Presenting Profunctors

Gabriel Goren Roig[1,2] Emilio Minichiello[3] Joshua Meyers[3]

[1]Departamento de Matemática, Universidad de Buenos Aires, Argentina

[2]Instituto de Ciencias de la Computación (ICC), CONICET

[3]Conexus AI

Applied Category Theory 2024

Introduction

- To compute with categories we need syntax.

Introduction

- To compute with categories we need syntax.
- There is a canonical way to present small categories by generators and relations (and morphisms between them).

Introduction

- To compute with categories we need syntax.
- There is a canonical way to present small categories by generators and relations (and morphisms between them).
- There is a canonical way to present **Set**-valued functors.

Introduction

- To compute with categories we need syntax.
- There is a canonical way to present small categories by generators and relations (and morphisms between them).
- There is a canonical way to present **Set**-valued functors.
- But what about profunctors?

- To compute with categories we need syntax.
- There is a canonical way to present small categories by generators and relations (and morphisms between them).
- There is a canonical way to present **Set**-valued functors.
- But what about profunctors?

A profunctor $\mathcal{P} : \mathcal{C} \nrightarrow \mathcal{D}$ is either

$$\mathcal{C}^{\mathsf{op}} \times \mathcal{D} \to \mathbf{Set} \qquad \text{or} \qquad \mathcal{C}^{\mathsf{op}} \to \mathbf{Set}^{\mathcal{D}}$$

but, surprisingly, the syntactic notions suggested by these alternatives are not equivalent. This is the subtlety that we explore in this work.

- To compute with categories we need syntax.
- There is a canonical way to present small categories by generators and relations (and morphisms between them).
- There is a canonical way to present **Set**-valued functors.
- But what about profunctors?

A profunctor $\mathcal{P} : \mathcal{C} \nrightarrow \mathcal{D}$ is either

$$\mathcal{C}^{\mathsf{op}} \times \mathcal{D} \to \mathbf{Set} \qquad \text{or} \qquad \mathcal{C}^{\mathsf{op}} \to \mathbf{Set}^{\mathcal{D}}$$

but, surprisingly, the syntactic notions suggested by these alternatives are not equivalent. This is the subtlety that we explore in this work.

But... why would we care?

Motivation: Categorical Database Theory

Our motivation comes from a **categorical data model** based on the following idea:

Database Schema	$\longleftrightarrow$	Category
Database Instance	$\longleftrightarrow$	Copresheaf

...every theorem about small categories becomes a theorem about databases. [Spi12]

Motivation: Categorical Database Theory

Our motivation comes from a **categorical data model** based on the following idea:

Database Schema	$\longleftrightarrow$	Category
Database Instance	$\longleftrightarrow$	Copresheaf

> *...every theorem about small categories becomes a theorem about databases. [Spi12]*

Remark: This data model (as presented in e.g. [Spi12]) has been superseded by another which incorporates data types and attributes [Sch+17], but the subtlety we are interested in arises already at this level.

Motivation: Categorical Database Theory

Our motivation comes from a **categorical data model** based on the following idea:

Database Schema	$\longleftrightarrow$	Category
Database Instance	$\longleftrightarrow$	Copresheaf

> *...every theorem about small categories becomes a theorem about databases. [Spi12]*

Remark: This data model (as presented in e.g. [Spi12]) has been superseded by another which incorporates data types and attributes [Sch+17], but the subtlety we are interested in arises already at this level.

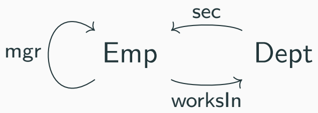
So, how does this work?

DB Schema $\longleftrightarrow$ Category

A category presentation consists of objects (a.k.a. "sorts"), generating arrows (a.k.a. "function symbols") and equations between parallel paths.

