Mathematics Interdisciplinary Research **10** (1) (2025) 49 − 70

Original Scientific Paper

Algorithms for Finding Specific Elements in Algebraic Hyperstructures with One Hyperoperation

Aboutorab Pourhaghani, Seid Mohammad Anvariyeh⋆ *and Bijan Davvaz*

Abstract

In this paper, first, we show how to define an algebraic hyperstructure by using algorithms. Then, we present algorithms that calculate specific elements in algebraic hyperstructures. These specific elements are: scalars, scalar identities, identities, inverses, zero elements, right simplifiable elements, left simplifiable elements, left absorbing-like elements and right absorbing-like elements. We also introduce some algorithms in algebraic hyperstructures to check properties or calculate specific members. These algorithms are presented for algebraic hyperstructures with one hyperoperation, i.e. hypergroupoids. However, they can be developed for other algebraic hyperstructures.

Keywords: Algorithm, Hypergroupoid, Algebraic hyperstructure, Specific element.

2020 Mathematics Subject Classification: 68W01, 68W30, 15B34, 20N99.

How to cite this article
A. Pourhaghani, S. M. Anvariyeh and B. Davvaz, Algorithms for finding specific elements in algebraic hyperstructures with one hyperoperation, *Math. Interdisc. Res.* **10** (1) (2025) 49-70.

⋆Corresponding author (E-mail: anvariyeh@yazd.ac.ir)
Academic Editor: Seyed Hassan Alavi
Received 4 July 2024, Accepted 12 December 2024
DOI: 10.22052/MIR.2024.255119.1470

1. Introduction

The concept of algebraic hyperstructures was introduced in 1934 by Marty [1], at the 8th Congress of Scandinavian Mathematicians and has been studied thenceforth 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 is a set [2, p. 1]. In recent decades, the development of algebraic hyperstructures has made significant progress in theoretical issues, and has less been addressed to practical and interdisciplinary issues. One of the features of this branch of algebra is the absence of the required algorithms so that the theoretical topics of this branch of algebra can be entered into mathematical software and provide key support to its development in applied and interdisciplinary research. For example, by using an algebraic software, such as "GAP [3]", we can find specific elements of a group, such as inverse of an element, but finding such elements in algebraic hyperstructures is not possible by applying softwares, because there are no suitable algorithms in this area.

So far, few algorithms have been presented in algebraic hyperstructures. For example, we can refer to [4–12]. Most of these algorithms only enumerate some specific algebraic hyperstructures (such as Rosenberg hypergroups) using combinatorial methods. But none of them have provided algorithms to find specific elements of an algebraic hyperstructure.

In this paper, we are going to present algorithms to find specific elements in algebraic hyperstructures with one hyperoperation. Algorithms will be presented in such a way that their implementation in mathematical softwares and programming languages is easily possible. Also, at the end of each part, Maple codes for algorithms are provided.

We can ask three important questions about these issues that we are trying to answer in this paper:

- How can we define a hypergroupoid $(H, \circ)$ by algorithms?
- How can we find specific elements, such as scalars, identities, invertible elements, in a hypergroupoid $(H, \circ)$ by algorithms?
- Can we develop these algorithms for algebraic hyperstructures with more than one hyperoperation, such as hyperrings?

We try to answer the above questions in the following frameworks:

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

b) Finding specific elements in a hypergroupoid $(H, \circ)$ by algorithm, which led to Algorithms 4, 5, 6, 7, 8, 9 and 10;

c) Developing above algorithms for algebraic hyperstructures with more than one hyperoperation.

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 the 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 output, which we have discussed in its place.

In issue b), using several algorithms, we can find specific elements in algebraic hyperstructures with one hyperoperation. These elements are: scalars, scalar identities, identities, inverses, zero elements, right simplifiable elements, left simplifiable elements, left absorbing-like elements and right absorbing-like elements. These elements are some of the most famous elements in algebraic hyperstructures that play important roles in the theorems of this field. For other elements, which are not mentioned in this paper, similar algorithms can be provided based on presented algorithms.

