Unified System Modeling and Simulation via Constraint Hypergraphs

John Morris[1]
Department of Mechanical Engineering
Clemson University
Clemson, SC, USA
email: jhmrrs@clemson.edu

Gregory Mocko
Department of Mechanical Engineering
Clemson University
Clemson, SC, USA
email: gmocko@clemson.edu

John Wagner, P.E.
Department of Mechanical Engineering
Clemson University
Clemson, SC, USA
email: jwagner@clemson.edu

This paper describes the theory behind constraint hypergraphs: a novel modeling framework that can be used to universally represent and simulate complex systems. Multi-domain system models are traditionally compiled from many diverse frameworks, each based in a single domain. Incompatibilities between these frameworks prevent information from being shared resulting in data silos, duplicate work, and knowledge gaps. A constraint hypergraph addresses these problems by providing a universal modeling framework within which all model prescriptions can be expressed. This methodology expands mathematical structures previously explored in abstract mathematics and systems theory into a new executable framework. Each hypergraph expresses the holistic behavior of a system in a declarative paradigm that describes the relationships between system properties. In addition to modeling, it is shown how constraint hypergraphs can be used for universal, cross-cutting simulation through principles of function composition. The theoretical framework of a constraint hypergraph is demonstrated with a practical representation of a hybrid system, combining a discrete-event simulation and continuous PID controller into a single model of an elevator lift system.

Keywords: Model-Based Systems Engineering, Hypergraphs, Constraint Graphs

1 Introduction

Everything in the universe is a system; in their efforts to understand those systems, scientists and engineers fabricate models that describe their behavior. These models are defined according to prescribed frameworks: the language of algebra, pseudocode, even visual diagrams such as circuit networks or architectural blueprints. Most frameworks are developed for a targeted domain such as geometry, artificial intelligence, or economics. Where these niche representations fall short is in the representation and simulation of systems spanning multiple domains [1], preventing modelers from understanding how a part of one domain, such as bird migrations, affects an object in another subsystem, such as the energy-production of a windmill.

To grapple with these cross-cutting interactions, systems theorists have devised general system modeling frameworks. Most of these, such as SysML or Modelica, employ an object-oriented paradigm [2,3]. They classify objects in the system, as well as interfaces for sharing information between them [4]. Object-oriented modeling is often preferred because it is easier for humans to read and understand; the cost is that models get broken into isolated subsystems where interactions between domains may not be fully captured. Because of this, object-oriented models can be difficult to integrate, with simulation limited to restricted domains [5].

Another class of model frameworks is declarative. A model following a declarative paradigm establishes the relationships comprising a system without enforcing a preferred order, connecting all system components at the same level of representation [6]. Domain-specific modeling frameworks include bond graphs for dynamic systems, the constraint networks of computer science [7] and the factor graphs applied primarily to coding theory [8,9]. Because they use functions as their underlying semantic unit, declarative models are far more interoperable than object-oriented frameworks, and consequently better suited for expressing system behavior [10]. This paper describes a new declarative framework for the universal modeling of heterogeneous systems, addressing the yet-unsolved problem of providing both generalized representation and execution of a system model. This framework is termed a constraint hypergraph (CH), borrowing phraseology from the field of constraint theory [7].

A CH is composed of nodes and edges. The nodes represent system properties such as position, speed, color, weight, etc. and are connected to each other via edges. Each edge represents an explicit constraint that maps a set of nodes to a single value of another node. This can be read as: "given values for A and B, the edge constrains the value of C to be x." Because such constraints are often multidimensional (such as $C = A + B$), an edge constraint may be derived from the values of multiple nodes resulting in hyperedges in a hypergraph. A basic example of a CH consisting of five nodes and two constraints describing a mass-spring system is shown in Fig. 1. The constraints for this dynamic system could also be represented algebraically as $\ddot{x} = \frac{k}{m}x$, where x, k and m denote displacement, stiffness, and mass respectively. The CH in the figure shows that $\ddot{x}$ can be calculated given known values for the nodes $\frac{k}{m}$ and x and that $\frac{k}{m}$ can be in turn calculated if k and m are known.

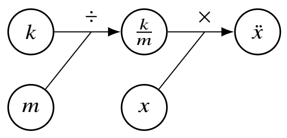


Fig. 1 An example of a CH for mass-spring system.

CHs reveal how each property in the system affects another. This is not limited to a single system; CHs also describe how systems interact with other systems, and how models of two different systems can be combined into a single whole. The composition of two CHs is the union of the set of nodes in each, joining the hypergraphs along their shared nodes. This is illustrated in Fig. 2, where the mass-spring system has been extended to include a damper.

In both of these figures, only one direction of the mapping has been made (showing relationships leading to the constraints on

[1]Corresponding Author.
Version 0.1, January 17, 2025

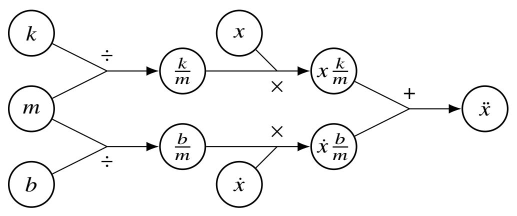


Fig. 2 An example of a CH for mass-spring-damper system, extending the system in Fig. 1.

$\ddot{x}$). Relationships in CHs are implicitly causal; acausal constraints can only be included by adding additional inverted edges. This leads to more edges and can crowd the diagrams. This makes for an important point: in contrast to a language like SysML, CHs are not diagrammatic nor intended for visual modeling. They are formulated as a robust, mathematical structure for capturing the fundamental relationships between system entities, allowing agents (both human and autonomous) to fully understand the behavior of the system and simulate its potential states.

CHs represent a new framework, but one whose structure is not entirely novel. The mathematics and methods of the framework amalgamate results from the fields of category theory, computer science, constraint theory, and logic programming. The contribution of this paper is to both describe how these findings can be combined to create a useful framework, and also demonstrate how the resulting strongly-coupled, multi-domain modeling scheme can be applied to the general representation and simulation of real-world systems. This is accomplished starting by providing a background of previous work on multi-domain system modeling (Section 2), followed by a robust definition of a CH. This definition is used to explore how CHs represent (Section 3) and simulate (Section 4) systems, as well as their limitations (Section 5). Finally, a case study using CHs to represent an elevator system is provided in Section 6.

2 Background

As long as engineers have considered systems, they have struggled to understand how objects of disjoint domains affect each other. From how forging affects the surface of a steel tool, to learning the ways in which music affects emotions, systems are by their nature heterogeneous, and consequently require methods of representing their heterogeneity. The section recounts only a fraction of the work in this field, with a focus on motivating CHs and their use by systems engineers.

2.1 System Modeling Frameworks. A system is an arrangement of things that together exhibit behavior that the individual components do not [11]. If systems are constructs of the real world, then their corollary in the virtual domain is a system model, which relates the information known about a reified system. A system model is composed of properties approximating attributes of the real system, which are commonly represented with variables [12]. Each property is allowed to vary over a set of values. The set of possible combinations of each property is known as the state space.

What establishes a system model as representing a system is the provision of a set of relationships between the system properties, such that the possible states for a system are restricted. The restriction of system states is the definition of behavior given by Willems [5,13], and is suitable for the purposes of this article. A system model is considered in this paper as a description of a system that provides rules governing the possible values for a set of properties. This general definition describes nearly every modeling framework as a system model. Geometric systems have properties of distance and orientation constrained by dimensions and geometric relationships, while finite element models have properties of

Table 1 A non-exhaustive list of graph-based system modeling frameworks

Framework	System Domain	Source
Block diagrams	Dynamic	[23]
Bond graphs	Dynamic	[24,25]
Linear graphs	Dynamic	[26,27]
Stock and Flow Diagrams	Dynamic	[28,29]
Organization charts	Knowledge	
Entity-Relationship Model	Knowledge	[30]
Circuit diagrams	Electronics	[18]
Flow charts	Processes	[31]
Petri nets	Discrete-event	[19]
Markov chains	Discrete-event	[32]
Bayesian networks	Stochastic	[33]
Causal models	Multi-Domain	[34]
Component objects (Paredis)	Multi-Domain	[35]
Composable Objects (COBS)	Multi-Domain	[36]
SysML	Multi-Domain	[37]
Constraint graphs	Multi-Domain	[38–40]
Factor graphs	Multi-Domain	[8,9]

nodal locations or thermal flow constrained by partial differential equations. Model frameworks applied to social [14], ecological [15], economic [16], and other domains also fit under this description.

The mathematics for describing system models are provided by category theory (see the introduction by Spivak and Fong [17]), which provides tools for defining the relationships inherent across systems. Efforts in this regard began with Fong in 2016 [18], who specifically introduced decorated cospans for deconstructing circuit networks into hypergraphs. This was built upon by Baez and Pollard, who applied decorated cospans to Petri nets [19], and by Patterson et al. in deconstructing dynamic systems [20,21]. The categories used in these works are extensions of graphs, and generalize the connections between a set of things [22]. This similarity to systems–also composed of entities connected to each other by constraints–leads to many system modeling frameworks being graph-based in nature. A selection of graph-based frameworks along with their primary domain of use is shown in Table 1. The diversity of modeling schemes stems from the varied interpretations of nodes and edges, where each might be assigned domain-specific system elements, relationship types, or composition rules.

