" and "$false$" values, instead of 0 and 1. In these codes, we denote the $|H|$ by variable $cardH$.

```
H:={O,A,AB,B};
cardH:=numelems(H);
AH:=Array([O,A,B,AB]);
#-------------------------------------------------------
TableO:=Matrix(cardH,cardH):
TableO[1,1]:={O}:
TableO[1,2]:={O,A}:
```

```
TableO[1,3]:={O,B}:
TableO[1,4]:={A,B}:
TableO[2,1]:={O,A}:
TableO[2,2]:={O,A}:
TableO[2,3]:={O,A,B,AB}:
TableO[2,4]:={A,B,AB}:
TableO[3,1]:={O,B}:
TableO[3,2]:={O,A,B,AB}:
TableO[3,3]:={O,B}:
TableO[3,4]:={A,B,AB}:
TableO[4,1]:={A,B}:
TableO[4,2]:={A,B,AB}:
TableO[4,3]:={A,B,AB}:
TableO[4,4]:={A,B,AB}:
TableO;
```

The following procedures are written in Maple based on Algorithm 1. In Maple, to find the index of a member in an array, there is a ready-made procedure called "Search", which is located in package "ListTools" and is called by command "with(ListTools):". This procedure uses a loop "for" to search the index of a member in an array and when it reaches the desired member, it returns the value of the loop variable as the index. We present the procedure WhatIndex(a,$\mathbf{A}_H$) in Maple based on procedure "Search" as follow.

Now, using the above codes, we can define the procedures of hyperoperation "$hypero$" as follows in Maple.

```
with(ListTools):
#-------------------------------------------------------
WhatIndex:=proc(a,AH::Array,$)::integer;
return Search(a,AH);
end proc:
#-------------------------------------------------------
hyperoTable:=proc(TableO::Matrix,a,b,$)::set;
 return TableO[WhatIndex(a,AH),WhatIndex(b,AH)];
 end proc:
#-------------------------------------------------------
hypero:=proc(a,b,$)::set;
 return hyperoTable(TableO,a,b);
 end proc:
#-------------------------------------------------------
```

For mapping $f : H \times H \to \mathcal{P}^*(H)$, the procedures of hyperoperation $\circ$, based on Algorithm 2, can be defined directly in Maple as follow.

```
hypero:=proc(a,b,$)::set;
 return f(a,b);
 end proc:
#-------------------------------------------------------
```

Now, we present algorithms for calculating expressions $a \circ b$, $a \circ B$, $A \circ b$ and $A \circ B$. In the algorithms in next subsections, we will use the procedures defined based on the Algorithm 3, instead of directly calling the procedure of hyperoperation "$\circ$", in the body of the algorithms. In Algorithm 3, with proper inputs, the sets $a \circ b$, $a \circ B$, $A \circ b$ and $A \circ B$, based on their definitions in section 2, are calculated and returned.

Algorithm 3: Calculating sets $a \circ b$, $a \circ B$, $A \circ b$, $A \circ B$

Input: Hyperoperation $\circ$, $a \in H$, $b \in H$
Output: Set $a \circ b$
Procedure: $aob(\circ : procedure, a, b) : set$
return $a \circ b$ /* $a \circ b$ = $\circ$(a,b) */

Input: Hyperoperation $\circ$, $a \in H$, Set $B \subseteq H$
Output: Set $a \circ B$
Procedure: $aoB(\circ : procedure, a, B : set) : set$
$aB := \emptyset$
for $b \in B$ **do**
  $aB := aB \cup a \circ b$ /* $a \circ b$ = aob($\circ$,a,b) */
end for
return aB

Input: Hyperoperation $\circ$, Set $A \subseteq H$, $b \in H$
Output: Set $A \circ b$
Procedure: $Aob(\circ : procedure, A : set, b) : set$
$Ab := \emptyset$
for $a \in A$ **do**
  $Ab := Ab \cup a \circ b$ /* $a \circ b$ = aob($\circ$,a,b) */
end for
return Ab

Input: Hyperoperation $\circ$, Set $A \subseteq H$, Set $B \subseteq H$
Output: Set $A \circ B$
Procedure: $AoB(\circ : procedure, A : set, B : set) : set$
$AB := \emptyset$
for $a \in A$ **do**
  $AB := AB \cup a \circ B$ /* $a \circ B$ = aoB($\circ$,a,B) */
end for
return AB

Procedure call: aob($\circ$, a, b), aoB($\circ$, a, B), Aob($\circ$, A, b), AoB($\circ$, A, B)