In issue c), if we have an algebraic hyperstructure with more than one hyperoperation, such as hyperrings, we should first define this algebraic hyperstructure appropriately. The definition of a hyperstructure with more than one hyperoperation is similar to hyperstructures with one hyperoperation. It is enough to define a procedure with a specific name for each operation or hyperoperation. Then, define the desired algorithms using the procedures of these operations and the above algorithms. The method of presenting the algorithms mentioned above can be used as a template for the algorithms of this topic. This topic is left to the reader.

2. Preliminaries

In this section, we recall some required definitions in algebraic hyperstructures. The content of this section is mostly adopted from [2, 13, 14].

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 finitely many elements.

Let A and B be 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 be recalled later on.

Example 2.1. ([15]). 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 a hypergroupoid $(H, \circ)$ (subsection 3.1). Then, in subsection 3.2, we present several algorithms for finding specific elements in algebraic hyperstructures. These elements are: scalars, scalar identities, identities, inverses, zero elements, right simplifiable elements, left simplifiable elements, left absorbing-like elements and right absorbing-like elements.

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 the 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 for other algorithms, 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 the 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 H is equal to the 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 the 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 the procedure $WhatIndex$ based on this procedure, or define by a "for" loop. For using it, we simply write $a_{\mathbf{index}} := WhatIndex(a, \mathbf{A}_H)$.

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

Example 3.2. Consider Example 2.1. We have $H = \{O, A, B, AB\}$ and the hyperoperation $\circ$ on H is given 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 the hyperoperation $\circ$. It doesn't matter by which method the hyperoperation $\circ$ is defined.
$H = \{O, A, B, AB\}$;
$|H| = cardinal\ of\ H$;
$\mathbf{A}_H = [O\ A\ B\ AB]$.
By the array $\mathbf{A}_H$, we calculate the index of each member in the set H according to its position of it in the array. For example, the index of B in $\mathbf{A}_H$ is equal to 3 and we write $B_{\mathbf{index}} = 3$. This index might be found 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 the 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 the 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 $\mathbf{Table}_\circ$, the hyperoperation $\circ$ is presented in Algorithm 1 and the 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 the hyperoperation $\circ$, we can easily define the 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 H, $\mathbf{A}_H$, $|H|$ and $\mathbf{Table}_\circ$ are gathered below. In Maple, Boolean variables are initialized with "$true$" and "$false$" values, instead of 0 and 1. In these codes, we denote $|H|$ 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:
#-----------------------------------------------------------
```

Although we can now use procedure of hyperoperation $\circ$ in future algorithms, it is better to use intermediate procedure *aob*(), instead of directly using procedure of hyperoperation $\circ$. The reason is that if it is necessary to check some things, e.g. the type of inputs, before using procedure of hyperoperation $\circ$ in the algorithms, it is possible by using the procedure *aob*(). Procedure *aob*() is presented very

simply, but the user can add any condition he deems necessary to the instructions of procedure $aob()$ before "return" line.

Algorithm 3: Calculating sets $a \circ b$.

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∘b = ∘(a,b) */`

Procedure call: `aob(∘, 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(hypero,b,a);
```

3.2 Algorithms for finding some specific elements in a hypergroupoid $(H, \circ)$

In this subsection, some specific elements in hypergroupoids are found using algorithms. Algorithms in this subsection can be a template for writing algorithms for other elements and other algebraic hyperstructures. The algorithms of this subsection are presented based on the following definitions. For any algorithm, if there is a statement, which is equivalent to the following definitions and is computationally more optimal, then the body of the algorithm can be easily updated according to the new statement. It is enough that the type and number of inputs and outputs are fixed and do not change, and the instructions of the algorithm body are changed to the desired instructions.

First, required definitions are recalled and then desired algorithms are presented. Definitions are given from [2, 13, 14]. Definitions are provided with necessary changes for hypergroupoids.

