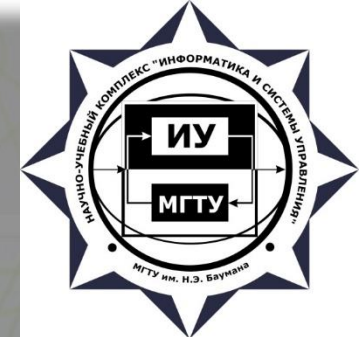




Bauman Moscow State Technical University
Computer Science and Control Systems Department



Logical Perspectives 2021 *Workshop*

Ideas of Metagraph-Based Types

Yuriy E. Gapanyuk, Anatoly N. Nardid, and Dmitry A. Zuev

Moscow, 19 June 2021

The outline

1. Complex graphs (network) models. The metagraph model.
2. The metagraph model processing.
3. Metagraph model vs. RDF model.
4. Metagraph model predicate representation.
5. Metagraph-based approach for types description.

Complex network models (1)

- Currently, models based on complex networks are increasingly used in various fields of science from mathematics and computer science to biology and sociology.
- A complex network is a graph (network) with non-trivial topological features – features that do not occur in simple networks such as lattices or random graphs but often occur in graphs modeling of real systems.
- The terms “complex network” and “complex graph” are often used synonymously. But the term “complex network” usually refers to real systems while the term “complex graph” is generally considered as the mathematical representation of a network. In our work, we also acknowledge these terms synonymously.

Complex network models (2)

- One of the most important types of such models is “complex graph (network) with emergence.”
- The term “emergence” is used in general system theory. The emergent element means a whole that cannot be separated into its component parts.
- Nowadays, there are two “complex networks with emergence” models exist: hypernetworks and metagraphs. The hypernetwork model is mature and it helps to understand many aspects of complex networks with an emergence.
- But the metagraph model is more flexible and convenient for use in information systems.

The metagraph model (1)

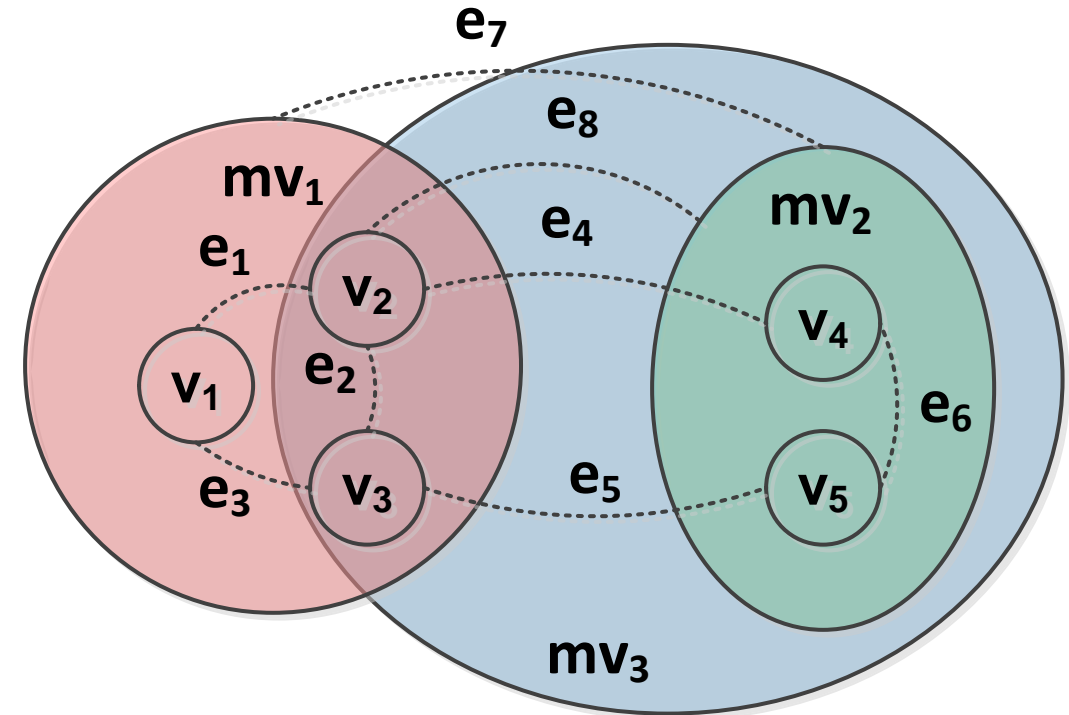
- A metagraph is a kind of complex network model, proposed by A. Basu and R. Blanning in their monography «*Basu A., Blanning R. Metagraphs and Their Applications, 2007*» and then adapted by the authors for information systems description.
- $MG = \langle V, MV, E \rangle$, where MG – metagraph; V – set of metagraph vertices; MV – set of metagraph metavertrices; E – set of metagraph edges.
- A metagraph vertex is described by the set of attributes: $v_i = \{atr_k\}$, $v_i \in V$, where v_i – metagraph vertex; atr_k – attribute.
- A metagraph edge is described by the set of attributes, the source and destination vertices and edge direction flag: $e_i = \langle v_S, v_E, eo, \{atr_k\} \rangle$, $e_i \in E$, $eo = true|false$, where e_i – metagraph edge; v_S – source vertex (metavertex) of the edge; v_E – destination vertex (metavertex) of the edge; eo – edge direction flag ($eo=true$ – directed edge, $eo=false$ – undirected edge); atr_k – attribute.

The metagraph model (2)

- The metagraph fragment: $MG_i = \{ev_j\}, ev_j \in (V \cup MV \cup E)$, where MG_i – metagraph fragment; ev_j – an element that belongs to union of vertices, metaverices and edges.
- The metagraph metavertex: $mv_i = \langle \{atr_k\}, MG_j \rangle, mv_i \in MV$, where mv_i – metagraph metavertex belongs to set of metagraph metaverices MV ; atr_k – attribute, MG_j – metagraph fragment.
- The metavertex in addition to the attributes includes a fragment of the metagraph. The presence of private attributes and connections for a metavertex is distinguishing feature of a metagraph. It makes the definition of metagraph holonic – a metavertex may include a number of lower level elements and in turn, may be included in a number of higher level elements.
- From the general system theory point of view, a metavertex is a special case of the manifestation of the emergence principle, which means that a metavertex with its private attributes and connections becomes a whole that cannot be separated into its component parts.

