Algebraic Property Graphs

Joshua Shinavier[1] Ryan Wisnesky[2] Joshua G. Meyers[2]

[1]Uber Technologies, Inc.[1]

[2]Conexus AI

July 19, 2022

[1]For the toolkit, applications, and original formulation of APG. The author's current affiliation is LinkedIn Corporation.

Overview

Abstract: We present a case study in applied category theory written from the point of view of an applied domain: the formalization of the widely-used *property graphs* data model in an enterprise setting using elementary constructions from type theory and category theory, including limit and co-limit sketches. Observing that *algebraic data types* are a common foundation of most of the enterprise schema languages we deal with in practice, for graph data or otherwise, we introduce a type theory for *algebraic* property graphs wherein the types denote both algebraic data types in the sense of functional programming and join-union E/R diagrams in the sense of database theory. We also provide theoretical foundations for graph transformation along schema mappings with by-construction guarantees of semantic consistency. Our data model originated as a formalization of a data integration toolkit developed at Uber which carries data and schemas along composable mappings between data interchange languages such as Apache Avro, Apache Thrift, and Protocol Buffers, and graph languages including RDF with OWL or SHACL-based schemas.

Design considerations:

- Formalize the labeled property graph data model
- Formalize data interchange languages including Protobuf, Thrift, and Avro
- Serve as a standard, unifying data model for Uber
- Serve as a bridge between graph and non-graph data sources

Applications:

- Basis for reusable, standardized data types
- Mapping of common types into multiple data languages
- Mapping of schemas and data across languages
- Language-agnostic schema and data validation
- Graph (PG and RDF) construction

Graph data models

- Property Graphs: edge-labeled and vertex-labeled graphs with properties. See Apache TinkerPop and graph databases such as Neo4j
- RDF: W3C standard graph data model. Graphs are sets of subject/predicate/object triples. Various schema formalisms (RDFS, OWL, SHACL, etc.)
- Hypergraphs: variations on the mathematical concept of a hypergraph. Less common than property graphs, but more compatible with relational databases

Property graph elements

- Vertex, edge: as in a classic labeled, directed multigraph
- Vertex property, edge property: key/value pair attached to a vertex or edge
- Metaproperty: property attached to another property
- etc.

Algebraic data types in practice

	Apache Thrift	Apache Avro	Protocol Buffers v3	GraphQL SDL
product types	yes	yes	yes	yes
sum types	yes	yes	yes	yes
interfaces				yes
enumerations	yes	yes	yes	yes
optionals	yes	yes		yes
typedefs	yes			
defaults	yes	yes	yes	yes
constants	yes			
lists/arrays	yes	yes	yes	yes
maps	yes	yes	yes	
sets	yes			

Figure: Comparison of selected data exchange languages

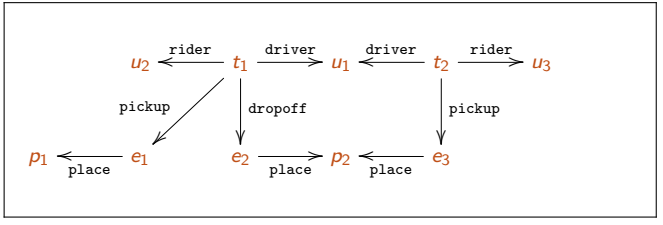
APGs: Definition 1

Fix a set P of primitive types p with extensions $\mathsf{PV}(p)$.

An *APG schema* S consists of:

- $\mathcal{L} : \textbf{Set}$
- $\sigma : \mathcal{L} \to \mathcal{T}$, where

$$\mathcal{T} \ni t \ ::= \ 0 \mid 1 \mid t_1 + t_2 \mid t_1 \times t_2 \mid \mathsf{Prim}\ p\ (p \in \mathsf{P}) \mid \mathsf{Lbl}\ l\ (l \in \mathcal{L})$$



Let $\mathsf{P} := \{\texttt{Integer}\}$, $\mathsf{PV}(\texttt{Integer}) = \mathbb{Z}$.