Let H be a non-empty set, $(H, \circ)$ be a hypergroupoid.

- An element $a \in H$ is called *scalar* if $|a \circ x| = |x \circ a| = 1$ for all $x \in H$;
- An element $e \in H$ is called *scalar identity* if $x \circ e = e \circ x = \{x\}$ for all $x \in H$;
- An element $e \in H$ is called *identity* if $x \in e \circ x \cap x \circ e$ for all $x \in H$;
- An element $a' \in H$ is called an *inverse* of $a \in H$ if there exists an identity $e \in H$ such that $e \in a \circ a' \cap a' \circ a$;
- An element $0 \in H$ is called *zero element* if $x \circ 0 = 0 \circ x = \{0\}$ for all $x \in H$;

- An element $a \in H$ is called *right simplifiable* element (respectively, *left*) if for all $x, y \in H$ we have $x \circ a = y \circ a \Rightarrow x = y$ (respectively, $a \circ x = a \circ y \Rightarrow x = y$);
- An element $a \in H$ is called *left absorbing-like* element (respectively, *right absorbing-like*) if $a \in a \circ x$ (respectively, $a \in x \circ a$) for all $x \in H$.

For each algorithm, except the Algorithm 7, two procedures are defined. The first procedure checks whether the input element has the desired property. The output of this procedure is "true" or "false". The second procedure, using the first procedure, returns the set of all elements that have the desired property. For example, Algorithm 4 consists of two procedures $IsScalar()$ and $AllScalar()$. Procedure $IsScalar()$ checks whether the element $a \in H$ is a scalar in hypergroupoid $(H, \circ)$ or not. Procedure $AllScalar()$ finds and returns the set of all scalars in hypergroupoid $(H, \circ)$ using procedure $IsScalar()$.

By Algorithm 4, we can determine all scalars in a hypergroupoid $(H, \circ)$.

Algorithm 4: Determining all scalars in a hypergroupoid $(H, \circ)$.

```
Input: Hyperoperation ∘, Set H, a ∈ H
Output: Is a ∈ H a scalar in (H, ∘)?
Procedure: IsScalar(∘ : procedure, H : set, a) : boolean
b := true
for x ∈ H do
    if |a ∘ x| ≠ 1 then
        b := false                                  /* a ∘ x = aob(∘,a,x) */
        break
    else if |x ∘ a| ≠ 1 then
        b := false                                  /* x ∘ a = aob(∘,x,a) */
        break
    end if
end for
return b
```

```
Input: Hyperoperation ∘, Set H
Output: Set of all scalars in (H, ∘)
Procedure: AllScalar(∘ : procedure, H : set) : set
T := ∅
for a ∈ H do
    if IsScalar(∘, H, a) = true then
        T := T ∪ {a}
    end if
end for
return T
```

Procedure call: `IsScalar`$(\circ, \mathtt{H}, \mathtt{a})$, `AllScalar`$(\circ, \mathtt{H})$

By Algorithm 5, we can determine all scalar identities in a hypergroupoid $(H, \circ)$.

Algorithm 5: Determining all scalar identities in a hypergroupoid $(H, \circ)$.

```
Input: Hyperoperation ∘, Set H, e ∈ H
Output: Is e ∈ H a scalar identity in (H, ∘)?
Procedure: IsScalarIdentity(∘ : procedure, H : set, e) : boolean
b := true
for x ∈ H do
    if e ∘ x ≠ {x} then
        b := false                                  /* e ∘ x = aob(∘,e,x) */
        break
    else if x ∘ e ≠ {x} then
        b := false                                  /* x ∘ e = aob(∘,x,e) */
        break
    end if
end for
return b
```

```
Input: Hyperoperation ∘, Set H
Output: Set of all scalar identities in (H, ∘)
Procedure: AllScalarIdentity(∘ : procedure, H : set) : set
T := ∅
for e ∈ H do
    if IsScalarIdentity(∘, H, e) = true then
        T := T ∪ {e}
    end if
end for
return T
```

Procedure call: `IsScalarIdentity`$(\circ, \mathtt{H}, \mathtt{e})$, `AllScalarIdentity`$(\circ, \mathtt{H})$

By Algorithm 6, we can determine all identities in a hypergroupoid $(H, \circ)$.