The Maple codes of Algorithm 3 is as follow:

```
aob:=proc(hypero::procedure,a,b,$)::set;
 return hypero(a,b);
 end proc:
#-------------------------------------------------------
aoB:=proc(hypero::procedure,a,B::set,$)::set;
 local aB::set,b;
 aB:={};
 for b in B do
  aB:=aB union aob(hypero,a,b);
 end do;
 return aB;
 end proc:
#-------------------------------------------------------
Aob:=proc(hypero::procedure,A::set,b,$)::set;
```

```
 local Ab::set,a;
 Ab:={};
 for a in A do
  Ab:=Ab union aob(hypero,a,b);
 end do;
 return Ab;
 end proc:
#-------------------------------------------------------
AoB:=proc(hypero::procedure,A::set,B::set,$)::set;
 local AB::set,a;
 AB:={};
 for a in A do
  AB:=AB union aoB(hypero,a,B);
 end do;
 return AB;
 end proc:
#-------------------------------------------------------
```

To call the above procedures in Maple, it can be done like the following commands:

```
aoB(hypero,a,{a,b});
```

3.2 Algorithms for some basic features

In previous subsection, a hypergroupoid $(H, \circ)$ was defined by algorithm. In this subsection, using the Algorithm 3, we can easily consider the basic features introduced for algebraic hyperstructures. This algorithms can be used as a template to providing algorithms to determine other features in algebraic hyperstructures.

Indeed, by algorithm, we can check the basic features of a hypergroupoid $(H, \circ)$, such as closedness under $\circ$, associativity, weak associativity, reproduction axiom, ..., and also we can determine the type of $(H, \circ)$, such as semihypergroup, hypergroup, etc. . The output of the algorithms in this subsection, except for Algorithm 11, is either **true** or **false**.

First, we need several definitions that are recalled from [5, 7, 3]. Some of them are presented with a slight change.

Let $(H, \circ)$ be a hypergroupoid and $K \subseteq H$. We say K in hypergroupoid $(H, \circ)$ is

- "*closed under* $\circ$" if for all $a, b \in K$ we have $a \circ b \in K$;
- "*associative*" if for all $a, b, c \in K$ we have $a \circ (b \circ c) = (a \circ b) \circ c$;
- "*weak associative*" if for all $a, b, c \in K$ we have $a \circ (b \circ c) \cap (a \circ b) \circ c \neq \emptyset$;
- "*commutative*" if for all $a, b \in K$ we have $a \circ b = b \circ a$;
- "*weak commutative*" if for all $a, b \in K$ we have $a \circ b \cap b \circ a \neq \emptyset$.

We say that set $K \subseteq H$ in hypergroupoid $(H, \circ)$ satisfy "*reproduction axiom*" if for all $a \in K$, we have $a \circ K = K \circ a = K$.

A hypergroupoid $(H, \circ)$ is called

- "*semihypergroup*" if H in hypergroupoid $(H, \circ)$ is associative;
- "*quasihypergroup*" if H in hypergroupoid $(H, \circ)$ satisfies reproduction axiom;
- "H_v*-semigroup*" if H in hypergroupoid $(H, \circ)$ is weak associative;
- "H_v*-group*" if H in hypergroupoid $(H, \circ)$ satisfies reproduction axiom and H in hypergroupoid $(H, \circ)$ is weak associative;

- "*hypergroup*" if H in hypergroupoid $(H, \circ)$ satisfies reproduction axiom and H in hypergroupoid $(H, \circ)$ is associative;

Algorithm 4 checks the closedness of $K \subseteq H$ in hypergroupoid $(H, \circ)$ under $\circ$, based on above definition.

Algorithm 4: Checking closedness of $K \subseteq H$ in hypergroupoid $(H, \circ)$ under $\circ$

```
Input: Hyperoperation ∘, Set K
Output: Is set K ⊆ H in (H, ∘) closed under hyperoperation ∘?
Procedure: IsClosed(∘ : procedure, K : set) : boolean
b := true
for x ∈ K do
   for y ∈ K do
      if x ∘ y ∉ K then                                   /* x ∘ y = aob(∘,x,y) */
         b := false
         break
         break
      end if
   end for
end for
return b
```

Procedure call: `IsClosed`$(\circ, K)$

In Algorithm 4, the closedness of set $K \subseteq H$ in hypergroupoid $(H, \circ)$ under $\circ$ is checked. It should be noted that in order for $(H, \circ)$ to be a hypergroupoid, it is enough that the set H to be closed under $\circ$. Therefore in Algorithm 4, it is enough to substitute the set H for K.

Now, we present algorithms to check the associativity and weak associativity of a set $K \subseteq H$ in a hypergroupoid $(H, \circ)$.

By Algorithm 5, we can check the associativity of set $K \subseteq H$ in hypergroupoid $(H, \circ)$. If set K in $(H, \circ)$ is associative, then it return "$true$", otherwise it return "$false$".

Algorithm 5: Associativity of a set $K \subseteq H$ in hypergroupoid $(H, \circ)$

```
Input: Hyperoperation ∘, Set K ⊆ H
Output: Is set K ⊆ H in (H, ∘) associative?
Procedure: IsAssociative(∘ : procedure, K : set) : boolean
b := true
for x ∈ K do
   for y ∈ K do
      for z ∈ K do
         if x ∘ (y ∘ z) ≠ (x ∘ y) ∘ z then      /* x ∘ (y ∘ z) = aoB(∘,x,aob(∘,y,z)) */
            b := false                          /* (x ∘ y) ∘ z = Aob(∘,aob(∘,x,y),z) */
            break
            break
            break
         end if
      end for
   end for
end for
return b
```

Procedure call: `IsAssociative`$(\circ, \mathtt{K})$

By Algorithm 6, we can check the weak associativity of a set $K \subseteq H$ in hypergroupoid $(H, \circ)$. If set K in $(H, \circ)$ is weak associative, it returns "$true$", otherwise it returns "$false$". This algorithm is similar to Algorithm 5 and only different is in the conditional statement.