Example:

$C :=$

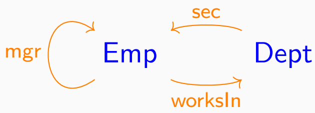


$$\text{mgr.worksIn} = \text{worksIn}$$
$$\text{sec.worksIn} = 1_{\text{Dept}}$$

A category presentation consists of objects (a.k.a. "sorts"), generating arrows (a.k.a. "function symbols") and equations between parallel paths.

Example:

$C := \{$



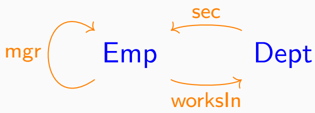
$$\text{mgr.worksIn} = \text{worksIn}$$
$$\text{sec.worksIn} = 1_{\text{Dept}}$$

$\}$

A category presentation consists of objects (a.k.a. "sorts"), generating arrows (a.k.a. "function symbols") and equations between parallel paths.

Example:

$C :=$



$$\text{mgr.worksIn} = \text{worksIn}$$
$$\text{sec.worksIn} = 1_{\text{Dept}}$$

The dots represent the associative concatenation of paths.

C presents the category $(\!|C|\!)$ ("the semantics of C") whose morphisms are the paths in C quotiented by the **provable equality relation** $\approx_C$ generated by the equations.

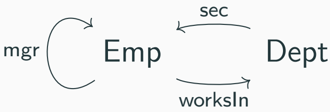
Explicitly, $\approx_C$ is the smallest equivalence relation that contains all the equations of C which is compatible with concatenation of paths ($p \approx_C q$ implies $f.p.g \approx_C f.q.g$ whenever the expression makes sense).

We write $[p]$ for the equivalence class of p.

DB Schema $\longleftrightarrow$ Category

In the example, we get:

$C :=$



$$\mathsf{mgr.worksIn} = \mathsf{worksIn}$$
$$\mathsf{sec.worksIn} = 1_{\mathsf{Dept}}$$

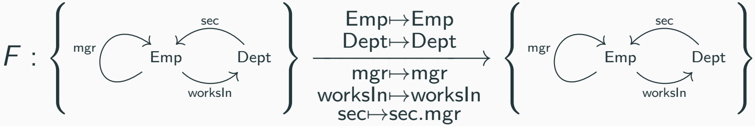
$$\mathrm{Obj}((\!|C|\!)) := \{\mathsf{Emp}, \mathsf{Dept}\}$$

$$\mathrm{Mor}((\!|C|\!)) := \{\underbrace{1_{\mathsf{Emp}}, 1_{\mathsf{Dept}},}_{\text{length } 0}, \underbrace{[\mathsf{worksIn}], [\mathsf{sec}], [\mathsf{mgr}]}_{\text{length } 1},$$
$$\underbrace{[\mathsf{mgr.mgr}], [\mathsf{worksIn.sec}], [\mathsf{sec.mgr}]}_{\text{length } 2}, \ldots$$
$$\underbrace{[mgr^{k}], [\mathsf{worksIn.sec.mgr}^{k-2}], [\mathsf{sec.mgr}^{k-1}]}_{\text{length } k,\ k \geq 3}, \quad \ldots\}$$

DB Schema $\longleftrightarrow$ Category

A **morphism of category presentations** $F : C \to D$ is an assignment on objects together with an assignment from generating arrows to paths in the target presentation.

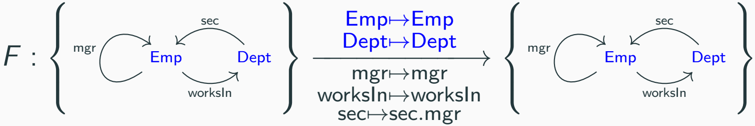
Example:



DB Schema $\longleftrightarrow$ Category

A **morphism of category presentations** $F : C \to D$ is an assignment on objects together with an assignment from generating arrows to paths in the target presentation.