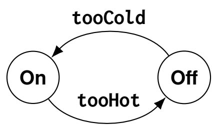
2.2 Model Interoperability. With such a diversity of modeling types, establishing interoperability becomes a critical challenge in multi-domain systems. When sub-systems are modeled within a domain-specific framework, inter-framework incompatibilities can result in modeling silos: where a holistic system is composed of individual subsystem models which cannot effectively share information. The primary result of sequestered models is the inability to capture relationships between elements in different silos. This is a critical challenge for decisions makers; for instance, a systems engineer may not be able to see how changing a sensor will influence the weight distribution in an aircraft because the sensor interfaces, electric circuits, and payload models are all siloed in incompatible frameworks. Secondary effects of model silos include information loss during data exchanges, duplication (where singular entities are represented multiple times throughout a system), and challenges modifying or scaling due to the inter-system complexities not being captured in the global model.

There are two general approaches to providing model interoperability: transformation or unification [41], which relate respectively to the competing concepts of object-oriented and declarative modeling. In the former, subsystems are treated as independent black boxes. Each subsystem has the ability to transform certain signals delivered via a defined interface (a port) [42,43]. This allows each subsystem to be uniquely defined according to its own domain-specific framework provided there is a standard interface for inter-component communication. The second approach is of system unification, where the components of each individual system are deconstructed and reconciled into a single holistic model. Of the two, transformer frameworks are more common, since it is simpler to define interfaces than fully deconstruct every member of a system.

Transformative frameworks, such as block diagrams, stock and flow diagrams, or Entity-Relation models, focus on the objects in a system. Because each class of object is unique, transformative frameworks must define unique methods for sharing information among the system [4]. These systems are often easier for designers to work with, since objects correspond to how a designer deconstructs a system [41]. Object-oriented solutions include SyDer [44], component objects [35], the Functional Mockup Interface (FMI) standard [45]–strictly for dynamic system simulations, though standards for hybrid systems are being developed [46]–and SysML, a multi-domain modeling framework developed in 2007 [37] with the objective of supporting systems engineering tasks [47]. These weakly-coupled frameworks may struggle to simulate reactive systems featuring complex relationships between many system components [48].

Strong coupling is provided by unifying models, which do not prescribe objects, but instead deal primarily with the system constraints and the properties they relate. The relationships of a unifying framework are functions, which are common across all modeling domains. Where object-oriented frameworks are nested, the use of functions results in unifying models being flat, with every relationship elevated to the status of "first-class citizen" [6]. This makes combining models simpler since information does not need to be transformed between system components. Friedman was the first to show how functional frameworks could be used to describe universal system behavior, writing about systems of mathematical expressions constraining a set of variables [38,49]. These constraint models, as labeled by Friedman, were used to expose system complexity by representing the system behaviors, though they were not formulated to provide execution. CHs, the framework introduced in this paper, are an expansion of Friedman's work adapted to universally represent and simulate system behavior.

2.3 Motivation. To motivate how system unification is accomplished by CHs, consider the example given by Gomes of a continuous and discrete system [50], where the discrete model is a state machine of a simple thermostat regulating the temperature in a room according to the following state machine:



Here the states "On" and "Off" refer to the status of a heater controlled by the thermostat, and the transitions `tooHot` and `tooCold` are triggered when the temperature in the room exceeds some threshold. The change in the room temperature is modeled by the following continuous equation, where T is the temperature, $r > 0$ is a constant rate of change, $q \in \{0, 1\}$ indicates whether the heater is on or off, and k is the rate of heating:

$$\dot{T} = -r(T - kq)$$

In their survey on cosimulation [50], Gomes found that the only established methods for coupling the discrete and continuous models into a hybrid system was to interpret subsystems from one domain in terms of the other, a messy translation process prone to information loss. Alternatively, by identifying the system properties and functional constraints, the system can be fully reconciled into the CH shown in Fig. 3.

Constraint hypergraphs are not a visual framework, but to illustrate their composition, the following diagrammatic schema is followed in Fig. 3 and the rest of this paper. In the figure, each circular node represents a variable, and each black box describes the function for an edge. Arrows wire nodes to edges (showing the function domain) and edges to a node (the codomain). Multiple domain arrows indicate a hyperedge, where the domain is the Cartesian product of all linked nodes. When order matters in the arguments, labels may be written on the domain arrows in the form $a, b, \ldots, z$. Finally, nodes that have only a single domain arrow are removed, replaced with wires connecting edges to edges, similar to a short circuit in an electrical diagram.

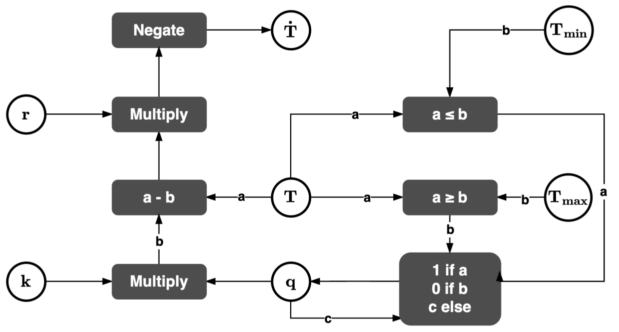


Fig. 3 Constraint hypergraph for a hybrid system modeling a thermostat controlling the room temperature x.

Building a CH reveals the two variables shared by the systems: q and T, allowing the systems to be connected along the respective nodes. The result is a fully coupled system, such that the relationships between the continuous and discrete models are completely expressed by the CH, exposing the behavior of the holistic system. In considering the hypergraph in Fig. 3, one notices that the discrete and continuous models are not easily distinguished. This is typical of a declarative modeling paradigm, where each model is fully deconstructed to the functional level. The categorization–and consequent isolation–of subsystems is largely a feature of object-oriented modeling, which treats each subsystem independently. Contrast this with a CH, where each function is considered and connected equally regardless of its subsystem of origin.

3 Structure of a Constraint Hypergraph

3.1 Formulation. The general definition of a CH is given as

Definition 1 *Constraint hypergraph: a graph $\mathcal{H}$ composed of a set V as vertices and a set of functions E as edges, where each vertex $v \in V$ is a set, and each function $e \in E$ maps between the Cartesian product of a subset of V: $V' \subseteq V$ and another element of V: $T \in V$ such that $\prod_{V'} \xrightarrow{e} T$.*

Let $\prod_X$ as used above be given as the Cartesian product of all sets in the superset X such that

$$\prod_X := x_1 \times x_2 \times \ldots \times x_n \text{ for } x \in X, n = ||X|| \tag{1}$$

The structure of a CH is more explicitly described using the syntax of category theory, although having an understanding of this abstract mathematical field is not necessary to understand their structure. Consequently, the limited discussions involving category theory are supplementary in nature. In that guise, a CH $\mathcal{H}$ is a subcategory of **Set** with objects derived from a set V where each element $v \in V$ is also a set. The objects $\text{Ob}(\mathcal{H})$ are given by the Cartesian product of each element of the power set of V, or $\text{Ob}(\mathcal{H}) := \{v_1 \times v_2 \times \ldots \times v_n \; \forall \, v_i \in P \; \forall \, P \in \mathcal{P}(V)\}$. Since $\text{Ob}(\mathcal{H})$ is a subcategory of **Set**, the morphisms of $\mathcal{H}$ are functions mapping

$\mathrm{Ob}(\mathcal{H}) \rightarrow \mathrm{Ob}(\mathcal{H})$. Composition is then given by typical function composition, $\circ$, and the identity by the identity function on a set. This defines $\mathcal{H}$ as a monoidal category.

Definition 2 *Edge: Given two functions* $\mathtt{cod}$ *and* $\mathtt{dom}$ *and two objects* a, b *(not necessarily unique) taken from the same set, let an edge* e *be defined such that* $a \in \mathtt{dom}(e)$ *and* $b \in \mathtt{cod}(e)$ *for at least one such pair* (a, b).

The functions $\mathtt{cod}$ and $\mathtt{dom}$ follow the definitions given by Mac Lane [22], with both mapping from $E \rightarrow \mathrm{Ob}(\mathcal{H})$, where E is a set of edges. In the sequel $\mathtt{dom}(e)$ is sometimes referred to as the *domain* of e, and similarly to $\mathtt{cod}(e)$ as the *codomain* of e. Similarly, the Cartesian product of each element in an edge's domain is referred to as S (the source set), and the codomain as T (the target set).

A hypergraph can be simply described as a collection of vertices connected by edges where the edges can connect any number of vertices. A CH adds to this general definition two important constraints:

(1) Each vertex in V is a set; and
(2) Each edge in E has only a single vertex in its codomain.

Note that here the definition of an edge is really that of a *directed* edge. This is more general than an undirected edge [51], and is important to the formulation of a CH. As non-directed graphs are not treated in this article, any such reference to a graph or hypergraph should be construed as referring to a *directed* graph and *directed* hypergraph.