Example schema $S = (\mathcal{L}, \sigma)$:

$$\begin{aligned}
\mathcal{L} &:= \{\texttt{User}, \texttt{Trip}, \texttt{PlaceEvent}, \texttt{Place}\} \\
\sigma(\texttt{User}) &:= \sigma(\texttt{Place}) := 1 \\
\sigma(\texttt{PlaceEvent}) &:= \texttt{Place} \times \texttt{Integer} \\
\sigma(\texttt{Trip}) &:= \texttt{User} \times \texttt{User} \\
&\quad \times (1 + \texttt{PlaceEvent}) \times (1 + \texttt{PlaceEvent})
\end{aligned}$$

Given an APG schema $S = (\mathcal{L}, \sigma)$, an *APG on* S consists of:

- $E : \mathcal{L} \to \textbf{Set}$
- For each $l \in \mathcal{L}$, a function $v_l : E(l) \to V(\sigma(l))$, where $V : \mathcal{T} \to \textbf{Set}$ is defined as follows:

$$\begin{aligned}
V(0) &:= 0 \quad V(1) := 1 \\
V(\mathsf{Prim}\ p) &:= \mathsf{PV}(p) \quad V(\mathsf{Lbl}\ l) := E(l) \\
V(t_1 + t_2) &:= V(t_1) + V(t_2) \\
V(t_1 \times t_2) &:= V(t_1) \times V(t_2)
\end{aligned}$$

Example APG on S:

$$\begin{aligned}
E(\texttt{User}) &:= \{u_1, u_2, u_3\} \\
E(\texttt{Trip}) &:= \{t_1, t_2\} \\
E(\texttt{PlaceEvent}) &:= \{e_1, e_2, e_3\} \\
E(\texttt{Place}) &:= \{p_1, p_2, p_3\} \\
v_{\texttt{User}}(u_1) &:= v_{\texttt{User}}(u_2) := v_{\texttt{User}}(u_3) := () \\
v_{\texttt{Place}}(p_1) &:= v_{\texttt{Place}}(p_2) := v_{\texttt{Place}}(p_3) := () \\
v_{\texttt{Trip}}(t_1) &:= (u_1, u_2, \mathsf{inr}(e_1), \mathsf{inr}(e_2)) \\
v_{\texttt{Trip}}(t_2) &:= (u_1, u_3, \mathsf{inr}(e_3), \mathsf{inl}(())) \\
v_{\texttt{PlaceEvent}}(e_1) &:= (p_1, 1602203601) \\
v_{\texttt{PlaceEvent}}(e_2) &:= (p_2, 1602203948) \\
v_{\texttt{PlaceEvent}}(e_3) &:= (p_2, 1602204122)
\end{aligned}$$

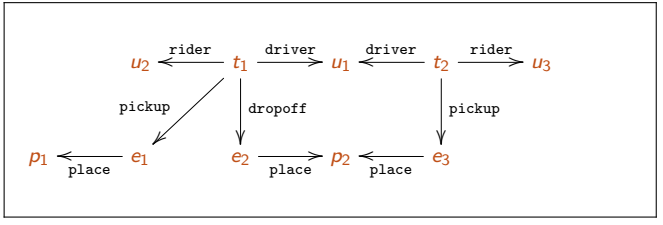
APGs: Definition 2 (type-theoretic flavor)

Fix a set P of primitive types p with extensions $\mathsf{PV}(p)$.

An *APG schema* S consists of:

- $\mathcal{L}$: **Set**
- $\sigma : \mathcal{L} \to \mathcal{T}$, where

$$\mathcal{T} \ni t \ ::= \ 0 \mid 1 \mid t_1 + t_2 \mid t_1 \times t_2 \mid \mathsf{Prim}\ p\ (p \in \mathsf{P}) \mid \mathsf{Lbl}\ l\ (l \in \mathcal{L})$$



Let $\mathsf{P} := \{\texttt{Integer}\}$, $\mathsf{PV}(\texttt{Integer}) = \mathbb{Z}$.