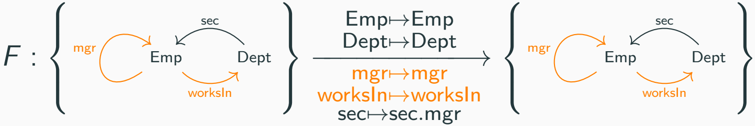
Example:



DB Schema $\longleftrightarrow$ Category

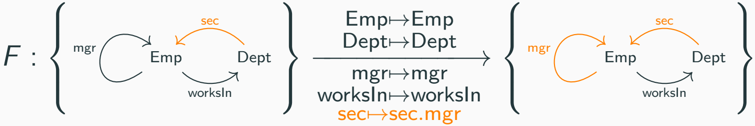
A **morphism of category presentations** $F : C \to D$ is an assignment on objects together with an assignment from generating arrows to paths in the target presentation.

Example:



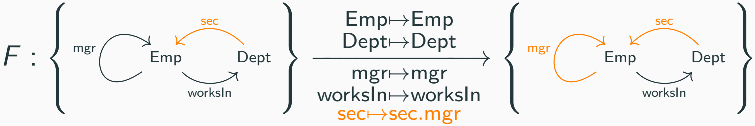
A **morphism of category presentations** $F : C \to D$ is an assignment on objects together with an assignment from generating arrows to paths in the target presentation.

Example:



A **morphism of category presentations** $F : C \to D$ is an assignment on objects together with an assignment from generating arrows to paths in the target presentation.

Example:



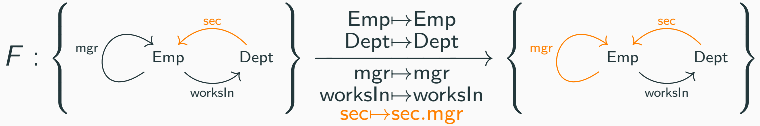
We require that if $p =_C q$, then $F(p) \approx_D F(q)$:

mgr.worksIn $=_C$ worksIn $\rightsquigarrow$ mgr.worksIn $\approx_C$ worksIn ✓

sec.worksIn $=_C 1_{\text{Dept}} \rightsquigarrow$ (sec.mgr).worksIn $\approx_C$ sec.worksIn $\approx_C 1_{\text{Dept}}$ ✓

A **morphism of category presentations** $F : C \to D$ is an assignment on objects together with an assignment from generating arrows to paths in the target presentation.

Example:



We require that if $p =_C q$, then $F(p) \approx_D F(q)$:

mgr.worksIn $=_C$ worksIn $\rightsquigarrow$ mgr.worksIn $\approx_C$ worksIn ✓

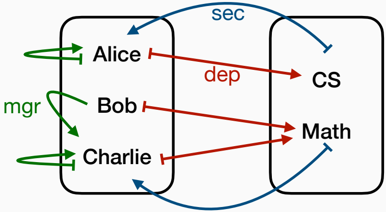
sec.worksIn $=_C 1_{\text{Dept}} \rightsquigarrow$ (sec.mgr).worksIn $\approx_C$ sec.worksIn $\approx_C 1_{\text{Dept}}$ ✓

In this way we obtain a category **CatPr** and a semantics functor $(\!|-|\!) : \textbf{CatPr} \to \textbf{Cat}$.

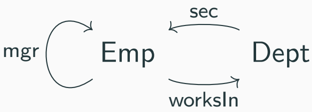
DB Instance $\longleftrightarrow$ Copresheaf

To understand this as a database schema, let us look at an example copresheaf $\mathcal{I}$ on $(\!|C|\!)$, which is determined by its action on objects and on the generating arrows:

$\mathcal{I} :=$

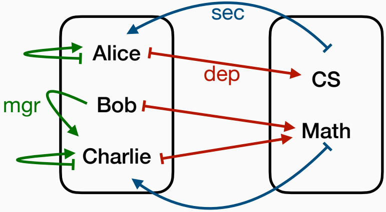


$: (\!|C|\!) \to \mathbf{Set}$



DB Instance $\longleftrightarrow$ Copresheaf

When we visualize with tables, we see that each object c corresponds to a table and each function symbol in C going out of c corresponds to a column in that table:



Emp	mgr	worksIn
Alice	Alice	CS
Bob	Charlie	Math
Charlie	Charlie	Math

Dept	sec
CS	Alice
Math	Charlie

DB Instance $\longleftrightarrow$ Copresheaf

From now on we will use the words "instance" and "copresheaf" interchangeably.

