Hyperalgorithms & Superhyperalgorithms: A Unified Framework for Higher-Order Computation

Takaaki Fujita [1,*]
[1]Independent Researcher, Shinjuku, Shinjuku-ku, Tokyo, Japan
Email: takaaki.fujita060@gmail.com

Abstract

An *algorithm* is a finite, well-defined computational procedure that transforms inputs into outputs through a structured sequence of steps, guaranteeing termination and correctness. A *multialgorithm* comprises multiple algorithms augmented with a selection mechanism that dynamically chooses the most appropriate procedure based on input characteristics or contextual conditions. While these concepts have deep roots in computer science and beyond, this paper introduces two novel generalizations: the *Hyperalgorithm* and the *Superhyperalgorithm*. By leveraging the mathematical frameworks of *hyperstructures* and *superhyperstructures*, respectively, we extend the classical notion of computation to higher-order operations on sets and iterated powersets. We present formal definitions, illustrative examples, and a preliminary analysis of their computational properties, laying the groundwork for a unified theory of higher-order algorithms.

Keywords: Algorithm; Multialgorithm; Hyperalgorithm; Superhyperalgorithm

1 Preliminaries and Definitions

This section provides an introduction to the foundational concepts and definitions required for the discussions in this paper. Throughout the paper, we restrict our attention to finite structures unless explicitly stated otherwise.

1.1 Powerset and n-th Powerset

In what follows, we employ the concepts of the powerset and the n-th powerset as fundamental building blocks for our later constructions.

Definition 1.1 (Set)**.** [14] A *set* is a collection of distinct objects, called elements, which are unambiguously defined. If A is a set and x is an element of A, we write $x \in A$. Sets are usually denoted by enclosing their elements in curly braces.

Definition 1.2 (Subset)**.** [14] For any two sets A and B, A is said to be a *subset* of B (written $A \subseteq B$) if every element of A is also an element of B:

$$\forall x \in A, \quad x \in B.$$

If additionally $A \neq B$, then A is called a *proper subset* of B, denoted $A \subset B$.

DOI: https://doi.org/10.54216/PAMDA.040104
Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

Definition 1.3 (Empty Set)**.** [14] The *empty set*, denoted $\emptyset$, is the unique set that contains no elements:

$$\forall x, \quad x \notin \emptyset.$$

It follows that $\emptyset$ is a subset of every set.

Definition 1.4 (Base Set)**.** [13] A *base set* S is the underlying set from which more elaborate structures, such as powersets and hyperstructures, are constructed. It is defined by

$$S = \{x \mid x \text{ belongs to a specified domain}\}.$$

All elements appearing in constructions like $\mathcal{P}(S)$ or $\mathcal{P}_n(S)$ are drawn from S.

Definition 1.5 (Powerset)**.** [13,19] The *powerset* of a set S, denoted $\mathcal{P}(S)$, is the collection of all subsets of S, including both $\emptyset$ and S itself:

$$\mathcal{P}(S) = \{A \mid A \subseteq S\}.$$

Definition 1.6 (n-th Powerset)**.** (cf.[24,25]) The n-th powerset of a set H, denoted $P_n(H)$, is defined recursively by:

$$P_1(H) = \mathcal{P}(H), \quad P_{n+1}(H) = \mathcal{P}(P_n(H)) \quad \text{for } n \geq 1.$$

Similarly, the n-th nonempty powerset, denoted $P_n^*(H)$, is given by:

$$P_1^*(H) = \mathcal{P}^*(H), \quad P_{n+1}^*(H) = \mathcal{P}^*(P_n^*(H)),$$

where $\mathcal{P}^*(H)$ denotes the powerset of H with the empty set omitted.

1.2 Hyperstructure and Superhyperstructure

To establish a robust theoretical foundation for hyperstructures[12,18] and superhyperstructures,[10,11] we introduce key definitions that formalize their properties.

Definition 1.7 (Classical Structure)**.** (cf.[23,25]) A *Classical Structure* is a mathematical system based on a nonempty set H that is endowed with one or more *classical operations* satisfying a prescribed set of axioms. A *classical operation* is a mapping

$$\#_0 : H^m \to H,$$

where $m \geq 1$ and H^m denotes the m-fold Cartesian product of H. Typical examples include operations like addition or multiplication in algebraic systems such as groups, rings, or fields.

Definition 1.8 (Hyperoperation)**.** (cf.[27,28])

A *hyperoperation* is a generalization of a binary operation in which the combination of two elements yields a set of elements rather than a single element. Formally, for a set S, a hyperoperation $\circ$ is defined by:

$$\circ : S \times S \to \mathcal{P}(S),$$

where $\mathcal{P}(S)$ represents the powerset of S.

Definition 1.9 (Hyperstructure)**.** (cf.[25,29])

A *hyperstructure* is an extension of a classical structure where the operations are defined on the powerset of a base set. It is formally given by:

$$\mathcal{H} = (\mathcal{P}(S), \circ),$$

with S as the base set and $\circ$ acting on subsets of S.

Definition 1.10 (SuperHyperOperations)**.** [25] Let H be a nonempty set and $P(H)$ its powerset. Define the n-th powerset $P^n(H)$ recursively by:

$$P^0(H) = H, \quad P^{k+1}(H) = P(P^k(H)) \quad \text{for } k \geq 0.$$

A *SuperHyperOperation* of order (m, n) is an m-ary operation

$$\circ^{(m,n)} : H^m \to P_*^n(H),$$

where $P_*^n(H)$ denotes the n-th powerset of H (either excluding the empty set, which we refer to as a classical-type operation, or including it, known as a Neutrosophic-type operation). These operations serve as higher-order generalizations of hyperoperations by capturing multi-level complexity through iterative powerset constructions.

