Composing knowledge graphs, inside and out

Spencer Breiner[1]

spencer.breiner@nist.gov

Joint with Blake Pollard[1], Peter Denno[1]

and Eswaran Subrahmanian[1,2]

[1]NIST [2]CMU

March 18, 2020

About me (Spencer Breiner)

National Institute of Standards and Technology

- Information Technology Lab - Software & Systems Division
- Ph.D., CMU, 2013 - Category theory (CT) and logic

Current work: *Applied* CT for systems modeling

- Knowledge representation
- Knowledge integration
- Multiple semantics

Outline for today:

- Graphs & categories
- Why not (just) graphs?
- Knowledge graphs as categories and functors

What's beneath a knowledge graph?

Knowledge Graphs:

> *"structured representations of semantic*
>
> *knowledge that are stored in a graph"*

What structure? Stored how?

Today, some possible answers from category theory.

Some themes:

- Bite-size ontologies
- Data/concept duality
- Internalized computation/proof

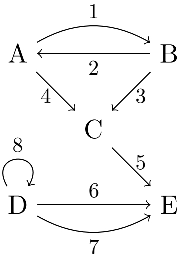
Graphs

For today, graphs are *directed* and (optionally) *multi*-.

Any graph can be represented as

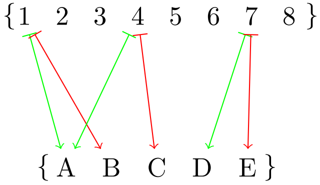
- A pair of sets $N = \texttt{Node}$ and $E = \texttt{Edge}$, and
- A pair of functions $s = \texttt{src}, t = \texttt{tgt} : E \rightrightarrows N$.

For example,



$\leftrightsquigarrow$

$$\begin{tikzcd} E \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] \\ N \end{tikzcd}$$



Categories

A category is a graph G together with

- Version 1: A partial associative operation (with identities)

$$\begin{array}{c} E \times E \\ \cup\!\mid \\ \{f.\mathtt{tgt} = g.\mathtt{src}\} \xrightarrow[f.g]{} E \end{array}$$

  Semantic categories: **Sets**, **Graph**, **Vect**, **Type**

- Version 2: A (concat-stable) equivalence relation over paths

$$\{\langle f_i \rangle \sim \langle g_j \rangle\} \subseteq \mathbf{Path}(G) \times \mathbf{Path}(G)$$

  Schemas: $\mathcal{G} = \langle E \rightrightarrows N \rangle$, $\mathcal{P} = \mathbf{OSProb}$, $\mathcal{S} = \mathbf{OSSoln}$

Free categories (!)

Upshot: Any graph G already "is" a category.

The relationship is mediated by a free/forgetful *adjunction*

$$\begin{tikzcd} \mathbf{Cat} \arrow[d, bend left, "\mathbf{Forget}"] \\ \mathbf{Graph} \arrow[u, bend left, "\mathbf{Free}"] \end{tikzcd} \qquad \begin{array}{c} \mathbf{Free}(G) \longrightarrow \mathbf{C} \\ \hline\hline G \longrightarrow \mathbf{Forget}(\mathbf{C}) \end{array} \quad \begin{array}{l} \in \mathbf{Cat} \\ \\ \in \mathbf{Graph} \end{array}$$

Two round trips:

A *monad* $\eta_G : G \to \mathbf{Path}(G)$ (concat)

A *comonad* $\epsilon_{\mathbf{C}} : \mathbf{Factor}(\mathbf{C}) \to \mathbf{C}$ (compute)

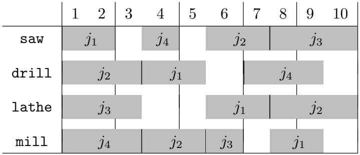
A bite-sized example

Open-shop scheduling

Problem

	Jobs	j_1	j_2	j_3	j_4
Machines	`saw`	2 hr	2 hr	2 hr	1 hr
	`drill`	2 hr	3 hr	0	3 hr
	`lathe`	2 hr	3 hr	3 hr	0
	`mill`	2 hr	2 hr	1 hr	3 hr

Schedule



Schematically,

$$\mathcal{P} = \langle \tau : J \times M \longrightarrow \mathbb{R}^+ \rangle \qquad \mathcal{S} = \langle s, t : J \times M \rightrightarrows \mathbb{R}^+ \mid \text{ax.} \rangle$$

The two are related via a functor

$$F : \mathcal{P} \longrightarrow \mathcal{S}$$
$$\tau \longmapsto t - s$$

Functorial semantics & duality

$F : \mathcal{P} \to \mathcal{S}$ encodes: "every schedule solves *some* problem."