Algorithm 6: Determining all identities in a hypergroupoid $(H, \circ)$.

```
Input: Hyperoperation ∘, Set H, e ∈ H
Output: Is e ∈ H a identity in (H, ∘)?
Procedure: IsIdentity(∘ : procedure, H : set, e) : boolean
b := true
for x ∈ H do
    if x ∉ e ∘ x then
        b := false                                  /* e ∘ x = aob(∘,e,x) */
        break
    else if x ∉ x ∘ e then
```

```
        b := false                                   /* x ∘ e = aob(∘,x,e) */
        break
    end if
end for
return b
```

```
Input: Hyperoperation ∘, Set H
Output: Set E (all identities in (H, ∘))
Procedure: AllIdentity(∘ : procedure, H : set) : set
T := ∅
for e ∈ H do
    if IsIdentity(∘, H, e) = true then
        T := T ∪ {e}
    end if
end for
return T
```

Procedure call: `IsIdentity`$(\circ, \mathtt{H}, \mathtt{e})$, `AllIdentity`$(\circ, \mathtt{H})$

The Algorithm 7 determines all inverses of an element in H and all invertible elements. This algorithm is more complicated than other algorithms in this subsection.

Algorithm 7: Determining all inverses in a hypergroupoid $(H, \circ)$.

```
Input: Hyperoperation ∘, Set H, Set E (all identities in (H, ∘)), a′ ∈ H, a ∈ H
Output: Is a′ ∈ H an inverse of H in (H, ∘)?
Procedure: IsInverse(∘ : procedure, H : set, E : set, a′, a) : boolean
b := false
for e ∈ E do
    if e ∈ a ∘ a′ then
        if e ∈ a′ ∘ a then
            b := true                        /* a ∘ a′ = aob(∘,a,a') and ...*/
            break
        end if
    end if
end for
return b
```

```
Input: Hyperoperation ∘, Set H, Set E (all identities in (H, ∘)), a′ ∈ H, a ∈ H
Output: Is a′ ∈ H invertible in (H, ∘)?
Procedure: IsInvertible(∘ : procedure, H : set, E : set, a′) : boolean
b := false
for a ∈ H do
    if IsInverse(∘, H, E, a′, a) = true then
```

```
        b := true
        break
    end if
end for
return b
```

Input: Hyperoperation $\circ$, Set H, Set E (all identities in $(H, \circ)$), $a' \in H$
Output: Set of all inverses of $a' \in H$ in $(H, \circ)$
Procedure: $InverseOf(\circ : procedure, H : set, E : set, a') : set$
$T := \emptyset$
for $a \in H$ **do**
    if $IsInverse(\circ, H, E, a', a) = true$ **then**
        $T := T \cup \{a\}$
    end if
end for
return T

Input: Hyperoperation $\circ$, Set H, Set E (all identities in $(H, \circ)$)
Output: Set of all invertible elements in $(H, \circ)$
Procedure: $AllInvertible(\circ : procedure, H : set, E : set) : set$
$T := \emptyset$
for $a' \in H$ **do**
    if $IsInvertible(\circ, H, E, a') = true$ **then**
        $T := T \cup \{a'\}$
    end if
end for
return T

Procedure call: `IsInverse`$(\circ, \mathtt{H}, \mathtt{E}, \mathtt{a}', \mathtt{a})$, `IsInvertible`$(\circ, \mathtt{H}, \mathtt{E}, \mathtt{a}')$, `InverseOf`$(\circ, \mathtt{H}, \mathtt{E}, \mathtt{a}')$, `AllInvertible`$(\circ, \mathtt{H}, \mathtt{E})$