Example schema $S = (\mathcal{L}, \sigma)$:

$$\begin{aligned}
\mathcal{L} &:= \{\texttt{User}, \texttt{Trip}, \texttt{PlaceEvent}, \texttt{Place}\} \\
\sigma(\texttt{User}) &:= \sigma(\texttt{Place}) := 1 \\
\sigma(\texttt{PlaceEvent}) &:= \texttt{Place} \times \texttt{Integer} \\
\sigma(\texttt{Trip}) &:= \texttt{User} \times \texttt{User} \\
&\quad \times (1 + \texttt{PlaceEvent}) \times (1 + \texttt{PlaceEvent})
\end{aligned}$$

Given an APG schema $S = (\mathcal{L}, \sigma)$, an *APG on* S consists of:

- $\mathcal{E}$: **Set**
- $\lambda : \mathcal{E} \to \mathcal{L}$
- $v : \mathcal{E} \to \mathcal{V}$ where

$$\begin{aligned}
\mathcal{V} \ni (v : t) \ ::= \ & () : 1 \mid \mathsf{inl}_{t_2}(v{:}t_1) : t_1 + t_2 \\
& \mid \mathsf{inr}_{t_1}(v{:}t_2) : t_1 + t_2 \\
& \mid (v_1{:}t_1, v_2{:}t_2) : t_1 \times t_2 \\
& \mid \mathsf{Prim}\ v_t : t \quad (t \in \mathsf{P}, v \in \mathsf{PV}(p)) \\
& \mid \mathsf{Elmt}\ e : \lambda(e) \quad (e \in \mathcal{E})
\end{aligned}$$

such that, where $\tau : \mathcal{V} \to \mathcal{T}$ is defined by by $\tau(v : t) := t$,

$$\begin{array}{ccc}
\mathcal{E} & \xrightarrow{\lambda} & \mathcal{L} \\
{\scriptstyle v}\downarrow & & \downarrow{\scriptstyle \sigma} \\
\mathcal{V} & \xrightarrow{\tau} & \mathcal{T}.
\end{array}$$

Example APG on S:

$$\begin{aligned}
\mathcal{E} &:= \{u_1, u_2, u_3, t_1, t_2, e_1, e_2, e_3, p_1, p_2, p_3\} \\
\lambda(u_1) &:= \lambda(u_2) := \lambda(u_3) := \texttt{User} \quad \lambda(t_1) := \lambda(t_2) := \texttt{Trip} \\
\lambda(p_1) &:= \lambda(p_2) := \lambda(p_3) := \texttt{Place} \\
\lambda(e_1) &:= \lambda(e_2) := \lambda(e_3) := \texttt{PlaceEvent} \\
v(u_1) &:= v(u_2) := v(u_3) := v(p_1) := v(p_2) := v(p_3) := () \\
v(t_1) &:= (u_1, u_2, \mathsf{inr}(e_1), \mathsf{inr}(e_2)) \\
v(t_2) &:= (u_1, u_3, \mathsf{inr}(e_3), \mathsf{inl}(())) \\
v(e_1) &:= (p_1, 1602203601) \quad v(e_2) := (p_2, 1602203948) \\
v(e_3) &:= (p_2, 1602204122)
\end{aligned}$$

APGs: Definition 3: APGs as Coalgebras

A *APG schema* S consists of:

- $\mathcal{L} : \mathbf{Set}$
- a polynomial functor $F : \mathbf{Set}^{\mathcal{L}} \to \mathbf{Set}^{\mathcal{L}}$

(This definition includes non-computable schemas, but it is mathematically very nice.)

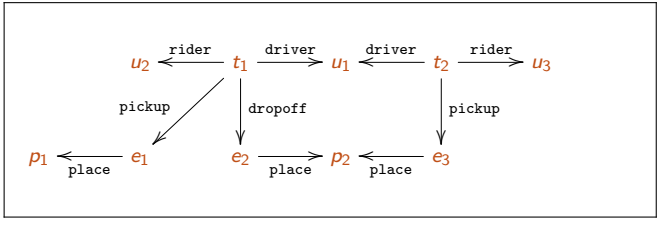
Given an APG schema $S = (\mathcal{L}, F)$, an *APG on* S consists of:

- $E : \mathcal{L} \to \mathrm{Set}$
- $\upsilon : E \Rightarrow F(E)$

In other words, it is a coalgebra of F.