Algorithm 6: Weak associativity of a set $K \subseteq H$ in hypergroupoid $(H, \circ)$

```
1  Input: Hyperoperation ∘, Set K ⊆ H
2  Output: Is set K ⊆ H in (H, ∘) weak associative?
3  Procedure: IsWeakAssociative(∘ : procedure, K : set) : boolean
4  b := true
5  for x ∈ K do
6     for y ∈ K do
7        for z ∈ K do
8           if x ∘ (y ∘ z) ∩ (x ∘ y) ∘ z = ∅ then        /* x ∘ (y ∘ z) = aoB(∘,x,aob(∘,y,z)) */
9              b := false                                /* (x ∘ y) ∘ z) = Aob(∘,aob(∘,x,y),z) */
10             break
11             break
12             break
13          end if
14       end for
15    end for
16 end for
17 return b
```

Procedure call: `IsWeakAssociative`$(\circ, \mathtt{K})$

The next two algorithms check the commutativity and weak commutativity of a set $K \subseteq H$ in hypergroupoid $(H, \circ)$ and return the result as "$true$" or "$false$".

Algorithm 7 checks the commutativity of a set $K \subseteq H$ in hypergroupoid $(H, \circ)$. If set K in $(H, \circ)$ is commutative, it return "$true$", otherwise it return "$false$".

Algorithm 7: Commutativity of a set $K \subseteq H$ in hypergroupoid $(H, \circ)$

```
1  Input: Hyperoperation ∘, Set K
2  Output: Is set K ⊆ H in (H, ∘) commutative?
3  Procedure: IsCommutative(∘ : procedure, K : set) : boolean
4  b := true
5  for x ∈ K do
6     for y ∈ K do
7        if x ∘ y ≠ y ∘ x then
8           b := false                                   /* x ∘ y = aob(∘,x,y) and ... */
9           break
10          break
11       end if
12    end for
13 end for
14 return b
```

Procedure call: `IsCommutative`$(\circ, \mathtt{K})$

By Algorithm 8, we can check the weak commutativity of a set $K \subseteq H$ in hypergroupoid $(H, \circ)$. If set K in $(H, \circ)$ is weak commutative, it returns "$true$", otherwise it returns "$false$". This algorithm is similar to Algorithm 7 and only different is in the conditional statement.

Algorithm 8: Weak commutativity of a set $K \subseteq H$ in hypergroupoid $(H, \circ)$

```
Input: Hyperoperation ∘, Set K
Output: Is set K ⊆ H in (H, ∘) weak commutative?
Procedure: IsWeakCommutative(∘ : procedure, K : set) : boolean
b := true
for x ∈ K do
   for y ∈ K do
      if x ∘ y ∩ y ∘ x = ∅ then
         b := false                                   /* x ∘ y = aob(∘,x,y) and ... */
         break
         break
      end if
   end for
end for
return b
```

Procedure call: `IsWeakCommutative(∘, K)`

Now, we present the algorithm for checking the reproduction axiom in a set $K \subseteq H$ in hypergroupoid $(H, \circ)$ using its definition.

Algorithm 9, using the definition, first checks the equality $a \circ K \neq K$ for each member $a \in K$. if it is not satisfied, it stores the result in variable b and break loop "for". Otherwise it checks the equality $K \circ a \neq K$. If it is not confirmed, it stores the result in variable b and break loop "for". Finally, it returns the variable b as the result of the algorithm.

Algorithm 9: Checking reproduction axiom in a set $K \subseteq H$ in hypergroupoid $(H, \circ)$

```
Input: Hyperoperation ∘, Set K
Output: Is reproduction axiom satisfied in set K ⊆ H in (H, ∘)?
Procedure: IsReproduction(∘ : procedure, K : set) : boolean
b := true
for a ∈ K do
   if a ∘ K ≠ K then                                /* a ∘ K=aoB(∘,a,K) */
      b := false
      break
   else if K ∘ a ≠ K then                           /* K ∘ a=Aob(∘,K,a) */
      b := false
      break
   end if
end for
return b
```

Procedure call: `IsReproduction(∘, K)`

By Algorithm 10, we can determine whether $(H, \circ)$ is a hypergroupoid. This algorithm determines whether H is a non-empty set and whether mapping $\circ$ is a hyperoperation on H.