In Algorithm 7, procedure $IsInverse()$ checks whether element $a' \in H$ is the inverse of the element $a \in H$. The output of this procedure is "true" or "false". Procedure $IsInvertible()$ checks whether element a' is invertible? The output of this procedure is also "true" or "false". Procedure $InverseOf()$ finds the set of all inverse elements of the element a'. The output of this procedure is empty if a' is not invertible, otherwise it will be the set of all inverses of a'. Finally, procedure $AllInvertible()$ returns the set of all invertible elements. The output can be empty, because the desired hypergroupoid $(H, \circ)$ may not have any inverse elements.

Algorithm 8 determines all zero elements in a hypergroupoid $(H, \circ)$.

Algorithm 8: Determining all zero elements in a hypergroupoid $(H, \circ)$.

Input: Hyperoperation $\circ$, Set H, $0 \in H$
Output: Is $0 \in H$ a zero element in $(H, \circ)$?

```
Procedure: IsZeroElems(∘ : procedure, H : set, 0) : boolean
b := true
for x ∈ H do
    if x ∘ 0 ≠ {0} then
        b := false                                  /* x ∘ 0 = aob(∘,x,0) */
        break
    else if 0 ∘ x ≠ {0} then
        b := false                                  /* 0 ∘ x = aob(∘,0,x) */
        break
    end if
end for
return b
```

```
Input: Hyperoperation ∘, Set H
Output: Set of all zero elements in (H, ∘)
Procedure: AllZeroElems(∘ : procedure, H : set) : set
T := ∅
for 0 ∈ H do
    if IsZeroElems(∘, H, 0) = true then
        T := T ∪ {0}
    end if
end for
return T
```

Procedure call: `IsZeroElems`$(\circ, \mathtt{H}, \mathtt{0})$, `AllZeroElems`$(\circ, \mathtt{H})$

By Algorithms 9, we can determine all right simplifiable elements in a hypergroupoid $(H, \circ)$.

Algorithm 9: Determining all right simplifiable elements in a hypergroupoid $(H, \circ)$.

```
Input: Hyperoperation ∘, Set H, a ∈ H
Output: Is a ∈ H a  simplifiable element in (H, ∘)?
Procedure: IsRightSimplifiable(∘ : procedure, H : set, a) : boolean
b := true
for x ∈ H do
    for y ∈ H do
        if (x ∘ a = y ∘ a) ∧ (x ≠ y) then
            b := false                          /* x ∘ a = aob(∘,x,a) and ... */
            break
        end if
    end for
end for
return b
```

Input: Hyperoperation $\circ$, Set H
Output: Set of all right simplifiable elements in $(H, \circ)$
Procedure: $AllRightSimplifiable(\circ : procedure, H : set) : set$
$T := \emptyset$
for $a \in H$ **do**
  if $IsRightSimplifiable(\circ, H, a) = true$ **then**
    $T := T \cup \{a\}$
  end if
end for
return T

Procedure call: `IsRightSimplifiable`$(\circ, \mathtt{H}, \mathtt{a})$, `AllRightSimplifiable`$(\circ, \mathtt{H})$

For left simplifiable elements, in Algorithm 9, it is enough to replace the expression "**if** $(x \circ a = y \circ a) \wedge (x \neq y)$ **then**" with "**if** $(a \circ x = a \circ y) \wedge (x \neq y)$ **then**" and word "$Right$" with "$Left$".

By Algorithm 10, we can determine all left absorbing-like elements in a hypergroupoid $(H, \circ)$.

Algorithm 10: Determining all left absorbing-like elements in a hypergroupoid $(H, \circ)$.

