GHG

May 1, 2019

Abstract

GHG

—

Computational and scientific data representation intrinsically balances two competing priorities: "semantic" expressiveness and computational tractability. On the one hand, representations should not obscure important details: the formal requirements on representational validity should not force representations into structures that necessitate the elimination of meaningful information. On the other hand, conversely, representations should have enough structural consistency that they are amenable to analysis and transformations driven by formal algorithms.

I have just left a lot of loose ends: in particular, these comments need to give some meaningful definition to "representation". There are many media wherein "data" can be represented: via graphics (e.g., charts, "maps" in the cartographical sense, or "graphs" in the sense of plotted visualizations of mathematical functions), via printed documents (like the logarithmic tables or astronomic records of early modern science), or via mathematical equations and formulae (if a mathematical theory correctly predicts and quantifies empirical data — e.g., fitting trajectories to elliptic or parabolic shapes — then the numeric structures of the theory are a proxy representation of the corresponding data). Moreover, aside from these relatively formal modes of representation, we have the capacity to indirectly describe information via natural language.

The modern age has a further notion of representation: *digital* representations which leverage the capabilities of computer storage and processing to create persistent data repositories, with the possibility of manipulating data via computer programming and displaying data via computer software. These digital representations incorporate features of the older representational media I alluded to: like Natural Language digital data often emanates from a textual representation, with data structures initialized from special "languages" like XML or JSON. As with structured "printed" documents (of the astronomical-table or even grocery-shopping-list variety), digital representations often build off of a rigid structural template, like a two-dimensional table with rows and columns, or a hierarchical document whose elements can be nested documents. And as with mathematical representations, digital representations can be analyzed as instances of spaces or structures with certain algebraic or syntactic properties and requirements. It is often possible to mathematically describe the full space of structures which conform to a given representational protocol, or the full space of transformations that can modify a given data structure while staying consistent with its enforced protocols. Also, each data structure potentially has some notion of aggregateness: of having different "parts", of being able to focus on one part at a time, and to "move" focus from one part to another. Given these qualities, digital representations extend and integrate the affordances of print, graphical, and linguistic media — but they do so in an electronic environment that permits digital archives to be accessed via computer software. Given this access, digital representations can inherit the formal properties of the software that uses them, so we can think of digital representations as formal structures themselves, amenable to something like a mathematical analysis.

In short, digital representations have several general criteria: *validity* (a formal model of conformant vs. invalid structures[1]); *traversability* (a notion of parthood

[1] An invalid graph might be a case where an edge has no incident nodes;

and iterating or refocusing among parts), and, let's say, *atomicity* (a notion of unitary parts that can be represented as integral wholes). These criteria ensure that digitally represented data structures can work within software and networking representations: information is presented to software users by displaying atomic units (textually or graphically) and traversing through data structures to fill in, via application code, a visual tableau presenting compound data, with different unitary displays visually separated and often organized into coherent visual patterns, like the grid-pattern of a spreadsheet. Meanwhile, atomic units can be textually encoded, and aggregate structure likewise notated through syntactic conventions that preserve atom's boundaries, yielding textual serializations of data structures that can be shared between computing environments, allowing information to be copied and distributed. Finally, digital representations of data structures canbe marshalled into different binary forms — encoded in byte- and bit-patterns — to enable both persistent storage in a database and "runtime" presence as binary data that can be accessed by software applications.

I take the time to lay out these basic principles because I want to emphasize the different contexts where digital representations can be found: in particular, textual encoding and serialization; graphical/interactive displays for software users; application runtimes; and long-term database storage. A given representation will morph and mutate to accommodate these different contexts. Moreover, these contexts correspond to distinct technical specializations: database engineers have a different perspective on data structures than network engineers designing protocols for encoding and exchanging data between network endpoints; and application designers focused on optimal human-computer-application understand digital representations as visual and interactive phenomena, whereas application *developers* need to focus on how to properly encode data structures for binary runtimes. These various perspectives all influence the theory and technology behind digital data representation: a successful representational paradigm needs to adapt to the operatoonal requirements of engineers in each of these disicplines.

Additionally, insofar as the point of digital archives is to encode empirical, "real-world" data, a proper representational protocol needs to promote a synergy between the information as humans understand it and the data structures recognized bu the technology. For instance, if a published data set shares scientific data, it should be represented in ways that preserve scintifically significant details — any derivations, descriptions, or observations which are intrinsic to the science's methodology, theory, assumptions, and experimental results. The data needs to be structured according to the "semantics of the science", so that the scientific background can be reconstructed along with future use of the data, even after a circuitous journey through different contexts, like through a database and over a network to a scientific-software application (maye years later). As formal models of data semantics have become more rigorous — e.g. in our century with Ontologies and the "Semantic Web" — this "semantic engineering" has become a further technical perspective needing consideration in the dsign and evaluation of digital representations.