DOI: https://doi.org/10.54216/PAMDA.040104

Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

Example 1.11 (Friend-of-Friend Operation as a SuperHyperOperation)**.** Let H be a finite set of people in a social network, and let $\mathcal{P}(H)$ denote its powerset. Define the binary SuperHyperOperation of order $(2,2)$,

$$\circ^{(2,2)} : H \times H \longrightarrow \mathcal{P}^2(H)$$

by

$$x \circ^{(2,2)} y = \big\{ \mathrm{Friends}(x),\ \mathrm{Friends}(y),\ \mathrm{Friends}(x) \cap \mathrm{Friends}(y) \big\},$$

where $\mathrm{Friends}(z) = \{\, w \in H : w \text{ is directly connected to } z \} \subseteq H$. Here:

- $x, y \in H$ are two individuals.
- $\mathrm{Friends}(x), \mathrm{Friends}(y) \in \mathcal{P}^1(H)$ are their direct friend sets.
- $\circ^{(2,2)}$ returns a 2-superset whose elements (each a subset of H) capture each person's friends and their mutual friends.

This SuperHyperOperation captures both first-order and second-order social connections in one higher-order operation.

Definition 1.12 (n-Superhyperstructure)**.** (cf.[9,25]) An *n-superhyperstructure* generalizes a hyperstructure by incorporating the n-th powerset of a base set. It is defined as:

$$\mathcal{SH}_n = (\mathcal{P}_n(S), \circ),$$

where S is the base set, $\mathcal{P}_n(S)$ is its n-th powerset, and $\circ$ is an operation on elements of $\mathcal{P}_n(S)$.

Example 1.13 (Cloud Infrastructure as a 3-Superhyperstructure)**.** Let S be the finite set of all physical servers in a global cloud provider. Form the iterated powersets:

$$\mathcal{P}^1(S) = \{\text{rack groupings}\}, \quad \mathcal{P}^2(S) = \{\text{data center assemblies}\}, \quad \mathcal{P}^3(S) = \{\text{regional partitions}\}.$$

Define the 3-superhyperstructure

$$\mathcal{SH}_3 = \big(\mathcal{P}_3(S), \cup\big),$$

where the operation $\cup$ acts on elements of $\mathcal{P}_3(S)$ by set-union at the region level. Concretely:

- Level 0 (S): individual servers.
- Level 1 ($\mathcal{P}^1(S)$): racks of servers.
- Level 2 ($\mathcal{P}^2(S)$): data centers composed of multiple racks.
- Level 3 ($\mathcal{P}^3(S)$): geographical regions grouping several data centers.
- The operation $\cup$ on two regions merges their data-center assemblies into a larger region.

This example illustrates a 3-superhyperstructure modeling hierarchical cloud infrastructure from servers up to global regions.

1.3 Algorithm

Algorithm is a finite, well-defined computational procedure that transforms inputs into outputs through a sequence of effective steps, ensuring termination and correctness.[4,20,22]

Definition 1.14 (Algorithm)**.** An *algorithm* is a finite, well-defined, and effective computational procedure that transforms an input into an output after a finite number of steps. An algorithm must satisfy the following properties:

DOI: https://doi.org/10.54216/PAMDA.040104
Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

(1) **Finiteness:** The algorithm terminates after a finite number of steps.

(2) **Definiteness:** Each step of the algorithm is precisely defined.

(3) **Input:** The algorithm accepts one or more inputs from a specified set I.

(4) **Output:** The algorithm produces one or more outputs in a set O.

(5) **Effectiveness:** Every operation is basic enough to be carried out in a finite amount of time.

Formally, an algorithm can be modeled as a total computable function

$$A : I \to O,$$

or equivalently as a Turing machine that halts on every input.

Example 1.15 (Euclidean Algorithm for GCD)**.** (cf.[8,17,21]) Consider the Euclidean algorithm for computing the greatest common divisor (GCD) of two positive integers a and b. The algorithm is defined by:

1. If $b = 0$, then output a and halt.
2. Otherwise, replace (a, b) with $(b, a \mod b)$ and repeat.

This algorithm terminates in a finite number of steps and returns $\gcd(a, b)$ as the output.

1.4 Multialgorithm and Parallel Algorithm

Multialgorithm is a collection of multiple algorithms with a selection mechanism that dynamically chooses the most suitable one based on input characteristics or conditions.[7,15,26]

Definition 1.16 (Multialgorithm)**.** A *multialgorithm* is a composite computational framework that consists of a finite collection of algorithms and a selection mechanism that determines, for each input $x \in I$, which algorithm to apply to produce the output. Formally, let

$$\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$$

be a finite set of algorithms, where each $A_i : I \to O$ is a total computable function. A *selection function*

$$\mu : I \to \{1, 2, \ldots, k\}$$

assigns to every input x an index $i = \mu(x)$, and the multialgorithm M is defined by

$$M(x) = A_{\mu(x)}(x) \quad \forall\, x \in I.$$

The selection function μ may depend on criteria such as input size, structure, or performance requirements.

Example 1.17 (Hybrid Sorting Algorithm)**.** (cf.[2,6,30]) A typical example of a multialgorithm is a hybrid sorting algorithm that chooses between Insertion Sort and Merge Sort based on the size of the input array. For an input array x of length n, define the selection function as:

$$\mu(x) = \begin{cases} 1, & \text{if } n \leq n_0, \\ 2, & \text{if } n > n_0, \end{cases}$$

where A_1 is Insertion Sort (efficient for small n) and A_2 is Merge Sort (efficient for large n), with n_0 as a threshold value. The multialgorithm then computes:

$$M(x) = A_{\mu(x)}(x).$$

This approach leverages the strengths of each algorithm based on the specific characteristics of the input.

DOI: https://doi.org/10.54216/PAMDA.040104

Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

A related concept is parallel algorithms.[1,16] For reference, the mathematical definition of parallel algorithms is provided below.