Input: Hyperoperation $\circ$, Set H, $a \in H$
Output: Is $a \in H$ a left absorbing−like element in $(H, \circ)$?
Procedure: $IsLeftAbsorb(\circ : procedure, H : set, a) : boolean$
$b := true$
for $x \in H$ **do**
  if $a \notin a \circ x$ **then**
    $b := false$ `/* a∘x = aob(∘,a,x) */`
    break
  end if
end for
return b

Input: Hyperoperation $\circ$, Set H
Output: Set of all left absorbing−like element in $(H, \circ)$
Procedure: $AllLeftAbsorb(\circ : procedure, H : set) : set$
$T := \emptyset$
for $a \in H$ **do**
  if $IsLeftAbsorb(\circ, H, 0) = true$ **then**
    $T := T \cup \{a\}$
  end if
end for
return T

Procedure call: `IsLeftAbsorb`$(\circ, \mathtt{H}, \mathtt{a})$, `AllLeftAbsorb`$(\circ, \mathtt{H})$

For right absorbing-like elements, in Algorithm 10, it is enough to replace the expression "**if** $a \notin a \circ x$ **then**" with "**if** $a \notin x \circ a$ **then**" and the word "$Right$" with "$Left$".

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

```
IsScalar:=proc(hypero::procedure,H::set,a,$)::boolean;
 local b::boolean,x;
 b:=true;
 for x in H do
  if numelems(aob(hypero,a,x))<>1 then
   b:=false;
   break;
  elif numelems(aob(hypero,x,a))<>1 then
   b:=false;
   break;
  end if;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
AllScalar:=proc(hypero::procedure,H::set,$)::set;
 local T::set,a;
 T:={};
 for a in H do
  if IsScalar(hypero,H,a) then
   T:=T union {a};
  end if;
 end do;
 return T;
 end proc:
#-------------------------------------------------------
IsScalarIdentity:=proc(hypero::procedure,H::set,e,$)::boolean;
 local b::boolean,x;
 b:=true;
 for x in H do
  if aob(hypero,e,x)<>{x} then
   b:=false;
   break;
  elif aob(hypero,x,e)<>{x} then
   b:=false;
   break;
  end if;
```

```
 end do;
 return b;
 end proc:
#-------------------------------------------------------
AllScalarIdentity:=proc(hypero::procedure,H::set,$)::set;
 local T::set,e;
 T:={};
 for e in H do
  if IsScalarIdentity(hypero,H,e) then
   T:=T union {e};
  end if;
 end do;
 return T:
 end proc:
#-------------------------------------------------------
IsIdentity:=proc(hypero::procedure,H::set,e,$)::boolean;
 local b::boolean,x;
 b:=true;
 for x in H do
  if (not (x in aob(hypero,e,x))) then
   b:=false;
   break;
  elif (not (x in aob(hypero,x,e))) then
   b:=false;
   break;
  end if;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
AllIdentity:=proc(hypero::procedure,H::set,$)::set;
 local T::set,e;
 T:={};
 for e in H do
  if IsIdentity(hypero,H,e) then
   T:=T union {e};
  end if;
 end do;
 return T;
 end proc:
#-------------------------------------------------------
IsInverse:=proc(hypero::procedure,H::set,E::set,aprime,a,$)::boolean;
 local b::boolean,e;
 b:=false;
```

```
 for e in E do
  if (e in aob(hypero,a,aprime)) then
   if (e in aob(hypero,aprime,a)) then
    b:=true;
    break;
   end if;
  end if;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
IsInvertible:=proc(hypero::procedure,H::set,E::set,aprime,$)::boolean;
 local b::boolean,a;
 b:=false;
 for a in H do
  if IsInverse(hypero,H,E,aprime,a) then
   b:=true;
   break;
  end if;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
InverseOf:=proc(hypero::procedure,H::set,E::set,aprime,$)::boolean;
 local T::set,a;
 T:={};
 for a in H do
  if IsInverse(hypero,H,E,aprime,a) then
   T:=T union {a};
  end if;
 end do;
 return T;
 end proc:
#-------------------------------------------------------
AllInvertible:=proc(hypero::procedure,H::set,E::set,$)::set;
 local T::set,aprime;
 T:={};
 for aprime in H do
  if IsInvertible(hypero,H,E,aprime) then
   T:=T union {aprime};
  end if;
 end do;
 return T;
 end proc:
```

```