Any concrete schedule (instance) defines a functor $P : \mathcal{P} \to \mathbf{Sets}$

Nodes map to sets:	$P(J) = \{j_1, j_2, j_3, j_4\}$
Edges map to functions:	$P(\tau) : (j_2, \texttt{lathe}) \mapsto 3$ hr

Every schema functor defines a *dual* transformation on instances

$$\begin{tikzcd} \mathbf{Inst}(\mathcal{S}) \arrow[r, "F^*"] & \mathbf{Inst}(\mathcal{P}) \\ S(s),\ S(t) \arrow[r, mapsto] & S(t){-}S(s) \end{tikzcd}$$

Duality is just precomposition:

$$\begin{tikzcd} \mathcal{S} \arrow[d, "S"'] & \mathcal{P} \arrow[l, "F"'] \arrow[d, "F^*(S)=F.S"] \\ \mathbf{Sets} \arrow[r, equal] & \mathbf{Sets} \end{tikzcd}$$

Why not (just) graphs?

- Structured nodes/edges: $J \times M$
- Built-in elements (libraries): $\texttt{diff} : \mathbb{R}^+ \times \mathbb{R}^+ \longrightarrow \mathbb{R}^+$
- First-class axioms/proofs: $s_{jm} \leq t_{jm} \;\vdash\; F(\tau)_{jm} \in \mathbb{R}^+$
- Not a graph homomorphism: $F(\tau) = \ell.p$

Structure in a category

The *Cartesian product* of two objects X and Y is a diagram $X \xleftarrow{\pi_1} P \xrightarrow{\pi_2} Y$ such that, for any object Z and any pair of arrows $x : Z \to X$ and $y : Z \to Y$, there exists a unique map $p = \langle x, y \rangle$ such that $p.\pi_1 = x$ and $p.\pi_2 = y$.

$$
\begin{tikzcd}
 & \forall Z \arrow[dl, "\forall x"'] \arrow[d, dashed, "\exists! p = \langle x,y \rangle"] \arrow[dr, "\forall y"] & \\
X & P \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & Y
\end{tikzcd}
$$

Generalized elements

A suggestive notation:

$$x : Z \to X \leftrightsquigarrow x \underset{Z}{\in} X$$

Compare:

In set theory	**In category theory**
$p \in X \times Y$	$p \underset{Z}{\in} X \times Y$
$x \in X,\ y \in Y,\ p = \langle x, y \rangle$	$x \underset{Z}{\in} X,\ y \underset{Z}{\in} Y,\ \underbrace{p = \langle x, y \rangle}_{p.\pi_1 = x,\ p.\pi_2 = y}$

Why generalize? In **Sets**, arrows $\{*\} \to X$ "see" everything in X, but...

In **Graph**, $\{*\}$ can't distinguish $\{* \quad *\}$ from $\{* \to *\}$.

In **Vect**, $\{*\} = \mathbb{R}^0$ can't see *anything* (zero object).

More structure

In programming, a function `f(x : X) : Y` is *pure* if

- It has no side effects (e.g., no IO, non-local variable mutation)
- It has consistent return values (e.g., no non-local variable dependence)

The pure fragment of a programming language defines a category **Type**.

The *exponential* adjunction mediates global/generalized elements

$$
\begin{tikzcd}
\mathbf{Type} \arrow[d, bend left, "(-)^Z"] \\
\mathbf{Type} \arrow[u, bend left, "-\times Z"]
\end{tikzcd}
\qquad
\begin{array}{rc}
f: & X \times Z \longrightarrow Y \\
\hline\hline
\ulcorner f(x,-) \urcorner : & X \longrightarrow Y^Z \\
\hline\hline
\ulcorner f \urcorner : & \{*\} \longrightarrow (Y^Z)^X \\
\hline\hline
\end{array}
$$

