Uncertainty Discourse and Applications

www.uda.reapress.com

Uncert. Disc. Appl.Vol. 3, No. 1 (2026) 33–53.

Paper Type: Original Article

Introduction for SuperHyperFunction and SuperHyperGraph with Some Algorithms

Takaaki Fujita[1,*], **Arif Mehmood**[2]

[1]Independent Researcher, Tokyo, Japan; takaaki.fujita060@gmail.com.

[2]Department of Mathematics, Institute of Numerical Sciences, Gomal University, Dera Ismail Khan29050, KPK, Pakistan; mehdaniyal@gmail.com.

Citation:

Received: 05 May 2025 *Revised: 23 August 2025* *Accepted: 16 October 2025*	Fujita, T., & Mehmood, A., (2026). Introduction for SuperHyperFunction and SuperHyperGraph with some algorithms. *Uncertainty discourse and applications, 3*(1), 33-53.

Abstract

A finite hypergraph generalizes the classical graph model by allowing hyperedges that can connect any nonempty subset of vertices. Building on this foundation, a finite SuperHyperGraph is obtained through iterative application of the powerset construction, thereby creating nested families of vertex and edge sets that capture multi-layered relationships.

In this paper, we introduce the concept of an h, k-ary m, n-SuperHyperGraph, which extends the already well-studied SuperHyperGraph by employing the framework of h, k-ary m, n-SuperHyperFunctions. An h, k-ary m, n-SuperHyperFunction is a structured mapping that takes h input sets at level m and produces k output sets at level n, enabling the representation of complex multi-input, multi-output relationships. An h, k-ary m, n-SuperHyperGraph is then defined as a higher-order hypergraph whose vertices are such superhyperfunctions, while its hyperedges group these functions to represent contextual and hierarchical dependencies among the mappings. In addition, recognition algorithms and construction algorithms of h, k-ary m, n-SuperHyperGraph are examined with respect to both their validity and their computational complexity.

Keywords: Superhypergraphs, Hypergraphs, SuperHyperFunction

Corresponding Author: takaaki.fujita060@gmail.com

https://doi.org/10.48313/uda.v3i1.96

1| Introduction

This section fixes the terminology and notation used throughout the paper. Unless stated otherwise, every graph considered here is *finite*.

1.1| SuperHyperGraphs

Graph theory is the mathematical study of graphs: structures of vertices and edges modeling pairwise relations, networks, and connectivity [1, 2]. A finite *hypergraph* generalizes the usual graph by permitting *hyperedges* that link arbitrary nonempty subsets of the vertex set [3, 4, 5]. Building on this idea, a finite *SuperHyperGraph* is obtained by iterating the powerset construction, producing nested families of vertex and edge sets that encode multi-layer relationships [6, 7, 8, 9, 10]. Such structures have proved useful in settings such as molecular design, complex-network analysis, and advanced signal-processing pipelines [11, 12]. Unless indicated otherwise, the index n in $\mathcal{P}_n\cdot$ or in an n–SuperHyperGraph is taken to be nonnegative.

Definition 1.1 (Base Set)**.** A *base set* S is the underlying universe of discourse:

$$S \;=\; \{\, x \mid x \text{ belongs to the context under consideration} \,\}.$$

All objects appearing in $\mathcal{P}S$ or in any iterated powerset $\mathcal{P}_nS$ are, by definition, subsets ultimately drawn from S.

Definition 1.2 (Powerset)**.** (see [13, 14, 15]) For a set S, the *powerset* $\mathcal{P}S$ is the collection of all subsets of S:

$$\mathcal{P}S \;=\; \{\, A \subseteq S \,\}.$$

In particular, both S and the empty set $\emptyset$ are elements of $\mathcal{P}S$.

Definition 1.3 (Hypergraph)**.** [16, 17] A *hypergraph* is an ordered pair $H = V, E$ where

- V is a finite set of vertices, and
- E is a finite family of nonempty subsets of V whose members are called *hyperedges*.

Hypergraphs naturally model interactions among more than two participants.

Example 1.4 (Hypergraph: project teams in a class)**.** Let the vertex set be the students

$$V = \{\text{Ayano}, \text{Masahiro}, \text{Carol}, \text{Dana}\}.$$

Define hyperedges (each a nonempty subset of V) to represent project teams:

$$e_1 = \{\text{Ayano}, \text{Masahiro}, \text{Carol}\},$$
$$e_2 = \{\text{Masahiro}, \text{Dana}\},$$
$$e_3 = \{\text{Ayano}, \text{Dana}\}.$$

Set $E = \{e_1, e_2, e_3\}$. Then $H = V, E$ is a hypergraph because every $e_i \subseteq V$ and $e_i \neq$, so $E \subseteq \mathcal{P}V \setminus \{\}$. This model captures multiway interactions: e_1 is a 3-person team, while e_2 and e_3 are 2-person teams. Cardinalities: $|V| = 4$ and $|E| = 3$.

Definition 1.5 (n-th Powerset)**.** [18, 19] Let X be a set. Put $\mathcal{P}_1X = \mathcal{P}X$ and, for $n \geq 1$,

$$\mathcal{P}_{n1}X \;=\; \mathcal{P}\big(\mathcal{P}_nX\big).$$

When excluding the empty set, we write $\mathcal{P}^*_nX = \mathcal{P}_nX \setminus \{\emptyset\}$.

Example 1.6 (n-th powerset for a concrete base set)**.** Take $X = \{a, b\}$. Then

$$\mathcal{P}_1X = \mathcal{P}X = \{\ ,\ \{a\},\ \{b\},\ \{a, b\}\ \}, \quad \text{so} \quad |\mathcal{P}_1X| = 4.$$

The second iterated powerset is the powerset of $\mathcal{P}_1X$:

$$\mathcal{P}_2X = \mathcal{P}\big(\mathcal{P}_1X\big),$$

which consists of *all* subsets of $\mathcal{P}_1X$. Enumerating by size,

$$\begin{aligned}
&\text{size } 0: \ \{\ \},\\
&\text{size } 1: \ \{\ \{\},\ \{\{a\}\},\ \{\{b\}\},\ \{\{a,b\}\}\ \},\\
&\text{size } 2: \ \{\ \{,\{a\}\},\ \{,\{b\}\},\ \{,\{a,b\}\},\ \{\{a\},\{b\}\},\ \{\{a\},\{a,b\}\},\ \{\{b\},\{a,b\}\}\ \},\\
&\text{size } 3: \ \{\ \{,\{a\},\{b\}\},\ \{,\{a\},\{a,b\}\},\ \{,\{b\},\{a,b\}\},\ \{\{a\},\{b\},\{a,b\}\}\ \},\\
&\text{size } 4: \ \{\ \{,\{a\},\{b\},\{a,b\}\}\ \}.
\end{aligned}$$

Hence $|\mathcal{P}_2X| = 16 = 2^4$. In general, with $|X| = 2$, one has $|\mathcal{P}_nX| = 2^{|\mathcal{P}_{n-1}X|}$; for example, $|\mathcal{P}_3X| = 2^{16}$.

Definition 1.7 (n–SuperHyperGraph)**.** (see [20, 21]) Fix a finite, nonempty base set V_0 and define

$$\mathcal{P}^0V_0 \coloneqq V_0, \qquad \mathcal{P}^{k1}V_0 \coloneqq \mathcal{P}\big(\mathcal{P}^kV_0\big) \quad k \in \mathbb{N}.$$

For $n \geq 0$, an *n–SuperHyperGraph on* V_0 is a pair

$$\mathrm{SHG}^n = V, E$$

with

$$V \subseteq \mathcal{P}^nV_0 \quad \text{and} \quad E \subseteq \mathcal{P}V \setminus \{\emptyset\}.$$

Elements of V are the *n–supervertices*, and elements of E are the *n–superedges* (each n–superedge is a nonempty subset of V).

Example 1.8 (Level $n = 1$ (Urban Mobility Bundles))**.** Let the base (atomic service) set be

$$V_0 = \{\mathrm{Bus\_A},\ \mathrm{Bus\_B},\ \mathrm{Subway\_C},\ \mathrm{BikeDock\_D},\ \mathrm{Taxi\_E}\}.$$

Form 1-supervertices as service *bundles* (each is a subset of V_0):

$$\begin{aligned}
M_1 &= \{\mathrm{Bus_A},\ \mathrm{Subway_C}\},\\
M_2 &= \{\mathrm{Subway_C},\ \mathrm{BikeDock_D}\},\\
M_3 &= \{\mathrm{Taxi_E}\},\\
M_4 &= \{\mathrm{Bus_B},\ \mathrm{Subway_C}\}.
\end{aligned}$$

Since $M_i \subseteq V_0$ for all i, we have $M_i \in \mathcal{P}V_0 = \mathcal{P}^1V_0$. Set the vertex and edge families by

$$V = \{M_1, M_2, M_3, M_4\} \subseteq \mathcal{P}^1V_0, \qquad \begin{aligned} E &= \{e_1, e_2, e_3\},\\ e_1 &= \{M_1, M_2\}, \quad e_2 = \{M_2, M_3\}, \quad e_3 = \{M_1, M_4\}. \end{aligned}$$

Each $e_j \subseteq V$ and $e_j \neq$, so $e_j \in \mathcal{P}V \setminus \{\}$. Therefore

$$\mathrm{SHG}^1 = V, E, \qquad V \subseteq \mathcal{P}^1V_0, \ \ E \subseteq \mathcal{P}V \setminus \{\}.$$

Supervertices are multimodal travel bundles (e.g., bus→subway); a superedge groups bundles that interoperate (e.g., share a transfer hub or a ticketing agreement). Cardinalities: $|V_0| = 5$, $|V| = 4$, $|E| = 3$.

Example 1.9 (Level $n = 2$ (Hospital Clinical Pathways))**.** Let the base set of atomic clinical events be

$$V_0 = \{\mathrm{Triage},\ \mathrm{CTScan},\ \mathrm{Antibiotics},\ \mathrm{Admission},\ \mathrm{Discharge}\}.$$

Form 1-supervertices (functional *modules* of care):

$$\begin{aligned}
U_1 &= \{\mathrm{Triage},\ \mathrm{CTScan}\} \quad \text{Diagnostics},\\
U_2 &= \{\mathrm{Antibiotics}\} \quad \text{Therapy},\\
U_3 &= \{\mathrm{Admission},\ \mathrm{Discharge}\} \quad \text{Disposition}.
\end{aligned}$$

Each $U_i \subseteq V_0$, hence $U_i \in \mathcal{P}V_0$. Now form 2-supervertices (care *pathways* as sets of modules):

$$W_1 = \{U_1, U_2\}, \qquad W_2 = \{U_1, U_3\}, \qquad W_3 = \{U_2, U_3\}.$$

Because $U_i \in \mathcal{P}V_0$, each $W_j \subseteq \mathcal{P}V_0$, i.e., $W_j \in \mathcal{P}\mathcal{P}V_0 = \mathcal{P}^2V_0$. Define

$$V = \{W_1, W_2, W_3\} \subseteq \mathcal{P}^2V_0, \qquad \begin{aligned} E &= \{e_1, e_2, e_3\},\\ e_1 &= \{W_1, W_2\}, \quad e_2 = \{W_1, W_3\}, \quad e_3 = \{W_2, W_3\}. \end{aligned}$$

Again $e_j \subseteq V$ and $e_j \neq$, so $E \subseteq \mathcal{P}V \setminus \{\}$. Thus

$$\mathrm{SHG}^2 = V, E, \qquad V \subseteq \mathcal{P}^2 V_0, \;\; E \subseteq \mathcal{P}V \setminus \{\}.$$

Supervertices are clinical pathways composed of care modules; superedges relate pathways that branch/merge (e.g., diagnosis→therapy vs. diagnosis→disposition). Cardinalities: $|V_0| = 5,\ |\{U_1, U_2, U_3\}| = 3,\ |V| = 3,\ |E| = 3.$

2| Reviews: h, k-ary m, n-SuperHyperGraph

A m, n-SuperHyperGraph is a mathematical structure in which each vertex corresponds to an m, n-superhyperfunction defined on a base set, while the hyperedges group such functions together to represent higher-order relationships and contextual connections. An h, k-ary m, n-SuperHyperGraph further generalizes this idea by taking vertices as h, k-ary m, n-superhyperfunctions [22].