(This definition includes non-computable APGs, but it is mathematically very nice.)

Now we can define morphisms of APGs on S simply as morphisms of coalgebras of F.



Example schema $S = (\mathcal{L}, F)$:

$$\mathcal{L} := \{\mathtt{User}, \mathtt{Trip}, \mathtt{PlaceEvent}, \mathtt{Place}\}$$

$$F\left(\begin{bmatrix} \mathtt{User} \mapsto E_{\mathtt{User}} \\ \mathtt{Trip} \mapsto E_{\mathtt{Trip}} \\ \mathtt{PlaceEvent} \mapsto E_{\mathtt{PlaceEvent}} \\ \mathtt{Place} \mapsto E_{\mathtt{Place}} \end{bmatrix}\right) :=$$

$$\begin{bmatrix} \mathtt{User} \mapsto 1 \\ \mathtt{Trip} \mapsto 1 \\ \mathtt{PlaceEvent} \mapsto E_{\mathtt{Place}} \times \mathbb{Z} \\ \mathtt{Place} \mapsto E_{\mathtt{User}} \times E_{\mathtt{User}} \times (1 + E_{\mathtt{PlaceEvent}}) \times (1 + E_{\mathtt{PlaceEvent}}) \end{bmatrix}$$

Example APG on S:

$$\begin{aligned}
E(\mathtt{User}) &:= \{u_1, u_2, u_3\} \\
E(\mathtt{Trip}) &:= \{t_1, t_2\} \\
E(\mathtt{PlaceEvent}) &:= \{e_1, e_2, e_3\} \\
E(\mathtt{Place}) &:= \{p_1, p_2, p_3\} \\
\upsilon_{\mathtt{User}}(u_1) &:= \upsilon_{\mathtt{User}}(u_2) := \upsilon_{\mathtt{User}}(u_3) := () \\
\upsilon_{\mathtt{Place}}(p_1) &:= \upsilon_{\mathtt{Place}}(p_2) := \upsilon_{\mathtt{Place}}(p_3) := () \\
\upsilon_{\mathtt{Trip}}(t_1) &:= (u_1, u_2, \mathrm{inr}(e_1), \mathrm{inr}(e_2)) \\
\upsilon_{\mathtt{Trip}}(t_2) &:= (u_1, u_3, \mathrm{inr}(e_3), \mathrm{inl}(())) \\
\upsilon_{\mathtt{PlaceEvent}}(e_1) &:= (p_1, 1602203601) \\
\upsilon_{\mathtt{PlaceEvent}}(e_2) &:= (p_2, 1602203948) \\
\upsilon_{\mathtt{PlaceEvent}}(e_3) &:= (p_2, 1602204122)
\end{aligned}$$

APGs: Definition 4: APGs as models

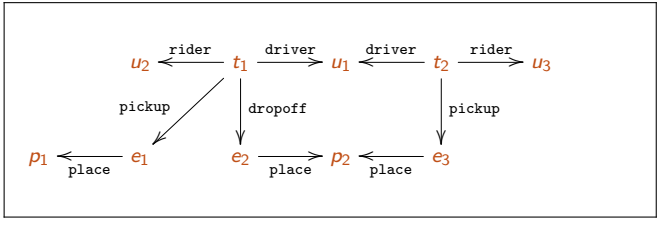
Let $S = (\mathcal{L} : \mathbf{Set}, \sigma : \mathcal{L} \to \mathcal{T})$ be an APG schema.