We can go further and define instance presentations similarly to presentations for actions of monoids/groups, e.g.

$$I = \langle \underbrace{e : \mathrm{Emp}}_{\text{generator(s)}} \mid \underbrace{\varnothing}_{\text{equations}} \rangle.$$

DB Instance $\longleftrightarrow$ Copresheaf

From now on we will use the words "instance" and "copresheaf" interchangeably.

We can go further and define instance presentations similarly to presentations for actions of monoids/groups, e.g.

$$I = \langle \underbrace{e : \mathrm{Emp}}_{\text{generator(s)}} \mid \underbrace{\varnothing}_{\text{equations}} \rangle.$$

This presents the infinite copresheaf $(\!|I|\!) : (\!|C|\!) \to \mathbf{Set}$ given by

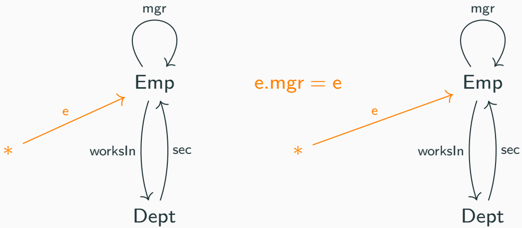
Emp	mgr	worksIn
[e]	[e.mgr]	[e.worksIn]
[e.mgr]	[e.mgr^2]	[e.worksIn]
...	...	[e.worksIn]
[e.worksIn.sec]	[e.worksIn.sec.mgr]	[e.worksIn]
[e.worksIn.sec.mgr]	[e.worksIn.sec.mgr^2]	[e.worksIn]
...	...	[e.worksIn]

Dept	sec
[e.worksIn]	[e.worksIn.sec]
[e.worksIn]	[e.worksIn.sec]

DB Instance $\longleftrightarrow$ Copresheaf

Formally we define an **instance presentation** on C to be a category presentation extending C with a unique object $*$, new arrows coming out of $*$, and some equations involving the newly added arrows, e.g.

$$I = \langle \text{e} : \text{Emp} \mid \varnothing \rangle \qquad I' = \langle \text{e} : \text{Emp} \mid \text{e.mgr} = \text{e} \rangle$$



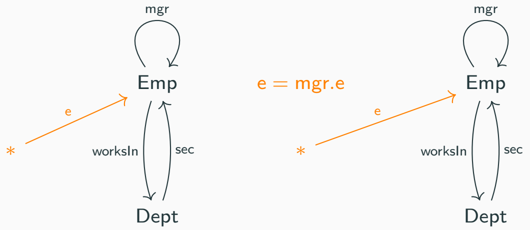
A **morphism of instance presentations** C $\phi : I \to J$ is an assignment of generators $x : * \to c$ in I to paths $y_1. \cdots .y_n : * \to c$ in D such that equations in I are respected.

DB Queries $\longleftrightarrow$ Mapping out of Instances?

A data model should also say how to *manipulate* the data. We will consider a finitely presented instance as a conjunctive query, using it to query other instances by mapping into them.

DB Queries $\longleftrightarrow$ Mapping out of Instances?

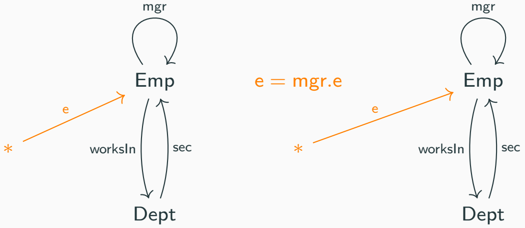
A data model should also say how to *manipulate* the data. We will consider a finitely presented instance as a conjunctive query, using it to query other instances by mapping into them.