#-------------------------------------------------------
IsZeroElems:=proc(hypero::procedure,H::set,zero,$)::boolean;
 local b::boolean,x;
 b:=true;
 for x in H do
  if aob(hypero,x,zero)<>{zero} then
   b:=false;
   break;
  elif aob(hypero,zero,x)<>{zero} then
   b:=false;
   break;
  end if;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
AllZeroElems:=proc(hypero::procedure,H::set,$)::set;
 local T::set,zero;
 T:={};
 for zero in H do
  if IsZeroElems(hypero,H,zero) then
   T:=T union {zero};
  end if;
 end do;
 return T;
 end proc:
#-------------------------------------------------------
IsRightSimplifable:=proc(hypero::procedure,H::set,a,$)::boolean;
 local b::boolean,x,y;
 b:=true;
 for x in H do
  for y in H do
   if ((aob(hypero,x,a)=aob(hypero,y,a)) and x<>y) then
    b:=false;
    break;
   end if;
  end do;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
AllRightSimplifable:=proc(hypero::procedure,H::set,$)::set;
 local T::set,a;
 T:={};
```

```
 for a in H do
  if IsRightSimplifable(hypero,H,a) then
   T:=T union {a};
  end if;
 end do;
 return T;
 end proc:
#-------------------------------------------------------
IsLeftSimplifable:=proc(hypero::procedure,H::set,a,$)::boolean;
 local b::boolean,x,y;
 b:=true;
 for x in H do
  for y in H do
   if ((aob(hypero,a,x)=aob(hypero,a,y)) and x<>y) then
    b:=false;
    break;
   end if;
  end do;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
AllLeftSimplifable:=proc(hypero::procedure,H::set,$)::set;
 local T::set,a;
 T:={};
 for a in H do
  if IsLeftSimplifable(hypero,H,a) then
   T:=T union {a};
  end if;
 end do;
 return T;
 end proc:
#-------------------------------------------------------
IsLeftAbsorb:=proc(hypero::procedure,H::set,a,$)::boolean;
 local b::boolean,x;
 b:=true;
 for x in H do
  if (not (a in aob(hypero,a,x))) then
   b:=false;
   break;
  end if;
 end do;
 return b;
 end proc:
```

```
#-------------------------------------------------------
AllLeftAbsorb:=proc(hypero::procedure,H::set,$)::set;
 local T::set,a;
 T:={};
 for a in H do
  if IsLeftAbsorb(hypero,H,a) then
   T:=T union {a};
  end if;
 end do;
 return T;
 end proc:
#-------------------------------------------------------
IsRightAbsorb:=proc(hypero::procedure,H::set,a,$)::boolean;
 local b::boolean,x;
 b:=true;
 for x in H do
  if (not (a in aob(hypero,x,a))) then
   b:=false;
   break;
  end if;
 end do;
 return b;
 end proc:
#-------------------------------------------------------
AllRightAbsorb:=proc(hypero::procedure,H::set,)::set;
 local T::set,a;
 T:={};
 for a in H do
  if IsRightAbsorb(hypero,H,a) then
   T:=T union {a};
  end if;
 end do;
 return T;
 end proc:
#-------------------------------------------------------
```

4. Conclusion

In this paper, first, we presented two methods to define a hypergroupoid by algorithm (subsection 3.1). These methods are simple, but any method can be used for this task, as long as the number and type of inputs and outputs are respected. Then, we presented algorithms for finding some specific elements in hypergroupoids (subsection 3.2). These specific elements are: scalars, scalar identities, identities,

inverses, zero elements, right simplifiable elements, left simplifiable elements, left absorbing-like elements and right absorbing-like elements.

Although, in this paper, all algorithms are presented for algebraic hyperstructures with one hyperoperation, but we can develop algorithms for algebraic hyperstructures with several operations and hyperoperations by defining procedures corresponding to each operation or hyperoperation in the algebraic hyperstructure and using the algorithms of this paper, both as a template and as a subroutine, for newer algorithms.