The utility of a CH lies primarily in finding sequences between sets of vertices. Traversing a CH is similar to pathfinding in a graph, with the added complexity of multiple dependencies present at each edge whose domain has a cardinality greater than 1. Traversal methods are discussed in greater detail in Section 4.

In order for a CH to be traversable, it must be paired with a computational engine capable of identifying the mapping given by each $S \xrightarrow{e} T$. Though such an engine is not required for the CH to be valid, the utility of a CH stems largely from its traversals. As such, the computational engine is a significant part of the CH. This engine may perform lookups similar to a querying agent in a relational database, or it may be a calculator capable of computing some rule that encodes the function mapping.

3.2 Characteristics. In addition to the description given in Definition 1, a CH is endued with several characteristics that increase its functionality. These characteristics do not alter the mathematical structure given by the more general definitions. Their inclusion is motivated by the application of CHs.

Definition 3 *Ordered: There should be an injective function* $\mathtt{idx}_e : \mathtt{dom}(e) \rightarrow \mathbb{N}_{||\mathtt{dom}(e)||}$ *given for all edges* $e \in E$ *in a CH, where* $\mathbb{N}_i$ *is the set of natural numbers up to* i.

The purpose of ordering is so each vertex connected to an edge can be uniquely identified along a hyperedge during traversal of the hypergraph. The ordering of elements of S by f enables non-commutative relationships to be codified in an edge. For example, a function rule that does not commute such as $\div(A, B) \rightarrow C$ (mapping each element of $A \times B$ with the quotient of its ordered pair in C) requires its operands to be assigned in a specific order, i.e. $\div(A, B) \neq \div(B, A)$.

Borrowing again from the language of Category Theory, an ordered CH is one in which each set $\mathtt{dom}(e)$ is totally ordered, such that for any objects $a, b \in \mathtt{dom}(e)$, $a \leq b$ is given by $\mathtt{idx}_e(a) \leq \mathtt{idx}_e(b)$, with $\leq$ defined as the typical magnitudinal ordering operation for the set of natural numbers. Note that equivalence between a and b indicates a commutative relationship espoused by e between the two objects. Consequently, an edge whose mapping was determined by addition might have $\mathtt{idx}_e(a) = 1\ \forall a \in \mathtt{dom}(e)$. Hence, $\mathtt{idx}_e$ is injective, not bijective.

Definition 4 *Conditional Viability: For each edge* $e \in E$ *in a CH, there should be a function* $\mathtt{via}_e : \prod_S \rightarrow \mathbb{B}$*, where* $\mathbb{B} := \{0, 1\}$ *is the set of Booleans, and* $S := \mathtt{dom}(e)$.

The value mapped by the function $\mathtt{via}_e$ determines whether e is viable. A non-viable edge is one that is known to exist, but for which the relationship is not currently sequenceable. This prevents traversals along a non-viable edge during simulation. A viable edge is equivalent with Definition 2.

As there are many relationships in the real world that exist for only a portion of their expressable dependent factors, such as a door permitting entrance only if it is unlocked, conditional viability has significant impact on the CH's ability to represent realistic relationships. In more practical terms, a conditionally viable edge is one whose traversability can change during sequencing. A secondary, but no less important effect of this is the ability to form sequencable cycles in the CH. In order for a cycle to form a part of a path, there must be some mechanism by which the processing agent can exit a cycle based on values encountered at run-time. Such a requirement is fulfilled by conditional viability, with the values provided by the elements mapped from $\mathtt{dom}(e)$. A better understanding of the mathematical basis for conditional viability is covered in Section 4.

3.3 Universality. In addition to their formal definition, the authors aim to show that CHs can be used to represent any real system, either singular or covering multiple domains. Such a general objective can be reduced to showing the following conditions: (a) that the framework can represent any system property; and (b) that the framework can represent any interaction between the system properties.

A system property represents an identifiable phenomenon of an entity. Consider two assumptions: that every phenomenon can be represented by a set of values, and only a single one of these values may be manifest for any distinct frame of consideration. The second assumption stems from considering the system to be deterministic: where only one possible outcome may result for any set of initial inputs [12]. A system's evolution is characterized as the manifestation of different values across unique frames of consideration. The behavior of a system is the set of restrictions on the possible values a system property can manifest for a given configuration of other system properties, following the definition given previously by Willems [5]. The goal of system representation is to express these system properties and inter-property restrictions.

A CH is composed solely of these elements. By refraining from prescribing the types of properties that can be represented, a CH is able to represent any property that can be described by a set of values. Due to the finitude of information, this should include all phenomena that can be considered, satisfying the first condition of system representation.

The second condition, considering system interactions, is motivated by the notion that a system property may manifest only a single value when considered. Consequently, any interaction between system properties must be given as the prescription of a unique value. A function is the correct mathematical construct for representing these interactions. By mapping the values of a system property to a single, distinct value of another, a function enforces the causality assumption implicit in system modeling. A CH encodes all possible system interactions by using functions to represent not just relations between a pair of properties, but also combinations of multiple properties. Consequently, any function that can be mapped between the properties of a system can be expressed by a CH, confirming the second condition.

This can be demonstrated by the example of the hybrid radiator system in Fig. 3. In that figure, there are seven properties of the system relating to temperature, settings, power flow, etc., each one represented by a node in the hypergraph. The set of every possible state for the system is given by all combinations of these properties, or $\prod_V$ where V is the set of properties (following the definition of $\prod_X$ in Equation 1). The behavior of the system is the restriction of these combinations; for example, q can not be 1 if T is greater than T_{max}. These restrictions come as a direct result of the interaction of system components. As the components in a system evolve, they affect other parts, constraining the associated properties.

To represent a system, a modeling framework must capture every constraint imposed by the association of the system's parts. It can be difficult to do this in a procedural (object-oriented) framework: trying to describe how a controller changes the continuous temperature of a room with a block diagram is possible, but not without converting the discrete controller output into a continuous signal. Contrast this to a CH, where the constraints between system properties are the natural language of the framework. The result is that a CH can not only represent any kind of real, deterministic system, but also any behavior espoused by the system.

4 System Simulation

An important purpose of representing a system is the ability to perform simulations, with simulation defined here as the measurement of a system property made without having first observed that property in reality. Under this definition, nearly every decision made by a modeler requires simulation. Before a decision is made, the state of a system is unset and cannot be observed since it does not yet exist. An architect deciding on the location of a building cannot observe that location in reality as the building has not been built. Rather, the architect simulates the system to identify the optimal setting. Consequently, any system representation that does not afford simulation is nearly useless to a decision maker. One of the principle benefits of using a CH is the ability to simulate a system generally, exposing the possible ways to simulate a property anywhere in the system as long as it has been constrained by a constraint.

Simulation of a CH is focused not on translating visual diagrams into executable scripts, but on creating a mathematical structure that aligns execution with the underlying system behavior. The purpose of simulating a constraint hypergraph is to constrain a system such that system data becomes evident. If data is generated from constraints in a model, then it is artificial, and it is called simulated data. The data that are known prior to the simulation are referred to as inputs, and the data that need to be generated are referred to as outputs. This terminology should not be confused with expressing a CH in terms of inputs and outputs. A CH model is a set of constraints and relationships. The concepts of inputs and outputs apply only during a simulation when it becomes necessary to process relationships in a certain order. The directed edges in a CH describe which direction such sequencing may occur. Fig. 4 shows an example of this: the model contains multiple different dependencies from different nodes, such that it's impossible to say which node depends on which. The exception to this is during a simulation, when a single node is set to be the input, and the graph describes which nodes can become possible outputs. The act of processing a CH from one node to another is referred to here as sequencing, and is synonymous with simulation.

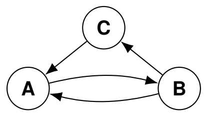


Fig. 4 A cyclic graph describing how causality can only be assigned within the context of a simulation.

The expressiveness of a CH is founded in two principles: composition and determinism. Composition is enforced in a CH by representing relationships as function. In order to be a valid mapping, a function must show how each element in one set (the domain) is associated with an element in another set (the codomain) [52]. This guarantees that if a value of each source node is known, then the target node for the edge can always be calculated by the encoded function. The solved target node can then be utilized for simulation of additional nodes. Each step in the simulation composes with the previous, allowing an agent to readily form sequences of simulation steps throughout the graph.

CHs are also deterministic, as described in Section 3.3, in that given a single input, each linked node is constrained to manifest only a single value. The two properties of composition and determinism are the foundation to forming simulatable sequences in a CH. As shown in Fig. 5, any sequence starting in the leftmost set is guaranteed to be able to arrive at the rightmost (composition), and given an element in either of the two leftmost sets, there is only one element in a set to the right corresponding to that value (determinism).

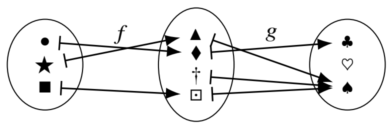


Fig. 5 Two functions mapping between sets demonstrating composition and determinism.

4.1 Sequences. In a CH, the fundamental construct of a simulation is a path. A path (or a chain) in a graph is some sequence of distinct vertices connected in a sequence of non-repeating edges [51]. The corollary for a hypergraph is a hyperpath, which connects a given set of vertices (the source) to another single vertex (the target) by a chain of hyperedges. In order for a hyperpath to be traversable, a value must be known or generated for every node in the path. Each edge describes a rule whose execution generates a new value, thereby advancing the simulating agent along the path.