Definition 1.18 (Parallel Algorithm)**.** Let I be a set of inputs, and let O be a set of outputs. A *parallel algorithm* consists of:

- A finite set of *processing units* $P = \{P_1, P_2, \ldots, P_m\}$, where each P_i executes a part of the computation independently.
- A *task decomposition function* $D : I \to \mathcal{P}(S)$, where S is the set of subtasks and $D(x)$ partitions an input $x \in I$ into subtasks $S_1, S_2, \ldots, S_k$ for parallel execution.
- A set of *local computations* $A_i : S_i \to O_i$ performed by each processing unit P_i, where O_i is the local output space of P_i.
- A *result aggregation function* $R : O_1 \times O_2 \times \cdots \times O_m \to O$ that combines local outputs to produce the final result.

Thus, a parallel algorithm is a tuple:

$$\mathcal{A} = (P, D, \{A_i\}_{i=1}^m, R)$$

that computes the final output as:

$$\mathcal{A}(x) = R(A_1(S_1), A_2(S_2), \ldots, A_m(S_m)), \quad \text{where } D(x) = \{S_1, S_2, \ldots, S_m\}.$$

Example 1.19 (Parallel Merge Sort)**.** (cf.[3,5]) Consider an unsorted array A of n elements that needs to be sorted. A parallel merge sort algorithm works as follows:

1. **Task Decomposition:** Divide the array A into m subarrays $A_1, A_2, \ldots, A_m$ of (approximately) equal size, where m is the number of available processing units.
2. **Local Computation:** Each processor P_i independently sorts its assigned subarray A_i using a sequential sorting algorithm (for example, merge sort).
3. **Aggregation:** In a parallel merging phase, the sorted subarrays are combined using a multiway merge operation to produce the final sorted array.

Under ideal conditions, the overall time complexity can be reduced from $O(n \log n)$ in the sequential case to approximately $O\left(\frac{n \log n}{m}\right)$ in the parallel implementation.

2 Result in this paper

In this section, we introduce the notions of a *Hyperalgorithm* and an n*-Superhyperalgorithm*.

2.1 Hyperalgorithm

The definition of a Hyperalgorithm is provided below.

Definition 2.1 (Hyperalgorithm)**.** Let I and O be nonempty sets. Suppose that $\mathcal{H} = (\mathcal{P}(O), \circ)$ is a *hyperstructure* on O (i.e., the operation $\circ$ is defined on the powerset $\mathcal{P}(O)$). Let

$$\mathcal{A} = \{A_i : I \to O \mid i \in J\}$$

be a finite collection of algorithms, where each A_i is a total computable function. A *Hyperalgorithm* is a function

$$A : I \to \mathcal{P}(O)$$

DOI: https://doi.org/10.54216/PAMDA.040104

Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

defined by

$$A(x) = \circ\Big(\{A_i(x) : i \in J\}\Big) \quad \text{for all } x \in I.$$

In other words, a Hyperalgorithm aggregates the outputs of a family of algorithms via a hyperoperation $\circ$ on O. Notice that if $\circ$ is chosen to be degenerate (i.e., always returns a singleton), then the Hyperalgorithm reduces to a conventional multialgorithm.

Example 2.2 (Automotive Fault Diagnosis as a Hyperalgorithm)**.** Let

$$I = \{\text{“engine won't start”, “battery light on”, “clicking noise”}\}$$

be the set of possible symptom-profiles, and let

$$O = \{\text{“dead battery”, “starter motor”, “alternator failure”, “fuel pump”}\}$$

be the set of diagnostic fault codes.

Define two diagnostic algorithms:

$$A_1 \colon I \to O, \quad A_1(x) = \begin{cases} \{\text{dead battery, starter motor}\}, & x = \{\text{engine won't start, clicking noise}\}, \\ \{\text{alternator failure}\}, & x \supseteq \{\text{battery light on}\}, \\ \emptyset, & \text{otherwise}, \end{cases}$$

$$A_2 \colon I \to O, \quad A_2(x) = \begin{cases} \{\text{dead battery, alternator failure}\}, & x \supseteq \{\text{battery light on}\}, \\ \{\text{fuel pump}\}, & x = \{\text{engine won't start}\}, \\ \emptyset, & \text{otherwise}. \end{cases}$$

We lift these to a Hyperalgorithm $A \colon I \to \mathcal{P}(O)$ by choosing the union as our hyperoperation:

$$\circ(S) \;=\; \bigcup S.$$

Then for any symptom profile $x \subseteq I$,

$$A(x) = \circ\big(\{A_1(x),\, A_2(x)\}\big) = A_1(x) \;\cup\; A_2(x).$$

For example, if $x = \{\text{engine won't start, battery light on}\}$, then

$$A_1(x) = \{\text{alternator failure}\}, \quad A_2(x) = \{\text{dead battery, alternator failure}\},$$

so

$$A(x) = \{\text{dead battery, alternator failure}\}.$$

This Hyperalgorithm thus captures both rule-based and data-driven diagnostics, aggregating their outputs into a combined set of candidate faults.

Example 2.3 (House Price Interval Prediction as a Hyperalgorithm)**.** Let

$$I = \mathbb{R}^d$$

be the set of d-dimensional feature vectors describing residential properties, and let

$$O = \mathbb{R}$$

denote predicted sale prices (in thousands of dollars). Consider three regression models:

$$\begin{aligned} A_1 &\colon I \to O, && \text{(linear regression)} \\ A_2 &\colon I \to O, && \text{(decision-tree regression)} \\ A_3 &\colon I \to O, && \text{(random-forest regression)} \end{aligned}$$

each of which is a total computable function on feature vectors.

DOI: https://doi.org/10.54216/PAMDA.040104
Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

Define the hyperoperation

$$\circ\colon \mathcal{P}(O) \to \mathcal{P}(O) \quad \text{by} \quad \circ\big(S\big) = \big[\min S,\ \max S\big] = \{\, y \in O : \min S \le y \le \max S\}\,,$$

which returns the closed interval spanning all predictions in S.