Round trips: $\texttt{eval}: \quad Y^Z \times Z \longrightarrow Y$

$\texttt{coeval}: \quad X \longrightarrow (X \times Z)^Z$

Types in a schema

We can think of schema libraries as

- A subschema $\mathcal{P}_0 \subseteq \mathcal{P}$, together with
- A fixed implementation functor $\mathbf{impl} : \mathcal{P}_0 \to \mathbf{Type}$

An instance should respect the behavior of the implementation:

$$\begin{tikzcd} \mathcal{P} \arrow[r, "P"] & \mathbf{Sets} \\ \mathcal{P}_0 \arrow[u, phantom, "\cup\mathrm{I}"] \arrow[r, "\mathbf{impl}"'] & \mathbf{Type} \arrow[u, "\mathbf{glob}=\mathrm{Hom}(\{*\},-)"'] \end{tikzcd}$$

Problem: We want $\tau \in \mathbb{R}^+$, but $s, t \in \mathbb{R}^+ \not\Rightarrow t - s \in \mathbb{R}^+$.

Logic in a schema

In general, formulas define subobjects, and inferences define sub-sub-objects:

$$\frac{\varphi(x) \vdash \psi(x)}{\begin{tikzcd} \llbracket\varphi\rrbracket \arrow[rr, dashed, tail] \arrow[dr, tail] & & \llbracket\psi\rrbracket \arrow[dl, tail] \\ & X & \end{tikzcd}}$$

Interpretations are defined recursively:

$x = y$	$\varphi \wedge \psi$	$\varphi \vee \psi$	$\exists y.\varphi$
Diagonal	Pullback	Pushout	Image

$$\begin{tikzcd} X \arrow[d, "\langle \mathrm{id}, \mathrm{id} \rangle"] \\ X \times X \end{tikzcd}
\qquad
\begin{tikzcd} \llbracket\varphi \wedge \psi\rrbracket \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \llbracket\psi\rrbracket \arrow[d] \\ \llbracket\varphi\rrbracket \arrow[r] & X \end{tikzcd}
\qquad
\begin{tikzcd} \llbracket\varphi \wedge \psi\rrbracket \arrow[r] \arrow[d] & \llbracket\psi\rrbracket \arrow[d] \\ \llbracket\varphi\rrbracket \arrow[r] & \llbracket\varphi \vee \psi\rrbracket \arrow[ul, phantom, "\ulcorner", very near start] \end{tikzcd}
\qquad
\begin{tikzcd} \llbracket\varphi\rrbracket \arrow[r, two heads] \arrow[d] & \llbracket\exists y.\varphi\rrbracket \arrow[d, tail] \\ X \times Y \arrow[r] & X \end{tikzcd}$$

Proofs as diagrams

Formulas are (sub)objects, inferences & proofs are arrows:

An axiom:

$$\vdash s_{jm} \leq t_{jm}$$

$$\begin{tikzcd} & \llbracket x \leq y \rrbracket \arrow[d, tail] \\ J \times M \arrow[ur, "\ell"] \arrow[r, "\langle s,t \rangle"'] & \mathbb{R}^+ \times \mathbb{R}^+ \end{tikzcd}$$

An inference:

$$x \leq y \ \vdash \ (y - x) \in \mathbb{R}^+$$

$$\begin{tikzcd} \llbracket x \leq y \rrbracket \arrow[r, "p"'] \arrow[d, tail] & \mathbb{R}^+ \arrow[d, tail] \\ \mathbb{R}^+ \times \mathbb{R}^+ \arrow[r, "\llbracket y-x \rrbracket"'] & \mathbb{R} \end{tikzcd}$$

The cut rule corresponds to concatenation of diagrams

e.g., $\vdash F(\tau)_{jm} \in \mathbb{R}^+$

Functors between graphs

Functors are more flexible than graph homomorphisms:

> Nodes map to nodes, but edges map to *paths*.

$$
\begin{tikzcd}
\mathcal{P} \arrow[d, "F"'] \\
\mathcal{S}
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
J \times M \arrow[rr, "\tau"] \arrow[d, mapsto] & & \mathbb{R}^+ \arrow[d, mapsto] \\
J \times M \arrow[r, "\ell"'] & \bullet \arrow[r, "p"'] & \mathbb{R}^+
\end{tikzcd}
$$

Usually interested in *structure-preserving* functors (instances, too!)

Constructions: $\quad F(J \times M) \cong F(J) \times F(M)$

Types:
$$
\begin{tikzcd}
\mathcal{P}_0 \arrow[rr, dashed, "F(\mathcal{P}_0)\subseteq\mathcal{S}_0"] \arrow[dr, "\mathbf{impl}"'] & & \mathcal{S}_0 \arrow[dl, "\mathbf{impl}"] \\
& \mathbf{Type} &
\end{tikzcd}
$$

Solutions as functors

Any solution algorithm a defines a matrix endomorphism

$$\begin{array}{ccc} (\mathbb{R}^+)^{J\times M} & \xrightarrow{\quad a \quad} & (\mathbb{R}^+)^{J\times M} \\ & & \\ (\tau_{jm}) & \longmapsto & (s_{jm}) \end{array}$$

From this, we can define a functor $A : \mathcal{S} \to \mathcal{P}$

$$A(s)_{jm} = \mathtt{eval}\big(a(\ulcorner\tau\urcorner), (j,m)\big)$$

$$A(t)_{jm} = A(s)_{jm} + \tau_{jm}$$

$$J \times M \xrightarrow{\langle \mathrm{id}, \ulcorner\tau\urcorner\rangle} (J\times M) \times (\mathbb{R}^+)^{J\times M} \xrightarrow{\mathrm{id}\times a} (J\times M)\times(\mathbb{R}^+)^{J\times M} \xrightarrow{\mathtt{eval}} \mathbb{R}^+$$

Defining A requires proof: a satisfies the axioms of $\mathcal{S}$.

Note: Equivalence $(\mathbb{R}^+)^{J\times M} \cong \mathrm{Mat}_{\mathbb{R}^+}(|J|, |M|)$ requires a *labeling*.

A knowledge "graph"

By duality, every problem $P \in \mathbf{Inst}(\mathcal{P})$ defines a solution $A^*(P) \in \mathbf{Inst}(\mathcal{S})$.

$$\begin{tikzcd} \mathcal{S} \arrow[d, bend left, "A"] \\ \mathcal{P} \arrow[u, bend left, "F"] \end{tikzcd} \qquad\qquad \begin{tikzcd} \mathbf{Inst}(\mathcal{S}) \arrow[d, bend right, "F^*"'] \\ \mathbf{Inst}(\mathcal{P}) \arrow[u, bend right, "A^*"'] \end{tikzcd}$$

The functors should satisfy $F.A = \mathrm{id}_{\mathcal{P}}$ ($\Rightarrow A^*.F^* = \mathrm{id}_{\mathbf{Inst}(\mathcal{P})}$):

$$\begin{aligned} A(F(\tau)) &= A(t - s) \\ &= A(t) - A(s) \\ &= A(s) + \tau - A(s) \\ &= \tau \end{aligned}$$

Variation I

What's the difference?

$$\mathcal{P}' := \left\langle \begin{tikzcd} & T \arrow[dl, "j"'] \arrow[dr, "m"] \arrow[rr, "\tau'"] & & \mathbb{R}^+ \\ J & & M & \end{tikzcd} \right\rangle \qquad \mathcal{S}' := \left\langle \begin{tikzcd} & T \arrow[dl, "j"'] \arrow[dr, "m"'] \arrow[rr, shift left, "s'"] \arrow[rr, shift right, "t'"'] & & \mathbb{R}^+ \\ J & & M & \end{tikzcd} \right\rangle$$

What's the same?

Problem generalization, functorially:

$$\begin{tikzcd} \mathcal{S}' \arrow[rr, dashed, "\exists!\ \overline{K}"] & & \mathcal{S} \\ \mathcal{P}' \arrow[u, "F'"] \arrow[rr, "K:T\mapsto J\times M"'] & & \mathcal{P} \arrow[u, "F"'] \end{tikzcd}$$

The other direction(s)?

Variation II

Duplicate machines (C=“capability”, a=“assignment”)

$$\mathcal{P}^d := \left\langle \begin{tikzcd} J \times C \arrow[r, "\tau^c"] & \mathbb{R}^+ \\ M \arrow[r, "c"'] & C \end{tikzcd} \right\rangle \qquad \mathcal{S}^d := \left\langle \begin{tikzcd} M \arrow[dd, "c"'] & & \\ & J \times C \arrow[ul, "a"'] \arrow[dl, "p_2"] \arrow[r, shift left, "s^c"] \arrow[r, shift right, "t^c"'] & \mathbb{R}^+ \\ C & & \end{tikzcd} \right\rangle$$

The arrow (bundle, dep. type) $M \to C$ represents a family of sets $\{M_c\}_{c \in C}$.