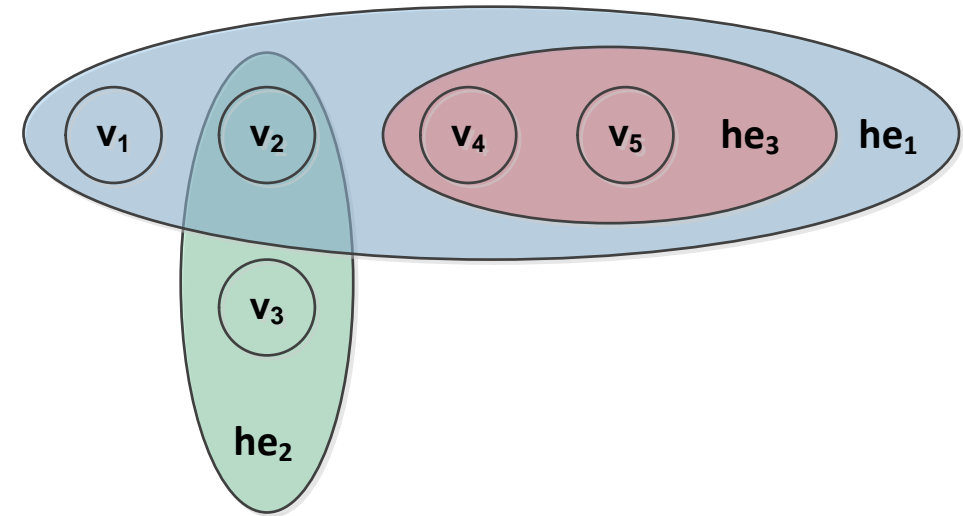
The example of metagraph

- This example contains three metavertrices: mv_1 , mv_2 and mv_3 .
- Metavertex mv_1 contains vertices v_1 , v_2 , v_3 and connecting them edges e_1 , e_2 , e_3 .
- Metavertex mv_2 contains vertices v_4 , v_5 and connecting them edge e_6 .
- Edges e_4 , e_5 are examples of edges connecting vertices v_2 - v_4 and v_3 - v_5 respectively, and are contained in different metavertrices mv_1 and mv_2 .
- Edge e_7 is an example of an edge connecting metavertrices mv_1 and mv_2 .
- Edge e_8 is an example of an edge connecting vertex v_2 and metavertex mv_2 .
- Metavertex mv_3 contains metavertex mv_2 , vertices v_2 , v_3 and edge e_2 from metavertex mv_1 and also edges e_4 , e_5 , e_8 showing the holonic nature of the metagraph structure.
- Thus, metagraph model allows describing complex data structures and it is the metavertex that allows implementing emergence principle in data structures.



Metagraph and hypergraph models comparison

- Hypergraph $HG = \langle V, HE \rangle, v_i \in V, he_j \in HE$, V – set of hypergraph vertices; HE – set of non-empty subsets of V called hyperedges; v_i – hypergraph vertex; he_j – hypergraph hyperedge.
- A hypergraph may be directed or undirected (shown in figure). A hyperedge in an undirected hypergraph only includes vertices whereas in a directed hypergraph, a hyperedge defines the order of traversal of vertices.
- The figure contains three hyperedges: he_1 , he_2 and he_3 .
- Hyperedge he_1 contains vertices v_1, v_2, v_4, v_5 .
- Hyperedge he_2 contains vertices v_2 and v_3 .
- Hyperedge he_3 contains vertices v_4 and v_5 .
- Hyperedges he_1 and he_2 have a common vertex v_2 .
- All vertices of hyperedge he_3 are also vertices of hyperedge he_1 .
- The inclusion of hyperedge he_3 in hyperedge he_1 is only graphical and informal, because according to hypergraph definition a hyperedge inclusion operation is not explicitly defined.

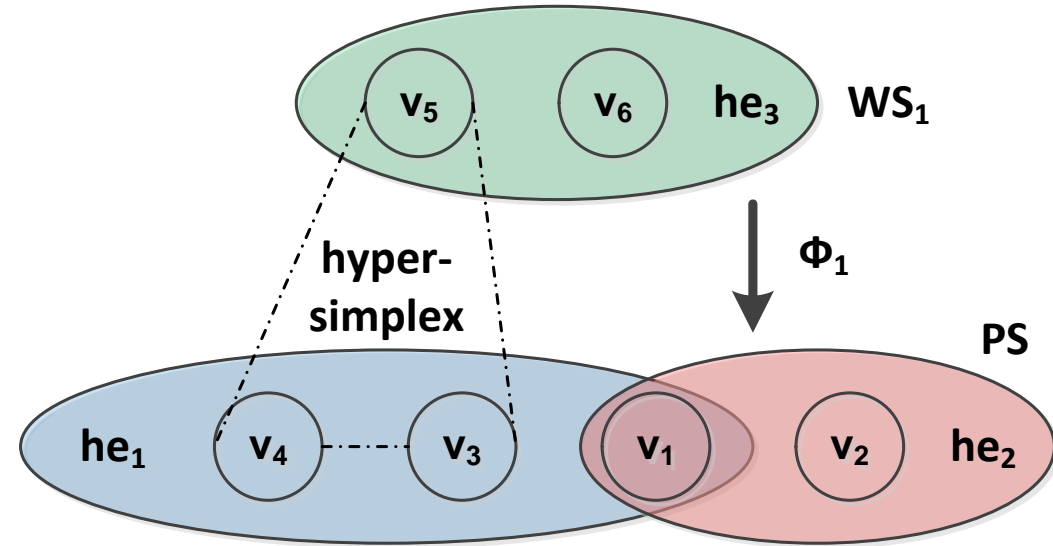


The hypernetwork model

- The hypernetwork model was invented twice.
- The first time the hypernetwork model was invented by Professor Vladimir Popkov with colleagues in 1980s. Professor V. Popkov proposes several kinds of hypernetwork models with complex formalization and therefore only main ideas of his hypernetworks model will be discussed.
- Given the hypergraphs $PS \equiv WS_0, WS_1, WS_2, \dots, WS_K$. The hypergraph $PS \equiv WS_0$ is called primary network. The hypergraph WS_i is called secondary network of order i . Also given the sequence of mappings between networks of different orders: $WS_K \xrightarrow{\Phi_K} WS_{K-1} \xrightarrow{\Phi_{K-1}} \dots WS_1 \xrightarrow{\Phi_1} PS$. Then the hierarchical abstract hypernetwork of order K may be defined as $AS^K = \langle PS, WS_1, \dots, WS_K; \Phi_1, \dots, \Phi_K \rangle$. The emergence in this model occurs because of the mappings Φ_i between the layers of hypergraphs.
- The second time the hypernetwork model was proposed by Professor Jeffrey Johnson in his monography in 2013.
- The main idea of Professor J. Johnson variant of hypernetwork model is the idea of hypersimplex (the term is adopted from polyhedral combinatorics). Hypersimplex is an ordered set of vertices with an explicit n -ary relation and hypernetwork is a set of hypersimplices. In hierarchical system the hypersimplex combines k elements at level N (base) with one element at level $N+1$ (apex). Thus, hypersimplex establishes an emergence between two adjoining levels.

The example of hypernetwork

- The primary network PS is formed by the vertices of hyperedges he_1 and he_2 .
- The first level WS_1 of secondary network is formed by the vertices of hyperedge he_3 .
- Mapping Φ_1 is shown with an arrow.
- The hypersimplex is emphasized with the dash-dotted line.
- The hypersimplex is formed by the base (vertices v_3 and v_4 of PS) and apex (vertex v_5 of WS_1).



Metagraph and hypernetwork models comparison

- Hypernetwork model may be considered as “horizontal” or layer-oriented. The emergence appears between adjoining levels using hypersimplices. The metagraph model may be considered as “vertical” or aspect-oriented. The emergence appears between any levels using metavertices.
- In hypernetwork model the elements are organized using hypergraphs inside layers and using mappings or hypersimplices between layers. In metagraph model metavertices are used for organizing elements both inside layers and between layers. Hypersimplex may be considered as a special case of metavertex.
- Metagraph model allows organizing the results of previous organizations. The fragments of flat graph may be organized into metavertices, metavertices may be organized in higher-level metavertices and so on. Metavertex organization is more flexible than hypersimplex organization because hypersimplex assumes base and apex usage and metavertex may include general form graph.
- Metavertex may represent a separate aspect of organization. The same fragment of flat graph may be included in different metavertices whether these metavertices are used for modelling different aspects.
- Thus, we can draw a conclusion that metagraph model is more flexible than hypernetwork model.

The metagraph model processing (1)

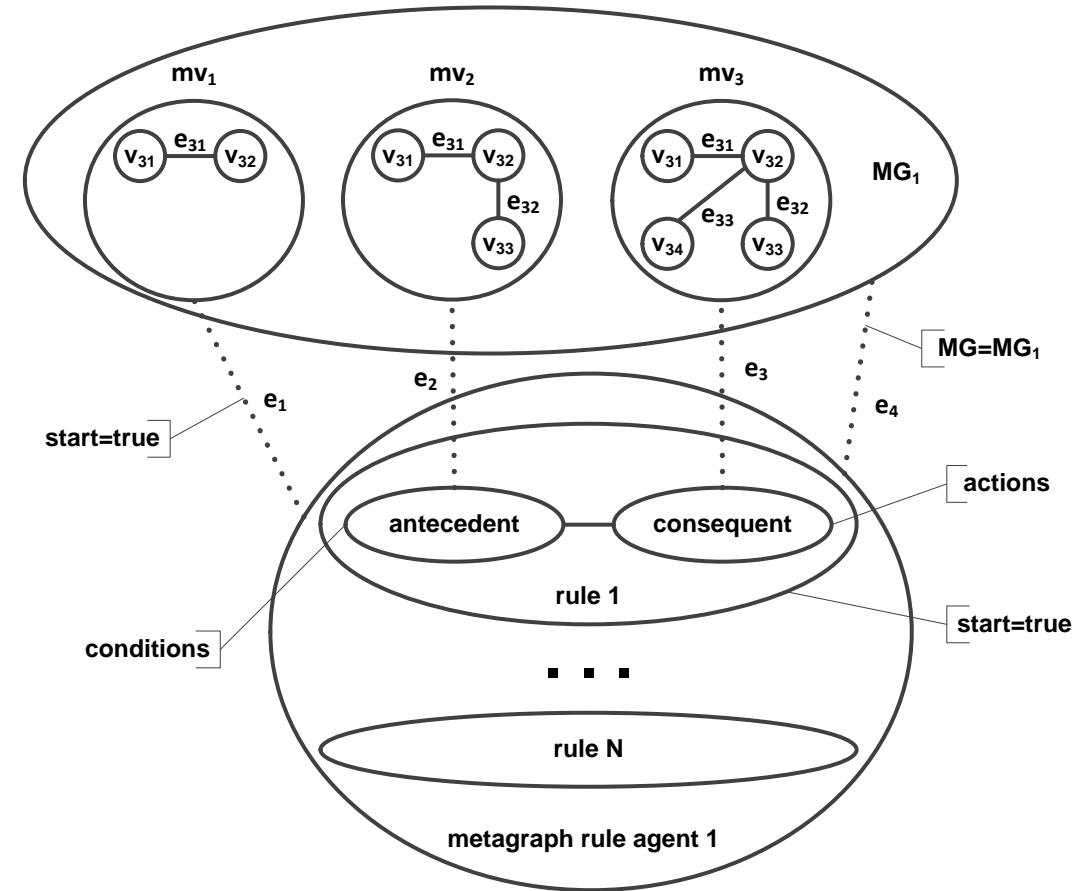
- The metagraph model is aimed for complex data description.
- But it is not aimed for data transformation.
- The idea for transforming the flat graphs with agents was proposed by Professor Vladimir Golenkov in his graphodynamics approach. By analogy with graphodynamics approach the metagraph agent (ag^{MG}) aimed for metagraph transformation is proposed.
- There are two kinds of metagraph agents: the metagraph function agent (ag^F) and the metagraph rule agent (ag^R). Thus $ag^{MG} = ag^F | ag^R$.
- The metagraph function agent serves as function with input and output parameter in form of metagraph:
- $ag^F = \langle MG_{IN}, MG_{OUT}, AST \rangle$, where ag^F – metagraph function agent; MG_{IN} – input parameter metagraph; MG_{OUT} – output parameter metagraph; AST – abstract syntax tree of metagraph function agent in form of metagraph.

The metagraph model processing (2)

- The metagraph rule agent is rule-based: $ag^R = \langle MG, R, AG^{ST} \rangle$, $R = \{r_i\}$, $r_i: MG_j \rightarrow OP^{MG}$, where ag^R – metagraph rule agent; MG – working metagraph, a metagraph on the basis of which the rules of agent are performed; R – set of rules r_i ; AG^{ST} – start condition (metagraph fragment for start rule check or start rule); MG_j – a metagraph fragment on the basis of which the rule is performed; OP^{MG} – set of actions performed on metagraph.
- The antecedent of rule is a condition over metagraph fragment, the consequent of rule is a set of actions performed on metagraph. Rules can be divided into open and closed.
- The consequent of open rule is not permitted to change metagraph fragment occurring in rule antecedent. In this case the input and output metagraph fragments may be separated. The open rule is similar to the template that generates the output metagraph based on the input metagraph.
- The consequent of closed rule is permitted to change metagraph fragment occurring in rule antecedent. The metagraph fragment changing in rule consequent cause to trigger the antecedents of other rules bound to the same metagraph fragment.
- Thus metagraph rule agent can generate the output metagraph based on the input metagraph (using open rules) or can modify the single metagraph (using closed rules).

The metagraph model processing (3)

- The metagraph rule agent “metagraph rule agent 1” is represented as metagraph metavertex. According to definition it is bound to the working metagraph MG_1 – a metagraph on the basis of which the rules of agent are performed. This binding is shown with edge e_4 .
- The metagraph rule agent description contains inner metavertices corresponds to agent rules (rule 1 ... rule N). Each rule metavertex contains antecedent and consequent inner vertices. In given example mv_2 metavertex bound with antecedent which is shown with edge e_2 and mv_3 metavertex bound with consequent which is shown with edge e_3 . Antecedent conditions and consequent actions are defined in form of attributes bound to antecedent and consequent corresponding vertices.
- The start condition is given in form of attribute “start=true”. If the start condition is defined as a start metagraph fragment then the edge bound start metagraph fragment to agent metavertex (edge e_1 in given example) is annotated with attribute “start=true”. If the start condition is defined as a start rule then the rule metavertex is annotated with attribute “start=true” (rule 1 in given example). Figure shows both cases corresponding to the start metagraph fragment and to the start rule.
- The distinguishing feature of metagraph agent is **homoiconicity** which means that it can be data structure for itself. This is due to the fact that according to definition metagraph agent may be represented as a set of metagraph fragments and this set can be combined in a single metagraph. Thus metagraph agent can change the structure of other metagraph agents.



The active metagraph

- In order to combine the data metagraph model and metagraph agent model, the concept of “active metagraph” is proposed:

$$MG^{ACTIVE} = \langle MG^D, AG^{MG} \rangle, AG^{MG} = \{ag_i\},$$

- where MG^{ACTIVE} – an active metagraph; MG^D – data metagraph; AG^{MG} – set of metagraph agents ag_i , attached to the data metagraph.
- Similar structures are often used in computer science. As an example, we can consider a class of object-oriented programming language, which contains data and methods of their processing. Another example is a relational DBMS table with an associated set of triggers for processing table entries.
- The main difference between an active metagraph and a single metagraph agent is that an active metagraph contains a set of metagraph agents that can use both closed and open rules. For example, one agent may change the structure of active metagraph using closed rules while the other may send metagraph data to another active metagraph using open rules. Agents work independently and can be started and stopped without affecting each other.

The metagraph model (4) – summary

- The metagraph is a true complex graph model whereas the hypergraph is a near flat graph model that does not fully implement the emergence principle. Therefore, hypergraph model doesn't fit well for complex data structures description.
- Hypernetwork model may be considered as “horizontal” or layer-oriented. The emergence appears between adjoining levels using hypersimplices. The metagraph model may be considered as “vertical” or aspect-oriented. The emergence appears between any levels using metavertices. From our point of view the metagraph model is more flexible than hypernetwork model.

Metagraph model vs. RDF model

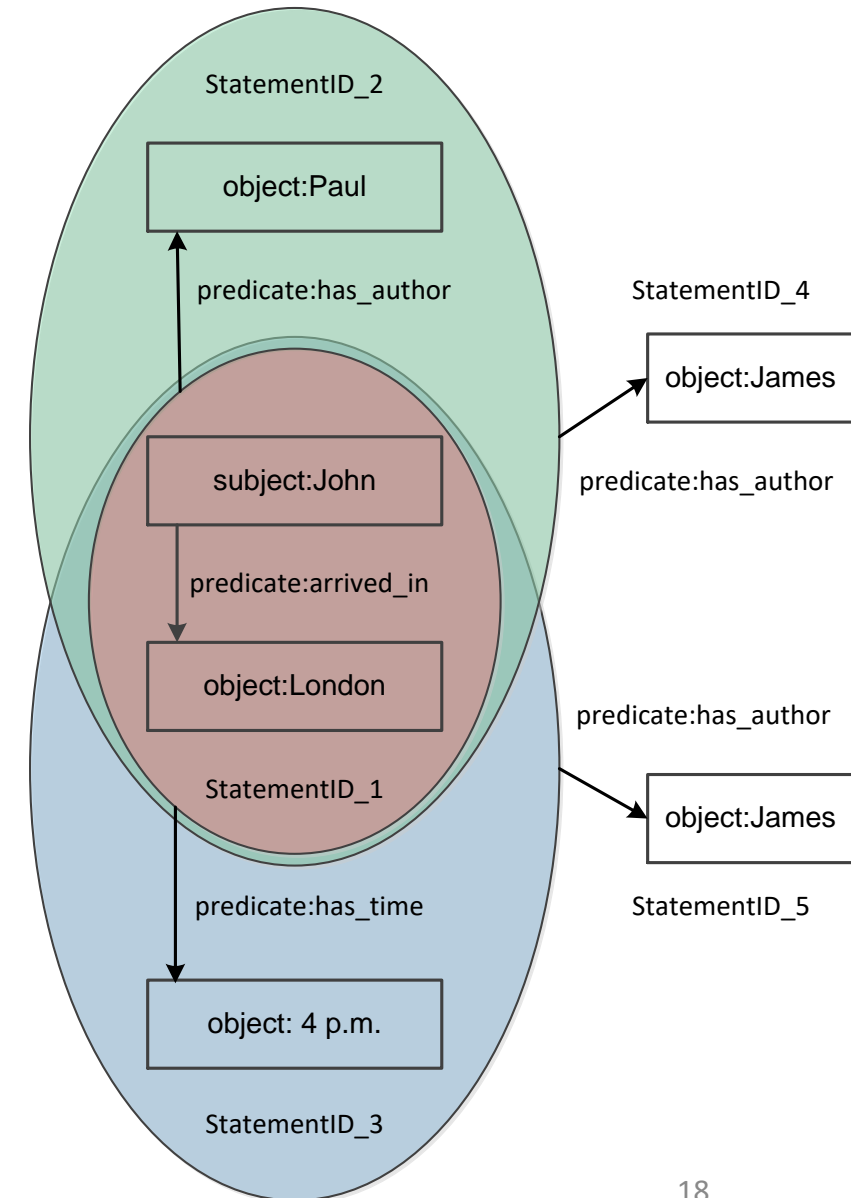
- Does emergence is a really key feature? May we have a convenient model without an emergence? Consider the example of RDF representation. We may consider RDF as “semi-emergent” graph model.
- In the beginning the key element of RDF model was a triple Subject->Predicate->Object.
- Now it’s evolved to reified triple StatementID (Subject->Predicate->Object).
- According to the RDF Primer: ‘the purpose of reification is to record information about when or where statements were made, who made them, or other similar information (this is sometimes referred to as “provenance” information)’.
- Consider the example of the complex statement: ‘James noted that Paul noted at 4 p.m. that John arrived in London’.

Reification limitation example (1)

- In the reified triples form, this example may be represented as follows:

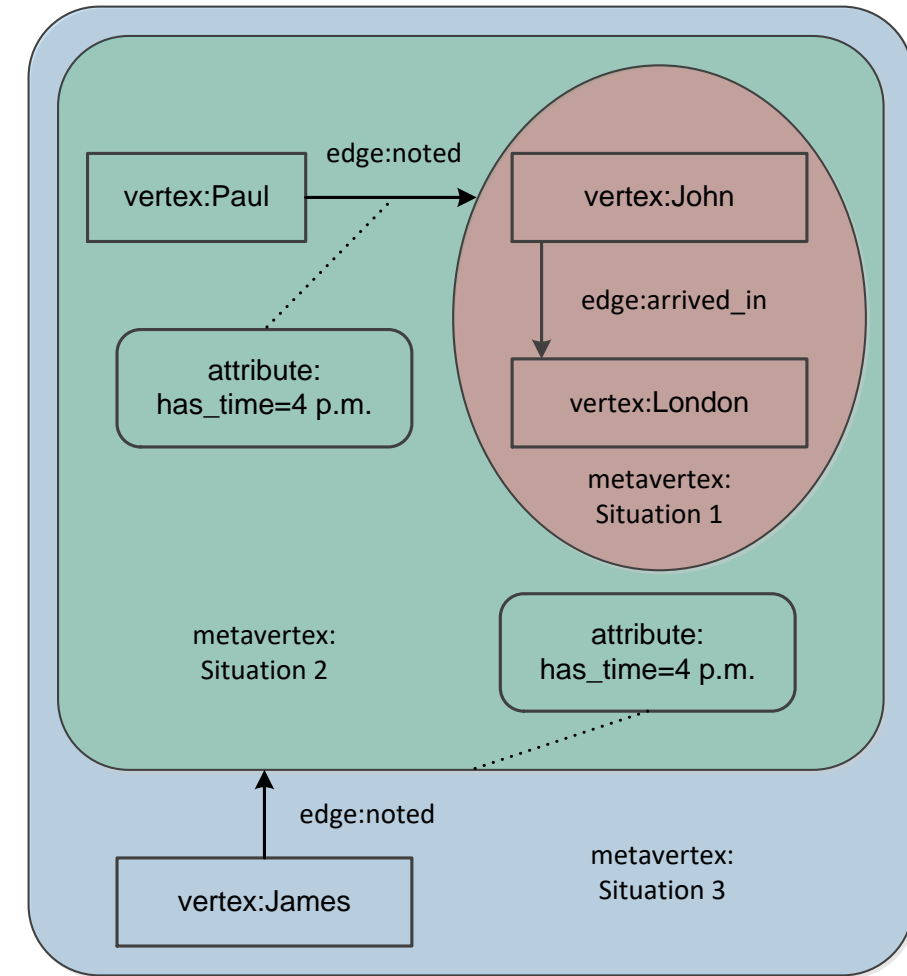
1. *StatementID_1 John arrived_in London*
2. *StatementID_2 StatementID_1 has_author Paul*
3. *StatementID_3 StatementID_1 has_time "4p.m."*
4. *StatementID_4 StatementID_2 has_author James*
5. *StatementID_5 StatementID_3 has_author James*

- The problem shown in this example is emergence loss because of artificial splitting of the whole situation into a few RDF statements. Statements 4 and 5 are represented by separate RDF statements, but they would more intuitively be represented by a single unit containing the whole situation.



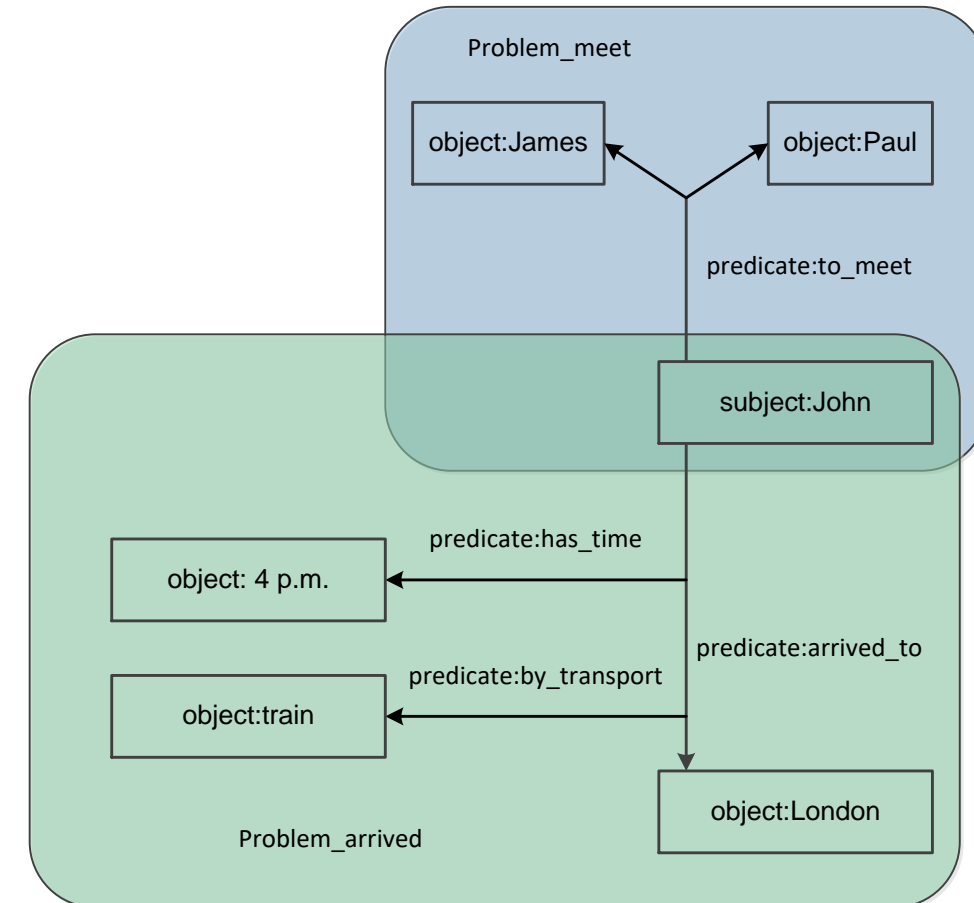
Reification limitation example (2)

- The metagraph approach helps to represent this example in a more natural and holistic way. From the metagraph point of view, this example contains three nested situations:
- Situation 1. John arrived in London.
- Situation 2. Paul noted at 4 p.m. situation 1.
- Situation 3. James noted situation 2.
- Each situation is represented by a metavertex as shown in figure. Attribute “has_time=4 p.m.” may be binded either to edge “noted” or to metavertex “Situation 2” (figure shows both cases).
- This considered example shows that the metagraph approach allows representing reification without emergence loss, keeping each nested situation in its own metavertex.
- This is because the emergence capability is explicitly built in metagraph model.



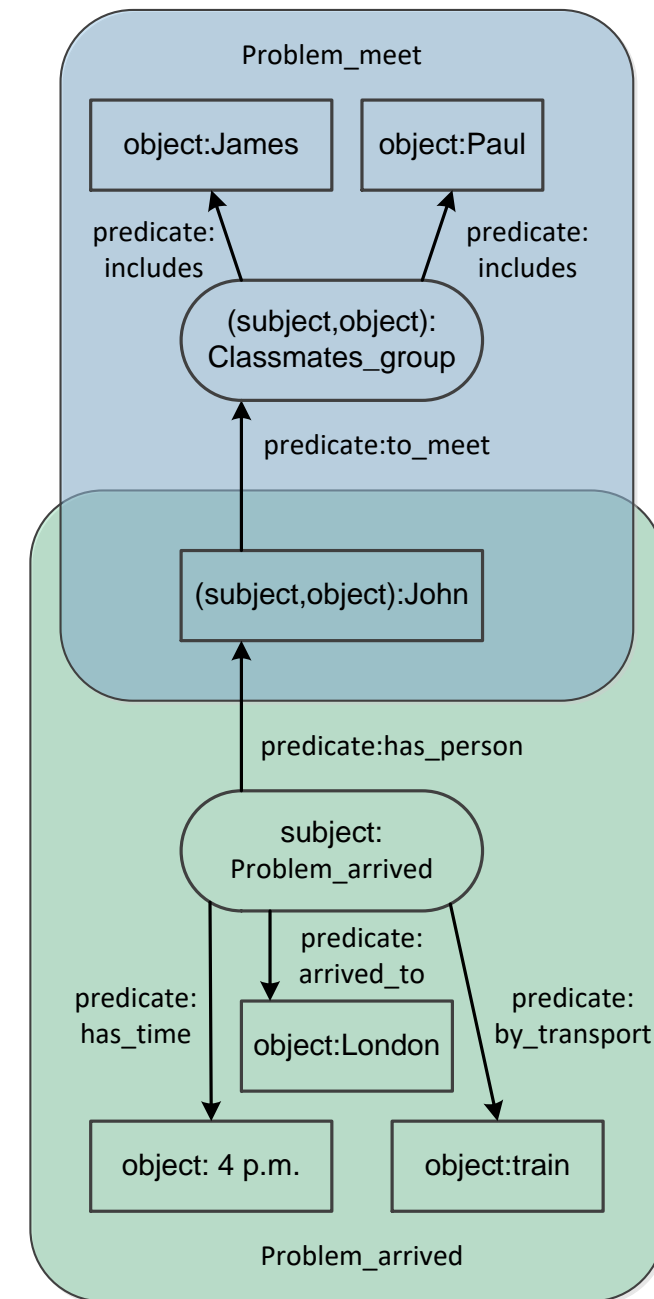
N-ary relationship limitation (1)

- An N-ary relationship is a situation where a predicate combines several subjects or objects, or has nested predicates. Such a situation is a problem from an RDF point of view. To address this problem, the 'Defining N-ary Relations on the Semantic Web' W3C Working Group Note was published.
- Consider the example of the complex statement: 'John arrived to London at 4 p.m. by train in order to meet his classmates James and Paul'. This is a typical example of an N-ary relationship. Both problems shown in figure cannot be represented by a pure RDF triplet model.
- The "Problem_arrived" is that the predicate "arrived_to" has nested predicates "has_time" and "by_transport". According to Note we are adding a supporting subject to "Problem_arrived" representing an instance of a relation.
- The "Problem_meet" is that the predicate "to_meet" has two objects "James" and "Paul". According to Note we have several ways to solve this problem. We may use the list construct of RDF or we may join object "James" and "Paul" into the classmates group. We do the latter in this example.



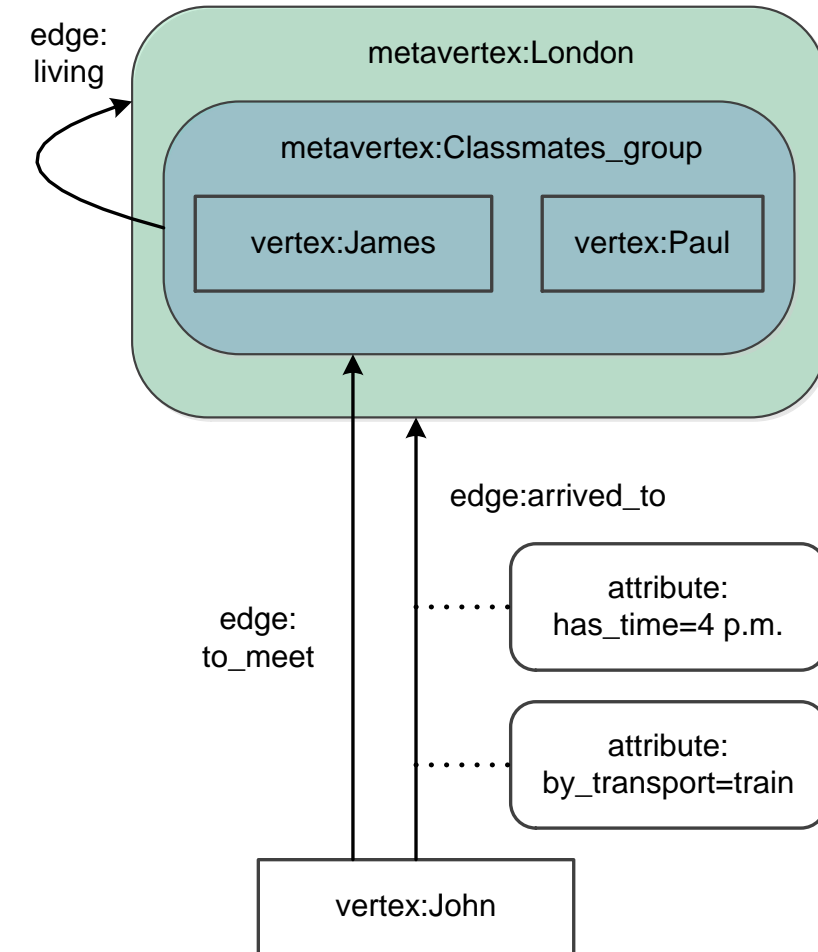
N-ary relationship limitation (2)

- The solution is shown in figure. We have added supporting vertices “Classmates_group” and “Problem_arrived”, which are shown in rounded boxes. In predicate “to_meet” the “Classmates_group” is an object while in predicate “includes” it is a subject. In predicate “has_person”, “John” is an object while in predicate “to_meet” he is a subject.
- Since we do not use reification, this may be represented in the RDF triple form “subject predicate object” as follows:
 1. *Problem_arrived has_person John*
 2. *Problem_arrived arrived_to London*
 3. *Problem_arrived by_transport train*
 4. *Problem_arrived has_time “4p.m.”*
 5. *John to_meet Classmates_group*
 6. *Classmates_group includes James*
 7. *Classmates_group includes Paul*
- As in the reification example, the problem here is in emergence loss due to the artificial splitting of the situation. The “Problem_arrived” vertex is added not because it describes the situation in a natural way, but because it is required to keep a consistent triplet structure. In a large RDF graph, many supporting vertices may obscure meaningful understanding the situation.



N-ary relationship limitation (3)

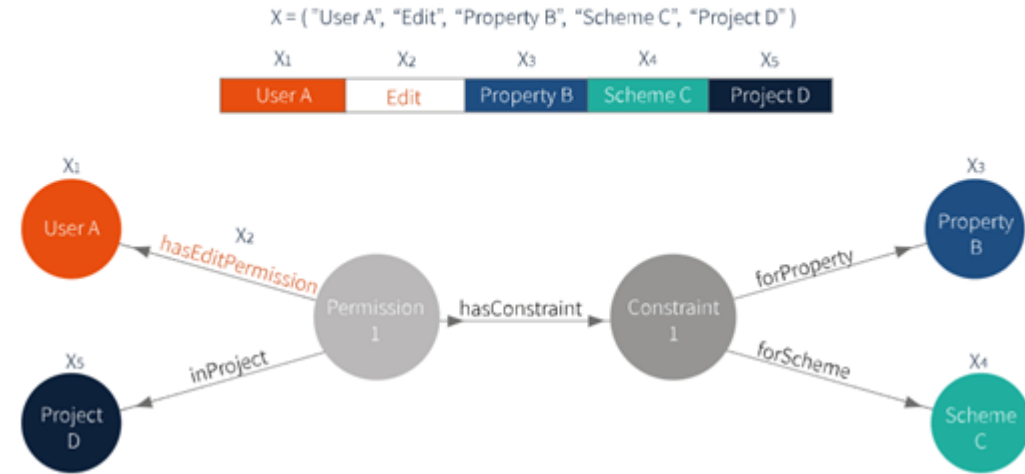
- As in the reification example, the metagraph approach helps to represent this example in a more natural and holistic way.
- The “Problem_arrived” is solved by binding attributes “has_time=4 p.m.” and “by_transport=train” to the edge “arrived_to”. The “Problem_meet” is solved by using metavertex “Classmates_group” which includes vertices “James” and “Paul”.
- The implicit knowledge about “Classmates_group” living in London may be shown either by the edge “living” or by inclusion of metavertex “Classmates_group” into metavertex “London” (figure shows both cases).
- This considered example shows that the metagraph approach allows the representation of N-ary relations without emergence loss, keeping each nested situation in its own metavertex.
- Summing up these examples, it should be noted that the metagraph model addresses RDF limitations in a natural way without emergence loss.
- The absence of the built-in emergence capability in RDF model increase the complexity of model description and may obscure meaningful understanding of the situation.



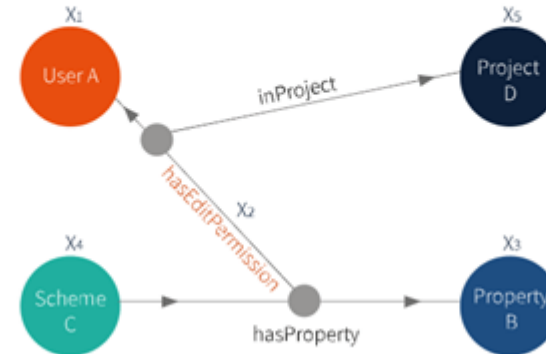
RDF-*

- The situation has now improved with the introduction of the RDF-Star specification. This specification allows the predicates of some statements to be used as subjects (objects) of other statements.
- But this still does not allow achieving the complexity of descriptions that metagraphs offer.
- Thus, the emergence is a key feature that must be explicitly built into the data model.

N-ary Relations



RDF*

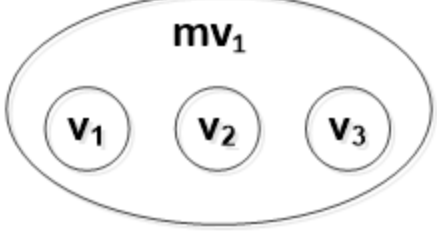
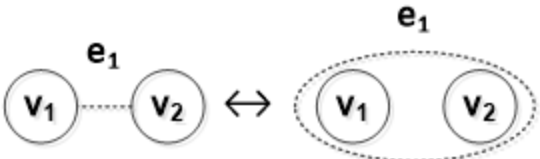

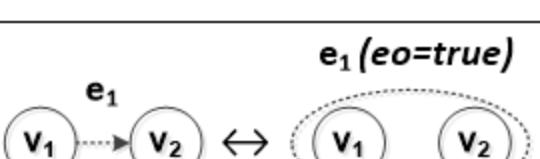
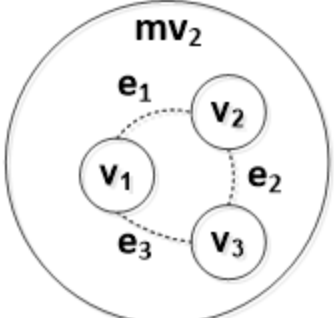


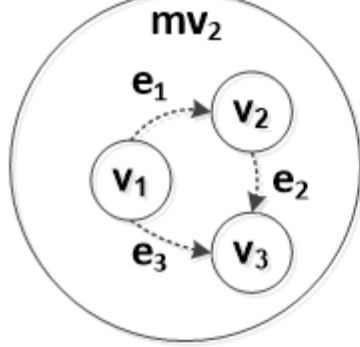
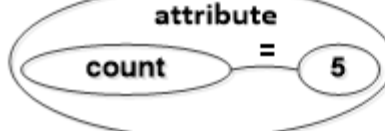
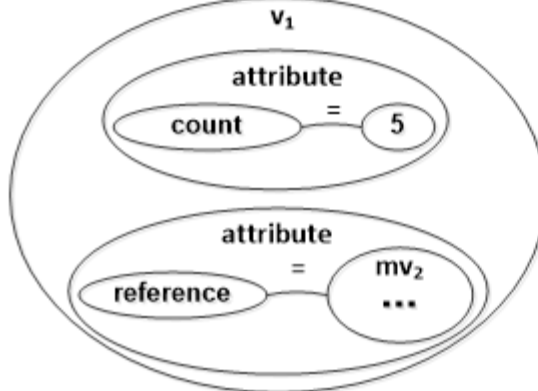
<https://www.synaptica.com/rdf-star-graphdb-graphite/>

Metagraph predicate representation (1)

- For the sake of clarity, we use the Prolog-like predicate textual representation. This representation may be easily converted into JSON or XML formats because it is compliant with JSON semantics and contains nested key-value pairs and collections.
- The classical Prolog uses the following form of the predicate: $predicate(atom_1, atom_2, \dots, atom_N)$. We used extended form of predicate where along with atoms predicate can also include key-value pairs and nested predicates: $predicate(atom, \dots, key = value, \dots, predicate(\dots), \dots)$.
- It can be noted that the use of nested predicates corresponds to higher-order logic.
- The mapping of metagraph model fragments into predicate representation is described in details in the next slide.

Metagraph predicate representation (2)

	Metavertex(Name=mv ₁ , v ₁ , v ₂ , v ₃)
	Edge(Name=e ₁ , v ₁ , v ₂)
	Edge(Name=e ₁ , v ₁ , v ₂ , eo=false)
	1. Edge(Name=e ₁ , v ₁ , v ₂ , eo=true) 2. Edge(Name=e ₁ , v _S =v ₁ , v _E =v ₂ , eo=true)
	Metavertex(Name=mv ₂ , v ₁ , v ₂ , v ₃ , Edge (Name=e ₁ , v ₁ , v ₂), Edge(Name=e ₂ , v ₂ , v ₃), Edge(Name=e ₃ , v ₁ , v ₃))

	Metavertex(Name=mv ₂ , v ₁ , v ₂ , v ₃ , Edge(Name=e ₁ , v _S =v ₁ , v _E =v ₂ , eo=true), Edge(Name=e ₂ , v _S =v ₂ , v _E =v ₃ , eo=true), Edge(Name=e ₃ , v _S =v ₁ , v _E =v ₃ , eo=true))
	Attribute(count, 5)
	Vertex(Name=v ₁ , Attribute(count, 5), Attribute(reference, mv ₂))

Metagraph-based approach for types description (1)

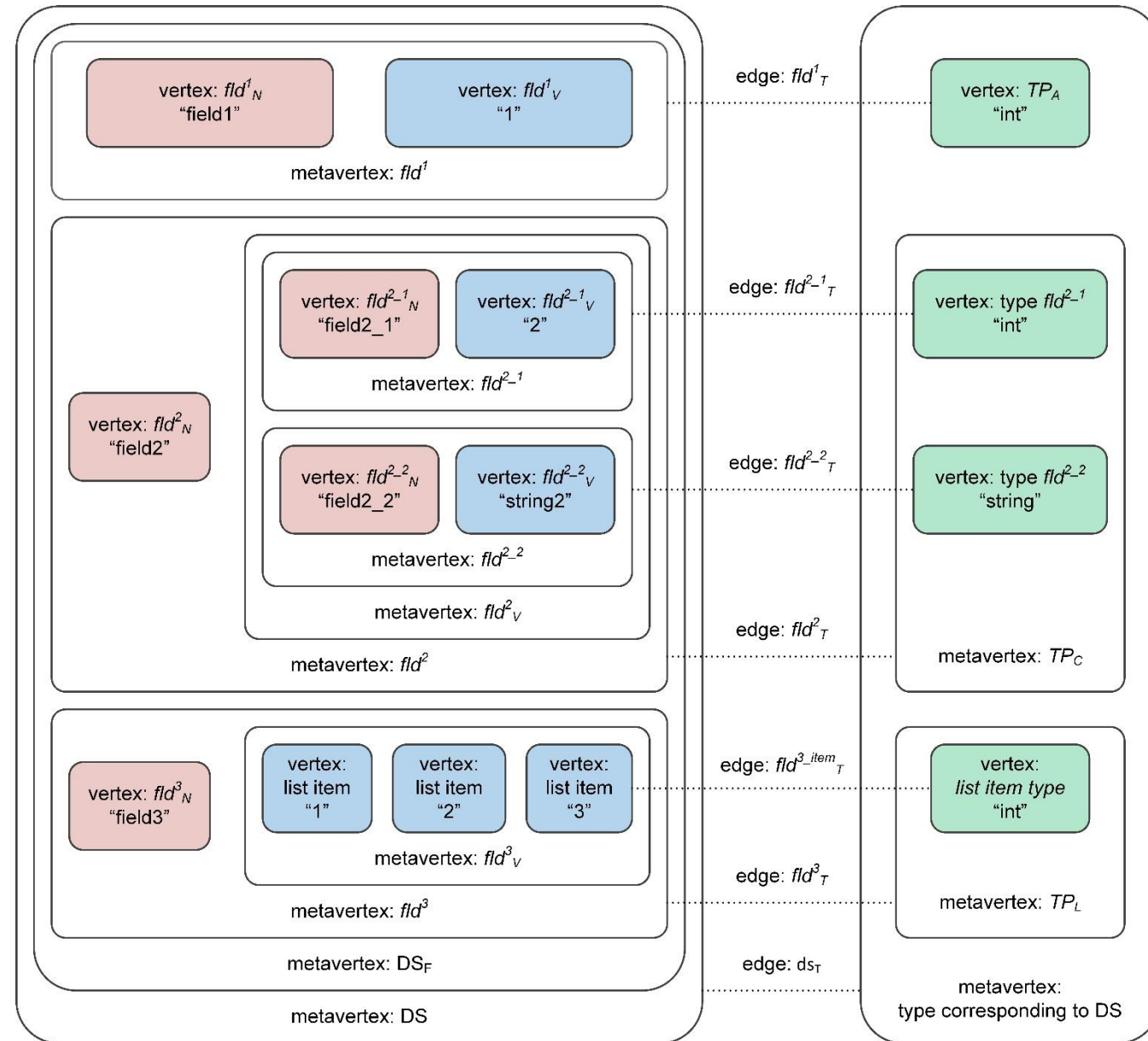
- According to the HOTT book: “the basic concept of type theory, that the term a is of type A , which is written: $a : A$. This expression is traditionally thought of as akin to: ‘ a is an element of the set A ’. However, in homotopy type theory we think of it instead as: ‘ a is a point of the space A ’.”
- From the point of view of the metagraph approach, a complex type is a metagraph description of a complex situation.
- When describing a complex situation, the relationships between elements and the nesting of elements are important, which is provided by the metagraph approach.
- Both type and instance may be represented as metagraph fragments.

Metagraph-based approach for types description (2)

- Consider data structures containing data fields in form *name:type:value* where type may be atomic type, complex type or list (collection) type.
- The data structure is defined as follows: $DS = \langle ds_T, DS_F \rangle$, $ds_T \in TP$, $DS_F = \{fld^i\}$, where DS – data structure; ds_T – data structure type belongs set of types TP ; DS_F – set of data structure fields fld^i .
- $fld^i = \langle fld_N, fld_T, fld_V \rangle$, $fld_T \in TP$, where fld_N – field name; fld_T – field type belongs set of types TP , fld_V – field value of type fld_T .
- $(\forall tp \in TP) tp = TP_A \mid TP_C = \{fld_T\} \mid TP_L = [TP]$. Every type tp belongs to set of types TP must be either atomic type TP_A or complex type TP_C or list (collection) type TP_L . The atomic type TP_A corresponds to the only value. The complex type TP_C contains set of corresponding field types fld_T . The list type TP_L is a collection of elements of any type.

Metagraph-based approach for types description (3)

- Data structure DS and its corresponding type are represented as a metaverices bound with edge ds_T . The set of data structure fields DS_F (also represented as a metavertex) consists of three fields fld^1 , fld^2 and fld^3 .
- Field fld^1 with name "field1" corresponds to the atomic type "int" with value "1". Field fld^1 is represented as a metavertex, field name fld^1_N and value fld^1_V are represented as inner vertices. Field type is represented as edge fld^1_T bound field metavertex with atomic type TP_A vertex.
- Field fld^2 with name "field2" corresponds to the complex type consists of fields "field2_1" of type "int" with value "2" and "field2_2" of type "string" with value "string2". Field fld^2 is represented as a metavertex, field name fld^2_N is represented as inner vertex and value fld^2_V is represented as inner metavertex containing metaverices fld^{2-1} and fld^{2-2} corresponding to subfields "field2_1" and "field2_2" with their values. Field fld^2 type is represented as edge fld^2_T bound field metavertex with complex type TP_C metavertex. The TP_C metavertex contains inner vertices corresponding to subfields fld^{2-1} and fld^{2-2} types. The edges fld^{2-1}_T and fld^{2-2}_T bound subfields fld^{2-1} and fld^{2-2} metaverices with corresponding subtypes vertices.
- Field fld^3 with name "field3" corresponds to the list (collection) type "list of int" with value "1,2,3". Field fld^3 is represented as a metavertex, field name fld^3_N is represented as inner vertex and value fld^3_V is represented as inner metavertex corresponding to the list containing vertices corresponding to the list items. Field type is represented as edge fld^3_T bound field metavertex with list (collection) type TP_L metavertex. The TP_L metavertex contains inner vertex corresponding to the list item type. List items bound with list item type with fld^{3-item}_T edge (shown only for list item "3" in order not to clutter the figure).
- The example shows that the object-oriented data structure may be represented using the metagraph approach without losing detailed information.



Conclusions

- From the point of view of the metagraph approach, a complex type is a metagraph description of a complex situation, in which not only the values of the vertices are important, but also the relationships between them. This makes the metagraph model related to the ontological knowledge model.
- When describing a complex situation, the relationships between elements and the nesting of elements are important, which is provided by the metagraph approach.
- Both type and instance may be represented as metagraph fragments.
- To process and transform metagraph data, the metagraph agents are used. The metagraph agent may be represented as a set of metagraph fragments. The distinguishing feature of the metagraph agent is its homoiconicity, which means that it can be data structure for itself.
- The combination of the metagraph data model and the metagraph agent model makes it possible to represent various type systems in the form of a metagraph model.

Thanks for your attention!
Any question?

gapyu@yandex.ru