Notation 2.1. *For a nonempty base set S define*

$$\mathcal{P}_0 S := S, \qquad \mathcal{P}_{m1} S := \mathcal{P}\big(\mathcal{P}_m S\big) \quad m \in \mathbb{N}_0,$$

so $\mathcal{P}_1 S = \mathcal{P}S$, $\mathcal{P}_2 S = \mathcal{P}\mathcal{P}S$, etc. We also use the Cartesian power $X^h := \underbrace{X \times \cdots \times X}_{h \text{ copies}}$ for $h \in \mathbb{N}$.

Definition 2.2 (m, n*-superhyperfunction*)**.** [23, 24] Let $m, n \in \mathbb{N}$ and $S \neq$. An m, n*-superhyperfunction on* S is a map

$$f : \mathcal{P}_m S \longrightarrow \mathcal{P}_n S.$$

Equivalently, $f \in \mathrm{Hom}\big(\mathcal{P}_m S, \mathcal{P}_n S\big)$ as functions of sets.

Example 2.3 (Home cooking assistant: pantry ⇒ feasible recipes $m, n = 1, 2$)**.** Let the base set of ingredients be

$$S = \{\text{egg}, \text{milk}, \text{flour}, \text{tomato}, \text{cheese}\}.$$

Consider a fixed family of recipes (each recipe is a subset of S)

$$\mathcal{R} = \{R_1 = \{\text{egg}, \text{milk}\},\ R_2 = \{\text{flour}, \text{egg}\},\ R_3 = \{\text{tomato}, \text{cheese}\}\} \subseteq \mathcal{P}_1 S.$$

Define

$$f : \mathcal{P}_1 S \longrightarrow \mathcal{P}_2 S, \qquad fA := \{\, R \in \mathcal{R} \mid R \subseteq A \,\}.$$

Typing check. For any $A \subseteq S$, the set fA is a set of recipes, hence $fA \subseteq \mathcal{P}_1 S$, i.e. $fA \in \mathcal{P}\mathcal{P}_1 S = \mathcal{P}_2 S$.

Concrete evaluation. With pantry $A = \{\text{egg}, \text{milk}, \text{flour}\}$,

$$fA = \{R_1, R_2\} = \{\{\text{egg}, \text{milk}\},\ \{\text{flour}, \text{egg}\}\} \in \mathcal{P}_2 S.$$

Property. If $A \subseteq B$ then $fA \subseteq fB$ (monotone w.r.t. $\subseteq$).

Example 2.4 (Playlist curator: set of playlists ⇒ common tracks $m, n = 2, 1$)**.** Let the base set of songs be $S = \{s_1, s_2, s_3\}$. Three user playlists (each a subset of songs) are

$$P_1 = \{s_1, s_2\}, \qquad P_2 = \{s_2, s_3\}, \qquad P_3 = \{s_1, s_2, s_3\} \in \mathcal{P}_1 S.$$

Define $f : \mathcal{P}_2 S \to \mathcal{P}_1 S$ by intersection of all playlists in the input family:

$$f\mathcal{U} := \begin{cases} \bigcap_{U \in \mathcal{U}} U, & \mathcal{U} \neq , \\ , & \mathcal{U} = . \end{cases}$$

Typing check. If $\mathcal{U} \in \mathcal{P}_2 S$, then $\mathcal{U} \subseteq \mathcal{P}_1 S$ and $\bigcup_{U \in \mathcal{U}} U \subseteq S$, so $f\mathcal{U} \in \mathcal{P}_1 S$.

Concrete evaluation. For $\mathcal{U} = \{P_1, P_2\} \in \mathcal{P}_2 S$,

$$f\{P_1, P_2\} = P_1 \cap P_2 = \{s_2\} \in \mathcal{P}_1 S.$$

Property. f is *antitone*: if $\mathcal{U} \subseteq \mathcal{V}$ then $f\mathcal{V} \subseteq f\mathcal{U}$.

Definition 2.5 (m, n-*SuperHyperGraph*)**.** [22] Fix $m, n \in \mathbb{N}$ and a nonempty base set S. Let

$$\mathfrak{F}_{m,n}S \;:=\; \{\, f : \mathcal{P}_m S \to \mathcal{P}_n S \,\}.$$

An m, n-*SuperHyperGraph* is a pair

$$\mathsf{SHG}^{m,n} \;:=\; V,\ \mathcal{E},$$

where $V \subseteq \mathfrak{F}_{m,n}S$ is a nonempty set of vertices (each vertex is a concrete m, n-superhyperfunction) and

$$\neq \mathcal{E} \;\subseteq\; \mathcal{P}V \setminus \{\}$$

is a nonempty family of nonempty *hyperedges*. Each hyperedge $E \in \mathcal{E}$ groups a finite, nonempty set of superhyperfunctions to encode higher-order relations/constraints among them.

Example 2.6 (Urban mobility planners: $m, n = 1, 2$)**.** Let the atomic services be

$$S = \{\text{Bus\_A},\ \text{Subway\_C},\ \text{BikeDock\_D},\ \text{Taxi\_E}\}.$$

Fix a catalogue of candidate itineraries (each a subset of S):

$$\mathcal{R} = \{R_1 = \{\text{Bus\_A}, \text{Subway\_C}\},\ R_2 = \{\text{Subway\_C}, \text{BikeDock\_D}\},\ R_3 = \{\text{Taxi\_E}\}\} \;\subseteq\; \mathcal{P}_1 S.$$

Define three $1, 2$-superhyperfunctions $f_{\text{ticket}}, f_{\text{green}}, f_{\text{fast}} : \mathcal{P}_1 S \to \mathcal{P}_2 S$ by

$$f_{\text{ticket}}A := \{\, R \in \mathcal{R} \mid \text{Subway\_C} \in R \text{ and } R \subseteq A \,\},$$
$$f_{\text{green}}A := \{\, R \in \mathcal{R} \mid \text{BikeDock\_D} \in R \text{ and } R \subseteq A \,\},$$
$$f_{\text{fast}}A := \{\, R \in \mathcal{R} \mid R \subseteq A \text{ and Taxi\_E} \in R \text{ or } |R| = 1 \,\}.$$

Then $V = \{f_{\text{ticket}}, f_{\text{green}}, f_{\text{fast}}\} \subseteq \mathfrak{F}_{1,2}S$ and we set

$$\mathcal{E} = \{\, \{f_{\text{ticket}}, f_{\text{green}}\},\ \{f_{\text{ticket}}, f_{\text{fast}}\} \,\}.$$

Hence $\mathsf{SHG}^{1,2} = V, \mathcal{E}$ is a valid m, n-SuperHyperGraph.

Concrete evaluation for $A = \{\text{Bus\_A}, \text{Subway\_C}, \text{BikeDock\_D}\}$:

$$f_{\text{ticket}}A = \{R_1, R_2\},$$
$$f_{\text{green}}A = \{R_2\},$$
$$f_{\text{fast}}A = \{R_1, R_2\}.$$

Each image is a subset of $\mathcal{P}_1 S$, hence lies in $\mathcal{P}_2 S$.

Example 2.7 (Hospital pathways: $m, n = 1, 2$)**.** Let the atomic clinical events be

$$S = \{\text{Triage},\ \text{CTScan},\ \text{Antibiotics},\ \text{Admission},\ \text{Discharge}\}.$$

Fix a small library of pathways $\mathcal{P}th \subseteq \mathcal{P}_1 S$:

$$P_1 = \{\text{Triage}, \text{CTScan}, \text{Admission}\},$$
$$P_2 = \{\text{Triage}, \text{Antibiotics}, \text{Admission}\},$$
$$P_3 = \{\text{Triage}, \text{CTScan}, \text{Discharge}\}.$$

Define $1, 2$-superhyperfunctions $f_{\text{diag}}, f_{\text{abx}}, f_{\text{admit}} : \mathcal{P}_1 S \to \mathcal{P}_2 S$ by

$$f_{\text{diag}}A := \{\, P \in \mathcal{P}th \mid \text{CTScan} \in P \text{ and } P \subseteq A \,\},$$
$$f_{\text{abx}}A := \{\, P \in \mathcal{P}th \mid \text{Antibiotics} \in P \text{ and } P \subseteq A \,\},$$
$$f_{\text{admit}}A := \{\, P \in \mathcal{P}th \mid \text{Admission} \in P \text{ and } P \subseteq A \,\}.$$

Let $V = \{f_{\text{diag}}, f_{\text{abx}}, f_{\text{admit}}\}$ and

$$\mathcal{E} = \{\, \{f_{\text{diag}}, f_{\text{abx}}\},\ \{f_{\text{diag}}, f_{\text{admit}}\},\ \{f_{\text{abx}}, f_{\text{admit}}\} \,\}.$$

Then $\mathsf{SHG}^{1,2} = V, \mathcal{E}$ is an m, n-SuperHyperGraph.

Concrete evaluation for $A = \{\text{Triage}, \text{CTScan}, \text{Antibiotics}, \text{Admission}\}$:

$$f_{\text{diag}}A = \{P_1\},$$
$$f_{\text{abx}}A = \{P_2\},$$

$$f_{\text{admit}}A = \{P_1, P_2\}.$$

Each output is a subset of $\mathcal{P}_1 S$, thus in $\mathcal{P}_2 S$.

Example 2.8 (Retail aggregation and cross-sell: $m, n = 2, 1$)**.** Let the atomic items be $S = \{\text{Phone},\ \text{Case},\ \text{Charger}\}$. Vertices are $2, 1$-superhyperfunctions $f : \mathcal{P}_2 S \to \mathcal{P}_1 S$:

$$f_{\cap}\mathcal{U} := \begin{cases} {}_{U\in\mathcal{U}}U, & \mathcal{U} \neq , \\ , & \mathcal{U} = , \end{cases}$$

$$f_{\cup}\mathcal{U} := {}_{U\in\mathcal{U}}U,$$

$$f_{\text{xsell}}\mathcal{U} := \begin{cases} \{\text{Case}\}, & \exists\, U \in \mathcal{U} \text{ with } \text{Phone} \in U, \\ , & \text{otherwise.} \end{cases}$$

Set

$$V = \{f_{\cap}, f_{\cup}, f_{\text{xsell}}\} \subseteq \mathfrak{F}_{2,1}S$$

and

$$\mathcal{E} = \{\ \{f_{\cap}, f_{\cup}\},\ \{f_{\cup}, f_{\text{xsell}}\}\ \}.$$

Thus $\mathsf{SHG}^{2,1} = V, \mathcal{E}$ is an m, n-SuperHyperGraph.

Concrete evaluation for

$$\mathcal{U} = \{\{\text{Phone}, \text{Case}\},\ \{\text{Phone}, \text{Charger}\}\} \in \mathcal{P}_2 S$$

:

$$f_{\cap}\mathcal{U} = \{\text{Phone}\},$$
$$f_{\cup}\mathcal{U} = \{\text{Phone}, \text{Case}, \text{Charger}\},$$
$$f_{\text{xsell}}\mathcal{U} = \{\text{Case}\}.$$

All outputs lie in $\mathcal{P}_1 S$ as required.

Definition 2.9 (h, k*-ary* m, n*-superhyperfunction*)**.** Let $h, k \in \mathbb{N}$ and $m, n \in \mathbb{N}$. An h, k*-ary* m, n*-superhyperfunction on* S is a map

$$F :\ \big(\mathcal{P}_m S\big)^h \ \longrightarrow\ \big(\mathcal{P}_n S\big)^k.$$

Writing $FA_1, \ldots, A_h = B_1, \ldots, B_k$, each component $B_j \in \mathcal{P}_n S$ is an n–level object determined by h many m–level inputs.