Over the decades, many general representation strategies and "layouts" have been envisioned, from tabular structures in the manner of Relational Databases, to tree-like documents whose morphology is inspired by markup languags, to structurally looser variations on row/column arrangements like multi-dimensional key-value spaces or "Big Column" and other "NoSQL"-database-inspired articulations. These various paradigms are subject to "selective" pressures based on how well the meet the different engineering needs I have identified. But the multiplicity of these needs complicates the "competition" between representational strategies: a paradigm optimized for one context (e.g., persistent databases) is not necessarily optiman for another (e.g., application and GUI development). As a result, developers and computer scientists must continue to explore and collaborate on new paradigms which work better across contexts.

In the plurality of representational paradigms, a significant evolution has been the emergent popularity of structures based on graphs. The predominant influence on this line of development has been the Semantic Web and the specific graph architecture codified by "Web Ontologies", although other graph-representation models (such as Conceptual Graph Semantics) were also candidates for potential technological "entrenchnment" before Semantic Web formats like RDF and OWL became

an invalid XML structure is one without a single root element or (insofar as such a structure would be representable) with mismatched tags, and so forth) standardized.[2] More recently, scientists have proposed generalizations and/or alternatives to the Semantic Web based on hypergraphs in lieu of ordinary (directed, labeled) graphs.

As might be ascertained from hypergraphs being the focus of this paper, I believe that representational paradigms based on hypergraphs can be superior to other formats and need to be studied with an integrated, generalized set of theories and computer tools. I have inrododucd the topic of hypergraph data structures via the general issues in digital representation in part to rest claims of hypergraphs' merit on explicit criteria. In particular, I believe hypergraphs can adapt to the different technological contexts prerequisute to a decentralized digital ecosystem — databases, networking, application implementation, visual/interactive software design, and semantic expressiveness — more effeectively than other paradigms, like regular graphs or SQL-style tables. I suspect other scientists and engineers have similar intuitions, because there has been a recent uptick in research on hypergraphs in various disciplines, such as Category Theory, Information Management, Artificial Intelligence, and Natural Language Processing. Compared to the Semantic Web, however, there is a noticeable lack of standard tools or formats for expressing hypergraph data or sharing hypergraph structures across multiple applications and environments. Where RDF and OWL are definitely associated with the Semantic Web, so that supporting these standards is a basic entry point for Semantic Web technologies, there is no comparable consensus on the underlying theory of hypergraph data in general.

This situation may be explained in part by subtle differences on what the word "hypergraph" is supposed to mean in different settings, so the overal terrain of hypergraphs is divided into distinct mathematical models, often without a rigorous theory of their interrelationships. Another problem is perhaps a failure to appreciate how hypergraphs are structurally different from ordinary graphs, so that hypergraph-shaped data may be imprecisely treated as a mere variation upon or a special structuration within ordinary graphs. For eample, Semantic Web structures such as RDF "bags" and "collections" introduce a kind of hierarchical organization to Semantic Web graphs, and can be considered a form of hypergraph, but these protocols are not described as enabling a transition from a networking framework based on directed labeled graphs to one based on hypergraphs. Instead, hypergraphs become *de facto* embedded within ordinary graphs, exploiting the representational flexibility which also makes the Semantic Web suitable for spreadsheets, XML, and other structures that are not graphs at a *prima facie* level. The problem is that while hypergraphs can be *encoded* in ordinary graphs with a suitable labeling convention, the distinct structural details which make hypergraphs superior to the Semantic Web no longer remain as explicit architectural features once hypergraphs are "encoded" in conventional Semantic Web formats.

Recent (computer-science-related) research into hypergraphs sems to emphasize two different topics: first, the representational potential for hypergraphs as a general morphology for representing structured information in general; and, second, the specific possibilities of using hypergraphs to model computer code and computational processes. In the second subject-area, hypergraphs are studied as a medium for expressing details about computer algorithms as a way to reason about executable computer code as a structured system, and perhaps even design execution engines (virtual machines to run computer code that is in a suitable representation). This latter research sometimes appears to present a mathematical picture of hypergraphs as an abstract model of computing procedures and evaluations — a kind of graph-based rinterpretation of the lambda calculus — but sometimes is also marshaled into implemented execution environments, as in the OpenCog and lmnTal frameworks.

My goal in this paper is to consider what a unified hypergraph paradigm can look like — how the different strands which in their own way embody hypergraph structures can be woven together into a multi-purpose whole. My emphasis is on computer implementations rather than mathematics — that is, I will not present axioms or theorems formalizing different sorts of hypergraphs, though I ackowledge that such descriptions are possible. Instead, my aim is to describe what might constitute a general software library or toolkit that can adapt to different hypergraph models and use-cases. This paper is accompanied by a "data set" which involves a

[2] Why RDF/OWL and not, say, Conceptual Graphs, or the hybrid Object-Database/Graph-Database models studied in the 1990s, became canonized in the Semantic Web, is an interesting question but perhaps of mostly historical interest insofar as Hypergaphs can unify each of these paradigms.

library (written in C++) for creating and manipulating hypergraphs of different varieties, and also a "virtual machine" for modeling and realizing computatational procedures via hypergraph structures. The point of the accompanying code is to demonstrate that a generalized hypergraph representation is possible, and that in addition to use-cases for modeling data structures such a representation can be used at the core of a virtual processor. The code includes simple tests and "scripts" that can be executed via the provided virtual-machine code.

Studying hypergraphs from a computational (and "implementational") perspective — not just as mathematical objects — introduces useful details that can add depth to our overall understanding of hypergraphs. For example, one question is how hypergraphs can be initialized — how software systems can build up hypergraph structures, piece by piece. This is related to the question of proper serialization formats for hypergraphs. Given some specific hypergraph data aggregate ″, it is important to have a textual encoding where ″ can be mapped to a character-string, shared as a document, and then reconstructed with the same hypergraph structure. This raises the question of when and whether two hypergraphs are properly isomorphic — such that serializing and then deserializing a hypergraph yields "the same" hypergraph — and also the problem of validating and parsing textual representations of hypergraphs. The theory of parsing *serializations* of hypergraphs — tectual encodings whose grammar and morphology are suitable to expressing and then rebuilding hypergraph structures — then becomes an extension of hypergraph theory itself.

In addition to creating hypergraphs via textual representations in the manner of markup languages (like XML or RDF), we can also consider the incremental accumulation of hypergraph structures via minimal units of transformation, providing a kind of "Intermediate Representation" embodying the form of a hypergraph via a sequence of effectively (at least on one scale of analysis) atomic operations. For any variety of hypergraph, then, we can consider which is a suitable Intermediate Representation for conformant spaces; for example, which set of primitive options can, iteratively, construct any representative of the space of instantations of a given kind of hypergraph. Insofar as a software framework seeks to work with sveral hypergraph varieties, we can consider how several different such operation-sets can be combined.

Moreover, we can consider both serialization and Intermediate Representation as alternative tactics for building hypergraphs, which can be combined. An Intermediate Representation language can be used to carry out transformations between hypergraphs — producing IR code based on the first hypergraph from which the second can be created. In conjunction with serialization and parsers, a hypergraph can be realized via several stages, with one form of hypergraph used as an intermediary because it has a convenient grammar and works with a parsing engine; from that preliminary graph a new graph can be created via IR code. Grammars, serializaion and deserialization, and IR thereby become part of the formal architecture defining a hypergraph ecosystem and inter-graph transformations (analogous to the trio of XML parsers, XML transforms, and the XML Document Object Model). Continuing the XML analogy, note that XML is not just a serialization specification: the full XML technology defined requirements on a series of tools operating on XML data, not just XML files: tools for traversing XML documents (including a programming interface for how traversal options are to be operationalized, e.g. via Object-Oriented methods like parentNode and nextSibling); for parsing XML files into traversable data structures (including requirements on the traversals which the derived structurees — the so-called "post-processing infoset" — must support); and for intr-document mapping (via XSLT or XML-FO). The Document Object Model and post-processing infoset is the structural core of the XML format, even more than any surface-level syntax (the grammar for tags, attributes, and so forth). Analogous specifictions would have to be developed for a generalized Hypergraph representation.

The code discussed here does not purport to provide a mature or decisive implementation of a general-purpose hypergraph ecosystem, but rather to demonstrate by example what components may comprise such a framework and how they may interoperat. The cod inclus hyprgraph models but also the preliminary and intermediary structures that can connct hypergraphs to a surrounding computational infrastructure — parsers, Intermediate Representation, application-integration logic (such as mapping hypergraph nodes to application-specific data types), and a "virtual machine" for realizing hypergraphs as executable code.

The remainder of this paper will focus on thre subjects: delimitating the proper range of hypergraph models; execution models and the "virtual machine"; and a review of formal structurs (like grammars and IR formats) which ar estructurally different than hypergraphs but can help tie hypergraph representations togther into a unified ecosystem. I have spoken only informally about hypergraphs themselves, taking for granted that we have a general picture of what hypergraphs are, but this now demands more rigorous treatment. There are actually several different research and engineering communities that talk about hypergraphs, in each case describing some sort of generalization of regular (often labeled and/or directed) graphs, but these generalizations fall into different kinds. Hence there are several different varieties of hypergraph, and I will try to outline a general theory of hypergraphs in these various forms.

1 Varieties of Hypergraphs

Any notion of hypergraphs contrasts with an underlying graph model, such that some element treated as singular or unstructurd in regular graphs becomes a multiplcity or compound structure in the hypergraph. So for example, an edge generalizes to a hyperedge with more than two incident nodes (which means, for directed graphs, more than one source and/or target nodes). Likewise, nodes can generalize to complex structures containing other nodes (including, potentially, nested graphs). The "elements" of a graph are nodes and edges, but also (for labeled/weighted and/or directed graphs) things like labels, weights (such as probability metrics associated with edges), and directions. Potentially, each of these elements can be transformed from a simple unit to a plural structure, a process I will call "diversification". That is, a hypergraph emerges from a graph by "diversifying" some elements, rendering as multiplicities what had preciously been a single entity — nodes become node-sets, edges become grouped into larger aggregates, labels generalize to complex structures (which we can call "annotations"), etc. Different avenues of diversification give rise to different varieties of hypergraphs, which I will review in the next several paragraphs.

Hyperedges Arguably the most common model of hypergraphs involves generalizing edges to hyperedges, which (potentially) connect more than two nodes. Directed hyperedges have a "source" node-set and a "target" node-set, either of which can (potentially) have more than one node. Note that this is actually a form of node-diversification — there is still (in this kind of hypergraph) just one edge at a time, but its incident node set (or source and target node-sets) are sets and not single nodes. Another way of looking at directed hyperedges is to see source and target node-sets as integral comple parts, or "hypernodes". So a (directed) hypergraph with hyperedges can also be seen as generalizing ordinary graphs by replacing nodes with hypernodes.

Recursive Graphs Whereas hyperedges embody a relatively simple node-diversification — nodes replaced by node-sets — so-called "recursive" graphs allow compound nodes (hypernodes) to contain entire nested graphs. Edges in this case can still connect two hypernodes as in ordinary graphs, but the hypernodes internally contain other graphs, with their own nodes and edges.

Hypergraph Categories and Link Grammar Since *labeled* graphs are an important model for computational models, we can also consider generalizations of labels to be compound structures rather than numeric or string labels (or, as in the Semantic Web, "predicate" terms, drawn from an Ontology, in "Subject-Predicate-Object" triples). Compound labels (which I will generically call "annotations") are encountered (sometimes implicitly) in several different branches of mathematics and other fields. In particular, compound annotations can represent the rules, justifications, or "compatibility" which allows to nodes to be connected. In this case the annotation may contain information about both incidnt nodes. An example in linguistics is morphosyntactic agreement between gramatically linked words, which involves details matched between both "ends" of a link (part-of-speech, plural/singular, gender, case/declension, etc.). This general phenomenon (I will mention further examples below) can be called a "diversification" on *labels*, transforming from labels as single units to labels as multi-part records.

Channels Hyperedges, which connect multiple nodes, are still generally seen as single edges (in contrast to multigraphs which allow multiple edges between two nodes). Analogous to the grouping of nodes into hypernodes, we can also consider structures where several different edges are unified into a larger totality, which I will call a *channel*. In the canonical case, a directed graph can group edges into composites (according to more fine-grained criteria than just distinguishing incoming and outgoing dges) which share a source or target node. For example, the set of incoming edges to the same target node may be partitioned into distinct "channels", so that at one scale of consideration the network structure can be analyzed via channels rather than via single edges. For a concrete example, consider graphs used to model Object-Oriented programming languages, where a single node can represent a single function call. The incoming edges are then "input parameters", and outgoing edges are procedural results or outputs. In the Object-Oriented paradigm, however, input paramters are organized into two groups: in addition to any number of "conventional" arguments, there is a single this or self object which has a distinct semantic status (vis-à-vis name resolution, function visibiloty, and polymorphism). This calls, under the graph model, for splitting incoming edges into two "channels", one reprsenting regular parameters and a separate channel for the distinguished or "receiver" object-value.