Then the Hyperalgorithm

$$A\colon I \longrightarrow \mathcal{P}(O) \quad \text{is defined by} \quad A(x) = \circ\big(\{A_1(x),\ A_2(x),\ A_3(x)\}\big).$$

Example. Suppose for a particular house feature vector x the models output

$$A_1(x) = 250, \quad A_2(x) = 275, \quad A_3(x) = 260.$$

Then

$$A(x) = \circ(\{250, 275, 260\}) = [\,250,\ 275\,],$$

indicating that the ensemble predicts the sale price will lie between \$250k and \$275k. This Hyperalgorithm thus encapsulates model disagreement as a continuous interval of plausible values.

Theorem 2.4 (Multialgorithm as a Special Case of Hyperalgorithm)**.** *Every multialgorithm is a special case of a hyperalgorithm.*

Proof. Let $M : I \to O$ be a multialgorithm defined by

$$M(x) = A_{\mu(x)}(x)$$

for a selection function $\mu : I \to J$ and a collection of algorithms $\{A_i\}_{i\in J}$. Define a hyperoperation $\circ$ on $\mathcal{P}(O)$ such that for any singleton subset $S = \{y\}$, $\circ(S) = S$. Now, define the hyperalgorithm

$$A(x) = \circ\Big(\{A_i(x) : i \in J\}\Big).$$

Since the selection function μ picks a unique algorithm for each input, we have

$$A(x) = \{A_{\mu(x)}(x)\},$$

which is a singleton. Thus, the multialgorithm M is recovered from A, showing that every multialgorithm is a special case of a hyperalgorithm. □

Theorem 2.5 (Computability of Hyperalgorithms)**.** *Let I and O be countable sets, and let $\mathcal{A} = \{A_i : I \to O \mid i \in J\}$ be a finite family of total computable functions. Suppose $\circ : \mathcal{P}(O) \to \mathcal{P}(O)$ is a computable hyperoperation on finite subsets of O. Then the Hyperalgorithm*

$$A : I \longrightarrow \mathcal{P}(O), \quad A(x) = \circ\big(\{A_i(x) : i \in J\}\big)$$

is itself a total computable function.

Proof. On input $x \in I$:

1. For each $i \in J$, compute $y_i = A_i(x)$. Since each A_i is total computable and J is finite, this yields the finite set $\{y_i : i \in J\} \subseteq O$.
2. Apply the computable hyperoperation $\circ$ to this finite set, producing $\circ(\{y_i\}) \in \mathcal{P}(O)$.

Each step is a computable procedure on finite data, so $A(x)$ is computed in finite time. Hence A is a total computable function from I to $\mathcal{P}(O)$. □

Theorem 2.6 (Monotonicity under Inclusion)**.** *Suppose the hyperoperation $\circ$ is* monotone *in the sense that whenever $S_1, S_2 \subseteq O$ satisfy $S_1 \subseteq S_2$, we have $\circ(S_1) \subseteq \circ(S_2)$. Let $\mathcal{A}_{J_1}$ and $\mathcal{A}_{J_2}$ be two finite algorithm families with $J_1 \subseteq J_2$, and let A_{J_1}, A_{J_2} be their corresponding Hyperalgorithms. Then for every $x \in I$,*

$$A_{J_1}(x) \ \subseteq\ A_{J_2}(x).$$

DOI: https://doi.org/10.54216/PAMDA.040104

Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

Proof. Fix $x \in I$. Since $J_1 \subseteq J_2$, the set of outputs $\{A_i(x) : i \in J_1\}$ is a subset of $\{A_i(x) : i \in J_2\}$. Monotonicity of $\circ$ then gives

$$A_{J_1}(x) = \circ\big(\{A_i(x) : i \in J_1\}\big) \subseteq \circ\big(\{A_i(x) : i \in J_2\}\big) = A_{J_2}(x).$$

Hence adding more component algorithms can only enlarge (or leave unchanged) the Hyperalgorithm's output. □

Theorem 2.7 (Idempotence under Self-Application)**.** *If the hyperoperation $\circ$ satisfies $\circ\big(\circ(S)\big) = \circ(S)$ for every finite $S \subseteq O$, then the Hyperalgorithm A is idempotent in the sense that*

$$A'(x) = \circ\big(\{\, A(x)\}\big) = A(x) \quad \textit{for all } x \in I.$$

Proof. For each $x \in I$, let $S_x = \{A_i(x) : i \in J\}$. By definition,

$$A(x) = \circ(S_x).$$

If we now form $\{A(x)\} \subseteq \mathcal{P}(O)$, then self-application of $\circ$ yields

$$A'(x) = \circ\big(\{A(x)\}\big) = \circ\big(\{\circ(S_x)\}\big) = \circ(S_x) = A(x),$$

where the third equality uses the idempotence hypothesis $\circ(\circ(S)) = \circ(S)$. Thus the Hyperalgorithm is unchanged by one additional round of hyperoperation. □

Theorem 2.8 (Time Complexity of Hyperalgorithms)**.** *Let I be a set of inputs encoded in bit-strings of length n, and let $\mathcal{A} = \{A_i \colon I \to O \mid i = 1, \ldots, k\}$ be a finite family of total computable component algorithms. Suppose:*

- *Each A_i runs in time $T_i(n)$.*
- *The hyperoperation $\circ \colon \mathcal{P}(O) \to \mathcal{P}(O)$ on k inputs runs in time $H(k)$.*

Then the Hyperalgorithm

$$A(x) = \circ\big(\{A_i(x) \colon i = 1, \ldots, k\}\big)$$

runs in time

$$T_{\text{total}}(n) = \sum_{i=1}^{k} T_i(n) \;+\; H(k).$$

In particular, if k is a fixed constant and each T_i and H are polynomial functions of n, then A runs in polynomial time.