Example 2.10 ($h, k = 2, 2, m, n = 1, 2$: Two-input / two-output lift)**.** Let the base set be $S = \{\text{egg}, \text{milk}, \text{flour}\}$. Define

$$F :\ \big(\mathcal{P}_1 S\big)^2 \longrightarrow \big(\mathcal{P}_2 S\big)^2, \qquad FA, B\ :=\ \big(B_1 A, B,\ B_2 A, B\big),$$

with components

$$B_1 A, B\ :=\ \{\ A,\ A \cup B\ \}, \qquad B_2 A, B\ :=\ \{\ A \cap B,\ B\ \}.$$

Typing check. Since $A, B \in \mathcal{P}_1 S$, we have $A \cup B \in \mathcal{P}_1 S$ and $A \cap B \in \mathcal{P}_1 S$. Thus $B_1 A, B, B_2 A, B \subseteq \mathcal{P}_1 S$, i.e. $B_1 A, B, B_2 A, B \in \mathcal{P}_2 S$, so F is well-typed: $\mathcal{P}_1 S^2 \to \mathcal{P}_2 S^2$.

Concrete evaluation. For $A = \{\text{egg}\}$ and $B = \{\text{milk}, \text{flour}\}$,

$$FA, B\ =\ \big(\ \{\{\text{egg}\},\ \{\text{egg}, \text{milk}, \text{flour}\}\},\ \{\ ,\ \{\text{milk}, \text{flour}\}\ \}\ \big)\ \in\ \mathcal{P}_2 S^2.$$

Example 2.11 ($h, k = 3, 1$, $m, n = 2, 1$: Three families to one set)**.** Let $S = \{x, y, z\}$ and consider three 2-level inputs $U, V, W \in \mathcal{P}_2 S$ (each is a set of subsets of S). Define

$$G :\ \big(\mathcal{P}_2 S\big)^3 \longrightarrow \mathcal{P}_1 S, \qquad GU, V, W\ :=\ \big(\ {}_{X\in U}X\ \big)\ \cap\ \big(\ {}_{Y\in V}Y\ \big)\ \cup\ \big(\ {}_{Z\in W}Z\ \big),$$

with the conventions $:= S$ and $:= .$

Typing check. Each union/intersection of members of $\mathcal{P}_1 S$ is again a subset of S; hence $GU, V, W \in \mathcal{P}_1 S$, so $G : \mathcal{P}_2 S^3 \to \mathcal{P}_1 S$ is well-typed.

Concrete evaluation. Let

$$U = \{\{x, y\}, \{y\}\}, \quad V = \{\{y, z\}, \{y\}\}, \quad W = \{\{z\}\}.$$

Then $U = \{x, y\}$, $V = \{y\}$, and $W = \{z\}$, so

$$GU, V, W \;=\; \{x, y\} \cap \{y\} \;\cup\; \{z\} \;=\; \{y, z\} \;\in\; \mathcal{P}_1 S.$$

The definition of an h, k-ary m, n-SuperHyperGraph is provided below.

Definition 2.12 (h, k-*ary* m, n-*SuperHyperGraph*)**.** Fix $m, n, h, k \in \mathbb{N}$ and $S \neq$. Let

$$\mathfrak{F}_{m,n}^{h,k} S \;:=\; \{\, F : \big(\mathcal{P}_m S\big)^h \to \big(\mathcal{P}_n S\big)^k \,\}.$$

An h, k-*ary* m, n-*SuperHyperGraph* is a pair

$$\mathsf{SHG}_{m,n}^{h,k} \;:=\; \big(V,\, \mathcal{E}\big), \qquad \neq V \subseteq \mathfrak{F}_{m,n}^{h,k} S, \quad \neq \mathcal{E} \subseteq \mathcal{P}V \setminus \{\}.$$

Example 2.13 (Multimodal travel planning as an h, k-ary m, n-SuperHyperGraph: $m, n = 1, 2$, $h = 2$, $k = 2$)**.** Let the atomic transport services be

$$S = \{\text{Bus\_A},\ \text{Subway\_C},\ \text{BikeDock\_D},\ \text{Taxi\_E}\}.$$

Fix a route catalogue (each route is a subset of S)

$$\mathcal{R} = \{R_1 = \{\text{Bus\_A}, \text{Subway\_C}\},\ R_2 = \{\text{Subway\_C}, \text{BikeDock\_D}\},\ R_3 = \{\text{Taxi\_E}\}\} \subseteq \mathcal{P}_1 S.$$

Inputs are two m=1 level sets: morning availability $A_{\text{am}} \in \mathcal{P}_1 S$ and evening availability $A_{\text{pm}} \in \mathcal{P}_1 S$. Vertices are h, k-ary m, n-superhyperfunctions $F : \mathcal{P}_1 S^2 \to \mathcal{P}_2 S^2$:

$$F_{\text{feas}} A_{\text{am}}, A_{\text{pm}} := \big(\ \{R \in \mathcal{R} \mid R \subseteq A_{\text{am}} \cup A_{\text{pm}}\},\ \{R \in \mathcal{R} \mid R \subseteq A_{\text{am}} \cup A_{\text{pm}} \ \wedge\ \text{BikeDock\_D} \in R\}\ \big),$$

$$F_{\text{subway}} A_{\text{am}}, A_{\text{pm}} := \big(\ \{R \in \mathcal{R} \mid \text{Subway\_C} \in R \subseteq A_{\text{am}} \cup A_{\text{pm}}\},\ \{R \in \mathcal{R} \mid \text{Taxi\_E} \notin R \subseteq A_{\text{am}} \cup A_{\text{pm}}\}\ \big),$$

$$F_{\text{two-hop}} A_{\text{am}}, A_{\text{pm}} := \big(\ \{R \in \mathcal{R} \mid |R| = 2 \ \wedge\ R \subseteq A_{\text{am}} \cup A_{\text{pm}}\},\ \{R \in \mathcal{R} \mid |R| = 1 \ \wedge\ R \subseteq A_{\text{am}} \cup A_{\text{pm}}\}\ \big).$$

Set the vertex set and hyperedges by

$$V = \{F_{\text{feas}}, F_{\text{subway}}, F_{\text{two-hop}}\}, \qquad \mathcal{E} = \{\ \{F_{\text{feas}}, F_{\text{subway}}\},\ \{F_{\text{feas}}, F_{\text{two-hop}}\}\ \}.$$

Concrete evaluation with $A_{\text{am}} = \{\text{Bus\_A}, \text{Subway\_C}\}$ and $A_{\text{pm}} = \{\text{BikeDock\_D}\}$:

$$F_{\text{feas}} A_{\text{am}}, A_{\text{pm}} = \big(\ \{R_1, R_2\},\ \{R_2\}\ \big),$$

$$F_{\text{subway}} A_{\text{am}}, A_{\text{pm}} = \big(\ \{R_1, R_2\},\ \{R_1, R_2\}\ \big),$$

$$F_{\text{two-hop}} A_{\text{am}}, A_{\text{pm}} = \big(\ \{R_1, R_2\},\ \ \big),$$

each component lying in $\mathcal{P}_2 S$.

Example 2.14 (Clinical triage aggregator as an h, k-ary m, n-SuperHyperGraph: $m, n = 1, 2$, $h = 3$, $k = 2$)**.** Let the atomic clinical actions be

$$S = \{\text{Triage},\ \text{CT},\ \text{Labs},\ \text{Antibiotics},\ \text{Admission},\ \text{Isolation}\}.$$

Pathway library $\mathcal{P}th \subseteq \mathcal{P}_1 S$:

$$P_1 = \{\text{Triage}, \text{CT}, \text{Admission}\}, \quad P_2 = \{\text{Triage}, \text{Labs}, \text{Antibiotics}\}, \quad P_3 = \{\text{Triage}, \text{Isolation}, \text{Admission}\}.$$

Inputs are observed actions $A \in \mathcal{P}_1 S$, available resources $B \in \mathcal{P}_1 S$, and risk flags $C \in \mathcal{P}_1 S$. Define vertices $F : \mathcal{P}_1 S^3 \to \mathcal{P}_2 S^2$:

$$F_{\text{cand}} A, B, C := \big(\{\, P \in \mathcal{P}th \mid P \subseteq A \cup B \cup C \ \wedge\ \text{Isolation} \in P \Rightarrow \text{Isolation} \in C\,\},$$
$$\{\,\{x\} \mid x \in \{\text{Triage}, \text{Isolation}\} \cap A \cup B \cup C\,\}\big).$$

$$F_{\text{split}} A, B, C := \big(\{\, P \in \mathcal{P}th \mid \text{CT} \in P \subseteq A \cup B\,\},$$
$$\{\, P \in \mathcal{P}th \mid \text{Antibiotics} \in P \subseteq A \cup B\,\}\big).$$

$$F_{\text{risk}} A, B, C := \big(\{\, P \in \mathcal{P}th \mid \text{Isolation} \in P \subseteq A \cup B \cup C\,\},$$
$$\{\, P \in \mathcal{P}th \mid \text{Admission} \notin P \ \wedge\ P \subseteq A \cup B\,\}\big).$$

Set $V = \{F_{\mathrm{cand}}, F_{\mathrm{split}}, F_{\mathrm{risk}}\}$ and

$$\mathcal{E} = \{\ \{F_{\mathrm{cand}}, F_{\mathrm{risk}}\},\ \{F_{\mathrm{split}}, F_{\mathrm{risk}}\}\ \}.$$

Concrete evaluation with $A = \{\mathrm{Triage}, \mathrm{Labs}\}$, $B = \{\mathrm{CT}, \mathrm{Admission}\}$, $C = \{\mathrm{Isolation}\}$:

$$\begin{aligned} F_{\mathrm{cand}}A, B, C &= \big(\ \{P_1, P_3\},\\ &\qquad \{\{\mathrm{Triage}\}, \{\mathrm{Isolation}\}\}\ \big),\\ F_{\mathrm{split}}A, B, C &= \big(\ \{P_1\},\ \ \big),\\ F_{\mathrm{risk}}A, B, C &= \big(\ \{P_3\},\ \ \big), \end{aligned}$$

all components in $\mathcal{P}_2 S$.

Example 2.15 (Document workflow compliance as an h, k-ary m, n-SuperHyperGraph: $m, n = 1, 1$, $h = 2$, $k = 3$)**.** Let the atomic steps be

$$S = \{\mathrm{Draft},\ \mathrm{Review},\ \mathrm{Approve},\ \mathrm{Publish},\ \mathrm{Archive}\}.$$

Inputs are policy-required steps $A \in \mathcal{P}_1 S$ and system-capable steps $B \in \mathcal{P}_1 S$. Vertices are $F : \mathcal{P}_1 S^2 \to \mathcal{P}_1 S^3$ returning *(Allowed, Required, Forbidden)*:

$$F_{\mathrm{base}}A, B := \big(B,\ A,\ S \setminus B\big).$$

$$F_{\mathrm{strict}}A, B := \big(B,\ A \cup \{\mathrm{Archive}\},\ S \setminus B\ \cup\ \big(\{\mathrm{Publish}\}\ \text{if Approve} \notin B\big)\big).$$

$$\begin{aligned} F_{\mathrm{fast}}A, B := \big(&B \setminus \{\mathrm{Review}\}\ \cup\ \big(\{\mathrm{Publish}\}\ \text{if Approve} \in B\big),\\ &A \setminus \{\mathrm{Review}\},\\ &S \setminus \big[B \setminus \{\mathrm{Review}\}\ \cup\ \big(\{\mathrm{Publish}\}\ \text{if Approve} \in B\big)\big]\big). \end{aligned}$$

Take $V = \{F_{\mathrm{base}}, F_{\mathrm{strict}}, F_{\mathrm{fast}}\}$ and

$$\mathcal{E} = \{\ \{F_{\mathrm{base}}, F_{\mathrm{strict}}\},\ \{F_{\mathrm{strict}}, F_{\mathrm{fast}}\}\ \}.$$

Concrete evaluation with $A = \{\mathrm{Draft}, \mathrm{Review}, \mathrm{Approve}\}$ and $B = \{\mathrm{Draft}, \mathrm{Review}, \mathrm{Approve}, \mathrm{Publish}\}$:

$$\begin{aligned} F_{\mathrm{base}}A, B &= \big(\ \{\mathrm{Draft}, \mathrm{Review}, \mathrm{Approve}, \mathrm{Publish}\},\ \{\mathrm{Draft}, \mathrm{Review}, \mathrm{Approve}\},\ \{\mathrm{Archive}\}\ \big),\\ F_{\mathrm{strict}}A, B &= \big(\ \{\mathrm{Draft}, \mathrm{Review}, \mathrm{Approve}, \mathrm{Publish}\},\ \{\mathrm{Draft}, \mathrm{Review}, \mathrm{Approve}, \mathrm{Archive}\},\ \{\mathrm{Archive}\}\ \big),\\ F_{\mathrm{fast}}A, B &= \big(\ \{\mathrm{Draft}, \mathrm{Approve}, \mathrm{Publish}\},\ \{\mathrm{Draft}, \mathrm{Approve}\},\ \{\mathrm{Review}, \mathrm{Archive}\}\ \big), \end{aligned}$$

each component lying in $\mathcal{P}_1 S$.