Algorithm 10: Ckecking hypergroupoidness of $(H, \circ)$

Input: Hyperoperation $\circ$, Set H
Output: Is $(H, \circ)$ a hypergroupoid?
Procedure: $IsHypergroupoid(\circ : procedure, H : set) : boolean$
$b := true$
if $H = \emptyset$ **then**
  $b := false$
else
  for $x \in H$ **do**
    for $y \in H$ **do**
      $T := x \circ y$ /* $x \circ y$ = aob(∘,x,y) */
      if $T = \emptyset$ **then**
        $b := false$
        break
        break
      else if $T \nsubseteq H$ **then**
        $b := false$
        break
        break
      end if
    end for
  end for
end if
return b

Procedure call: `IsHypergroupoid(∘,H)`

Now, we can determine the type of a hypergroupoid $(H, \circ)$ among the hypergroupoid, semi-hypergroup, quasi-hypergroup, H_v-semigroup, H_v-group and hypergroup. Algorithm 11 dœs this. It is better to use this algorithm after we have made sure that $(H, \circ)$ is a hypergroupoid (for example, using Algorithms 10), because the precondition of Algorithm 11 is that their input is a hypergroupoid. This algorithm can be extended to include other types of algebraic hyperstructures.

Algorithm 11 determines the type of an algebraic hyperstructure $(H, \circ)$.

Algorithm 11: Determining the type of an algebraic hyperstructure $(H, \circ)$

Input: Hyperoperation $\circ$, Set H
Output: What is the type of $(H, \circ)$?
Procedure: $WhatIsTypeOf(\circ : procedure, H : set) : string$
$associative := IsAssociative(\circ, H)$
if $associative = true$ **then** $weakassociative := true$
else $weakassociative := IsWeakAssociative(\circ, H)$
end if
$commutative := IsCommutative(\circ, H)$
if $commutative = true$ **then** $weakcommutative := true$
else $weakcommutative := IsWeakCommutative(\circ, H)$
end if
$reproduction := IsReproduction(\circ, H)$
if $associative = weakassociative = reproduction = true$ **then**

$BaseType :=$ "**Hypergroup**"
else if $(weakassociative = reproduction = true) \wedge (associative = false)$ **then**
$BaseType :=$ "$H_v -$ **group**"
else if $(associative = weakassociative = true) \wedge (reproduction = false)$ **then**
$BaseType :=$ "**Semihypergroup**"
else if $(weakassociative = true) \wedge (associative = reproduction = false)$ **then**
$BaseType :=$ "$H_v -$ **semigroup**"
else if $(reproduction = true) \wedge (associative = weakassociative = false)$ **then**
$BaseType :=$ "**Quasihypergroup**"
else $BaseType :=$ "**Hypergroupoid**"
end if
if $commutative = weakcommutative = true$ **then**
$CommuataiveType :=$ "**Commutative**"
else if $(weakcommutative = true) \wedge (commutative = false)$ **then**
$CommuataiveType :=$ "**Weak Commutative**"
else $CommuataiveType :=$ "**Non** $-$ **Commutative**"
end if
return $CommuataiveType + BaseType$