This time, we can go both ways (sort of)

$$\begin{tikzcd} \mathcal{S}^d \arrow[r, "\exists!\ \overline{I}:a\mapsto p_2"] & \mathcal{S} \arrow[r, "H:s,t\mapsto s|^a,t|^a"] & \mathcal{S}^d \\ \mathcal{P}^d \arrow[u, "F^d"] \arrow[r, "I:c\mapsto \mathrm{id}_M"'] & \mathcal{P} \arrow[u, "F"] \arrow[r, "G:M\mapsto C"'] \arrow[ur, phantom, "\times"] & \mathcal{P}^d \arrow[u, "F^d"'] \end{tikzcd}$$

Here $H : M \mapsto M$ and $s|^a, t|^a$ denote extension by zero along a.

Variation III

Duplicate jobs ($\mathcal{P}$=Process *catalog*, $\mathcal{O}$=Orders database)

$$\mathcal{O} := \left\langle \begin{tikzcd} C \arrow[d, bend right, "b"'] \arrow[d, "r"] & O \arrow[l, "c"'] \arrow[dl, "d"] & I \arrow[l, "o"'] \arrow[d, "j"] \\ \mathbb{R}^+ & & J \arrow[ll, "k"] \end{tikzcd} \right\rangle, \qquad b = r + \textstyle\sum_{o:O_c} \sum_{i:I_o} k(j(i))$$

Extract the daily schedule by mapping to a pushout:

$$\begin{tikzcd} \mathcal{P} \arrow[rr] & & \mathcal{O} \underset{J}{+} \mathcal{P} \\ J \times M \arrow[rr, mapsto] \arrow[dd, "\tau"'] & & I \times M \arrow[d, "j \times \mathrm{id}"] \\ & & J \times M \arrow[d, "\tau"] \\ \mathbb{R}^+ \arrow[rr, mapsto] & & \mathbb{R}^+ \end{tikzcd}$$

Variation IV

Duplicate jobs and machines ($\mathcal{M}$=Shop floor model)

$$
\begin{tikzcd}
 & & & \mathcal{S}^{co} := \mathcal{S}^c \underset{\mathcal{P}^c}{+} \mathcal{P}^{co} \arrow[dl, phantom, "\lrcorner"] & & \\
\mathcal{S}^c \arrow[urrr, bend left, "\overline{D^c}"] & & & \mathcal{P}^{co} := \mathcal{M} \underset{C_{\mathcal{M}}}{+} \mathcal{P} \underset{J}{+} \mathcal{O} \arrow[u] \arrow[d, phantom, "\vee"] & & \\
\mathcal{P}^c \arrow[u, "F^c"] \arrow[urrr, bend left, "D^c : J \mapsto I"] \arrow[rr] & & \mathcal{M} \underset{C}{+} \mathcal{P} \arrow[ur, "C \mapsto C_{\mathcal{M}}"] \arrow[d, phantom, "\vee"] & & \mathcal{P} \underset{J}{+} \mathcal{O} \arrow[ul, "C \mapsto C_{\mathcal{O}}"'] \arrow[d, phantom, "\vee"] & \\
 & \mathcal{M} \arrow[ur] & & \mathcal{P} \arrow[ul, "M \mapsto C"'] \arrow[ur] & & \mathcal{O} \arrow[ul] \\
 & & \{C\} \arrow[ul] \arrow[ur, "C \mapsto M"] & & \{J\} \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

Wrapping Up

Recap:

- Bite-sized semantic models & functorial instances
- Built-in logic & computation via (preservation of) structure
- Knowledge graphs as schemas & functors.

More goodies:

- Build-your-own semantics (presheaves)
- Internal concepts generate external schemas (Yoneda/slice cat.)
- Relationships between relationships (Natural transformations)

The bad news...

- Limited tooling
- Steep learning curve

Thank you!

PS. This talk is based on a paper under review, but a draft is available on request from `spencer.breiner@nist.gov`.