Proof. On input x of length n, we first compute each $y_i = A_i(x)$ for $i = 1, \ldots, k$; this takes $\sum_{i=1}^{k} T_i(n)$ steps since the A_i run sequentially. We then collect the finite set $\{y_1, \ldots, y_k\}$ and apply the hyperoperation $\circ$, which by hypothesis takes $H(k)$ steps. Summing these two phases yields the stated bound. □

Theorem 2.9 (Space Complexity of Hyperalgorithms)**.** *Under the same setup, suppose moreover that:*

- *Each output $A_i(x)$ can be represented in space at most $S_i(n)$.*
- *The hyperoperation $\circ$ on k inputs requires working space $W(k)$ (in addition to storing its inputs and output).*

Then the Hyperalgorithm A can be implemented in space

$$S_{\text{total}}(n) = \sum_{i=1}^{k} S_i(n) \;+\; W(k).$$

If k is fixed and each S_i and W are polynomial in n, then A uses polynomial space.

Proof. To compute $A(x)$, we must store each intermediate result $y_i = A_i(x)$, using at most $\sum_{i=1}^{k} S_i(n)$ space. Then applying $\circ$ may use an additional $W(k)$ workspace (e.g. for temporary registers or indexing), but does not overwrite the stored y_i. Finally, we produce the output hyper-set, whose storage is already accounted for within $\sum_i S_i(n)$. Therefore the total space required is the sum of the component storage plus $W(k)$, as claimed. □

DOI: https://doi.org/10.54216/PAMDA.040104

Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

2.2 n-Superhyperalgorithm

The definition of a n-Superhyperalgorithm is provided below.

Definition 2.10 (n-Superhyperalgorithm)**.** Let I and O be nonempty sets. Suppose that $\mathcal{SH}_n = (\mathcal{P}_n(O), \circ^{(n)})$ is an n-superhyperstructure on O, where $\mathcal{P}_n(O)$ denotes the n-th powerset of O (constructed recursively as in the preliminaries) and $\circ^{(n)}$ is an n-superhyperoperation on $\mathcal{P}_n(O)$. Let

$$\mathcal{A} = \{A_i : I \to O \mid i \in J\}$$

be a finite collection of algorithms. An *n-Superhyperalgorithm* is a function

$$A^{(n)} : I \to \mathcal{P}_n(O)$$

defined by

$$A^{(n)}(x) = \circ^{(n)}\Big(\{A_i(x) : i \in J\}\Big) \quad \text{for all } x \in I.$$

This definition generalizes the hyperalgorithm by operating within the richer structure of the n-th powerset. In particular, when $n = 1$ (so that $\mathcal{P}_1(O) = \mathcal{P}(O)$), the n-superhyperalgorithm coincides with the hyperalgorithm.

Example 2.11 (Automotive Fault Diagnosis via a 2-Superhyperalgorithm)**.** Let

$$I = \{\text{ “engine won't start”, “battery light on”, “clicking noise”}\}$$

be the set of symptom profiles, and let

$$O = \{\text{ dead battery, starter motor, alternator failure, fuel pump}\}$$

be the set of candidate faults.

Define two diagnostic sub-algorithms:

$$A_1(x) = \begin{cases} \{\text{dead battery, starter motor}\}, & x \supseteq \{\text{engine won't start, clicking noise}\}, \\ \{\text{alternator failure}\}, & x \supseteq \{\text{battery light on}\}, \\ \emptyset, & \text{otherwise}, \end{cases}$$

$$A_2(x) = \begin{cases} \{\text{dead battery, alternator failure}\}, & x \supseteq \{\text{battery light on}\}, \\ \{\text{fuel pump}\}, & x = \{\text{engine won't start}\}, \\ \emptyset, & \text{otherwise}. \end{cases}$$

We embed each output as a singleton-set:

$$S(x) = \big\{\{A_1(x)\},\, \{A_2(x)\}\big\} \;\subset\; \mathcal{P}^1\big(\mathcal{P}^1(O)\big) = \mathcal{P}_2(O).$$

Define the 2-superhyperoperation

$$\circ^{(2)}(S) = \big\{\, U \cup V,\; U \cap V,\; U \triangle V \mid \{U, V\} \subseteq S\big\},$$

which returns, for any two singleton-sets $U, V \subseteq O$, their union, intersection, and symmetric difference.

Then the 2-Superhyperalgorithm

$$A^{(2)} \colon I \;\longrightarrow\; \mathcal{P}_2(O) \quad \text{is} \quad A^{(2)}(x) = \circ^{(2)}\big(\{\{A_1(x)\}, \{A_2(x)\}\}\big).$$

For instance, if $x = \{\text{engine won't start, battery light on}\}$, then

$$A_1(x) = \{\text{alternator failure}\}, \quad A_2(x) = \{\text{dead battery, alternator failure}\},$$

so

$$S(x) = \big\{\{\text{alternator failure}\},\, \{\text{dead battery, alternator failure}\}\big\},$$

and

$$A^{(2)}(x) = \Big\{\{\text{dead battery, alternator failure}\}, \{\text{alternator failure}\}, \{\text{dead battery}\}\Big\},$$

explicitly listing the union, intersection, and symmetric difference of the two candidate-fault sets. This output lies in $\mathcal{P}_2(O)$ and demonstrates how a 2-Superhyperalgorithm encodes both individual diagnoses and their combined relationships.

DOI: https://doi.org/10.54216/PAMDA.040104

Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

Example 2.12 (3-Superhyperalgorithm for Power Computation)**.** Let $I = \mathbb{N}$ and $O = \mathbb{N}$. Define two base algorithms:

$$A_1(x) = x^2, \qquad A_2(x) = x^3.$$

For each $x \in \mathbb{N}$, form the embedded singleton outputs

$$S_0(x) = \big\{\{A_1(x)\}, \{A_2(x)\}\big\} = \big\{\{x^2\}, \{x^3\}\big\} \subseteq \mathcal{P}^1(O).$$

Second-level (2-super) step: apply the 2-superhyperoperation $\circ^{(2)}(S) = \mathcal{P}(S)$. Thus

$$S_1(x) = \circ^{(2)}\big(S_0(x)\big) = \mathcal{P}\big(\{\{x^2\}, \{x^3\}\}\big) = \Big\{\emptyset,\ \{\{x^2\}\},\ \{\{x^3\}\},\ \{\{x^2\}, \{x^3\}\}\Big\} \subseteq \mathcal{P}^2(O).$$