Procedure call: `WhatIsTypeOf`$(\circ, H)$

The following procedures are written in Maple based on algorithms in this subsection.

```
IsClosed:=proc(hypero::procedure,K::set,$)::boolean;
 local b::boolean,x,y;
 b:=true;
 for x in K do
  for y in K do
   if not(aob(hypero,x,y) subset K) then
    b:=false;
        break;
        break;
   end if;
  end do;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
IsAssociative:=proc(hypero::procedure,K::set,$)::boolean;
 local b::boolean,x,y,z;
 b:=true;
 for x in K do
  for y in K do
   for z in K do
    if evalb(aoB(hypero,x,aob(hypero,y,z))<>Aob(hypero,aob(hypero,x,y),z)) then
     b:=false;
     break;
     break;
     break;
    end if;
```

```
   end do;
  end do;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
IsWeakAssociative:=proc(hypero::procedure,K::set,$)::boolean;
 local b::boolean,x,y,z;
 b:=true;
 for x in K do
  for y in K do
   for z in K do
    if (aoB(hypero,x,aob(hypero,y,z)) intersect Aob(hypero,aob(hypero,x,y),z))={} then
     b:=false;
     break;
     break;
     break;
    end if;
   end do;
  end do;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
IsCommutative:=proc(hypero::procedure,K::set,$)::boolean;
 local b::boolean,x,y;
 b:=true;
 for x in K do
  for y in K do
   if aob(hypero,x,y)<>aob(hypero,y,x) then
    b:=false;
        break;
        break;
   end if;
  end do;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
IsWeakCommutative:=proc(hypero::procedure,K::set,$)::boolean;
 local b::boolean,x,y;
 b:=true;
 for x in K do
  for y in K do
   if (aob(hypero,x,y) intersect aob(hypero,y,x))={} then
    b:=false;
        break;
        break;
   end if;
```

```
  end do;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
IsReproduction:=proc(hypero::procedure,K::set,$)::boolean;
 local b::boolean,a;
 b:=true;
 for a in K do
  if aoB(hypero,a,K)<>K then
   b:=false;
   break;
  elif Aob(hypero,K,a)<>K then
    b:=false;
        break;
  end if;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
IsHypergroupoid:=proc(hypero::procedure,H::set,$)::boolean;
 local b::boolean,x,y,T::set;
 b:=true;
 if H={} then
  b:=false;
 else
  for x in H do
   for y in H do
    T:=aob(hypero,a,b);
        if T={} then
         b:=false;
         break;
         break;
        elif not (T subset H) then
         b:=false;
         break;
         break;
        end if;
   end do;
  end do;
 end if;
 return b;
 end proc:
#-------------------------------------------------------
with(StringTools):
#-------------------------------------------------------
WhatIsTypeOf:=proc(hypero::procedure,H::set,$)::string;
 local BaseType::string,CommuataiveType::string,associative::boolean,weakassociative::
    boolean,commutative::boolean,weakcommutative::boolean,reproduction::boolean;
```

```
 associative:=IsAssociative(hypero,H);
 if associative=true then weakassociative:=true;
 else weakassociative:=IsWeakAssociative(hypero,H);
 end if;
 commutative:=IsCommutative(hypero,H);
 if commutative=true then weakcommutative:=true;
 else weakcommutative:=IsWeakCommutative(hypero,H);
 end if;
 reproduction:=IsReproduction(hypero,H);
 if (associative and weakassociative and reproduction) then
 BaseType:="Hypergroup";
 elif (weakassociative and reproduction) and (not associative) then
 BaseType:="Hv-group";
 elif (associative and weakassociative) and (not reproduction) then
 BaseType:="Semihypergroup";
 elif weakassociative and ((not associative) and (not reproduction)) then
 BaseType:="Hv-semigroup";
 elif reproduction and ((not associative) and  (not weakassociative)) then
 BaseType:="Quasihypergroup";
 else BaseType:="Hypergroupoid";
 end if;
 if (commutative and weakcommutative) then
 CommuataiveType:="Commutative";
 elif weakcommutative and (not commutative) then
 CommuataiveType:="Weak Commutative";
 else CommuataiveType:="Non-Commutative";
 end if;
 return Join([CommuataiveType,BaseType]," ");
 end proc:
#-------------------------------------------------------
```

To call the above procedures in Maple, it can be done like the following commands:

```
IsClosed(hypero,K);
WhatIsTypeOf(hypero,H);
```

3.3 An algorithm for morphisms in hypergroupoids

In this subsection, we present an algorithm for morphisms. The algorithm in this subsection can be easily extended to other types of morphisms in other algebraic hyperstructures. Required definition are given from [5, 7, 3].

Let $(H_1, \circ)$ and $(H_2, \star)$ be two hypergroupoids. A map $f : H_1 \to H_2$ is called

- a *homomorphism* or *inclusion homomorphism* if we have $f(x \circ y) \subseteq f(x) \star f(y)$ for all $x, y \in H_1$;
- a *good (strong) homomorphism* if we have $f(x \circ y) = f(x) \star f(y)$ for all $x, y \in H_1$;
- an *isomorphism* if it is one to one and onto good homomorphism.
  If f is an isomorphism, then H_1 and H_2 are said to be *isomorphic*, which is denoted by $H_1 \cong H_2$.

Remark 3.3. *If $f : H_1 \to H_2$ is a map on the finite sets H_1 and H_2, then the necessary and sufficient condition for f to be onto is that $|f(H_1)| = |H_2|$, which $f(H_1) = \{f(x)|x \in H_1\}$, and to be one to one is that $|f(H_1)| = |H_1|$. Therefore, f is one to one and onto if we have $|f(H_1)| = |H_1| = |H_2|$.*

Now, using Remark 3.3 in Algorithm 12, we can easily determine the type of a mapping between two hypergroupoid $(H_1, \circ)$ and $(H_2, \star)$.

Algorithm 12: Determining the type of a mapping $f : H_1 \to H_2$ between two hypergroupoids $(H_1, \circ)$ and $(H_2, \star)$

Input: Hyperoperation $\circ$, Set H_1, Hyperoperation $\star$, Set H_2, Mapping $f : H_1 \to H_2$
Output: Is f a mapping from $(H_1, \circ)$ to $(H_2, \star)$?
Procedure: $IsMapping(\circ : procedure, H_1 : set, \star : procedure, H_2 : set, f : procedure) : booelan$
$b := true$
if $(H_1 = \emptyset) \vee (H_2 = \emptyset)$ **then**
  $b := false$
else
  for $x \in H_1$ **do**
    if $f(x) \notin H_2$ **then**
      $b := fasle$
      break
    end if
  end for
end if
return b