The *APG theory* C_S corresponding to S is the free category C_S with terminal and initial object, products, and coproducts, on the following generators:[2]

$$\frac{l \in \mathcal{L}}{(\text{Lbl } l) \in C_S} \qquad \frac{p \in \mathsf{P}}{(\text{Prim } p) \in C_S} \qquad \frac{l \in \mathcal{L}}{\delta_l : l \to \sigma(l)}$$

Note that the objects of C_S are exactly $\mathcal{T}$.

(We can also extend APGs by adding more generators to C_S, e.g. user-defined functions and constraints.)



Let $\mathsf{P} := \{\texttt{Integer}\}$, $\mathsf{PV}(\texttt{Integer}) = \mathbb{Z}$.

Example schema $S = (\mathcal{L}, \sigma)$:

$$\begin{aligned}
\mathcal{L} &:= \{\texttt{User}, \texttt{Trip}, \texttt{PlaceEvent}, \texttt{Place}\} \\
\sigma(\texttt{User}) &:= \sigma(\texttt{Place}) := 1 \\
\sigma(\texttt{PlaceEvent}) &:= \texttt{Place} \times \texttt{Integer} \\
\sigma(\texttt{Trip}) &:= \texttt{User} \times \texttt{User} \\
&\quad \times (1 + \texttt{PlaceEvent}) \times (1 + \texttt{PlaceEvent})
\end{aligned}$$

Given an APG schema C_S, a *APG on S* is a model of C_S, that is, a functor $V : C_S \to \mathbf{Set}$ which preserves products, coproducts, terminal object, and initial object, and which sends $\text{Prim } p \mapsto \mathsf{PV}(p)$.

Note that V is determined completely by where it sends objects of the form $\text{Lbl } l$ and morphisms of the form δ_l.

(This definition includes non-computable APGs, but it is mathematically very nice.)

Now we can define morphisms of APGs V and V' on S as natural transformations $\eta : V \Rightarrow V'$ such that

$$\eta_{\text{Prim } p} = \text{id}_{\text{Prim } p} \qquad \eta_{t+t'} = \eta_t + \eta_{t'} \qquad \eta_{t \times t'} = \eta_t \times \eta_{t'}$$

Example APG on S:

$$\begin{aligned}
V(\text{Lbl } \texttt{User}) &:= \{u_1, u_2, u_3\} \\
V(\text{Lbl } \texttt{Trip}) &:= \{t_1, t_2\} \\
V(\text{Lbl } \texttt{PlaceEvent}) &:= \{e_1, e_2, e_3\} \\
V(\text{Lbl } \texttt{Place}) &:= \{p_1, p_2, p_3\} \\
V(\delta_{\texttt{User}}) &:= \begin{bmatrix} u_1 \mapsto () \\ u_2 \mapsto () \\ u_3 \mapsto () \end{bmatrix} \qquad V(\delta_{\texttt{Place}}) := \begin{bmatrix} p_1 \mapsto () \\ p_2 \mapsto () \\ p_3 \mapsto () \end{bmatrix} \\
V(\delta_{\texttt{Trip}}) &:= \begin{bmatrix} t_1 \mapsto (u_1, u_2, \text{inr}(e_1), \text{inr}(e_2)) \\ t_2 \mapsto (u_1, u_3, \text{inr}(e_3), \text{inl}(())) \end{bmatrix} \\
V(\delta_{\texttt{PlaceEvent}}) &:= \begin{bmatrix} e_1 \mapsto (p_1, 1602203601) \\ e_2 \mapsto (p_2, 1602203948) \\ e_3 \mapsto (p_2, 1602204122) \end{bmatrix}
\end{aligned}$$

[2] See Cockett and Seely 2001 for the details of this construction. We also could choose to require that in C_S, the canonical maps $t \times u + t \times v \to t \times (u + v)$ and $t \to t + 0$ are isomorphisms, making C_S a distributive category. This has no effect on the models of C_S, since Set is distributive, but it does afford additional morphisms of theories (see next slide). The downside is that it complicates term rewriting.

Morphisms of APG Theories and Schemas

Morphisms of APG theories

A morphism of APG theories C_S and $C_{S'}$ is defined as a functor $\Phi : C_S \to C_{S'}$ preserving products, coproducts, terminal object, initial object, and primitive types.