The actual execution of this rule–e.g. performing a table lookup, or executing a sequence of mathematical operations–is performed by a simulation engine paired with each function. Part of the process of preparing a constraint hypergraph is connecting each function rule to a simulation engine that can process it. During simulation, the engine is passed the value of the source set along with the rule to be calculated. It then returns the output value, which is assigned to the target node.

Structurally, hyperpaths are constructed as trees. In a hyperpath tree, branching occurs along every edge whose domain has a cardinality greater than one. By making the output node the root, and placing the source nodes as the leaves, the tree shows all the functions and nodes which must be traversed to complete the simulation. A hyperpath (as a tree) is shown in Fig. 6 for a hypergraph with two source nodes S_1, and S_2 and a target node T.

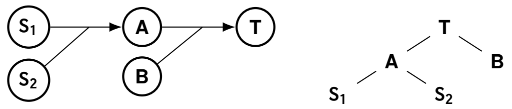


Fig. 6 A hypergraph with a hyperpath from $\{S_1, S_2\}$ to T represented as a tree.

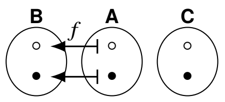
4.2 Pathfinding. In addition to sequencing a hyperpath, a means for determining which hyperpath to sequence must be provided. The philosophy of a CH is that the model contains all relevant relationships between elements of the system, which can create multiple ways to travel in between the same nodes. It is a nontrivial task to discover a valid path between nodes in a hypergraph, and even more complex to find an optimal one. To do this, the modeler needs to quantify the optimality of each edge, generally by assigning weights. Ausiello et al. described ways of traversing directed hypergraphs using a modification of Djikstra's algorithm [53,54] to identify the minimum-cost path between all nodes in the hypergraph. Though this works well for simple hypergraphs, CHs are not simple due to their inclusion of two features: cycles and conditional edges. Ausiello's work must be adapted to handle these conditions. Since this adaptation is not straight forward, it is worthwhile to motivate the provision of these graph features in a CH.

4.2.1 Conditional Viability of Edges. Consider the following example for a system of three switches with states on and off represented respectively as ∘ and •. There are eight possible states of the system as construed.

$$\begin{array}{cccc} (\circ\circ\circ) & (\circ\bullet\circ) & (\bullet\circ\circ) & (\bullet\bullet\circ) \\ (\circ\circ\bullet) & (\circ\bullet\bullet) & (\bullet\circ\bullet) & (\bullet\bullet\bullet) \end{array}$$

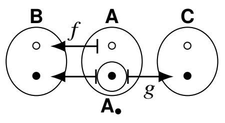
The most intuitive way to represent these state values is by associating a variable with each switch, labeled A, B, C. Each variable will be represented by a node in a CH, with $A, B, C := \{\circ, \bullet\}$.

Providing a constraint decreases the degrees of freedom for the system. For example, a function f constraining A and B to be equivalent reduces the degrees of freedom from three to two, and the number of possible system states from eight to four, as shown below.



$$\begin{array}{cc} (\circ\circ\circ) & (\bullet\bullet\circ) \\ (\circ\circ\bullet) & (\bullet\bullet\bullet) \end{array}$$

Because the way system states are categorized is arbitrary, oftentimes a system may exhibit behavior that does not perfectly correlate to the prescribed variables. For example, if switch C is always off if switch A is off, then the system behavior cannot be correctly expressed by a function mapping $A \rightarrow C$, as C is not constrained by the ∘ state of A. Instead, this behavior should be represented by a function $g : A_\bullet \rightarrow C$, where $A_\bullet$ only represents the • state of A. Accordingly, $A_\bullet$ is a subset of A, as shown below.



$$\begin{array}{cc} (\circ\circ\circ) & \\ (\circ\circ\bullet) & (\bullet\bullet\bullet) \end{array}$$

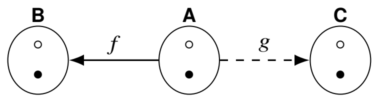
Including $A_\bullet$ changes the paradigm of the CH by introducing nested nodes. The main value of nesting is that the intuitive sets of A, B and C can be maintained while also describing relationships between individual members of A. Duplication of entities is also avoided since every function that maps from A implicitly maps from $A_\bullet$ also, without requiring f to be redefined. Nested relationships represent constraints that only hold for a limited subset of the domain's values. This is such a common occurrence in system modeling that conditional edges were identified by Peak et al. as a fundamental requisite for system simulation [36]. An example is Hooke's law relating the force of a spring to its deflection, which is valid only if the deflection is in the spring's linear regime.

The complexity of conditional edges comes as a result of violating composition. Let X' be a subset of a set X. Any function f that points from X also points from every value in X'. However, the inverse of this is not true: if X' is the domain of a function g, then a simulation sequence arriving at X is not guaranteed to be ably to traverse g unless the assigned value for X happens to also be in X'. The result of this is that the simulation path becomes conditional on the specific value solved for X at runtime. This greatly increases the computational expense of simulation, as a path through the CH must be rediscovered for each unique set of inputs, in contrast with the general paths found by Djikstra's algorithm that are optimal for any node values.

In order to enable sequencing, the use of a conditional viability function $\mathtt{via}_e$ for any edge e was provided in Definition 4. This function is used to determine whether a value x is part of the domain of e, such that:

$$\mathtt{via}_e(x) = 1 \iff x \in X \land X \in \mathtt{dom}(e) \tag{2}$$

An edge is described as being conditionally viable if $\mathtt{via}_e$ is not 1 for all values of $\mathtt{dom}(e)$. The set of values for which $\mathtt{via}_e$ is 1 is referred to as the *viable set of e*. Conditional viability can be used for path switching, where the node a simulation sequence next reaches changes based on the values of the latest inputs. In this case, there are two or more edges with disjoint viable sets, so that only one edge is valid for any given value. In this paper, conditionally viable edges are typically shown as dashed, as in the figure below (which expresses the switching example described previously).



4.2.2 Cycles. One of the most vexing issues of a graph is dealing with cycles: a path that begins and ends on the same node [55]. In a CH, a cycle in a simulation sequence would seemingly indicate that the value of a node is in someway constrained by itself, a violation of causality not found in the real world.

Despite this, cycles are still introduced as a matter of convenience to the modeler in performing simulations. This is due to many system variables being iterations of previous system states, especially as a system evolves through time. In a procedural model, such as those espoused by Wymore, a system has several independent states which repeat at each instance of time [56]. A CH, in contrast, has no sense of state. Each instance of a system variable is considered unique and independent, and must have some sequence of hyperedges connecting it to the input values to be successfully solved for. For instance, a hypergraph representing a body moving at a constant velocity along a single dimension is shown at the top of Fig. 7, where the position at each time step i is given by x_i.

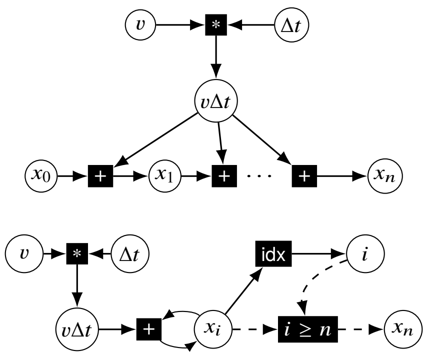


Fig. 7 A hypergraph representing the position (x_i) of a moving body at multiple points in time as an extended path (top) and as a cycle (bottom).

Since a modeler may wish to simulate x at an arbitrary time step, this hypergraph should arguably be extended to infinity. This impossible expression can be enabled through the use of cycles, as shown at the bottom of Fig. 7. In the cyclical hypergraph, the addition of the index i as well as a conditional edge $i \geq n$ provides a mechanism for exiting the cycle after n iterations. Note the essential functionality of the conditional edge, without which a pathfinder would not be able to identify a valid path that exits x_i for only a subset of the node's values.

4.2.3 Pathfinding Process. Having motivated the inclusion of conditional edges and cycles, the methods of pathfinding can now be properly described. Pathfinding is the act of tracing a hyperpath from a set of source nodes S to a target node T in the CH. Assuming that conditionally viable edges are present in the hypergraph, pathfinding must be performed for each unique set of input values. Consequently, the first step in the pathfinding sequence is to pare down the hypergraph to a subhypergraph consisting only of the nodes and edges that are possible candidates for a path. Given S and T, a solver can perform a basic search for a subhypergraph $\mathcal{H}'$, where each node in $\mathcal{H}'$ is reachable from at least one $s \in S$ and from which T is likewise reachable. A node b is reachable from (or connected to) another node a if there is a hyperpath from a to b [55].