Input: Hyperoperation $\circ$, Set H_1, Hyperoperation $\star$, Set H_2, Mapping $f : H_1 \to H_2$
Output: Is mapping f a homomorphism from $(H_1, \circ)$ to $(H_2, \star)$?
Procedure: $IsHomomorphism(\circ : procedure, H_1 : set, \star : procedure, H_2 : set, f : procedure) : booelan$
$b := true$
for $x \in H_1$ **do**
  for $y \in H_1$ **do**
    $fxoy := \emptyset$
    $xoy := x \circ y$ /* $x \circ y$ = aob(∘,x,y) */
    for $z \in xoy$ **do**
      $fxoy := fxoy \cup \{f(z)\}$ /* $fxoy$ is equal to $f(x \circ y)$ */
    end for
    if $fxoy \nsubseteq f(x) \star f(y)$ **then** /* $f(x) \star f(y)$ = aob(⋆, f(x),f(y)) */
      $b := false$
      break
    end if
  end for
end for
return b

Input: Hyperoperation $\circ$, Set H_1, Hyperoperation $\star$, Set H_2, Mapping $f : H_1 \to H_2$
Output: Is mapping f a good homomorphism from $(H_1, \circ)$ to $(H_2, \star)$?
Procedure: $IsGoodHomomorphism(\circ : procedure, H_1 : set, \star : procedure, H_2 : set, f : procedure) : booelan$
$b := true$
for $x \in H_1$ **do**
  for $y \in H_1$ **do**
    $fxoy := \emptyset$
    $xoy := x \circ y$ /* $x \circ y$ = aob(∘,x,y) */

```
        for z ∈ xoy do
            fxoy := fxoy ∪ {f(z)}                          /* fxoy is equal to f(x ∘ y) */
        end for
        if fxoy ≠ f(x) ⋆ f(y) then                         /* f(x) ⋆ f(y) = aob(⋆, f(x),f(y)) */
            b := false
            break
        end if
    end for
end for
return b
```

Input: Hyperoperation $\circ$, Set H_1, Hyperoperation $\star$, Set H_2, Mapping $f : H_1 \to H_2$
Output: Is mapping f onto from $(H_1, \circ)$ to $(H_2, \star)$?
Procedure: $IsOnto(\circ : procedure, H_1 : set, \star : procedure, H_2 : set, f : procedure) : booelan$

```
b := true
fH1 := ∅
for x ∈ H1 do
    fH1 := fH1 ∪ {f(x)}                                    /* Set fH1 is equal to f(H1) */
end for
if |fH1| ≠ |H2| then
    b := false
end if
return b
```

Input: Hyperoperation $\circ$, Set H_1, Hyperoperation $\star$, Set H_2, Mapping $f : H_1 \to H_2$
Output: Is mapping f one to one from $(H_1, \circ)$ to $(H_2, \star)$?
Procedure: $IsOneToOne(\circ : procedure, H_1 : set, \star : procedure, H_2 : set, f : procedure) : booelan$

```
b := true
fH1 := ∅
for x ∈ H1 do
    fH1 := fH1 ∪ {f(x)}                                    /* Set fH1 is equal to f(H1) */
end for
if |fH1| ≠ |H1| then
    b := false
end if
return b
```

Input: Hyperoperation $\circ$, Set H_1, Hyperoperation $\star$, Set H_2, Mapping $f : H_1 \to H_2$
Output: Is mapping f an isomorphism from $(H_1, \circ)$ to $(H_2, \star)$?
Procedure: $IsIsomorphism(\circ : procedure, H_1 : set, \star : procedure, H_2 : set, f : procedure) : booelan$

```
b := true
if IsGoodHomomorphism(∘, H1, ⋆, H2, f) = false then b := false
else if IsOnto(∘, H1, ⋆, H2, f) = false then b := false
else if |H1| ≠ |H2| then b := false
end if
return b
```

Input: Hyperoperation $\circ$, Set H_1, Hyperoperation $\star$, Set H_2, Mapping $f : H_1 \to H_2$
Output: What is the type of mapping f from $(H_1, \circ)$ to $(H_2, \star)$?

Procedure: $WhatTypeOfMapping(\circ : procedure, H_1 : set, \star : procedure, H_2 : set, f : procedure) : boolean$
$goodhomo := IsGoodHomomorphism(\circ, H_1, \star, H_2, f)$
if $goodhomo = true$ **then**
  $homo := true$
else $homo := IsHomomorphism(\circ, H_1, \star, H_2, f)$
end if
$onto := IsOnto(\circ, H_1, \star, H_2, f)$
$cardH_1 := |H_1|$
$cardH_2 := |H_2|$
$BaseType :=$ ""
$SubType :=$ ""
If $goodhomo = true$ **then**
  $BaseType :=$ "**GoodHomomorphism**"
else if $(goodhomo = false) \wedge (homo = true)$ **then**
  $BaseType :=$ "**Homomorphism**"
else $BaseType :=$ "**Mapping**"
end if
if $(goodhomo = true) \wedge (onto = true) \wedge (cardH_1 = cardH_2)$ **then**
  $BaseType :=$ "**Isomorphism**"
  $SubType :=$ ""
else if $(onto = true) \wedge (cardH_1 = cardH_2)$ **then**
  $SubType :=$ "**One to one and onto**"
else if $(onto = true) \wedge (cardH_1 \neq cardH_2)$ **then**
  $SubType :=$ "**Onto**"
else
  $onetoone := IsOneToOne(\circ, H_1, \star, H_2, f)$
  if $(onto = false) \wedge (onetoone = true)$ **then**
    $SubType :=$ "**One to one**"
  else $SubType :=$ ""
  end if
end if
return $SubType + BaseType$