Now we have a functor $\Delta : \mathbf{Theory}^{\mathrm{op}} \to \mathbf{Cat}$, where **Theory** is the category of APG theories, defined by:

- $\Delta(C_S) := S\text{-}\mathbf{APG}$, where S-**APG** is the category of APGs on S
- $\Delta(\Phi) := (V \mapsto V \circ \Phi)$

Simple example

Source schema $S = (\mathcal{L}, \boldsymbol{\sigma})$:

- $\mathcal{L} := \{l\}$, $\boldsymbol{\sigma}(l) := \texttt{String} \times \texttt{Nat} \times \texttt{Integer}$

Target schema $S' = (\mathcal{L}', \boldsymbol{\sigma}')$:

- $\mathcal{L} := \{l'\}$, $\boldsymbol{\sigma}'(l') := \texttt{Nat} \times \texttt{String}$

Theory mapping $\Phi : C_S \to C_{S'}$:

- $\Phi(\mathrm{Lbl})\ l := \mathrm{Lbl}\ l'$
- $\Phi(\delta_l) := l' \xrightarrow{\delta_{l'}} \texttt{Nat} \times \texttt{String} \xrightarrow{\langle \mathrm{snd}, \mathrm{fst}, 42 \circ ! \rangle} \texttt{String} \times \texttt{Nat} \times \texttt{Integer}$

(Here we assume a user-defined function $42 : 1 \to \texttt{String}$ sending $() \mapsto 42$).

The functor $\Delta(\Phi)$ converts APGs on S' to schema S by permuting projections and adding 42.

Morphisms of APG schemas

A morphism of schemas $(\mathcal{L}, F)$ and $(\mathcal{L}', F')$ is a pair $(\Phi : \mathrm{Set}^{\mathcal{L}'} \to \mathrm{Set}^{\mathcal{L}}$ polynomial functor, $\phi : \Phi F_{S'} \Rightarrow F_S \Phi)$, as shown:

$$\begin{array}{ccc} \mathrm{Set}^{\mathcal{L}'} & \xrightarrow{F_{S'}} & \mathrm{Set}^{\mathcal{L}'} \\ \downarrow \Phi & \swarrow \phi & \downarrow \Phi \\ \mathrm{Set}^{\mathcal{L}} & \xrightarrow{F_S} & \mathrm{Set}^{\mathcal{L}} \end{array}$$

Now we have a functor $\Delta : \mathbf{Schema}^{\mathrm{op}} \to \mathbf{Cat}$, where **Schema** is the category of APG schemas, defined by:

- $\Delta(C_S) := S\text{-}\mathbf{APG}$, where S-**APG** is the category of APGs on S
- $\Delta(\Phi, \phi) := ((E' : \mathcal{L}' \to \mathbf{Set}, \upsilon' : E' \Rightarrow F'(E')$
  $\mapsto \left(\Phi(E'), \left(\Phi(E') \xrightarrow{\Phi(\upsilon')} \Phi F'(E') \xrightarrow{\phi_{E'}} F\Phi(E')\right)\right)$

These morphisms can be translated to morphisms of the corresponding theories by a functor $C : \mathbf{Schema} \to \mathbf{Theory}$ sending $S \mapsto C_S$ and satisfying

$$\begin{array}{ccc} \mathbf{Schema}^{\mathrm{op}} & \xrightarrow{C} & \mathbf{Theory}^{\mathrm{op}} \\ & {\scriptstyle\Delta} \searrow \quad \swarrow {\scriptstyle\Delta} & \\ & \mathbf{Cat} & \end{array}$$

Though faithful, C is not full, making morphisms of APG theories strictly more general than morphisms of APG schemas!

In the paper on arXiv, there is a conjecture on how morphisms of APG schemas can be generalized in order to extend this functor to an equivalence.