Theorem 2.16 (Product–Hom decompositions)**.** *For any sets X, Y and $h, k \in \mathbb{N}$ there are natural bijections*

$$\mathrm{Hom}\big(X^h,\ Y^k\big)\ \cong\ \big(\mathrm{Hom}X^h, Y\big)^k\ \cong\ \mathrm{Hom}\big(X,\ \mathrm{Hom}X^{h-1}, Y^k\big).$$

Specializing to $X = \mathcal{P}_m S$ and $Y = \mathcal{P}_n S$, every h, k-ary m, n-superhyperfunction $F : X^h \to Y^k$ can be viewed (i) as a k–tuple of h–ary m, n-superhyperfunctions, and (ii) in fully curried form.

Proof: The first bijection sends F to its component tuple $\pi_1 \circ F, \ldots, \pi_k \circ F$ and is inverted by stacking components. The second is the standard currying bijection. Naturality is routine. □

Definition 2.17 (Componentwise order and monotonicity)**.** Equip $\mathcal{P}_m S$ with $\subseteq$, and $\big(\mathcal{P}_n S\big)^k$ with the product order:

$$B_1, \ldots, B_k\ \preceq\ B'_1, \ldots, B'_k\ \iff\ \forall j\ B_j \subseteq B'_j.$$

An h, k-ary m, n-superhyperfunction F is *monotone* if

$$A_1, \ldots, A_h\ \preceq\ A'_1, \ldots, A'_h\ \implies\ FA_1, \ldots, A_h\ \preceq\ FA'_1, \ldots, A'_h.$$

Example 2.18 (Componentwise order and monotonicity)**.** Let $S = \{a, b, c\}$, take $m = n = 1$, $h = 2$, $k = 2$, and consider

$$F : \left(\mathcal{P}_1 S\right)^2 \longrightarrow \left(\mathcal{P}_1 S\right)^2, \qquad FA, B := \left(A \cap B,\ A \cup B\right).$$

If $A, B \preceq A', B'$ (i.e. $A \subseteq A'$ and $B \subseteq B'$), then

$$A \cap B \subseteq A' \cap B' \quad \text{and} \quad A \cup B \subseteq A' \cup B',$$

hence $FA, B \preceq FA', B'$ in the product order. Therefore F is *monotone*.

With $A = \{a\}$, $B = \{a, b\}$, $A' = \{a, b\}$, $B' = \{a, b, c\}$:

$$A, B \preceq A', B', \quad FA, B = \{a\}, \{a, b\} \preceq \{a, b\}, \{a, b, c\} = FA', B'.$$

Theorem 2.19 (Closure under typed composition)**.** *Let*

$$F : \left(\mathcal{P}_m S\right)^h \longrightarrow \left(\mathcal{P}_n S\right)^k, \qquad G : \left(\mathcal{P}_n S\right)^k \longrightarrow \left(\mathcal{P}_r S\right)^\ell.$$

Then the composite

$$G \circ F : \left(\mathcal{P}_m S\right)^h \longrightarrow \left(\mathcal{P}_r S\right)^\ell$$

is an h, ℓ-ary m, r-superhyperfunction. If F and G are monotone (Definition 2.17), then $G \circ F$ is monotone.

Proof: Typing follows from function composition. Monotonicity is preserved by composition in product posets. □

3| Result: Algorithm of Recognizing h, k-ary m, n-SuperHyperGraph

In this section, we present the Algorithm of Recognizing an h, k-ary m, n-SuperHyperGraph, together with its subroutine algorithms. We then provide a detailed evaluation of their validity and analyze the computational complexity in order to establish both soundness and efficiency of the overall recognition procedure.

Notation 3.1. *Fix a finite nonempty base set S and integers $m, n, h, k \in \mathbb{N}$. Let V be a finite set of candidate vertices (each intended to be a total function $F : \mathcal{P}_m S^h \to \mathcal{P}_n S^k$), and let $\mathcal{E}$ be a finite family of candidate hyperedges (intended to be nonempty subsets of V). Write $\mathcal{P}_0 S = S$ and $\mathcal{P}_{t1} S = \mathcal{P}\mathcal{P}_t S$.*

IsInIterPowerset decides membership in the t-th iterated powerset recursively. If $t = 0$, it checks whether $X \in S$. Otherwise, it requires X to be finite and ensures that every element $Y \in X$ recursively satisfies IsInIterPowerset$Y, t-1, S$. The outline of the algorithms and the corresponding theorems are presented below.

Algorithm 1: IsInIterPowersetX, t, S

Input: Finite S; integer $t \geq 0$; object X.
Output: **true** iff $X \in \mathcal{P}_t S$.

```
if t = 0 then return X ∈ S
else
    if X is not a finite set then return false
    foreach Y ∈ X do
        if IsInIterPowersetY, t − 1, S = false then return false
    end
    return true
end
```

Example 3.2 (Real-world use of IsInIterPowerset: validating a folder collection)**.** Let $S = \{\mathrm{Readme}, \mathrm{Report}, \mathrm{Data}\}$ denote atomic files. Then $\mathcal{P}_1 S$ are *folders* (sets of files) and $\mathcal{P}_2 S$ are *collections of folders*. Define

$$A = \{\mathrm{Readme}, \mathrm{Report}\}, \quad B = \{\mathrm{Data}\}, \quad X = \{A, B\}.$$

Running IsInIterPowerset$X, 2, S$ returns **true**: X is finite and each $Y \in X$ satisfies IsInIterPowerset$Y, 1, S$, since every $y \in Y$ passes IsInIterPowerset$y, 0, S$ with $y \in S$. Thus $X \in \mathcal{P}_2 S$, i.e., the folder collection is well-typed.

Theorem 3.3 (Correctness of IsInIterPowerset)**.** *For any finite S, integer $t \geq 0$, and object X, Algorithm 1 returns **true** if and only if $X \in \mathcal{P}_t S$.*

Proof: By induction on t. Base $t = 0$: the algorithm returns **true** exactly when $X \in S = \mathcal{P}_0 S$. Inductive step: for $t \geq 1$, $\mathcal{P}_t S$ consists of *finite* sets all of whose elements lie in $\mathcal{P}_{t-1} S$. The algorithm enforces finiteness and recursively verifies each member with parameter $t - 1$. Hence it accepts precisely the members of $\mathcal{P}_t S$. □

Theorem 3.4 (Time and space of IsInIterPowerset)**.** *On input X, t, S with $X \in \mathcal{P}_t S$, Algorithm 1 runs in time $\Theta\big(\mathrm{size}_t X\big)$ and uses Ot stack space. In particular, in the worst case (taking $X = \mathcal{P}_{t-1} S$) the time is*

$$U_t \;:=\; 1 \;N_{t-1}\,U_{t-1} \quad U_0 = 1 \quad \Rightarrow \quad U_t \;=\; 1 \;N_{t-1} \;N_{t-1}N_{t-2} \;\cdots \prod_{i=0}^{t-1} N_i,$$

which is dominated by the product term and thus grows tower-exponentially with t.

Proof: Each node of the membership tree (the set itself plus all nested members down to depth t) is visited once with $O1$ local work, yielding the $\Theta\mathrm{size}_t X$ bound. Recursion depth is exactly t, hence Ot stack space. For the worst case, set $X = \mathcal{P}_{t-1} S$ so $|X| = N_{t-1}$ and each recursive child can be chosen worst-case. This gives $U_t = 1 \;N_{t-1}U_{t-1}$ with $U_0 = 1$, solving to the stated series. □

CheckVertexType iterates over all possible input tuples, verifies function F is defined everywhere, ensures outputs are valid k-tuples, and confirms each component belongs to the n-th iterated powerset. The outline of the algorithms and the corresponding theorems are presented below.

Algorithm 2: CheckVertexTypeF, m, n, h, k, S

Input: Finite S; integers m, n, h, k; function F.
Output: **true** iff $F : \mathcal{P}_m S^h \to \mathcal{P}_n S^k$ is total and well-typed.

```
Generate an iterator over all h-tuples A_1, ..., A_h with A_i ∈ P_m S.
foreach A_1, ..., A_h ∈ do
    if F is undefined on A_1, ..., A_h then return false
    Set Y := F A_1, ..., A_h.
    if Y is not a k-tuple then return false
    for j ← 1 to k do
        if IsInIterPowersetY_j, n, S = false then return false
    end
end
return true
```

Example 3.5 (Real-world use of CheckVertexType: validating a notification policy)**.** Let $S = \{\mathrm{Email}, \mathrm{SMS}, \mathrm{Push}\}$ and take $m = n = 1$, $h = 1$, $k = 2$. Define a policy

$$F : \mathcal{P}_1 S \longrightarrow \big(\mathcal{P}_1 S\big)^2, \qquad FA \;:=\; \big(A \setminus \{\mathrm{SMS}\},\;\; S \setminus A\big),$$

interpreted as *(Allowed, Forbidden)* channels.

*Why CheckVertexType returns **true**.* For every $A \subseteq S$, F is defined, FA is a 2-tuple, and each component is a subset of S, hence belongs to $\mathcal{P}_1 S$. Therefore the loop in CheckVertexType never triggers a failure, and the algorithm returns **true**.

Concrete check. For $A = \{\mathrm{Email}, \mathrm{SMS}\}$,

$$Y = FA = \big(\{\mathrm{Email}\},\; \{\mathrm{Push}\}\big),$$

which is a 2-tuple with both entries in $\mathcal{P}_1 S$, so all tests pass.

Theorem 3.6 (Correctness of CheckVertexType)**.** *Let $m, n, h, k \in \mathbb{N}$ and finite S. The Algorithm returns **true** if and only if $F : \mathcal{P}_m S^h \to \mathcal{P}_n S^k$ is total on $\mathcal{P}_m S^h$ and every output component lies in $\mathcal{P}_n S$.*

Proof: (*If*) If the algorithm returns **true**, it has iterated over every domain element $A_1, \ldots, A_h \in \mathcal{P}_m S^h$, found F defined there, confirmed the result is a k-tuple, and verified each component via Algorithm 1 that it belongs to $\mathcal{P}_n S$. Hence F is total and well-typed. (*Only if*) Conversely, if F is total and well-typed, each loop iteration succeeds and Algorithm 1 returns **true** for every component; thus the algorithm returns **true**. □

Theorem 3.7 (Time of CheckVertexType)**.** *Let $N_0 = |S|$ and $N_{t1} = 2^{N_t}$, so $|\mathcal{P}_t S| = N_t$. Set $D = N_m^h = |\mathcal{P}_m S^h|$. If one evaluation of F costs T_{eval}, then*

$$T\big(\textit{CheckVertexType}\big) \;=\; \Theta\Big(D \cdot \big(T_{\mathrm{eval}} \;\sum_{j=1}^{k} T_n Y_j\big)\Big),$$

where $T_n Y_j = \Theta\big(\mathrm{size}_n Y_j\big)$ by Theorem 3.4. In the worst case, $T_n Y_j \le U_n$, giving the crude bound

$$T\big(\textit{CheckVertexType}\big) \;=\; O\Big(D \cdot \big(T_{\mathrm{eval}} \;k\, U_n\big)\Big),$$

with recursion depth at most n per component check.