Once $\mathcal{H}'$ has been prepared, a solver should be run using standard algorithms for pathfinding (such as A*). At each node, the solver calculates a step along a hyperpath through $\mathcal{H}'$. In doing so, the solver creates a hyper-dimensional search tree. Each node in the search tree corresponds to a viable edge, as depicted in Fig. 6. Additional hyper dimensions indicate parallel paths established whenever a conditional edge is encountered. This suggests that the optimal path may run through the conditional edge, but because that path cannot be guaranteed to exist during runtime, other paths must also be explored. As the solver searches $\mathcal{H}'$, these parallel path are pruned from the search tree if either of the following conditions are met:

(1) There are no unprocessed edges remaining from the root node (and the root node is not T).
(2) The total cost of the branch is less optimal than another branch comprised of simple edges.

Pruning in such a way is a description of backtracking, the most common method of solving a constraint problem [39]. The result of such exploration is a tree of all possible candidates for an optimal path from S to T. If the lowest cost path is made up of simple edges then the tree will be a normal search tree optimal for any input values. Otherwise, during simulation, the solver will proceed through the tree. At any branches onto parallel paths, the solver should step down the lowest-cost branch until the solver either reaches T or the path becomes non-viable. In the later case, the solver resets to the latest solved node and traverses the next-most optimal path.

Although repeated pathfinding can become computationally expensive, it provides the unexpected benefit of gracefully handling discontinuities. Rather than aborting the simulation, solvers encountering a discontinuity (such as an ill-formed mapping or missing value) are able to switch to the next viable path as long as one is available.

4.3 Weighting Schemes and Model Selection. A heretofore undiscussed aspect of CHs is the application of weights to their edges. While weights are a common practice in graph theory, it will be useful to consider their interpretation with respect to modeling systems. Weights are primarily used to depict a cost of traversing an edge; in the classic traveling salesman problem, weights depict distances between cities on an imaginary map. Search algorithms require weights to discriminate between parallel paths, preferring the path with the lowest summed weight. In the case of a CH, the cost of traversing an edge is the cost of executing a constraint on the system model. This cost could be interpreted a number of ways, such as the computational cost of calculation, the distance to the desired target node, the cost of excluding some alternate constraint, or even the uncertainty associated with the modeling constraint.

The utility of interpreting edge weights as modeling uncertainty is not in the quantification; assigning uncertainty is still as arduous in a CH as any other framework. Instead, CHs provide an advantage in composing uncertainty for different simulations. A CH breaks down every relationship in a system model into a single, traceable function. For a systems engineer, each of these functions can be thought of as assumption. By listing each edge in a path, a modeler can systematically consider each assumption made in a simulation. Because each function is independent of all others, uncertainty can be assigned without having to consider side effects or duplicated calculations. And because the total uncertainty of any arbitrary path can be trivially calculated, a pathfinding algorithm can search for a simulation path that minimizes uncertainty.

When the edge weights are interpreted as computational costs, then the weighting scheme can be used to determine the least expensive simulation to compute. Weighting schemes are not exclusive either, they can be combined to compare simulation paths by different metrics. The resulting paths can then be optimized via a multi-optimization method to minimize multiple objectives.

5 Limitations

Though constraint hypergraphs have been used to great effect, they by no means represent a silver bullet to modeling challenges. It has already been mentioned that flat, complex system models can be visually overwhelming. Object-oriented frameworks, which provide methods for abstraction and encapsulation, are often better for decomposing a system for a human modeler. It is for this reason that this article has not focused on establishing hypergraph diagrams. This section aims to set forth some of their other known limitations.

Related to the challenges of visualization, CHs are not optimal for model development. Because of their generality, any relationship can be formulated in a CH, including impossible relationships. The specificity of domain-specific frameworks discourage modelers from creating invalid systems, such as connecting gravity to a battery terminal. There is no such restriction imposed by CHs. From this the conclusion is drawn that CHs are used most effectively to unify system models that have already been developed in more specialized frameworks.

Perhaps the greatest limitation of CHs is that of syntactic interoperability. Although the semantics of a system are perfectly captured by a CH, this is dependent upon syntactic agreement between models. For instance, it may be impossible to say whether the label of "Speed" for a node in one model refers to the same data as the "Velocity" label of another. Though the use of ontologies can help prevent naming conflicts, the lack of tools for reconciling nodal identities greatly inconveniences the adoption of CHs. The authors are intrigued by the possibility of providing syntactic interoperability through reasoners based on graph similarity metrics.

Another major barrier to convenient adoption is the heavy processing time of constraint hypergraphs. Because optimal simulation paths cannot be autonomously determined a priori, path searching often must occur during each simulation run. Though constraint programming has developed great tools and methods for searching, this processing overhead can limit CHs use in real-time environments. This is especially true considering that solving a constraint problem is NP-hard [57], and CH networks can become incredibly complex. There are some instances that paths can be pre-determined, though in such cases simulation success often cannot be guaranteed unless the hypergraph contains no conditionally viable edges.

Finally, constraint hypergraphs are very good at exposing system behavior; they are less good at hiding it. There are many instances in which a modeler may which to obscure the sensitive behavior of a subsystem in a shared model, such as with proprietary technology. While it is certainly possible to form black boxes in a CH (simply by aggregating nodes into a single node and reconnecting graph edges), such actions prevent subsystem properties from being connected with other system elements. This greatly reduces the efficacy of a CH. Encapsulation and abstraction are both primary features of object-oriented systems, but they reduce the expressiveness of the more functional CHs. If such features are needed, interface-based frameworks may need to be used; privacy-motivated black boxing is one of the many use cases of FMIs

[58,59].

6 Case Study

A CH for an elevator lift system is provided to partially validate the constraint hypergraph structure. The unified model is an aggregation of a discrete-event simulation (DES) and continuous state space model (built around a PID controller). Descriptions for each node are provided in Table 3 in the Appendix. At the risk of repetition, the reader is reminded that CHs are not generally a good choice for visualizing system models due to their high complexity and abundance of lines. However, for the purpose of communicating their use, a diagram for the CH has been provided in Fig. 8, with the model scoped to be as simple as possible without sacrificing functionality. The diagrammatic scheme follows that of Fig. 3, with the added stylizations of zigzag hashing for nodes that are shown multiple times in the figure (for clarity) and double arrows (↠) for edges that increase the iteration of a node in a cycle. There should be one such edge for every cycle in the graph; Fig. 8 has three cycles with corresponding iterative edges. The weight of every edge is set to one, resulting in the solver preferring the path with the minimum number of steps during simulation.

There are three primary subsystems represented in the CH: the dynamic system, the PID controller, and the DES of the passenger actions. The properties and edges for these subsystems are roughly gathered in the right, top left, and bottom left of the figure respectively. These classifiers for the various subsystems are only useful for a human modeler as the CH focuses solely on the holistic system. Consequently, it is not clear in Fig. 8 where one subsystem ends and another begins–at least not as clear as it would be in an object-oriented framework. The mock elevator consists of a carriage moving between three floors referenced by integers 0, 1, and 2. The forces acting on the carriage include the empty weight as well as the summed weight of each passenger (with the assumption that each passenger weighs the same). The driving force of the elevator is given by a PID controller using a first-order Euler integrator with a fixed step-size.

The DES subsystem is formalized for four passengers referred to as `A` through `D`. There are three properties associated with a passenger: the floor they start on (`startX`), their desired destination floor (`goalX`), and whether they are on or off the elevator carriage (`onX`). Recall that the CH does not include a "passenger" object, only these properties. Also note that it is required to explicitly model the properties for each passenger, as each new passenger extends the possible states (or degrees of freedom) of a system. When executed, each node should be duplicated as a new passenger is simulated. To avoid complicated, redundant figures (beyond the graph already depicted), the nodes for only a single passenger are expressed, contained by the dotted box in the lower left corner of Fig. 8. As shown by that sequence, an ordered tuple (`onX, startX, goalX`) is made for each passenger. The `Set Boarding` function counts the tuples for which `onX` is `False` and `Start` is `Current Floor`. In this case there are no passengers meeting the criteria, so `Num Boarding` is zero. `Set Exiting` does the same except the conditions are `onX` being `True` and `Goal` equaling `Current Floor`. `Occupancy` is then derived as the previous value of `Occupancy` minus `Num Exiting` and added to `Num Boarding`. The `Set Pass Status` edge shown in Fig. 8 is given by the following conditional function:

$$\begin{gathered}\text{False if } \texttt{Current Floor} = \texttt{StartX} \\ \text{True if } \texttt{Current Floor} = \texttt{GoalX} \\ \texttt{onX} \text{ else}\end{gathered} \tag{3}$$

The number of properties comprising the system state is given by the number of nodes in the CH, which in this case is 45. There are several nodes for which constraints are not provided, these must be treated as inputs for a valid simulation, such as physical constants like gravitational acceleration. Initial properties for each passenger have also been given in Table 2. In the example the elevator initially starts on floor 0 and moves incrementally to floor 2 (only ascending for simplicity).

Table 2 Initial values for properties associated with elevator passengers.

Person ID	Start Floor	Goal Floor	On Elevator?
A	1	2	False
B	1	2	False
C	0	2	True
D	2	0	False

Having visualized the CH, the next goal of this case study is to demonstrate a full system simulation. Any node in the hypergraph can be chosen to be simulated, but selecting `Occupancy` allows a more interesting example. The goal of a simulation is to predict the value of `Occupancy` at every landing to which the elevator arrives. In practice, this simulation would be conducted using a computational tool, however, this study is a demonstration of theory. The enactment of this simulation is consequently performed by a theoretical agent capable of processing a CH, which will be referred to as CH AGENT. The inputs to CH AGENT are the graph in Fig. 8, as well as a list of input values to several nodes, as tabulated in Table 3.

Assuming the inputs have been seeded correctly, a pathfinding algorithm would attempt to find a valid path from a subset of the input nodes to `Occupancy`. Pathfinding is conducted with some searching strategy such as a depth-first or breadth-first search, with the search starting from nodes with known values (the input set). If all nodes in the source set of an edge are known, then the edge can be traversed. Traversing an edge results in CH AGENT solving for the value of the target node by performing the function calculation represented by the edge. This new node is then added to the list of known values. The process repeats until CH AGENT is able to solve for the goal node, at which point the simulation terminates.

Because `Occupancy` is given as a known node, the initial simulation is trivial. However, `Occupancy` is included inside a cycle, meaning that additional iterations of the node can be solved for if all other values in the cycle are found. The cycle for `Occupancy` involves a hyperedge requiring the number of passengers boarding and exiting the carriage to be calculated, consequently, CH AGENT must solve for these nodes before it can find the next value of `Occupancy`, written here as $\texttt{Occupancy}_2$. The process for doing so is given by tracing the starting position of each passenger to determine whether they are on the carriage, boarding, or exiting while the carriage is at its initial floor, according to Equation 3. The result is a chain of equations that ultimately transforms `X Start` and `X Goal` into $\texttt{Occupancy}_2$, which when solved returns a value of two.

Solving for $\texttt{Occupancy}_3$ requires a much longer process. The conditional edge relating Δ`Passengers` to `Occupancy` requires that the iteration of the source of `Occupancy` be only one iteration greater than the iteration levels of `Num Boarding` and `Num Exiting`, but the path found by CH AGENT includes only $\texttt{Num Boarding}_1$ and $\texttt{Num Exiting}_1$. Because no iterative edge (indicated by ↠) was encountered in the simulation path found previously by CH AGENT, the path cannot be reused to solve for the node's next value. Tracing the cycle back in Fig. 8, the iterative edge is a hyperedge relating $\texttt{Floor Gap}_1$, $\texttt{Ht Tolerance}_1$, and $\texttt{Height}_1$ to $\texttt{Elev Current Floor}_2$. In order to increment the state of each node in the DES simulation just traversed, CH AGENT must find a path that connects through this edge, so that each source node will also be in their second iteration upon solving for $\texttt{Occupancy}_3$.

To necessary path takes CH AGENT through the entirety of the hypergraph, starting from $\texttt{Occupancy}_2$ and ending on $\texttt{Height}_2$. After the iterative edge is traversed, the original path is retraced but at a higher iteration level, resulting in solving $\texttt{Occupancy}_3$. In

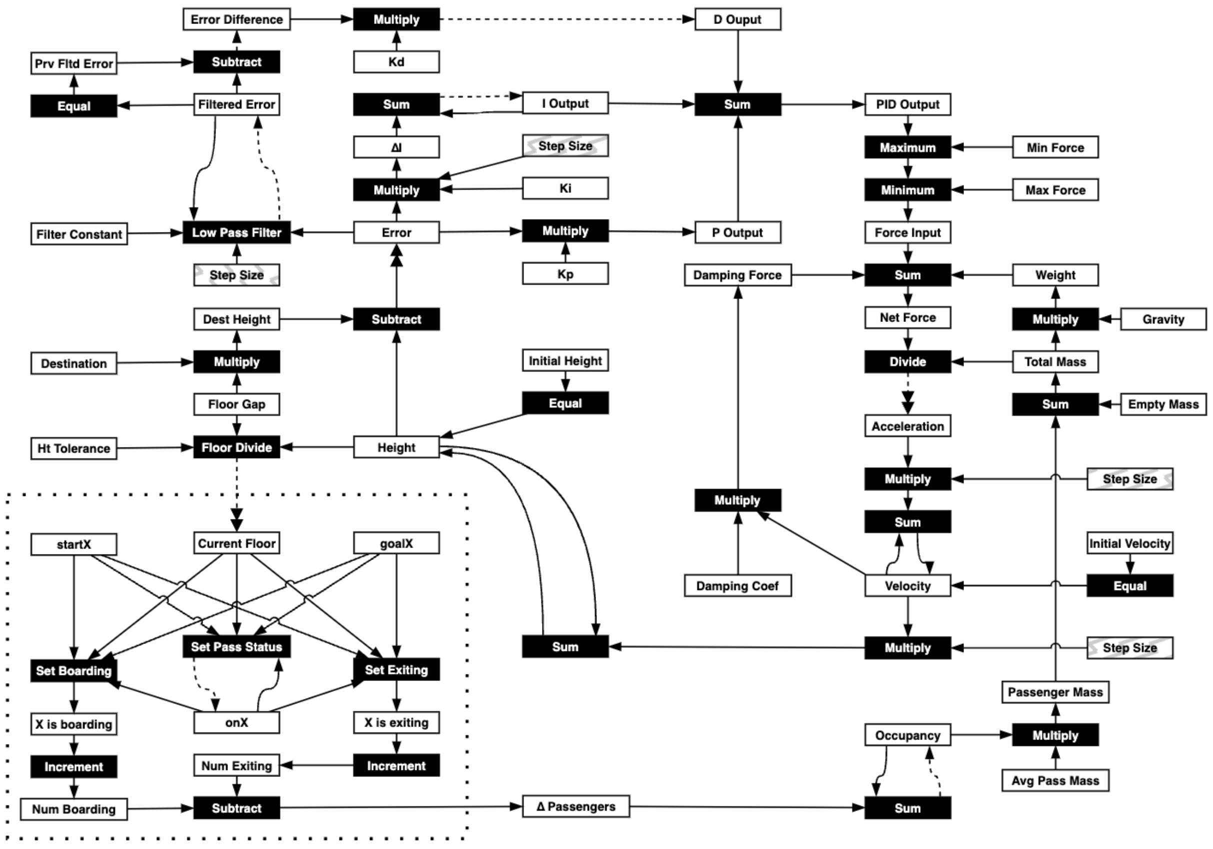


Fig. 8 Constraint hypergraph model for a hybrid elevator lift system.

this case, CH AGENT traverses 42 edges to reveal that the value of $\texttt{Occupancy}_3$ is unchanged from the previous iteration. Along the way, values are found for $\texttt{Height}_2$, $\texttt{PID Output}_1$, and other nodes that are necessary for the general simulation.

By structuring the hybrid elevator system as a constraint hypergraph, CH AGENT demonstrates the universality of simulation, moving easily between the various subsystems without concern for ports and type specifications. It should be emphasized that this is a theoretical demonstration; computational complexity, run time, and other computing metrics are dependent upon the practical instantiation of CH AGENT and the specific search strategies employed by the encoded algorithms. For validation, a Python-based implementation of CH AGENT employing a breadth-first search strategy [60] was run to calculate the `Height` of the elevator over 100 iterations, returning the results shown in Fig. 9. As a true instantiation of CH AGENT, no additional programming is employed apart from generic plotting software, conducting the simulation without any for or while loops, go-to statements, or conditional logic–all behavior of the system is encapsulated in the CH and executed by CH AGENT. While these results do not validate the models, which are overly simplified for demonstration purposes, they do indicate the ability of CH AGENT to integrate complex systems without relying on manually ported connections.

7 Future Work

A robust, unified system model has many possible applications, some of which the authors hope to explore in future work; these include decision modeling and digital twins. Decision modeling applications stem from the fact that the decisions made by engineers and other agents often must be made before relevant information about a system is known; this is especially true early in the design stage. Constraint hypergraphs make it easier for the information

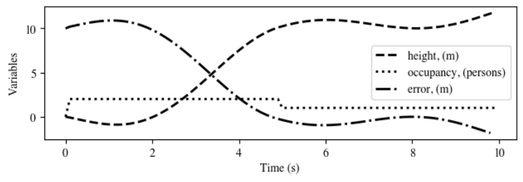


Fig. 9 Results of a simulation for three variables covering 100 cycles.

that is known about a system to be understood and analyzed, but the process of quantifying uncertainty and validating simulation predictions remains a topic for further investigation.

Digital twins, defined as the virtual representation of some real system, are a confluence of data streams and models. Their popularity has driven use cases in a variety of systems, such as global environments [61] and national infrastructure [62,63]. This rich variety of system domains requires the combination of disparate models as well as the processing of heterogeneous data streams. As digital twins grow in fidelity these multifarious facets only increase. Constraint hypergraphs may promote the interoperability of digital twins [64] on both of these fronts: by unifying siloed system models and also by enabling sequencing of data across the holistic system. The author's envision a constraint hypergraph expanding the role currently played by knowledge graphs from information banks [65] to system orchestrators: capturing and distributing information connected to system behavior with a multiplicity of connected agents.

8 Conclusion

The principle factors that contribute to a simulatable system model are composability and determinism. These two principles are embedded uniquely in various system models, but are robustly defined in constraint hypergraphs. Constraint hypergraphs, in their declarative description of property relationships, embody the structure of a system. The claim that CHs are suitable for general inter-system reconciliation was supported by showing that CHs support composability and determinism through functional composition, and further describing how every system behavior can be both represented and sequenced within the CH structure.

In the attempt to make this framework more practical, the mathematical structures have been related in the language of systems theory, showing how CHs communicate information and how the relevant notions of objects, states, and behavior are described. CHs are not only useful for semantic representation, they are also simulatable, and can be used to derive unknown system properties or solution spaces. It was shown that this depended on both the provision of functions and the representation of nodes as subsets of other nodes. A mechanism was then built out for building executable sequences and even cycles using edges with conditional viability.

In addition to establishing what CHs are, it was established how they can be used, including how weightings can represent uncertainty or computational costs, as well as applications to autonomous decision-making. These use cases were contrasted with some known limitations of CHs. Finally, these principles were demonstrated with an example of an elevator lifting system, with multiple system domains unified by a single CH. The authors hope to build upon this work through additional refinement of the CH framework, as well as the provision of integrable tools that can assist with CH modeling in established engineering applications.

Acknowledgements

A topic as general as systems modeling requires perspectives from multiple domains. As the authors are not experts in every domain, they would like to thank Dr. Wayne Goddard, Dr. Matthew Macauley, Dr. Satchit Ramnath, Edward Louis, Evan Taylor, and Nikhil Raj for their invaluable discussions regarding these topics and for expanding our perspectives.

References

[1] Boulding, K., Ap 1956, "General Systems Theory: The Skeleton of Science," Management Science, **2**(3), pp. 197–208.

[2] Friedenthal, S., Moore, A., and Steiner, R., 2015, *A Practical Guide to SysML : The Systems Modeling Language*, 3rd ed., Elsevier, Waltham.

[3] Modelica Association, 2000, "Modelica - A Unified Object-Oriented Language for Physical Systems Modeling," accessed 2024-07-10, https://modelica.org/documents/ModelicaTutorial14.pdf

[4] Larsson, J., 2003, "Interoperability in Modeling and Simulation," Ph.D. thesis, Linköpings Universitet, Linköping.

[5] Willems, J. C., 2007, "The Behavioral Approach to Open and Interconnected Systems," IEEE Control Systems Magazine, **27**(6), pp. 46–99.

[6] Hudak, P., 1989, "Conception, Evolution, and Application of Functional Programming Languages," ACM Comput. Surv., **21**(3), pp. 359–411.

[7] Rossi, F., van Beek, P., and Walsh, T., 2008, "Constraint Programming," *Foundations of Artificial Intelligence*, F. van Harmelen, V. Lifschitz, and B. Porter, eds., Vol. 3 of Handbook of Knowledge Representation, Elsevier, pp. 181–211.

[8] Kschischang, F., Frey, B., and Loeliger, H.-A., 2001, "Factor Graphs and the Sum-Product Algorithm," IEEE Trans. Inform. Theory, **47**(2), pp. 498–519.

[9] Loeliger, H.-A., 2004, "An Introduction to Factor Graphs," IEEE Signal Processing Magazine, **21**(1), pp. 28–41.

[10] MacLennan, B. J., 1990, *Functional Programming: Practice and Theory*, Addison-Wesley, Reading, Mass.

[11] International Organization for Standardization, 2023, "Systems and Software Engineering - System Life Cycle Processes," accessed 2024-09-21, https://www.iso.org/standard/81702.html

[12] Lee, E. A., 2021, "Determinism," ACM Trans. Embed. Comput. Syst., **20**(5), pp. 38:1–38:34.

[13] Willems, J. C., 1989, "Models for Dynamics," *Dynamics Reported*, U. Kirchgraber and H. O. Walther, eds., Vol. 2, John Wiley & Sons Ltd and B.G. Teubner, pp. 171–266.

[14] Hauser, R. M. and Warren, J. R., 1997, "Socioeconomic Indexes for Occupations: A Review, Update, and Critique," Sociological Methodology, **27**(1), pp. 177–298.

[15] Lotka, A. J., 1920, "Analytical Note on Certain Rhythmic Relations in Organic Systems," Proc Natl Acad Sci U S A, **6**(7), pp. 410–415.

[16] Ouliaris, S., "Economic Models: Simulations of Reality," accessed 2024-09-25, https://www.imf.org/external/pubs/ft/fandd/basics/models.htm

[17] Fong, B. and Spivak, D. I., 2018, "Seven Sketches in Compositionality: An Invitation to Applied Category Theory," doi: 10.48550/arXiv.1803.05316, 1803.05316

[18] Fong, B., 2016, "The Algebra of Open and Interconnected Systems," Ph.D. thesis, arXiv, Oxford University, doi: 10.48550/arXiv.1609.05382, 1609.05382

[19] Baez, J. C. and Pollard, B. S., 2017, "A Compositional Framework for Reaction Networks," Rev. Math. Phys., **29**(09), p. 1750028.

[20] Patterson, E., Spivak, D. I., and Vagner, D., 2021, "Wiring Diagrams as Normal Forms for Computing in Symmetric Monoidal Categories," Electron. Proc. Theor. Comput. Sci., **333**, pp. 49–64.

[21] Patterson, E., Baas, A., Hosgood, T., and Fairbanks, J., 2022, "A Diagrammatic View of Differential Equations in Physics," MINE, **5**(2), pp. 1–59.

[22] Mac Lane, S., 1971, *Categories for the Working Mathematician*, No. 5 in Graduate Texts in Mathematics, Springer-Verlag, New York.

[23] Palm, W. J., 2014, "Block Diagrams, State-Variable Models, and Simulation Methods," *System Dynamics*, Third edition ed., McGraw-Hill Science, New York, NY, pp. 250–318.

[24] Borutzky, W., ed., 2011, *Bond Graph Modelling of Engineering Systems: Theory, Applications and Software Support*, Springer New York, New York, NY.

[25] Paytner, H. M., 2000, "The Gestation and Birth of Bond Graphs," accessed 2024-06-17, https://sites.utexas.edu/longoria/files/2020/10/Birth_of_-Bond_Graphs.pdf

[26] McPhee, J. J., 1996, "On the Use of Linear Graph Theory in Multibody System Dynamics," Nonlinear Dyn, **9**(1), pp. 73–90.

[27] Trent, H. M., 1955, "Isomorphisms between Oriented Linear Graphs and Lumped Physical Systems," The Journal of the Acoustical Society of America, **27**(3), pp. 500–527.

[28] Fisher, I., 1896, "What Is Capital?" The Economic Journal, **6**(24), pp. 509–534.

[29] Baez, J., Li, X., Libkind, S., Osgood, N. D., and Patterson, E., 2023, "Compositional Modeling with Stock and Flow Diagrams," Electron. Proc. Theor. Comput. Sci., **380**, pp. 77–96.

[30] Chen, P. P.-S., 1976, "The Entity-Relationship Model—toward a Unified View of Data," ACM Trans. Database Syst., **1**(1), pp. 9–36.

[31] Gilbreth, F. B. and Gilbreth, L. M., 1921, "Process Charts," *Annual Meeting of The American Society of Mechanical Engineers*, American Society of Mechanical Engineers, New York, 5 Dec 1921, accessed 2024-09-25, https://web.archive.org/web/20150509222833/https://engineering.purdue.edu/IE/GilbrethLibrary/gilbrethproject/processcharts.pdf

[32] Ibe, O. C., 2013, *Markov Processes for Stochastic Modeling*, 2nd ed., Elsevier Insights, Elsevier, London.

[33] Daly, R., Shen, Q., and Aitken, S., 2011, "Learning Bayesian Networks: Approaches and Issues," The Knowledge Engineering Review, **26**(2), pp. 99–157.

[34] Pearl, J., 2009, *Causality*, Cambridge University Press.

[35] Paredis, C., Diaz-Calderon, A., Sinha, R., and Khosla, P., 2001, "Composable Models for Simulation-Based Design," EWC, **17**(2), pp. 112–128.

[36] Peak, R., Paredis, C., Tamburini, D., and Waterbury, S., 2005, "The Composable Object (COB) Knowledge Representation: Enabling Advanced Collaborative Engineering Environments (CEEs)," National Aeronautics and Space Administration, accessed 2024-09-19, https://www.eislab.gatech.edu/projects/nasa-ngcobs/COB_Requirements_v1.0.pdf

[37] 2007, "OMG Systems Modeling Language (OMG SysML)," accessed 2024-09-21, https://www.omg.org/spec/SysML/1.0/PDF

[38] Friedman, G. J. and Leondes, C. T., 1969, "Constraint Theory, Part I: Fundamentals," IEEE Transactions on Systems Science and Cybernetics, **5**(1), pp. 48–56.

[39] Dechter, R., 1992, "Constraint Networks," *Encyclopedia of Artificial Intelligence*, 2nd ed., S. C. Shapiro, ed., Vol. 1, John Wiley & Sons Inc, New York, pp. 276–285.

[40] Lecoutre, C., 2013, *Constraint Networks: Targeting Simplicity for Techniques and Algorithms*, John Wiley & Sons.

[41] Sinha, R., Paredis, C. J. J., Liang, V.-C., and Khosla, P. K., 2001, "Modeling and Simulation Methods for Design of Engineering Systems," J. Comput. Inf. Sci. Eng, **1**(1), pp. 84–91.

[42] Andersson, C., 2016, "Methods and Tools for Co-Simulation of Dynamic Systems with the Functional Mock-up Interface," Doctoral Dissertation in Mathematical Sciences, Lund University, Lund, Sweden, accessed 2024-03-12, https://www.maths.lth.se/na/staff/chria/phdthesis.pdf

[43] de Vries, T. J. A., Weustink, P. B. T., and Cremer, J. A., 1997, "Improving Dynamic System Model Building Through Constraints," *Computer Aided Conceptual Design*, Lancaster, 1997, pp. 83–95, accessed 2024-09-25, https://ris.utwente.nl/ws/portalfiles/portal/169842019/DeVries1997improving.pdf

[44] Feldkamp, F., Heinrich, M., and Meyer-Gramann, K. D., 1998, "SyDeR—System Design for Reusability," AIEDAM, **12**(4), pp. 373–382.

[45] Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Clauß, C., Elmqvist, H., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J.-v., and Wolf, S., 2011, "The Functional Mockup Interface for Tool Independent Exchange of Simulation Models," *Proceedings 8th Modelica Conference*, Dresden, March 20 2011, accessed 2022-01-13, http://www.functional-mockup-interface.org

[46] Broman, D., Greenberg, L., Lee, E. A., Masin, M., Tripakis, S., and Wetter, M., 2015, "Requirements for Hybrid Cosimulation Standards," *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, Association for Computing Machinery, New York, NY, USA, April 14, 2015, pp. 179–188, doi: 10.1145/2728606.2728629.

[47] Wolny, S., Mazak, A., Carpella, C., Geist, V., and Wimmer, M., 2020, "Thirteen Years of SysML: A Systematic Mapping Study," Softw Syst Model, **19**(1), pp. 111–169.

[48] Harel, D. and Pnueli, A., 1985, "Reactive Systems," *Logics and Models of Concurrent Systems*, K. Apt, ed., NATO ASI Series, Vol. 13, Springer, Berlin, Heidelberg, Jan 1985, accessed 2024-09-16, https://link.springer.com/chapter/10.1007/978-3-642-82453-1_17

[49] Friedman, G. J. and Phan, P., 2017, *Constraint Theory*, Vol. 23 of IFSR International Series on Systems Science and Engineering, Springer International Publishing, Cham.

[50] Gomes, C., Thule, C., Broman, D., Larsen, P. G., and Vangheluwe, H., 2018, "Co-Simulation: A Survey," ACM Comput. Surv., **51**(3), pp. 49:1–49:33.

[51] Berge, C., 1973, *Graphs and hypergraphs*, Vol. 6 of North-Holland mathematical library, v. 6, North-Holland Pub. Co.; American Elsevier Pub. Co, Amsterdam, New York.

[52] Herstein, I. N., 1964, *Topics in Algebra*, 1st ed., Blaisdell Publishing Company, Waltham, MA.

[53] Ausiello, G., Giaccio, R., Italiano, G., and Nanni, U., 1992, "Optimal Traversal of Directed Hypergraphs," International Computer Science Institute, Berkeley, CA, ICSI Technical Report ICSI TR-92-073.

[54] Ausiello, G. and Laura, L., 2017, "Directed Hypergraphs: Introduction and Fundamental Algorithms—A Survey," Theoretical Computer Science, **658**, pp. 293–306.

[55] Diestel, R., 2017, *Graph Theory*, Vol. 173 of Graduate Texts in Mathematics, Springer, Berlin, Heidelberg.

[56] Wymore, A. W., 1993, *Model-Based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*, Systems Engineering Series, CRC Press, Boca Raton, Fla.

[57] Rossi, F., van Beek, P., and Walsh, T., eds., 2007, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, Elsevier, Amsterdam Heidelberg.

[58] Gomes, C., Meyers, B., Denil, J., Thule, C., Lausdahl, K., Vangheluwe, H., and De Meulenaere, P., 2019, "Semantic Adaptation for FMI Co-Simulation with Hierarchical Simulators," SIMULATION, **95**(3), pp. 241–269.

[59] Wolf, C., Schleipen, M., and Frey, G., 2023, "Secure Exchange of Black-Box Simulation Models Using FMI in the Industrial Context," Modelica Conferences, pp. 487–496.

[60] Morris, J., 2024, "ConstraintHg," https://github.com/jmorris335/ConstraintHg

[61] Moigne, J. L. and Smith, B., 2022, "Advanced Information Systems Technology (AIST) Earth Systems Digital Twin (ESDT) Workshop Report," NASA, accessed 2024-05-21, https://esto.nasa.gov/files/ESDT_Workshop_Report.pdf

[62] Walters, A., 2019, "National Digital Twin Programme," accessed 2023-04-12, https://www.cdbb.cam.ac.uk/what-we-did/national-digital-twin-programme

[63] Walker, A., 2023, "Singapore's Digital Twin – from Science Fiction to Hi-Tech Reality," accessed 2024-05-22, https://infra.global/singapores-digital-twin-from-science-fiction-to-hi-tech-reality/

[64] Budiardjo, A. and Migliori, D., 2021, "Digital Twin System Interoperability Framework," Digital Twin Consortium, accessed 2021-12-07, https://www.digitaltwinconsortium.org/wp-content/uploads/sites/3/2022/06/Digital-Twin-System-Interoperability-Framework-12072021.pdf

[65] Akroyd, J., Mosbach, S., Bhave, A., and Kraft, M., 2020, "National Digital Twin of the UK – a Knowledge-Graph Approach," accessed 2024-05-23, https://como.ceb.cam.ac.uk/media/preprints/c4e-preprint-264.pdf

Appendix A: Detailed Information for Case Study

Table 3 describes each node (or variable) shown in Fig. 8. Input values, useful for simulation, are provided for nodes if applicable.

Table 3 Descriptions of nodes of the CH shown in Fig. 8.

Node Name	Description	Subsystem	Input Value	Units
`Δ Passengers`	Change in carriage occupancy	DES		Persons
`Elev Current Floor`	Current floor of carriage	DES	0	Floor
`Num Boarding`	Number of passengers boarding carriage	DES		Persons
`Num Exiting`	Number of passengers exiting carriage	DES		Persons
`Occupancy`	Number of passengers in carriage	DES	0	Persons
`X Goal`	Goal floor for passenger X	DES		Floor
`X is Boarding`	True if passenger X is currently boarding carriage	DES		Boolean
`X is Exiting`	True if passenger X is currently exiting carriage	DES		Boolean
`X is Riding`	True if passenger X is currently on the carriage	DES		Floor
`X Start`	Starting floor for passenger X	DES		Floor
`Acceleration`	Current acceleration of carriage	Dynamics		m/s^2
`Avg Pass Mass`	Average mass of a passenger	Dynamics	75	kg
`Counterweight`	Mass of counterbalancing weight	Dynamics	850	kg
`Damping Coef`	Damping coefficient	Dynamics	10	kg/s
`Damping Force`	Force due to system damping	Dynamics		N
`Empty Mass`	Mass of empty carriage	Dynamics	1000	kg
`Gravity`	Gravitational Acceleration	Dynamics	9.8	m/s^2
`Height`	Current position of carriage	Dynamics		m
`Initial Height`	Initial position of the carriage	Dynamics	0	m
`Initial Velocity`	Initial velocity of carriage	Dynamics	0	m/s
`Net Force`	Total force on carriage	Dynamics		N
`Passenger Mass`	Total mass of passengers on carriage	Dynamics		kg
`Total Mass`	Total mass of carriage	Dynamics		kg
`Velocity`	Current velocity of carriage	Dynamics		m/s
`Weight`	Weight of carriage	Dynamics		N
`Step Size`	Time between simulation steps	Many	0.1	s
`ΔI`	The change in integrative outputs	PID		N
`D Output`	Force output set by derivitive controller	PID		N
`Dest Height`	Height of destination floor	PID		m
`Destination`	Current floor carriage is moving to	PID	1	m
`Error`	Distance between current and goal locations	PID		m
`Error Difference`	Difference between current and previous step	PID		m
`Filter Constant`	Low-pass filter constant	PID	0.5	s
`Filtered Error`	Current filtered error signal	PID		m
`Floor Gap`	Distance between floors	PID	4	m
`Force Input`	Force input given to motor	PID		N
`Ht Tolerance`	Distance where carriage is considered at a floor	PID	0.2	m
`I Output`	Force output set by integrative controller	PID		N
`Kd`	Derivitive gain for PID controller	PID	-0.79	s
`Ki`	Integral gain for PID controller	PID	35	s^{-1}
`Kp`	Proportional gain for PID controller	PID	271	
`Max Force`	Maximum force reachable by the PID controller	PID	10000	N
`Min Force`	Minimum force reachable by the PID controller	PID	-1000	N
`P Output`	Force output set by proportional controller	PID		N
`PID Output`	Total force ouput of the PID controller	PID		N
`Prv Fltd Error`	Filtered error signal of previous step	PID		m

List of Figures

List of Tables