Third-level (3-super) step: apply the 3-superhyperoperation $\circ^{(3)}(T) = \mathcal{P}(T)$. Hence the 3-Superhyperalgorithm $A^{(3)} : I \to \mathcal{P}^3(O)$ is

$$A^{(3)}(x) = \circ^{(3)}\big(S_1(x)\big) = \mathcal{P}\big(S_1(x)\big) = \big\{\, U : U \subseteq S_1(x)\big\},$$

a subset of $\mathcal{P}^3(O)$. In particular, for $x = 2$:

$$S_0(2) = \{\{4\}, \{8\}\}, \quad S_1(2) = \big\{\emptyset, \{\{4\}\}, \{\{8\}\}, \{\{4\}, \{8\}\}\big\},$$

$$A^{(3)}(2) = \mathcal{P}\big(S_1(2)\big) = \big\{\, U : U \subseteq S_1(2)\big\}, \quad \big|A^{(3)}(2)\big| = 16.$$

Thus $A^{(3)}(2)$ is the collection of all 16 possible subcollections of $\{\emptyset, \{\{4\}\}, \{\{8\}\}, \{\{4\}, \{8\}\}\}$, vividly illustrating how a 3-Superhyperalgorithm returns richly nested multilevel information about the inputs.

Theorem 2.13 (n-Superhyperalgorithm Generalizes Hyperalgorithm)**.** *Every hyperalgorithm is a special case of an n-superhyperalgorithm.*

Proof. Let $A : I \to \mathcal{P}(O)$ be a hyperalgorithm defined as in the previous section. Notice that $\mathcal{P}(O)$ is naturally embedded in $\mathcal{P}_n(O)$ for any $n \geq 1$ (since the first powerset is a subset of the n-th powerset). Define the mapping $\iota : \mathcal{P}(O) \to \mathcal{P}_n(O)$ by

$$\iota(S) = S \quad \text{for every } S \in \mathcal{P}(O).$$

Then, define the n-superhyperalgorithm $A^{(n)} : I \to \mathcal{P}_n(O)$ by

$$A^{(n)}(x) = \iota\big(A(x)\big).$$

For every $x \in I$, we have $A^{(n)}(x) = A(x)$ (viewed as an element of $\mathcal{P}_n(O)$). Therefore, the hyperalgorithm A is embedded as a special case of the n-superhyperalgorithm $A^{(n)}$. □

Theorem 2.14 (Computability of n-Superhyperalgorithms)**.** *Let I and O be countable sets, and let $\mathcal{A} = \{A_i : I \to O \mid i \in J\}$ be a finite family of total computable functions. Suppose the n-superhyperoperation*

$$\circ^{(n)} : \mathcal{P}_n(O) \longrightarrow \mathcal{P}_n(O)$$

is computable on finite inputs. Then the induced n-Superhyperalgorithm

$$A^{(n)} : I \longrightarrow \mathcal{P}_n(O), \qquad A^{(n)}(x) = \circ^{(n)}\big(\{A_i(x) : i \in J\}\big)$$

is a total computable function.

Proof. On input $x \in I$:

1. Compute each $y_i = A_i(x)$. Since J is finite and each A_i is total computable, this yields the finite multiset $\{y_i\} \subseteq O$.
2. Embed each y_i into $\mathcal{P}_n(O)$ by iterated singletons, forming an input in $\mathcal{P}_n(O)$.

DOI: https://doi.org/10.54216/PAMDA.040104

Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

3. Apply the computable n-superhyperoperation $\circ^{(n)}$ to this finite collection, producing $A^{(n)}(x) \in \mathcal{P}_n(O)$.

Each step terminates in finite time; hence $A^{(n)}$ is total computable. □

Theorem 2.15 (Monotonicity under Algorithm Inclusion)**.** *Assume $\circ^{(n)}$ is monotone in the sense that whenever $\{X_i\}, \{Y_i\} \subseteq \mathcal{P}_n(O)$ satisfy $X_i \subseteq Y_i$ for all i, then $\circ^{(n)}(\{X_i\}) \subseteq \circ^{(n)}(\{Y_i\})$. Let $\mathcal{A}_1 \subseteq \mathcal{A}_2$ be two finite families of algorithms, with corresponding n-Superhyperalgorithms $A_1^{(n)}$, $A_2^{(n)}$. Then for every $x \in I$,*

$$A_1^{(n)}(x) \ \subseteq \ A_2^{(n)}(x).$$

Proof. Fix x. Since $\mathcal{A}_1 \subseteq \mathcal{A}_2$, we have $\{A_i(x) : i \in \mathcal{A}_1\} \subseteq \{A_i(x) : i \in \mathcal{A}_2\}$ as subsets of O, and hence their embeddings into $\mathcal{P}_n(O)$ satisfy componentwise inclusion. Monotonicity of $\circ^{(n)}$ implies

$$A_1^{(n)}(x) = \circ^{(n)}\big(\{A_i(x) : i \in \mathcal{A}_1\}\big) \ \subseteq \ \circ^{(n)}\big(\{A_i(x) : i \in \mathcal{A}_2\}\big) = A_2^{(n)}(x).$$

□

Theorem 2.16 (Flattening to Hyperalgorithms)**.** *Let $\varphi_k : \mathcal{P}_n(O) \to \mathcal{P}_{n-k}(O)$ be the k-fold "flattening" map given by successive union. If $A^{(n)}$ is an n-Superhyperalgorithm, then for each $k \leq n$, the composition*

$$A^{(n-k)} = \varphi_k \circ A^{(n)} : I \longrightarrow \mathcal{P}_{n-k}(O)$$

is a well-defined $(n-k)$-Superhyperalgorithm. In particular, for $k = n-1$ it yields an ordinary hyperalgorithm.