Proof: The algorithm performs a constant amount of overhead per domain element, one evaluation of F, and k calls to Algorithm 1. Summing over the D inputs yields the stated expression. By Theorem 3.4, each component check costs $\Theta\mathrm{size}_n\cdot$, and the worst-case over $\mathcal{P}_n S$ is U_n. □

Recognize_HK_MN_SuperHyperGraph verifies nonempty vertices and edges, type-checks every vertex, rejects edges containing non-vertices or empties, and accepts exactly when all structural conditions are satisfied. The outline of the algorithms and the corresponding theorems are presented below.

Algorithm 3: RecognizeHKMNSuperHyperGraph$S, m, n, h, k, V, \mathcal{E}$

Input: Finite S; integers m, n, h, k; finite vertex set V; finite family $\mathcal{E}$.
Output: Accept iff $V, \mathcal{E}$ is an h, k-ary m, n-SuperHyperGraph; otherwise Reject with a certificate.

if $V =$ **then**
  return Reject (empty vertex set).
end
foreach $F \in V$ **do**
  if *CheckVertexType*$F, m, n, h, k, S =$ ***false*** **then return** Reject (ill-typed vertex F).
end
if $\mathcal{E} =$ **then**
  return Reject (no hyperedges).
end
foreach $E \in \mathcal{E}$ **do**
  if $E =$ **then return** Reject (empty hyperedge).
  foreach $F \in E$ **do**
    if $F \notin V$ **then return** Reject (hyperedge contains non-vertex).
  end
end
return Accept

Example 3.8 (Real-world use of RecognizeHKMNSuperHyperGraph: validating policy graphs)**.** Let $S = \{\mathrm{Email}, \mathrm{SMS}, \mathrm{Push}\}$ and take $m = n = h = k = 1$. Define two total, well-typed policies $F_{\mathrm{id}} A = A$ and $F_{\mathrm{noSMS}} A = A \setminus \{\mathrm{SMS}\}$, so $F_{\mathrm{id}}, F_{\mathrm{noSMS}} : \mathcal{P}_1 S \to \mathcal{P}_1 S$. Set

$$V = \{F_{\mathrm{id}}, F_{\mathrm{noSMS}}\}, \qquad \mathcal{E} = \{\, \{F_{\mathrm{id}}\},\ \{F_{\mathrm{id}}, F_{\mathrm{noSMS}}\}\,\}.$$

Recognition. The algorithm finds $V \neq$; both vertices pass CheckVertexType; $\mathcal{E} \neq$ and every $E \in \mathcal{E}$ is nonempty with members in V. Hence it returns Accept, confirming an $1, 1$–$1, 1$ SuperHyperGraph.

Theorem 3.9 (Correctness (Soundness and Completeness))**.** *Let S be finite and $m, n, h, k \in \mathbb{N}$. On input $S, m, n, h, k, V, \mathcal{E}$, the algorithm RecognizeHKMNSuperHyperGraph returns Accept if and only if $V, \mathcal{E}$ is an h, k-ary m, n-SuperHyperGraph on S.*

Proof: (*Soundness*) If the algorithm accepts, then: (i) $V \neq$ by the first check. (ii) For every $F \in V$, CheckVertexType verified that F is total on $\mathcal{P}_m S^h$ and that each image is a k-tuple with components in $\mathcal{P}_n S$. Hence $F : \mathcal{P}_m S^h \to \mathcal{P}_n S^k$, so $V \subseteq \mathfrak{F}_{m,n}^{h,k} S$. (iii) $\mathcal{E} \neq$, and each $E \in \mathcal{E}$ is nonempty with all elements in V, hence $\mathcal{E} \subseteq \mathcal{P}V \setminus \{\}$. Therefore, by definition, $V, \mathcal{E}$ is an h, k-ary m, n-SuperHyperGraph.

(*Completeness*) Conversely, suppose $V, \mathcal{E}$ is an h, k-ary m, n-SuperHyperGraph. Then $V \neq$ and $V \subseteq \mathfrak{F}_{m,n}^{h,k} S$; thus every $F \in V$ is a total, well-typed map from $\mathcal{P}_m S^h$ to $\mathcal{P}_n S^k$, so CheckVertexType returns true. Moreover, $\mathcal{E} \neq$ and each $E \in \mathcal{E}$ is a nonempty subset of V, so the edge checks pass. Hence the algorithm accepts. □

Theorem 3.10 (Time Complexity)**.** *Let $N_0 = |S|$ and $N_{t1} = 2^{N_t}$ for $t \geq 0$; then $|\mathcal{P}_t S| = N_t$. Let $D := N_m^h$ be the size of the domain $\mathcal{P}_m S^h$. Assume a single evaluation of a vertex F costs T_{eval} time. Then the worst-case running time of RECOGNIZE_HK_MN_SUPERHYPERGRAPH is*

$$\mathcal{O}\Bigg(|V| \cdot D \cdot \big(T_{\text{eval}} \;\; k \cdot T_n\big) \quad |\mathcal{E}| \;\; {}_{E \in \mathcal{E}} |E| \Bigg),$$

where T_n is the cost of IsInIterPowerset on level n, which satisfies $T_n = \mathcal{O}N_n$. In particular, the recognition problem is (nonuniformly) tower-exponential in m and n via N_m and N_n.

Proof: The vertex-type phase processes each $F \in V$ and all $D = N_m^h$ inputs. For each input, it performs one evaluation (T_{eval}) and k membership checks in $\mathcal{P}_n S$. By induction on n, IsInIterPowerset examines each element one level down and recurses, yielding $T_n = \mathcal{O}N_n$ in the worst case (since a maximal element of $\mathcal{P}_n S$ can contain N_{n-1} elements, each with a worst-case membership cost $\mathcal{O}N_{n-1}$, etc., unfolding to $\mathcal{O}N_n$). Thus the vertex phase costs $\mathcal{O}\big(|V| \cdot D \cdot T_{\text{eval}} \;\; kT_n\big)$. The edge phase checks emptiness and membership of each element of each hyperedge in V, costing $\mathcal{O}\big(|\mathcal{E}| \;\; {}_E |E|\big)$. Summing both parts gives the stated bound. The dependence on m and n is through N_m and N_n, which are exponential towers by definition, so the overall worst-case complexity is tower-exponential. □

4| Result: Algorithm of Constructing h, k-ary m, n-SuperHyperGraph

The algorithm of constructing an h, k-ary m, n-SuperHyperGraph systematically verifies candidate functions, selects valid vertices, and organizes property-based hyperedges to build a consistent higher-order graph structure. The following section defines the algorithms, including the necessary subroutines, and evaluates their correctness to ensure that they faithfully implement the intended construction process.

Notation 4.1. *Fix a finite, nonempty base set S and integers $m, n, h, k \in \mathbb{N}$. We are given a finite library of candidate mappings $F : \mathcal{P}_m S^h \to \mathcal{P}_n S^k$ (templates or concrete functions), and a finite family of vertex predicates $\mathbf{\Phi} = \{\Phi_1, \ldots, \Phi_p\}$, where each Φ_i is a (decidable) property on such F (e.g., monotonicity, diagonal idempotence, symmetry). We construct a vertex set $V \subseteq \mathfrak{F}_{m,n}^{h,k} S$ by type-checking and a hyperedge family $\mathcal{E}$ by grouping vertices that satisfy the same property.*

Notation 4.2. *Let $N_0 = |S|$ and $N_{t1} = 2^{N_t}$ so $|\mathcal{P}_t S| = N_t$. The domain size is $D := |\ \mathcal{P}_m S^h\ | = N_m^h$. Write U_n for a worst-case membership-check cost on $\mathcal{P}_n S$ (e.g., as in a structural check via IsInIterPowerset).*

EvaluatePredicates iterates through the predicate family, tests each predicate on F, collects indices of those evaluating true into set I, and returns resulting index set. The outline of the algorithms and the corresponding theorems are presented below.

Algorithm 4: EvaluatePredicates$F, \mathbf{\Phi}$

Input: Function $F : \mathcal{P}_m S^h \to \mathcal{P}_n S^k$; predicate family $\mathbf{\Phi} = \{\Phi_1, \ldots, \Phi_p\}$.
Output: Index set $IF = \{\, i \in \{1, \ldots, p\} \mid \Phi_i F = \textbf{true} \,\}$.

```
I ←
for i ← 1 to p do
    if Φ_i F = true then
        I ← I ∪ {i}
    end
end
return I
```

Example 4.3 (Real-world use of EvaluatePredicates: notification policy)**.** Let $S = \{\mathrm{Email}, \mathrm{SMS}, \mathrm{Push}\}$ and take $m = n = h = k = 1$. Define the policy (function) $F : \mathcal{P}_1 S \to \mathcal{P}_1 S$ by

$$FA \;:=\; A \setminus \{\mathrm{SMS}\} \qquad \text{“never send SMS”.}$$

Consider the predicate family $\mathbf{\Phi} = \{\Phi_1, \Phi_2, \Phi_3, \Phi_4\}$:

$$\begin{aligned}
&\Phi_1 : \forall A \subseteq S \;\; FA \subseteq A \quad \text{(opt-out safety)},\\
&\Phi_2 : \forall A \subseteq S \;\; \mathrm{SMS} \notin FA \quad \text{(no-SMS policy)},\\
&\Phi_3 : \forall A \subseteq S \;\; FFA = FA \quad \text{(idempotent)},\\
&\Phi_4 : \forall A \subseteq B \subseteq S \;\; FA \subseteq FB \quad \text{(monotone)}.
\end{aligned}$$

Each predicate holds for this F (e.g., F removes a fixed element, preserving inclusion and idempotence). Hence

$$\textsc{EvaluatePredicates}F, \mathbf{\Phi} \;=\; \{1, 2, 3, 4\}.$$

Concrete check: with $A = \{\mathrm{Email}, \mathrm{SMS}\}$, $FA = \{\mathrm{Email}\}$ and $FFA = \{\mathrm{Email}\} = FA$; if $A \subseteq B$, then removing SMS from both preserves inclusion, so Φ_4 holds.

Theorem 4.4 (Correctness of EvaluatePredicates)**.** *For any $F : \mathcal{P}_m S^h \to \mathcal{P}_n S^k$ and predicate family $\mathbf{\Phi} = \{\Phi_1, \ldots, \Phi_p\}$, Algorithm 4 returns*

$$IF = \{\, i \in \{1, \ldots, p\} \mid \Phi_i F = \textbf{true} \,\}.$$

Proof: Initialize $I \leftarrow$. The loop inspects each $i = 1, \ldots, p$ once and adds i to I iff the condition $\Phi_i F =$ **true** holds. Thus, upon termination, I contains exactly those indices whose predicates are true. □

Theorem 4.5 (Time and Space of EvaluatePredicates)**.** *The running time is*

$$T\big(\textsc{EvaluatePredicates}F, \mathbf{\Phi}\big) \;=\; \sum_{i=1}^{p} C_{\Phi_i} F \;\le\; p\, C_{\Phi}^{\max},$$

and the extra space is $O|IF|$ to store the output set (plus $O1$ overhead).

Proof: Each predicate is evaluated exactly once; summing their costs yields the stated time. Only the index set IF is stored, whose size is at most p; other storage is constant. □

ConstructVertices scans the library, uses CheckVertexType to verify each function's total, well-typed mapping, accumulates valid ones into vertex set V, and returns the set. The outline of the algorithms and the corresponding theorems are presented below.

Algorithm 5: ConstructVertices$S, m, n, h, k,$

Input: Finite S; integers m, n, h, k; library of candidate functions.
Output: Vertex set $V \subseteq \mathfrak{F}_{m,n}^{h,k} S$.

$V \leftarrow$
foreach $F \in$ **do**
  if *CheckVertexType*F, m, n, h, k, S **then**
    $V \leftarrow V \cup \{F\}$
  end
end
return V

Example 4.6 (Real-world use of ConstructVertices: selecting valid notification policies)**.** Let $S = \{\text{Email}, \text{SMS}, \text{Push}\}$ and take $m = n = h = k = 1$. Consider a library $= \{F_1, F_2, F_3, F_4\}$ of candidate policies $F : \mathcal{P}_1 S \to \mathcal{P}_1 S$:

$$
\begin{aligned}
F_1 A &:= A \setminus \{\text{SMS}\} \quad \text{(never send SMS)},\\
F_2 A &:= \begin{cases} \{\text{SMS}\}, & \text{Email} \in A,\\ \text{undefined}, & \text{otherwise}, \end{cases}\\
F_3 A &:= A \cup \{\text{Fax}\} \quad \text{Fax} \notin S,\\
F_4 A &:= \quad \text{(mute all)}.
\end{aligned}
$$

Running ConstructVertices keeps exactly those F that are total and well-typed.

Checks. F_1 is total and $F_1 A \subseteq S$, so it is accepted. F_2 is undefined at $A =$, hence rejected. $F_3 = \{\text{Fax}\} \not\subseteq S$, hence rejected as ill-typed. F_4 is total with image $\in \mathcal{P}_1 S$, so it is accepted.

Therefore the algorithm returns

$$V = \{F_1, F_4\} \subseteq \mathfrak{F}_{1,1}^{1,1} S.$$

Theorem 4.7 (Correctness of ConstructVertices)**.** *Algorithm 5 returns*

$$V = \{\, F \in \ \mid \textit{CheckVertexType}F, m, n, h, k, S = \textbf{\textit{true}} \,\}.$$

Consequently, $V \subseteq \mathfrak{F}_{m,n}^{h,k} S$ *and every* $F \in V$ *is a total, well-typed map* $\mathcal{P}_m S^h \to \mathcal{P}_n S^k$.

Proof: The algorithm scans all $F \in$ and inserts F into V iff CheckVertexType returns **true**. By the specification of CheckVertexType, this holds precisely for total, well-typed functions $\mathcal{P}_m S^h \to \mathcal{P}_n S^k$. Hence the characterization and inclusion. □

Theorem 4.8 (Time and Space of ConstructVertices)**.** *Let* $T_{\text{type}} F$ *denote the running time of CheckVertexType on* F*, and set* $T_{\text{type}}^{\max} := \max_{F \in} T_{\text{type}} F$*. Then*

$$T\big(\textit{ConstructVertices}\big) = \sum_{F \in} T_{\text{type}} F \le M\, T_{\text{type}}^{\max}, \qquad \textit{Space} = O|V| \ \textit{for storing accepted vertices}.$$

Proof: Each candidate is tested once by CheckVertexType; summing gives the time bound and the maximal bound follows immediately. Apart from V, the routine uses only constant extra space. □

BuildPropertyEdges initializes edge buckets, evaluates each vertex against all predicates, inserts vertices into satisfied buckets, removes empty ones, and returns resulting family of property-induced hyperedges. The outline of the algorithms and the corresponding theorems are presented below.

Algorithm 6: BUILDPROPERTYEDGES$V, \mathbf{\Phi}$

Input: Vertex set V; predicate family $\mathbf{\Phi} = \{\Phi_1, \ldots, \Phi_p\}$.
Output: Hyperedge family $\mathcal{E}$.

for $i \leftarrow 1$ **to** p **do**
  $E_i \leftarrow$
end
foreach $F \in V$ **do**
  $IF \leftarrow$EVALUATEPREDICATES$F, \mathbf{\Phi}$
  foreach $i \in IF$ **do**
    $E_i \leftarrow E_i \cup \{F\}$
  end
end
$\mathcal{E} \leftarrow$
for $i \leftarrow 1$ **to** p **do**
  if $E_i \neq$ **then**
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{E_i\}$
  end
end
return $\mathcal{E}$

Example 4.9 (Real-world use of BUILDPROPERTYEDGES: grouping notification policies)**.** Let $S = \{\mathrm{Email}, \mathrm{SMS}, \mathrm{Push}\}$ with $m = n = h = k = 1$. Consider vertices $V = \{F_1, F_2, F_3\}$ where

$$F_1A = A \setminus \{\mathrm{SMS}\}, \qquad F_2A = , \qquad F_3A = \begin{cases} \{\mathrm{Push}\}, & \mathrm{Push} \in A, \\ , & \text{otherwise.} \end{cases}$$

Predicates $\mathbf{\Phi} = \{\Phi_1, \Phi_2, \Phi_3, \Phi_4\}$:

$$\Phi_1 : \forall A\ \mathrm{SMS} \notin FA \quad \text{(no-SMS)},$$
$$\Phi_2 : \exists A\ \mathrm{Push} \in FA \quad \text{(push-enabled)},$$
$$\Phi_3 : \forall A\ FFA = FA \quad \text{(idempotent)},$$
$$\Phi_4 : \forall A\ FA = \quad \text{(mute-all)}.$$

Evaluations yield

	Φ_1	Φ_2	Φ_3	Φ_4
F_1	✓	✓	✓	×
F_2	✓	×	✓	✓
F_3	✓	✓	✓	×

Hence BUILDPROPERTYEDGES returns the nonempty hyperedges

$$E_1 = \{F_1, F_2, F_3\}, \quad E_2 = \{F_1, F_3\}, \quad E_3 = \{F_1, F_2, F_3\}, \quad E_4 = \{F_2\}.$$

These edges group policies by shared properties (no-SMS, push-enabled, idempotent, mute-all).

Theorem 4.10 (Correctness of BUILDPROPERTYEDGES)**.** *Let V be any vertex set and $\mathbf{\Phi} = \{\Phi_1, \ldots, \Phi_p\}$. Algorithm 6 returns a family $\mathcal{E}$ such that for each i,*

$$E_i \ = \ \{\, F \in V \mid \Phi_i F = \textbf{\textit{true}} \,\},$$

and $\mathcal{E}$ consists exactly of the nonempty E_i's. Hence $\mathcal{E} \subseteq \mathcal{P}V \setminus \{\}$.

Proof: For each $F \in V$, the call $IF \leftarrow$ EVALUATEPREDICATES$F, \mathbf{\Phi}$ (Thm. 4.4) returns the *exact* set of indices of true predicates. The algorithm inserts F into E_i for every $i \in IF$, so when all vertices are processed, each E_i is precisely $\{F \in V \mid \Phi_i F = \textbf{true}\}$. Empty E_i's are discarded; the remaining edges are nonempty subsets of V. □

Theorem 4.11 (Time and Space of BUILDPROPERTYEDGES)**.** *Let $T_{\mathrm{eval}}F := T\big($EVALUATEPREDICATES$F, \mathbf{\Phi}\big)$. Then*

$$T\big(\textit{BUILDPROPERTYEDGES}\big) \ = \ O\big(p \ \max_{F \in V} T_{\mathrm{eval}}F \ \sum_{i=1}^{p} |E_i|\big), \qquad \textit{Space} \ = \ O\big(p \ \sum_{i=1}^{p} |E_i|\big).$$

If $C_{\Phi}^{\max}$ is a uniform predicate bound, then $T\big(\textsc{BuildPropertyEdges}\big) = O\big(p\ |V| \cdot p \cdot C_{\Phi}^{\max}\ {}_i\,|E_i|\big)$.

Proof: Initialization and final filtering cost Op. For each $F \in V$, EvaluatePredicates costs $T_{\mathrm{eval}}F$ (Theorem 4.5), and inserting F into E_i for each $i \in IF$ yields a total of ${}_i\,|E_i|$ insertions across all vertices. Space is dominated by storing the p edge-containers and their members. With a uniform bound $C_{\Phi}^{\max}$, $T_{\mathrm{eval}}F \leq p\,C_{\Phi}^{\max}$. □

UniformizeEdges traverses each hyperedge, orders its vertices, partitions them into consecutive blocks of size r, optionally emits a remainder block, and returns the r-uniform family. The outline of the algorithms and the corresponding theorems are presented below.

Algorithm 7: UniformizeEdges$\mathcal{E}, r$

Input: Hyperedge family $\mathcal{E}$; integer $r \geq 1$.
Output: r-uniform hyperedge family $\mathcal{E}^r$.

$\mathcal{E}^r \leftarrow$
foreach $E \in \mathcal{E}$ **do**
  Let $F_1, \ldots, F_q$ be an arbitrary ordering of E.
  for $j \leftarrow 1$ **to** $\lfloor qr \rfloor$ **do**
    $\mathcal{E}^r \leftarrow \mathcal{E}^r \cup \{\{F_{j-1r1}, \ldots, F_{jr}\}\}$
  end
  if $q \bmod r \neq 0$ **then**
    Create a final edge with the remaining elements (optional, if nonempty).
  end
end
return $\mathcal{E}^r$

Example 4.12 (Real-world use of UniformizeEdges: forming breakout rooms)**.** Let the vertices be participants $V = \{A, B, C, D, E, F, G, H, I\}$. Two interest clusters (hyperedges) are

$$E_{\mathrm{Data}} = \{A, B, C, D, E, F, G, H\}, \qquad E_{\mathrm{UI}} = \{C, D, E, F, I\}.$$

With room size $r = 3$, UniformizeEdges partitions each hyperedge into consecutive blocks of three (and a possible remainder):

$$E^3_{\mathrm{Data}} = \{\{A, B, C\},\ \{D, E, F\},\ \{G, H\}\} \quad \text{last is a remainder of size 2,}$$
$$E^3_{\mathrm{UI}} = \{\{C, D, E\},\ \{F, I\}\} \quad \text{remainder of size 2.}$$

If remainders are omitted, the family is strictly 3-uniform; if included, at most one smaller block per original edge appears.

Theorem 4.13 (Validity of UniformizeEdges)**.** *Fix $r \geq 1$. For each input hyperedge $E \in \mathcal{E}$ with $|E| = q$, Algorithm 7 partitions E into $\lfloor qr \rfloor$ disjoint blocks of size exactly r (and optionally one remainder block of size $q \bmod r$). Hence the output family $\mathcal{E}^r$ is r-uniform if the remainder is omitted; if the remainder is included, then $\mathcal{E}^r$ is r-uniform* except *for at most one smaller block per input edge.*

Proof: The procedure orders the elements of E and groups them consecutively into $\lfloor qr \rfloor$ sets of r elements; these are pairwise disjoint and their union covers the first $r\lfloor qr \rfloor$ elements. The leftover $q \bmod r$ elements, if any, are either ignored (strict r-uniformity) or placed in a final block (single non-r remainder). Applying this independently to each $E \in \mathcal{E}$ produces the stated structure. □

Theorem 4.14 (Time and Space of UniformizeEdges)**.** *Let $Q := {}_{E\in\mathcal{E}}\,|E|$ be the total size of all input hyperedges. Then Algorithm 7 runs in OQ time and uses OQ space to store the output (plus Or transient workspace per processed edge).*

Proof: Each element of each E is visited once to form blocks, so the total time is linear in Q. The output stores every element at most once in some block, hence OQ space. At any moment we buffer at most one partial block of size r and loop-local variables, giving Or extra transient space per edge. □

Construct_HK_MN_SuperHyperGraph builds vertices via ConstructVertices, forms property-induced edges with BuildPropertyEdges, optionally uniformizes edges to size r, and returns a valid h, k–m, n SuperHyperGraph or fails. The outline of the algorithms and the corresponding theorems are presented below.

Algorithm 8: Construct_HK_MN_SuperHyperGraph$S, m, n, h, k, , \mathbf{\Phi}, r$

Input: Finite S; $m, n, h, k \in \mathbb{N}$; library ; predicates $\mathbf{\Phi}$; optional $r \in \mathbb{N}$.
Output: An h, k-ary m, n-SuperHyperGraph $V, \mathcal{E}'$.

$V \leftarrow$ConstructVertices$S, m, n, h, k,$
if $V =$ **then**
  return Fail (no well-typed vertices)
end
$\mathcal{E} \leftarrow$BuildPropertyEdges$V, \mathbf{\Phi}$
if $\mathcal{E} =$ **then**
  return $V, \{ \{F\} \mid F \in V \}$ `// Fallback: singleton edges`
end
if r *is provided* **then**
  $\mathcal{E}' \leftarrow$UniformizeEdges$\mathcal{E}, r$
  if $\mathcal{E}' =$ **then**
    return $V, \mathcal{E}$
  end
  return $V, \mathcal{E}'$
end
else
  return $V, \mathcal{E}$
end

Example 4.15 (Real-world use of Construct_HK_MN_SuperHyperGraph: smart-home policies)**.** Let $S = \{\text{Light}, \text{HVAC}, \text{DoorLock}, \text{Camera}\}$ and take $m = n = h = k = 1$. Candidate policies $= \{F_{\text{auto}}, F_{\text{secure}}, F_{\text{comfort}}\}$ with

$$F_{\text{auto}}A = A, \quad F_{\text{secure}}A = A \cup \{\text{DoorLock}\} \setminus \{\text{Light}\}, \quad F_{\text{comfort}}A = A \cup \{\text{HVAC}\}.$$

All are total maps $\mathcal{P}_1 S \to \mathcal{P}_1 S$, so ConstructVertices returns $V = $. Predicates $\mathbf{\Phi} = \{\Phi_1, \Phi_2, \Phi_3\}$:

$$\Phi_1 : \exists A \ \text{DoorLock} \in FA, \quad \Phi_2 : \exists A \ \text{HVAC} \in FA, \quad \Phi_3 : \forall A \ FFA = FA.$$

Evaluations yield

$$E_1 = \{F_{\text{secure}}\}, \quad E_2 = \{F_{\text{comfort}}\}, \quad E_3 = \{F_{\text{auto}}, F_{\text{secure}}, F_{\text{comfort}}\}.$$

Thus BuildPropertyEdges$V, \mathbf{\Phi}$ returns $\mathcal{E} = \{E_1, E_2, E_3\}$. With $r = 2$, UniformizeEdges$\mathcal{E}, 2$ gives

$$E_3^2 = \{\{F_{\text{auto}}, F_{\text{secure}}\}\} \quad \text{remainder } \{F_{\text{comfort}}\} \text{ optional},$$

and the algorithm returns the $1, 1$–$1, 1$ SuperHyperGraph $V, \mathcal{E}'$ ready for A/B testing of policy pairs.

Theorem 4.16 (Validity)**.** *Let S be finite and nonempty, and $m, n, h, k \in \mathbb{N}$. If Algorithm 8 returns $V, \mathcal{E}'$, then $V \neq$ and $V, \mathcal{E}'$ is an h, k-ary m, n-SuperHyperGraph on S. Moreover, if r is provided and $\mathcal{E}' =$ UniformizeEdges$\mathcal{E}, r$ is returned, then $V, \mathcal{E}'$ is r-uniform.*

Proof: By Algorithm 5, every $F \in V$ satisfies CheckVertexType, hence $F : \mathcal{P}_m S^h \to \mathcal{P}_n S^k$ is total and well-typed. Therefore $V \subseteq \mathfrak{F}_{m,n}^{h,k} S$ and $V \neq$ by the explicit check. Algorithm 6 forms each hyperedge E_i as a set of vertices that satisfy Φ_i; by construction $E_i \subseteq V$, and empty sets are discarded, so $\mathcal{E} \subseteq \mathcal{P}V \setminus \{\}$. If r is not provided, $V, \mathcal{E}$ is an h, k-ary m, n-SuperHyperGraph by definition. If r is provided, Algorithm 7 partitions each $E \in \mathcal{E}$ into blocks of size r (except possibly the final remainder, which is either included or dropped by the stated option), hence every hyperedge in $\mathcal{E}'$ has size r (or the remainder policy yields nonempty edges by construction). Thus $V, \mathcal{E}'$ is r-uniform and still a family of nonempty subsets of V. □

Theorem 4.17 (Time Complexity)**.** *Let $M := ||$ and $p := |\mathbf{\Phi}|$. Write T_{eval} for the time to evaluate F once, and let C_{Φ} be a uniform upper bound on the time to decide $\Phi_i F$ for any i and F. Then the running time of Algorithm 8 satisfies*

$$T = \underbrace{O\big(M \cdot D \cdot T_{\text{eval}}\ \ k\,U_n\big)}_{\text{vertex typing}}\ \underbrace{O\big(M \cdot p \cdot C_{\Phi}\big)}_{\text{predicate evaluation}}\ \underbrace{O\big(\sum_{i=1}^{p} |E_i|\big)}_{\text{edge assembly}}\ \underbrace{O\big(|\mathcal{E}|\ \ |V|\big)}_{\text{bookkeeping}}\ \underbrace{O\big(\sum_{E\in\mathcal{E}} |E|\big)}_{\text{uniformization (if used)}}\ .$$

In particular, the dependence on m and n is carried by $D = N_m^h$ and U_n, which grow tower-exponentially with m and n.

Proof: *Vertex typing.* For each $F \in$, the Algorithm iterates over all D inputs, performs one F-evaluation (T_{eval}), and checks k components for membership in $\mathcal{P}_n S$, each costing at most U_n in the worst case; hence $O\big(D \cdot T_{\text{eval}}\ \ k\,U_n\big)$ per F, and M such F's.

Predicate evaluation. For each $F \in V \subseteq$, each of the p predicates is tested once, costing at most C_{Φ} each; hence $OMpC_{\Phi}$.

Edge assembly. Each F contributes to E_i precisely when $\Phi_i F$ holds; in total, writing E_i for the constructed property edges, the insertion work is $O_i\,|E_i|$, plus linear bookkeeping for collecting nonempty edges.

Uniformization. Algorithm 7 scans each E once and produces $\lfloor |E| r \rfloor$ r-sized blocks (and optionally one remainder), hence $O|E|$ per E, summing to $O_{E\in\mathcal{E}}\,|E|$. □

Theorem 4.18 (Space Complexity)**.** *The space usage of Algorithm 8 is*

$$O\big(\ |V|\quad \sum_{i=1}^{p} |E_i|\quad D\big),$$

dominated by storage for vertices, property–induced edges, and the working memory needed to enumerate and type–check domain tuples. If uniformization is requested, an additional $O\big(\sum_{E\in\mathcal{E}} |E|\big)$ space is needed transiently.

Proof: We decompose peak memory by phases of Algorithm 8.

(1) Persistent structures. (i) The algorithm stores the accepted vertices V explicitly, contributing $O|V|$ words (we count handles/pointers to functions). (ii) In BuildPropertyEdges, it constructs one container $E_i \subseteq V$ for each predicate Φ_i and discards empty ones at the end. Hence the total edge storage is $\sum_{i=1}^{p} |E_i|$, i.e., $O\big(\sum_i |E_i|\big)$.

(2) Working memory for type checking. In ConstructVertices every candidate F is tested by CheckVertexType, which enumerates the domain $\mathcal{P}_m S^h$ of size $D = N_m^h$ with $N_{t1} = 2^{N_t}$ and $N_0 = |S|$. A streaming (lexicographic or mixed–radix) iterator needs:

$$\text{(a) one current input tuple } \mathbf{A} = A_1, \ldots, A_h, \quad \text{(b) a constant number of loop counters/indices.}$$

Each $A_i \in \mathcal{P}_m S$ is itself a subset of $\mathcal{P}_{m-1} S$, so storing $\mathbf{A}$ uses at most $h\,N_{m-1}$ atomic references in the worst case. The check of the output $Y = F\mathbf{A} = Y_1, \ldots, Y_k$ can be performed *componentwise and on the fly*: for each j, we test $Y_j \in \mathcal{P}_n S$ using a structural membership routine that streams through Y_j and then discards it; therefore no accumulation over different inputs occurs. The peak extra buffer while inspecting a single component is ON_{n-1}, but since we process at most one component at a time and immediately release it, the global peak is bounded by the iterator's working set plus the current tuple buffer.

Combining these, the transient workspace during type checking is

$$O\big(h\,N_{m-1}\big) \quad \text{(for } \mathbf{A}\text{)} \qquad \text{plus} \qquad O1 \text{ (counters).}$$

Because $D = 2^{h\,N_{m-1}}$ (since $N_m = 2^{N_{m-1}}$), the inequality $2^x \geq x$ for $x \geq 1$ yields $D \geq h\,N_{m-1}$ whenever $h\,N_{m-1} \geq 1$. Thus the iterator/buffer footprint is upper–bounded by OD, which we use as a coarse but valid bound that subsumes the working set.

(3) Optional uniformization. If UniformizeEdges is invoked, it scans each $E \in \mathcal{E}$ once and emits r–blocks. At any moment it retains only a partial block (size $\leq r$) and loop variables; the output family $\mathcal{E}^r$ itself is not counted as transient. Hence the additional *transient* space is $O\big(\sum_{E\in\mathcal{E}} |E|\big)$ across the pass (the output storage is charged to the final edge family).

(4) Peak aggregation. Summing persistent and transient contributions, the peak space is

$$O\big(\ |V|\quad \sum_{i=1}^{p} |E_i|\quad \underbrace{D}_{\text{iterator / buffers}}\ \big),$$

plus the optional $O\big(_{E\in\mathcal{E}}\,|E|\big)$ during uniformization. This proves the claim.

Sharper (optional) bound. If one prefers to avoid the coarse OD term, the working memory in (2) can be bounded more tightly by $O\big(h\,N_{m-1}\,N_{n-1}\big)$ (current tuple plus one streamed output component), so the peak becomes $O\big(|V|_i\,|E_i|h\,N_{m-1}\,N_{n-1}\big)$, which is asymptotically $\subseteq O\big(|V|\ _i\,|E_i|\ D\big)$ because $D = 2^{h\,N_{m-1}} \geq h\,N_{m-1}$ and, in typical regimes, $N_{n-1} \leq D$. We keep the theorem's statement for simplicity. □

5| Conclusion

In this paper, we introduced the concept of an h, k-*ary* m, n-***SuperHyperGraph***, which extends the already well-studied SuperHyperGraph by employing the framework of h, k-*ary* m, n-***SuperHyperFunctions***. In future work, we aim to explore further extensions of the h, k-*ary* m, n-***SuperHyperGraph*** using different paradigms such as Fuzzy Graphs[25, 26, 27], Intuitionistic Fuzzy Graphs[28, 29], Picture Fuzzy Graphs[30, 31], Linear Diophantine Fuzzy Graphs [32, 33], Hesitant Fuzzy Graphs[34, 35], Neutrosophic Graphs[36, 37, 38, 39, 40, 41, 42], and Plithogenic Graphs[43, 44].

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.

Consent to Publish declaration

The author approved to Publish declarations.

Ethics Approval and Consent to Participate

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

Funding

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

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.

Research Integrity

The authors hereby confirm that, to the best of their knowledge, this manuscript is their original work, has not been published in any other journal, and is not currently under consideration for publication elsewhere at this stage.

Disclaimer (Note on Computational Tools)

No computer-assisted proof, symbolic computation, or automated theorem proving tools (e.g., Mathematica, SageMath, Coq, etc.) were used in the development or verification of the results presented in this paper. All proofs and derivations were carried out manually and analytically by the authors.

Disclaimer (Limitations and Claims)

The theoretical concepts presented in this paper have not yet been subject to practical implementation or empirical validation. Future researchers are invited to explore these ideas in applied or experimental settings. Although every effort has been made to ensure the accuracy of the content and the proper citation of sources, unintentional errors or omissions may persist. Readers should independently verify any referenced materials.

To the best of the authors' knowledge, all mathematical statements and proofs contained herein are correct and have been thoroughly vetted. Should you identify any potential errors or ambiguities, please feel free to contact the authors for clarification.