Stream Example

Streams can be considered as APGs on the schema $S = (\mathcal{L}, \sigma)$:

- $\mathcal{L} := \{\mathtt{A}\}$
- $\sigma(\mathtt{A}) := \mathtt{A} + 1$

Finite stream:

- $E(\mathtt{A}) := \{e_1, e_2, e_3, \ldots, e_{10}\}$
- $\upsilon(e_n) := \mathsf{inl}(e_{n+1})$ for $n = 1, \ldots, 9$
- $\upsilon(e_{10}) := \mathsf{inr}(())$

Infinite stream:

- $E(\mathtt{A}) := \{e_1, e_2, e_3, \ldots, e_{10}\}$
- $\upsilon(e_n) := \mathsf{inl}(e_{n+1})$ for $n = 1, \ldots, 9$
- $\upsilon(e_{10}) := \mathsf{inl}(e_4)$

A morphism from the theory C_S to itself (assume C_S is taken as distributive).

$$\Phi(A) := A \times A$$

$$\Phi(\delta_A) : A \times A \to A \times A + 1$$

$$\begin{aligned}\Phi(\delta_A) := \Big(A \times A &\xrightarrow{\delta_A \times \delta_A} (A+1) \times (A+1) \\ &\xrightarrow{(\delta_A + 1) \times \mathsf{id}} ((A+1)+1) \times (A+1) \\ &\xrightarrow{\mathrm{dist},\mathrm{assoc}} A \times A + (1+1) \times A + A \times 1 + (1+1) \times 1 \\ &\xrightarrow{[\mathsf{id},!,!,!]} A \times A + 1\Big)\end{aligned}$$

We apply this morphism to the finite stream (E, υ) to obtain the stream $\Delta(\Phi)(E, \upsilon) =: (E', \upsilon')$:

$$E'(A) := \{(e_i, e_j) \mid i, j = 1, \ldots, 10\}$$

$$\upsilon'((e_i, e_j)) := \mathsf{inl}((e_{i+2}, e_{j+1})) \text{ for } i = 1, \ldots, 8 \text{ and } j = 1, \ldots, 9$$

$$\upsilon'((e_i, e_{10})) := \mathsf{inr}(()) \text{ for } i = 1, \ldots, 10$$

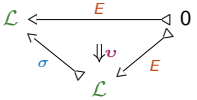
$$\upsilon'((e_i, e_j)) := \mathsf{inr}(()) \text{ for } i = 9, 10 \text{ and } j = 1, \ldots, 10$$

We can say that $\Delta(\Phi)$ converts a stream into a new stream whose elements are ordered pairs of elements of the old stream, and where the first element advances twice as fast as in the old stream, and the second element advances just as in the old stream.

Generalized APGs (GAPGs)

In the language of David Spivak (cf. *The Poly Book*), polynomial functors $\mathbf{Set}^{\mathcal{L}} \to \mathbf{Set}^{\mathcal{L}}$ can be described as bicomodules in Poly from a discrete category (aka comonoid) to itself: $\mathcal{L}y \xleftarrow{\sigma} \mathcal{L}y$.

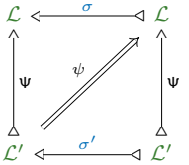
We immediately generalize to the case of an arbitrary category $\mathcal{L}$ of labels, to obtain a *generalized APG (GAPG) schema*, defined as a category $\mathcal{L}$ and a bicomodule (aka prafunctor) $\mathcal{L} \xleftarrow{\sigma} \mathcal{L}$. A *GAPG* on the schema $(\mathcal{L}, \sigma)$ is then an ordered pair (E, υ) where E is a copresheaf on $\mathcal{L}$, aka a bicomodule $\mathcal{L} \xleftarrow{E} 0$, and υ is a bicomodule homomorphism:



Use cases and implementation possibilities of GAPGs have yet to be explored.

Morphisms of GAPG schemas

A schema morphism $(\mathcal{L}, \sigma) \to (\mathcal{L}', \sigma')$ is an ordered pair (Ψ, ψ):



Then the schema morphism (Ψ, ψ) can be applied to a GAPG (E', υ') on $(\mathcal{L}', \sigma')$ in a straightforward way: