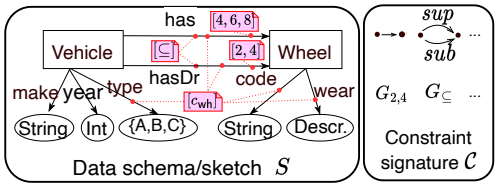
Cartesian institutions with evidence: Data and system modelling with diagrammatic constraints and generalized sketches

Zinovy Diskin
McMaster University, Canada
zdiskin@gmail.com

Constraints are fundamental for data modelling: they keep data integrity and ensure safety. Many constraint specification languages were developed, e.g., FOL and its fragments, the OCL (Object Constraint Language) widely used in the UML/EML software development ecosystem [7], or the diagrammatic language of lifting constraints [6] popular within the ACT community. These languages are successfully employed within their own ecosystems, but create severe interoperability problems when used in a heterogeneous environment [5]. Unification via XML solves the problem for only simple constraints and does not help when complex constraints modelling complex requirements appearing in system engineering (SE) are involved. In contrast, the *Generalized Sketch Framework (GSF)* can manage arbitrary constraints as soon as they have a specified *scope*: collection of elements over which the constraint is declared. This condition does hold for constraint languages used in the SE practice, and specifications in any of the languages above can be interpreted as *generalized sketches* (further just *sketches*).



The inset figure shows a small example of a data schema specified as a sketch S. It is a pair (G_S, C_S) of a graph G_S (with 7 nodes and 7 arrows, arrow hasDr points to the *driving* wheels of a vehicle), and a set C_S of constraint declarations (or just constraints) shown as dog-eared notes ((borrowed from the UML); each constraint has its scope shown with red dotted lines.[1] In detail, a *constraint declaration* is a pair $[c] = (c, \mathsf{b}_{[c]})$ of a *constraint name/symbol* c (e.g., $2,4$ or $\subseteq$) taken from a predefined signature $\mathscr{C}$, and a binding map $\mathsf{b}_{[c]}\colon G_c \to G_S$, where G_c is the *arity* graph of c (e.g., $G_\subseteq$ consists of two parallel arrows *sub* and *sup*, and $\mathsf{b}_{[\subseteq]}(sup) = \mathsf{has}$, $\mathsf{b}_{[\subseteq]}(sub) = \mathsf{hasDr}$). Each constraint name c has its predefined semantics: a class $[\![c]\!]$ of data instances over G_c considered *valid*, i.e., $[\![c]\!]$ is a class of graph morphisms into G_c. E.g., for constraint $\subseteq$, instance $t\colon G_t \to G_\subseteq$ is *valid* if $[\![sub]\!]^t \subseteq [\![sup]\!]^t$, where $[\![\cdot]\!]^t\colon G_\subseteq \to |\mathsf{Span}|$ is the graph morphism into the graph underlying category $\mathsf{Span} = \mathsf{Span}(\mathsf{Set})$ obtained by inverting mapping t. We say that an instance $t\colon G_t \to G_S$ over G_S *satisfies* a constraint declaration $[c]$ iff pulling t back along $\mathsf{b}_{[c]}$ results in $t\restriction_{\mathsf{b}_{[c]}} \in [\![c]\!]$; then we write $t \models [c]$. We will use the term 'constraint' for both constraint names and constraint declarations, which is ambiguous but follows the tradition; the ambiguity can always be resolved by the context.

Semantics of multiplicity constraints ($[2,4]$ *etc*) can be specified in a similar way by logical means, but constraint $[c_{\mathsf{wh}}]$ is essentially different. It is a very special customized constraint with a complex arity graph $G_{c_{\mathsf{wh}}}$ isomorphic to its image in G_S, and a complex custom semantics. To specify this semantics, we need to ensure that nodes and arrows in graph $G_{c_{\mathsf{wh}}}$ are interpreted as intended, to wit: we introduce

[1] Actually each of the 5 attribute arrows has an attached default multiplicity constraint [1] (each vehicle has exactly one make, year, *etc*). Moreover, oval nodes are constraints rather than node names, which prescribe these nodes to be interpreted by predefined sets (of strings, integers *etc*), Thus, there are 13 constraints in C_S, and later we will add five more.

Submitted to:
arxiv

constraints type, code, *etc*, whose arity is a single arrow so that, e.g., arrow named 'type' actually denotes a constraint [type] declared for an unnamed arrow. (This is exactly similar to oval nodes, whose names are names of the constraints declared for the underlying unnamed nodes.) This forces the arity graph $G_{c_{\mathsf{wh}}}$ to be a sketch (a graph with constraint declarations), and a correct binding mapping $\mathsf{b}_{[c_{\mathsf{wh}}]}\colon G_{c_{\mathsf{wh}}} \to G_S$ is to be a sketch morphism and map arrow 'type' in $G_{c_{\mathsf{wh}}}$ to arrow 'type' in G_S, and the same for other four arrows involved. Thus, constraint c_{wh} *depends* on constraints type, code, *etc* to be defined before c_{wh} is defined. This makes a signature of constraints a category with a hierarchical structure rather than a set.

An instance $[\![\cdot]\!]^t\colon G_{c_{\mathsf{wh}}} \to |\mathsf{Span}|$ is considered valid, if for each vehicle $v \in [\![\mathsf{Vehicle}]\!]^t$, the collection of its parameters, $\left(v.[\![\mathsf{type}]\!]^t, \ldots, (w.[\![\mathsf{code}]\!]^t)_{w\in v.[\![\mathsf{has}]\!]^t}, \ldots,\right)$ satisfies some predefined conditions specified in a normative document. Moreover, it is not excluded that the final decision whether a vehicle v satisfies the condition is delegated to an expert e, so that we should write $t \models^e [c_{\mathsf{wh}}]$ to assert t's validity.

Considering logical satisfaction $\models$ as as a ternary span of functors rather than a binary relation is the main novelty of the GSF developed in paper [2] (to be presented in the talk). Another new feature is considering indexed structures rather than subcategories, e.g., in the example above, if $\mathbb{G}$ denotes the category of graphs, and $\mathbb{G}/_{\mathscr{C}}G_S$ is the category of all possible $\mathscr{C}$-labelled binding maps into G_S, then constraints of sketch S are given by a general functor $\#_S\colon C_S \to \mathbb{G}/_{\mathscr{C}}G_S$ rather than embedding $C_S \hookrightarrow \mathbb{G}/_{\mathscr{C}}G_S$ (our previous treatment was simplified to explain the idea). Similarly, valid instances of a constraint c are indexed by functor $\mathsf{t}^c_{\_}\colon \mathit{Inst}^c_{\mathscr{C}} \to \mathbb{G}/_{\Delta}G_c$, which is not necessarily injective, where $\mathbb{G}/_{\Delta}G_c$ refers to a special slice category whose morphisms are spans of morphisms in $\mathbb{G}/G_c$, which model instance updates/deltas. Dealing with categories $\mathbb{G}/_{\Delta}G$ rather than $\mathbb{G}/G$ is another new feature of our sketches compared with [3].

Fig. 1 semi-formally presents the general idea behind the framework. Its nodes denote collections of objects named in them, and arrows are mappings. Three mappings provides an instance index with its three basic ingredients: its data schema providing types, its carrier structure (that can typically be modelled by a typed-graph structure), and (crucially in the context of SE and assurance) a structure of pieces of evidence supporting the claim that the instance conforms to the schema. Sketchy block arrows *Compatibility 1,2* refer to unspecified interconnections between the components.

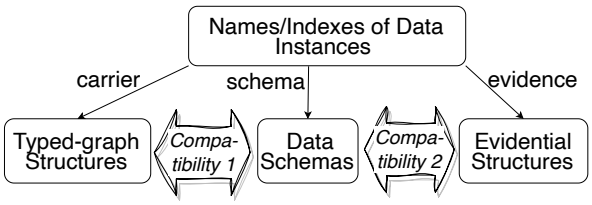


Fig. 1: Conceptual schema of data modelling, $\mathscr{M}$

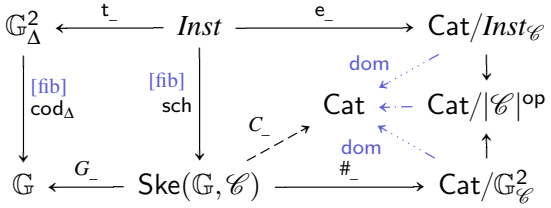


Fig. 2: $\mathscr{M}$ implemented vis sketches ($2 := \{\cdot \to \cdot\}$)

A major result [2, Th.2] states that any constraint signature $\mathscr{C}$ over a category $\mathbb{G}$ with pullbacks (providing arities for $\mathscr{C}$-constraints) gives rise to a commutative diagram in Cat specified in Fig. 2. In this diagram, $\mathsf{Ske}(\mathbb{G}, \mathscr{C})$ is the category of all $\mathscr{C}$-sketches, categories $\mathbb{G}^2_\Delta$ and $\mathbb{G}^2_{\mathscr{C}}$ are global counterparts of families $\mathbb{G}/_{\Delta}G$ and $\mathbb{G}/_{\mathscr{C}}G$ indexed by $G \in \mathsf{Ob}\mathbb{G}$, and pair $(\mathsf{t}_{\_}, G_{\_})$ is a fibrations morphism (which implements *Compatibility1* condition while the right-half subdiagram implements *Compatibility2*).

With a suitable formalization of schema $\mathscr{M}$ in Fig. 1(work in progress), the diagram in Fig. 2 would formally be an instance of $\mathscr{M}$. A version of this idea is realized for the indexed-category-based counterpart of fibrationally-defined $\mathscr{M}$: the notion of an *institution with evidence (e-institution)* is defined in [2], which enriches ordinary institutions[2] with a structure modelling evidence. The second major result [2, Th.1] states that any constraint signature $\mathscr{C}$ gives rise to an e-institution.

[2] introduced by Goguen and Burstall [4] to adapt so called abstract model theory [1] to computer science and programming

References

[1] John Barwise (1974): *Axioms for abstract model theory*. *Annals of Mathematical Logic* 7, pp. 221–265.

[2] Zinovy Diskin (2023): *Cartesian institutions with evidence: Data and system modelling with diagrammatic constraints and generalized sketches*. Available at `https://arxiv.org/abs/2306.16284`.

[3] Zinovy Diskin & Uwe Wolter (2008): *A Diagrammatic Logic for Object-Oriented Visual Modeling*. *Electr. Notes Theor. Comput. Sci.* 203(6), pp. 19–41. Available at `http://dx.doi.org/10.1016/j.entcs.2008.10.041`.

[4] J.A. Goguen & R.M. Burstall (1992): *Institutions: Abstract model theory for specification and programming*. *Journal of ACM* 39(1), pp. 95–146.

[5] Esther Guerra, Juan de Lara, Dimitrios S. Kolovos & Richard F. Paige (2010): *Inter-modelling: From Theory to Practice*. In: *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Lecture Notes in Computer Science* 6394, Springer, pp. 376–391, doi:10.1007/978-3-642-16145-2_26. Available at `https://doi.org/10.1007/978-3-642-16145-2_26`.

[6] David I. Spivak (2014): *Database queries and constraints via lifting problems*. *Mathematical Structures in Computer Science* 24(6), p. e240602, doi:10.1017/S0960129513000479.

[7] Jos Warmer & Anneke Kleppe (1999): *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.