The results presented are valid only under the specific assumptions and conditions detailed in the manuscript. Extending these findings to broader mathematical structures may require additional research. The opinions and conclusions expressed in this work are those of the authors alone and do not necessarily reflect the official positions of their affiliated institutions.

Competing interests

Author has declared that no competing interests exist.

References

[1] Gross, J. L., Yellen, J., & Anderson, M. (2018). *Graph theory and its applications*. Chapman and hall/CRC. https://www.amazon.com/Graph-Theory-Applications-Textbooks-Mathematics/dp/1482249480

[2] Diestel, R. (2025). *Graph theory* (6th ed.). Springer. https://doi.org/10.1007/978-3-662-70107-2

[3] Gao, Y., Zhang, Z., Lin, H., Zhao, X., Du, S., & Zou, C. (2020). Hypergraph learning: Methods and practices. *IEEE transactions on pattern analysis and machine intelligence, 44*(5), 2548-2566. https://doi.org/10.1109/tpami.2020.3039374

[4] Feng, Y., You, H., Zhang, Z., Ji, R., & Gao, Y. (2019, July). Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 33, No. 01, pp. 3558-3565). https://doi.org/10.1609/aaai.v33i01.33013558

[5] Jose, B. K., & Tuza, Z. (2009). Hypergraph domination and strong independence. *Applicable analysis and discrete mathematics, 3*(2), 347-358. https://doi.org/10.2298/AADM0902347J

[6] Campoverde Valencia, E. M., Chuisaca Vásquez, J. P., & Becerra Lois, F. Á. (2025). Multineutrosophic analysis of the relationship between survival and business growth in the manufacturing sector of azuay province, 2020–2023, using plithogenic n-superhypergraphs. *Neutrosophic sets and systems, 84*(1), 28. https://www.researchgate.net/publication/402314471

[7] Méndez Bravo, J. C., Bolanos Piedrahita, C. J., Méndez Bravo, M. A., & Pilacuan Bonete, L. M. (2025). Integrating smed and industry 4.0 to optimize processes with plithogenic n-superhypergraphs. *Neutrosophic sets and systems, 84*(1), 27. https://doi.org/10.1109/tpami.2020.3039374

[8] Fujita, T. (2025). Multi-superhypergraph neural networks: A generalization of multi-hypergraph neural networks. *Neutrosophic computing and machine learning, 39*(1), 328-347. https://doi.org/10.1504/IJCAST.2025.10073956

[9] Mogro Cepeda, Y. V., Riofrío Guevara, M. A., Jácome Mogro, E. J., & Piovanelli Tizano, R. (2024). Impact of irrigation water technification on seven directories of the san juan-patoa river using plithogenic n-superhypergraphs based on environmental indicators in the canton of pujilí, 2021. *Neutrosophic sets and systems, 74*(1), 6. https://doi.org/10.5281/zenodo.14417807

[10] Fujita, T., & Smarandache, F. (2024). *Advancing uncertain combinatorics through graphization, hyperization, and uncertainization: Fuzzy, neutrosophic, soft, rough, and beyond: Second volume*. Infinite study. https://www.researchgate.net/publication/386082978

[11] Berrocal Villegas, S. M., Montalvo Fritas, W., Berrocal Villegas, C. R., Flores Fuentes Rivera, M. Y., Espejo Rivera, R., Bautista Puma, L. D., & Macazana Fernández, D. M. (2025). Using plithogenic n-superhypergraphs to assess the degree of relationship between information skills and digital competencies. *Neutrosophic sets and systems, 84*(1), 41. https://digitalrepository.unm.edu/nss_journal/vol84/iss1/41/

[12] Zhu, S. (2025). Neutrosophic n-superhypernetwork: A new approach for evaluating short video communication effectiveness in media convergence. *Neutrosophic sets and systems, 85*(1), 58. https://doi.org/10.5281/zenodo.15420570

[13] Nacaroglu, Y., Akgunes, N., Pak, S., & Cangul, I. N. (2021). Some graph parameters of power set graphs. *Advances & applications in discrete mathematics, 26*(2). https://doi.org/10.17654/DM026020211

[14] Shalu, M. A., & Yamini, S. D. (2014). Counting maximal independent sets in power set graphs. *Indian institute of information technology design & manufacturing (IIITD&M) Kancheepuram, India*. https://www.researchgate.net/profile/Shalu-M-A/publication/262603561

[15] Jech, T. (2003). *Set theory*. Berlin, heidelberg: Springer Berlin heidelberg. https://doi.org/10.1007/3-540-44761-X

[16] Bretto, A. (2013). Hypergraph theory. *An introduction. Mathematical engineering. cham: Springer, 1*, 209-216. https://doi.org/10.1007/978-3-319-00080-0

[17] Berge, C. (1984). *Hypergraphs: Combinatorics of finite sets* (Vol. 45). Elsevier. https://www.amazon.com/Hypergraphs-45-Combinatorics-North-Holland-Mathematical/dp/0444874895

[18] Smarandache, F. (2024). Foundation of superhyperstructure & neutrosophic superhyperstructure. *Neutrosophic sets and systems, 63*(2024), 367-381. https://digitalrepository.unm.edu/nss_journal/vol63/iss1/21/

[19]Khali, H. E., Güngör, G. D., & Zaina, M. A. N. (2022). Neutrosophic superhyper bi-topological spaces: Original notions and new insights. *Neutrosophic sets and systems, 51*(1), 3. https://fs.unm.edu/NSS/SuperHyperStructure.pdf

[20]Smarandache, F. (2020). *Extension of hypergraph to n-superhypergraph and to plithogenic n-superhypergraph, and extension of hyperalgebra to n-ary (classical-/neutro-/anti-) hyperalgebra*. Infinite study. https://www.researchgate.net/publication/343862930

[21]Smarandache, F. (2022). Introduction to the n-superhypergraph-the most general form of graph today. *Infinite study*. https://doi.org/10.5281/zenodo.3783103

[22]Fujita, T. (2024). Hypergraph-and superhypergraph-based models for patient care, disease transmission, and healthcare collaboration networks (with an exploration of (m, n)-superhypergraphs).

[23]Jdid, M., Smarandache, F., & Fujita, T. (2025). A linear mathematical model of the vocational training problem in a company using neutrosophic logic, hyperfunctions, and superhyperfunction. *Neutrosophic sets and systems, 87*(1), 1. https://www.researchgate.net/profile/T-Fujita-2/publication/392238093

[24]Smarandache, F. S. (2023). Neutrosophic superhyperstructure: Current understanding and future directions. *Infinite study*.

[25]Rosenfeld, A. (1975). Fuzzy graphs. *In fuzzy sets and their applications to cognitive and decision processes* (pp. 77-95). Academic press. https://doi.org/10.1016/B978-0-12-775260-0.50008-6

[26]Mordeson, J. N., & Nair, P. S. (2012). *Fuzzy graphs and fuzzy hypergraphs* (Vol. 46). Physica. https://www.amazon.nl/-/en/John-N-Mordeson/dp/3790812862

[27]Al-Hawary, T. (2011). Complete fuzzy graphs. *International journal of mathematical combinatorics, 4*, 26. https://fs.unm.edu/IJMC/CompleteFuzzyGraphs.pdf

[28]Kishore Kumar, P. K., Lavanya, S., Rashmanlou, H., & Jouybari, M. N. (2017). Magic labeling on interval-valued intuitionistic fuzzy graphs. *Journal of intelligent & fuzzy systems, 33*(6), 3999-4006. https://doi.org/10.3233/JIFS-17847

[29]Akram, M., Davvaz, B., & Feng, F. (2013). Intuitionistic fuzzy soft k-algebras. *Mathematics in computer science, 7*(3), 353-365. https://doi.org/10.1007/s11786-013-0158-5

[30]Shoaib, M., Mahmood, W., Al-Kenani, A. N., & Islam, S. (2022). Notes on upper and lower truncation of picture fuzzy graphs. *Discrete dynamics in nature and society, 2022*(1), 7646828. https://doi.org/10.1155/2022/7646828

[31]Rosyida, I., & Indrati, C. R. (2023). An algorithm for coloring of picture fuzzy graphs based on strong and weak adjacencies, and its application. Algorithms, 16(12), 551. https://doi.org/10.3390/a16120551

[32]Hanif, M. Z., Yaqoob, N., Riaz, M., & Aslam, M. (2022). Linear Diophantine fuzzy graphs with new decision-making approach. AIMS Mathematics, 7(8), 14532–14556. https://doi.org/10.3934/math.2022798

[33]Parimala, M., & Jafari, S. (2024). Spherical linear Diophantine fuzzy graphs: Unleashing the power of fuzzy logic for uncertainty modeling and real-world applications. Axioms, 13(3), 153. https://doi.org/10.3390/axioms13030153

[34]Alkouri, A., AbuHijleh, E. A., Alafifi, G., Almuhur, E., & Al-Zubi, F. M. A. (2023). More on complex hesitant fuzzy graphs. AIMS Mathematics, 8(11), 27653–27674. https://doi.org/10.3934/math.20231408

[35]Gong, Z., & Wang, J. (2021). Hesitant fuzzy graphs, hesitant fuzzy hypergraphs and fuzzy graph decisions. Journal of Intelligent & Fuzzy Systems, 40(1), 865–875. https://doi.org/10.3233/JIFS-189203

[36]Broumi, S., Talea, M., Bakali, A., & Smarandache, F. (2016). Single valued neutrosophic graphs. Journal of New Theory, 10, 86–101. https://dergipark.org.tr/en/pub/jnt/issue/24472/259409

[37]Broumi, S., Talea, M., Bakali, A., Smarandache, F., & Kumar, P. K. (2017). Shortest path problem on single valued neutrosophic graphs. In 2017 International Symposium on Networks, Computers and Communications (ISNCC) (pp. 1–6). IEEE. https://doi.org/10.1109/ISNCC.2017.8072015

[38]Broumi, S., Bakali, A., & Bahnasse, A. (2018). Neutrosophic sets: An overview. Infinite Study. https://fs.unm.edu/NeutrosophicSetsAnOverview.pdf

[39]Shi, X., Kosari, S., Rashmanlou, H., Broumi, S., & Hussain, S. S. (2023). Properties of interval-valued quadripartitioned neutrosophic graphs with real-life application. Journal of Intelligent & Fuzzy Systems, 44(5), 7683–7697. https://doi.org/10.3233/JIFS-223455

[40]Quek, S. G., Selvachandran, G., Ajay, D., Chellamani, P., Taniar, D., Fujita, H., Duong, P., Son, L. H., & Giang, N. L. (2022). New concepts of pentapartitioned neutrosophic graphs and applications for determining safest paths and towns in response to COVID-19. Computational and Applied Mathematics, 41(4), 151. https://doi.org/10.1007/s40314-022-01835-4

[41]Broumi, S., Bakali, A., Talea, M., Smarandache, F., & Singh, P. K. (2019). Properties of interval-valued neutrosophic graphs. In Fuzzy Multi-criteria Decision-Making Using Neutrosophic Sets (pp. 173–202). Springer. https://doi.org/10.1007/978-3-030-11471-8_8

[42]Broumi, S., Bakali, A., Talea, M., & Smarandache, F. (2018). An isolated bipolar single-valued neutrosophic graphs. In Information Systems Design and Intelligent Applications (pp. 816–822). Springer. https://doi.org/10.1007/978-981-10-4748-8_78

[43]Sultana, F., Gulistan, M., Ali, M., Yaqoob, N., Khan, M., Rashid, T., & Ahmed, T. (2023). A study of plithogenic graphs: Applications in spreading coronavirus disease (COVID-19) globally. Journal of Ambient Intelligence and Humanized Computing, 14(10), 13139–13159. https://doi.org/10.1007/s12652-021-03384-7

[44]Kandasamy, W. B. V., Ilanthenral, K., & Smarandache, F. (2020). Plithogenic graphs. Infinite Study. https://fs.unm.edu/PlithogenicGraphs.pdf