Procedure call: `IsMapping`$(\circ, H_1, \star, H_2, f)$, `IsHomomorphism`$(\circ, H_1, \star, H_2, f)$, `IsGoodHomomorphism`$(\circ, H_1, \star, H_2, f)$, `IsOnto`$(\circ, H_1, \star, H_2, f)$, `IsOneToOne`$(\circ, H_1, \star, H_2, f)$, `IsIsomorphism`$(\circ, H_1, \star, H_2, f)$, `WhatTypeOfMapping`$(\circ, H_1, \star, H_2, f)$

To call procedure $WhatTypeOfMapping$ in Algorithm 12, we must first show f is a mapping (for example by procedure $IsMapping$), and then determine its type by procedure $WhatTypeOfMapping$.

The following procedures are written in Maple based on Algorithm 12.

```
with(StringTools):
#-------------------------------------------------------
IsMapping:=proc(hypero1::procedure,H1::set,hypero2::procedure,H2::set,fmap::procedure,$)::
    boolean;
 local b::boolean,x;
 b:=true;
 if (H1={} or H2={}) then
  b:=false;
 else
```

```
  for x in H1 do
   if not (fmap(x) in H2) then
    b:=false;
        break;
   end if;
  end do;
 end if;
 return b;
 end proc:
#-------------------------------------------------------
IsHomomorphism:=proc(hypero1::procedure,H1::set,hypero2::procedure,H2::set,fmap::procedure
    ,$)::boolean;
 local b::boolean,x,y,z,xoy,fxoy;
 b:=true;
 for x in H1 do
  for y in H2 do
   fxoy:={};
   xoy:=aob(hypero1,x,y);
   for z in xoy do
    fxoy:=fxoy union {fmap(z)};
   end do;
   if not(fxoy subset aob(hypero2,fmap(x),fmap(y))) then
    b:=false;
        break;
   end if;
  end do;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
IsGoodHomomorphism:=proc(hypero1::procedure,H1::set,hypero2::procedure,H2::set,fmap::
    procedure,$)::boolean;
 local b::boolean,x,y,z,xoy,fxoy;
 b:=true;
 for x in H1 do
  for y in H2 do
   fxoy:={};
   xoy:=aob(hypero1,x,y);
   for z in xoy do
    fxoy:=fxoy union {fmap(z)};
   end do;
   if fxoy<>aob(hypero2,fmap(x),fmap(y)) then
    b:=false;
        break;
   end if;
  end do;
 end do;
 return b;
 end proc:
```

```
#-------------------------------------------------------
IsOnto:=proc(hypero1::procedure,H1::set,hypero2::procedure,H2::set,fmap::procedure,$)::
    boolean;
 local b::boolean,x,fH1;
 b:=true;
 for x in H1 do
  fH1:=fH1 union {f(x)};
 end do;
 if numelems(fH1)<>numelems(H2) then
  b:=false;
 end if;
 return b;
 end proc:
#-------------------------------------------------------
IsOneToOne:=proc(hypero1::procedure,H1::set,hypero2::procedure,H2::set,fmap::procedure,$)
    ::boolean;
 local b::boolean,x,fH1;
 b:=true;
 for x in H1 do
  fH1:=fH1 union {f(x)};
 end do;
 if numelems(fH1)<>numelems(H1) then
  b:=false;
 end if;
 return b;
 end proc:
#-------------------------------------------------------
IsIsomorphism:=proc(hypero1::procedure,H1::set,hypero2::procedure,H2::set,fmap::procedure,
    $)::boolean;
 local b::boolean;
 b:=true;
 if not IsGoodHomomorphism(hypero1,H1,hypero2,H2,fmap) then b:=false;
 elif IsOnto(hypero1,H1,hypero2,H2,fmap) then b:=false;
 elif numelems(H1)<>numelems(H2) then b:=false;
 end if;
 return b;
 end proc:
#-------------------------------------------------------
WhatTypeOfMapping:=proc(hypero1::procedure,H1::set,hypero2::procedure,H2::set,fmap::
    procedure,$)::boolean;
 local goodhomo::boolean,homo::boolean,onto::boolean,onetoone::boolean,cardH1,cardH2,
    BaseType::string,SubType::string;
 goodhomo:=IsGoodHomomorphism(hypero1,H1,hypero2,H2,fmap);
 if goodhomo then
  homo:=true;
 else homo:=IsHomomorphism(hypero1,H1,hypero2,H2,fmap);
 end if;
 onto:=IsOnto(hypero1,H1,hypero2,H2,fmap);
 cardH1:=numelems(H1);
```

```
 cardH2:=numelems(H2);
 BaseType:="";
 SubType:="";
 if goodhomo then BaseType:="GoodHmomomorphism";
 elif (not goodhomo) and homo then BaseType:="Hmomomorphism";
 else BaseType:="Mapping";
 end if;
 if goodhomo and onto and (cardH1=cardH2) then
  BaseType:="Isomorphism";
  SubType:="";
 elif onto and (cardH1=cardH2) then SubType:="One to one and onto";
 elif onto and (cardH1<>cardH2) then SubType:="Onto";
 else
  onetoone:=IsOneToOne(hypero1,H1,hypero2,H2,fmap);
  if (not onto) and onetoone then
   SubType:="One to one";
  else SubType:="";
  end if;
 end if;
 return Join([SubType,BaseType]," ");
 end proc:
#-------------------------------------------------------
```

To call the above procedures in Maple, it can be done like the following commands:

```
IsMapping(hypero1,H1,hypero2,H2,fmap);
WhatTypeOfMapping(hypero1,H1,hypero2,H2,fmap);
```

4 Conclusion

In this paper, first, we presented two methods to define a hypergroupoid by algorithm (subsection 3.1). These methods are simply, but any methods can be used for this, with the condition that the number and type of inputs and output are observed. Then, we presented algorithms for checking some basic features of algebraic hyperstructures with one hyperoperation (hypergroupoids) and determining the type of algebraic hyperstructures (subsection 3.2). The features that we provided algorithm to check are: being closed under $\circ$, associativity, weak associativity, commutativity, weak commutativity, establishing the reproduction axiom, determining the type of a hypergroupoid $(H, \circ)$. Also, we present an algorithm to determine the type of morphisms in hypergroupoids (subsection 3.3).

Although in this paper, we presented algorithms only for algebraic hyperstructures with one hyperoperation, but using the methods provided in the algorithms, it is easy to provide algorithms for other algebraic hyperstructures.

In this paper, we did not examine the complexity and optimization of the algorithms, and these issues can be explored in future research. It is also possible to study the application of these algorithms in interdisciplinary research. The presentation of algorithms based on definitions encourages people to investigate the effect of these algorithms in teaching the desired concepts in algebraic hyperstructures.

5 Data Availability Statement

NA

Acknowledgement(s): I would like to thank the referee(s) for his comments and suggestions on the manuscript.

References

[1] Aghabozorgi, H., Jafarpour, M., Dolatabadi, M. K., Cristea, I. (2019). An algorithm to compute the number of Rosenberg hypergroups of order less than 7. Italian journal of pure and applied mathematics, 42, 262–270.

[2] Cristea, I., Jafarpour, M., Mousavi, S. S., Soleymani, A. (2010). Enumeration of Rosenberg hypergroups. Computers & Mathematics with Applications, 60(10), 2753–2763.

[3] Davvaz, B. (2016). Semihypergroup Theory. Cambridge, MA, USA: Academic Press.

[4] Davvaz, B., Dehghan Nezhad, A., Heidari, M. (2013). Inheritance examples of algebraic hyperstructures. Information Sciences, 224, 180–187.

[5] Davvaz, B., Leoreanu-Fotea, V. (2022). Hypergroup Theory. World Scientific.

[6] Davvaz, B., Leoreanu-Fotea, V. (2024). Krasner Hyperring Theory. World Scientific.

[7] Davvaz, B., Vougiouklis, T. (2018). A Walk Through Weak Hyperstructures. Singapore: World Scientific Publishing Company.

[8] GAP (2022). GAP – Groups, Algorithms, and Programming, Version 4.12.2. The GAP Group.

[9] Marty, F. (1934). Sur une generalization de la notion de groups. 8th congress Math. Scandinaves, Stockholm, (pp. 45–49).

[10] Massouros, C. G., Tsitouras, C. (2011). Enumeration of hypercompositional structures defined by binary relations. Italian Journal of Pure and Applied Mathematics, 28, 51–62.

[11] Nordo, G. (1997). An algorithm on number of isomorphism classes of hypergroups of order 3. Italian J. Pure Appl. Math, 2, 37–42.

[12] Park, J., Chung, S.-C. (2000). On algorithms to compute some H_v-groups. Korean J. Comput. & Appl. Math., 7(2), 433–453.

[13] Safari, M., Davvaz, B., Leoreanu-Fotea, V. (2015). Enumeration of 3- and 4-hypergroups on sets with two elements. European Journal of Combinatorics, 44, 298–306. Recent Researches in Hyperstructures.

[14] Spartalis, S., Mamaloukas, C. (2006). Hyperstructures associated with binary relations. Computers & Mathematics with Applications, 51(1), 41–50.

[15] Tsitouras, C., Massouros, C. (2012). Enumeration of Rosenberg-type hypercompositional structures defined by binary relations. European Journal of Combinatorics, 33(8), 1777–1786.

[16] Tsitouras, C., Massouros, C. G. (2010). On enumeration of hypergroups of order 3. Computers & Mathematics with Applications, 59(1), 519–523.