Proof. Since $A^{(n)}(x) \in \mathcal{P}_n(O)$ for every $x \in I$, applying φ_k produces an element of $\mathcal{P}_{n-k}(O)$. The operation $\circ^{(n-k)}$ on $\mathcal{P}_{n-k}(O)$ can be defined by $\circ^{(n-k)} = \varphi_k \circ \circ^{(n)}$, and one checks directly that this satisfies the axioms of an $(n-k)$-superhyperoperation. Hence $A^{(n-k)}$ is a valid $(n-k)$-Superhyperalgorithm. □

Theorem 2.17 (Idempotence under Self-Application)**.** *If $\circ^{(n)}$ satisfies $\circ^{(n)}(\{\circ^{(n)}(S)\}) = \circ^{(n)}(S)$ for all finite $S \subseteq \mathcal{P}_n(O)$, then the associated Superhyperalgorithm $A^{(n)}$ is idempotent:*

$$\forall x \in I, \qquad \circ^{(n)}\big(\{A^{(n)}(x)\}\big) = A^{(n)}(x).$$

Proof. Let $S_x = \{A_i(x) : i \in J\} \subseteq O$ embedded into $\mathcal{P}_n(O)$. Then

$$A^{(n)}(x) = \circ^{(n)}(S_x),$$

and self-application gives

$$\circ^{(n)}\big(\{A^{(n)}(x)\}\big) = \circ^{(n)}\big(\{\circ^{(n)}(S_x)\}\big) = \circ^{(n)}(S_x) = A^{(n)}(x).$$

The second equality uses the idempotence property of $\circ^{(n)}$. □

Theorem 2.18 (Closure under Composition)**.** *Let*

$$A^{(n)} : I \to \mathcal{P}_n(O) \quad \textit{and} \quad B^{(n)} : O \to \mathcal{P}_n(P)$$

be an n-Superhyperalgorithm on (I, O) with superhyperoperation $\circ_O^{(n)}$, and another on (O, P) with superhyperoperation $\circ_P^{(n)}$. Then the composite

$$(B \circ A)^{(n)} : I \ \longrightarrow \ \mathcal{P}_n(P), \quad (B \circ A)^{(n)}(x) = \circ_P^{(n)}\Big(\big\{\, b : b \in B^{(n)}(y),\, y \in A^{(n)}(x)\big\}\Big)$$

is again an n-Superhyperalgorithm.

Proof. For each $x \in I$, $A^{(n)}(x) \subseteq \mathcal{P}_n(O)$. Write $\mathcal{Y} = A^{(n)}(x)$. Then each $y \in \mathcal{Y} \subseteq O$ yields $B^{(n)}(y) \subseteq \mathcal{P}_n(P)$. Collecting all outputs $\{B^{(n)}(y) : y \in \mathcal{Y}\}$ gives a finite subset of $\mathcal{P}_n(P)$, to which we apply $\circ_P^{(n)}$. This follows the same pattern as the definition of an n-Superhyperalgorithm, so the composite is of the required form. □

DOI: https://doi.org/10.54216/PAMDA.040104

Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

Theorem 2.19 (Time Complexity Bound)**.** *Assume each component algorithm $A_i \in \mathcal{A}$ runs in time at most $T_i(|x|)$ on input x, and that the n-superhyperoperation $\circ^{(n)}$ on $k = |\mathcal{A}|$ inputs runs in time $H^{(n)}(k)$. Then the n-Superhyperalgorithm*

$$A^{(n)}(x) = \circ^{(n)}\big(\{A_i(x)\}\big)$$

runs in time

$$T_{\text{total}}(x) = \sum_{i \in J} T_i(|x|) \; + \; H^{(n)}\big(|J|\big).$$

In particular, if $|J|$ and n are fixed constants and each T_i and $H^{(n)}$ are polynomial, then $A^{(n)}$ runs in polynomial time.

Proof. Computing each $A_i(x)$ takes $T_i(|x|)$, summing over $i \in J$. Gathering these outputs into the input for $\circ^{(n)}$ is $O(|J|)$, and applying the superhyperoperation takes $H^{(n)}(|J|)$. Adding these yields the stated bound. □

Theorem 2.20 (Space Complexity and Output-Size Bound)**.** *Let $|O| = m$ be the size of the output domain. Then $|\mathcal{P}_n(O)| \leq 2^{2^{\cdots^{2^m}}}$ (an n-fold tower of 2's). Consequently, in the worst case, the space required to represent $A^{(n)}(x)$ is $O\big(\big|\mathcal{P}_n(O)\big|\big)$. If each $A_i(x)$ produces at most s bits and $\circ^{(n)}$ produces an output of size at most $S(k)$ for $k = |J|$ inputs, then the total space is*

$$S\big(|J|\big) \; + \; \sum_{i \in J} s,$$

which is constant if n, $|J|$, and s are fixed.

Proof. By definition, $\mathcal{P}^1(O)$ has at most 2^m elements, $\mathcal{P}^2(O)$ at most 2^{2^m}, and so on, yielding the n-fold tower. Representing $A^{(n)}(x)$ thus requires space proportional to the size of its encoding within $\mathcal{P}_n(O)$, bounded by this tower. If we restrict to fixed n and small outputs, the space reduces to the sum of the sizes of each $A_i(x)$ plus the space to store $\circ^{(n)}$'s result. □

3 Conclusion and Future Works

In this paper, we introduced two novel computational paradigms: the *Hyperalgorithm* and the *Superhyperalgorithm*. By building on the algebraic frameworks of *hyperstructures* and *superhyperstructures*, we have extended classical algorithms to operate on higher-order sets and iterated powersets, opening new avenues for representing and combining computational processes.

Funding

This study did not receive any financial or external support from organizations or individuals.

Acknowledgments

We extend our sincere gratitude to everyone who provided insights, inspiration, and assistance throughout this research. We particularly thank our readers for their interest and acknowledge the authors of the cited works for laying the foundation that made our study possible. We also appreciate the support from individuals and institutions that provided the resources and infrastructure needed to produce and share this paper. Finally, we are grateful to all those who supported us in various ways during this project.

Data Availability

This research is purely theoretical, involving no data collection or analysis. We encourage future researchers to pursue empirical investigations to further develop and validate the concepts introduced here.

Ethical Approval

As this research is entirely theoretical in nature and does not involve human participants or animal subjects, no ethical approval is required.

Conflicts of Interest

The authors confirm that there are no conflicts of interest related to the research or its publication.

Disclaimer

This work presents theoretical concepts that have not yet undergone practical testing or validation. Future researchers are encouraged to apply and assess these ideas in empirical contexts. While every effort has been made to ensure accuracy and appropriate referencing, unintentional errors or omissions may still exist. Readers are advised to verify referenced materials on their own. The views and conclusions expressed here are the authors' own and do not necessarily reflect those of their affiliated organizations.

References

[1] Guy E Blelloch. Programming parallel algorithms. *Communications of the ACM*, 39(3):85–97, 1996.

[2] You-Rong Chen, Chien-Chia Ho, Wei-Ting Chen, and Pei-Yin Chen. A low-cost pipelined architecture based on a hybrid sorting algorithm. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 71(2):717–730, 2023.

[3] Richard Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.

[4] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[5] Andrew Davidson, David Tarjan, Michael Garland, and John D Owens. *Efficient parallel merge sort for fixed and variable length keys*. IEEE, 2012.

[6] Luciana De Micco, Mariano L Acosta, and Maximiliano Antonelli. Hybrid sorting algorithm implemented by high level synthesis. *IEEE Latin America Transactions*, 18(02):430–437, 2019.

[7] Pedro S de Souza and Sarosh N Talukdar. Asynchronous organizations for multi-algorithm problems. In *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, pages 286–293, 1993.

[8] John D Dixon. The number of steps in the euclidean algorithm. *Journal of number theory*, 2(4):414–422, 1970.

[9] Takaaki Fujita. *Advancing Uncertain Combinatorics through Graphization, Hyperization, and Uncertainization: Fuzzy, Neutrosophic, Soft, Rough, and Beyond*. Biblio Publishing, 2025.

[10] Takaaki Fujita. A concise review on various concepts of superhyperstructures. *Preprint*, 2025.

[11] Takaaki Fujita. Expanding horizons of plithogenic superhyperstructures: Applications in decision-making, control, and neuro systems. *Advancing Uncertain Combinatorics through Graphization, Hyperization, and Uncertainization: Fuzzy, Neutrosophic, Soft, Rough, and Beyond*, page 416, 2025.

[12] Takaaki Fujita. A theoretical exploration of hyperconcepts: Hyperfunctions, hyperrandomness, hyperdecision-making, and beyond (including a survey of hyperstructures). *Advancing Uncertain Combinatorics through Graphization, Hyperization, and Uncertainization: Fuzzy, Neutrosophic, Soft, Rough, and Beyond*, 344(498):111, 2025.

[13] Takaaki Fujita and Florentin Smarandache. Superhypergraph neural networks and plithogenic graph neural networks: Theoretical foundations. 2025.

[14] Thomas Jech. *Set theory: The third millennium edition, revised and expanded*. Springer, 2003.

DOI: https://doi.org/10.54216/PAMDA.040104
Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024

[15] EJC Kelkboom, Xuebing Zhou, Jeroen Breebaart, Raymond NJ Veldhuis, and Christoph Busch. Multi-algorithm fusion with template protection. In *2009 IEEE 3rd international conference on biometrics: Theory, applications, and systems*, pages 1–8. IEEE, 2009.

[16] F Thomson Leighton. *Introduction to parallel algorithms and architectures: Arrays· trees· hypercubes*. Elsevier, 2014.

[17] Th Motzkin. The euclidean algorithm. 1949.

[18] Michal Novák, Štepán Křehlík, and Kyriakos Ovaliadis. Elements of hyperstructure theory in uwsn design and data aggregation. *Symmetry*, 11(6):734, 2019.

[19] Judith Roitman. *Introduction to modern set theory*, volume 8. John Wiley & Sons, 1990.

[20] Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-wesley professional, 2011.

[21] Jeffrey Shallit. Origins of the analysis of the euclidean algorithm. *Historia Mathematica*, 21(4):401–419, 1994.

[22] Steven S Skiena. *The algorithm design manual*, volume 2. Springer, 2008.

[23] F. Smarandache. Introduction to superhyperalgebra and neutrosophic superhyperalgebra. *Journal of Algebraic Hyperstructures and Logical Algebras*, 2022.

[24] Florentin Smarandache. Extension of hyperalgebra to superhyperalgebra and neutrosophic superhyperalgebra (revisited). In *International Conference on Computers Communications and Control*, pages 427–432. Springer, 2022.

[25] Florentin Smarandache. Foundation of superhyperstructure & neutrosophic superhyperstructure. *Neutrosophic Sets and Systems*, 63(1):21, 2024.

[26] Kouichi Takahashi, Kazunari Kaizu, Bin Hu, and Masaru Tomita. A multi-algorithm, multi-timescale method for cell simulation. *Bioinformatics*, 20(4):538–546, 2004.

[27] Souzana Vougioukli. Helix hyperoperation in teaching research. *Science & Philosophy*, 8(2):157–163, 2020.

[28] Souzana Vougioukli. Hyperoperations defined on sets of s -helix matrices. 2020.

[29] Thomas Vougiouklis. *Hyperstructures and their representations*. Hadronic Press, 1994.

[30] Ming Xu, Xianbin Xu, Fang Zheng, Yuanhua Yang, and Mengjia Yin. A hybrid sorting algorithm on heterogeneous architectures. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 13(4):1399–1407, 2015.

DOI: https://doi.org/10.54216/PAMDA.040104

Received: January 18, 2024 Revised: March 03, 2024 Accepted: June 05, 2024