The complexity and optimization of the algorithms were not among the goals of the paper, so we did not study them. But these issues can be a suitable topic for future research. Also, we did not pay attention to the application of these algorithms in pure and interdisciplinary research. These issues can also be investigated in future research. Presenting algorithms based on definitions helps to understand their performance when applying definitions on algebraic hyperstructures. It can help to teach the concepts of algebraic hyperstructures.

Conflicts of Interest. The authors declare that they have no conflicts of interest regarding the publication of this article.

References

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

[2] B. Davvaz and V. Leoreanu-Fotea, *Hypergroup Theory*, World Scientific, 2022.

[3] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.12.2*, 2022, https://www. gap-system. org.

[4] H. Aghabozorgi, M. Jafarpour, M. K. Dolatabadi and I. Cristea, An algorithm to compute the number of Rosenberg hypergroups of order less than 7, *Ital. J. Pure Appl. Math.* **42** (2019) 262 − 270.

[5] M. B. Safari, B. Davvaz and V. Leoreanu-Fotea, Enumeration of 3- and 4-hypergroups on sets with two elements, *European J. Combin.* **44** (2015) 298 − 306, https://doi.org/10.1016/j.ejc.2014.08.01.

[6] G. Nordo, An algorithm on number of isomorphism classes of hypergroups of order 3, *Ital. J. Pure Appl. Math.* **2** (1997) 37 − 42.

[7] C. G. Massouros and C. Tsitouras, Enumeration of hypercompositional structures defined by binary relations, *Ital. J. Pure Appl. Math.* **28** (2011) 43 − 54.

[8] I. Cristea, M. Jafarpour, S. S. Mousavi and A. Soleymani, Enumeration of rosenberg hypergroups, *Comput. Math. Appl.* **60** (2010) 2753 − 2763, https://doi.org/10.1016/j.camwa.2010.09.027.

[9] C. Tsitouras and C. G. Massouros, Enumeration of rosenberg-type hypercompositional structures defined by binary relations, *European J. Combin.* **33** (2012) 1777 – 1786, https://doi.org/10.1016/j.ejc.2012.03.032.

[10] C. Tsitouras and C. G. Massouros, On enumeration of hypergroups of order 3, *Comput. Math. Appl.* **59** (2010) 519 – 523, https://doi.org/10.1016/j.camwa.2009.06.013.

[11] S. I. Spartalis and C. Mamaloukas, Hyperstructures associated with binary relations, *Comput. Math. Appl.* **51** (2006) 41 – 50, https://doi.org/10.1016/j.camwa.2005.07.011.

[12] J. Park and S.-C. Chung, On algorithms to compute some H_v-groups, *Korean J. Comput. Appl. Math.* **7** (2000) 433 – 453, https://doi.org/10.1007/BF03012204.

[13] B. Davvaz and T. Vougiouklis, *A Walk Through Weak Hyperstructures*, World Scientific Publishing Company, Singapore, 2018.

[14] B. Davvaz, *Semihypergroup Theory*, Academic Press, 2016.

[15] B. Davvaz, A. Dehghan Nezhad and M. M. Heidari, Inheritance examples of algebraic hyperstructures, *Inform. Sci.* **224** (2013) 180 – 187, https://doi.org/10.1016/j.ins.2012.10.023.

Aboutorab Pourhaghani
Department of Mathematical Sciences,
Yazd University,
Yazd, I. R. Iran
e-mail: pourhaghani@stu.yazd.ac.ir

Seid Mohammad Anvariyeh
Department of Mathematical Sciences,
Yazd University,
Yazd, I. R. Iran
e-mail: anvariyeh@yazd.ac.ir

Bijan Davvaz
Department of Mathematical Sciences,
Yazd University,
Yazd, I. R. Iran
e-mail: davvaz@yazd.ac.ir