- I presents a representable on Emp: $\mathbf{Set}^{(\!|C|\!)}((\!|I|\!), \mathcal{J}) \cong \mathcal{J}(\mathrm{Emp})$ (i.e. representables are the free (co)presheaves on one generator).

DB Queries $\longleftrightarrow$ Mapping out of Instances?

A data model should also say how to *manipulate* the data. We will consider a finitely presented instance as a conjunctive query, using it to query other instances by mapping into them.



- I presents a representable on Emp: $\mathbf{Set}^{(\!|C|\!)}((\!|I|\!), \mathcal{J}) \cong \mathcal{J}(\mathrm{Emp})$ (i.e. representables are the free (co)presheaves on one generator).
- $\mathbf{Set}^{(\!|C|\!)}((\!|I'|\!), \mathcal{J})$ is the set of employees in $\mathcal{J}$ who are their own managers.

DB Queries $\longleftrightarrow$ Profunctors!

This is nice for basic querying, but it's not expressive enough if instead of a set of answers, we actually want to produce a new instance (e.g. implement data migration).

This is nice for basic querying, but it's not expressive enough if instead of a set of answers, we actually want to produce a new instance (e.g. implement data migration).

Profunctors let us query and transform in more complex ways: given a profunctor $\mathcal{P} : \mathcal{C} \nrightarrow \mathcal{D}$ seen as $\mathcal{P} : \mathcal{C}^{\mathsf{op}} \to \mathbf{Set}^{\mathcal{D}}$, define

$$\mathsf{Eval}_{\mathcal{P}} : \mathbf{Set}^{\mathcal{D}} \to \mathbf{Set}^{\mathcal{C}}$$
$$\mathsf{Eval}_{\mathcal{P}}(\mathcal{J}) \coloneqq \mathbf{Set}^{\mathcal{D}}(\mathcal{P}(-), \mathcal{J})$$

This contains the previous situation as a particular case by setting $\mathcal{C} = \mathbf{1}$, since a profunctor $\mathbf{1} \nrightarrow \mathcal{D}$ is simply a copresheaf on $\mathcal{D}$.

Composing Queries

Moreover it is crucial to be able to compose queries before evaluating them (i.e. without taking a look at the data).

Recall the usual composition rule for profunctors:

$$\mathcal{C} \xrightarrow{\mathcal{P}} \mathcal{D} \xrightarrow{\mathcal{Q}} \mathcal{E} \quad \rightsquigarrow \quad \mathcal{C} \xrightarrow{\mathcal{P} \odot \mathcal{Q}} \mathcal{E}$$

$$(\mathcal{P} \odot \mathcal{Q})(c, e) \cong \int^{d \in \mathcal{D}} \mathcal{P}(c, d) \times \mathcal{Q}(d, e).$$

Profunctor composition implements composition of queries, because

$$\mathsf{Eval}_{\mathcal{P}} \circ \mathsf{Eval}_{\mathcal{Q}} \cong \mathsf{Eval}_{\mathcal{P} \odot \mathcal{Q}}.$$

Composing Queries

Moreover it is crucial to be able to compose queries before evaluating them (i.e. without taking a look at the data).

Recall the usual composition rule for profunctors:

$$\mathcal{C} \overset{\mathcal{P}}{\nrightarrow} \mathcal{D} \overset{\mathcal{Q}}{\nrightarrow} \mathcal{E} \quad \rightsquigarrow \quad \mathcal{C} \overset{\mathcal{P} \odot \mathcal{Q}}{\nrightarrow} \mathcal{E}$$

$$(\mathcal{P} \odot \mathcal{Q})(c, e) \cong \int^{d \in \mathcal{D}} \mathcal{P}(c, d) \times \mathcal{Q}(d, e).$$

Profunctor composition implements composition of queries, because

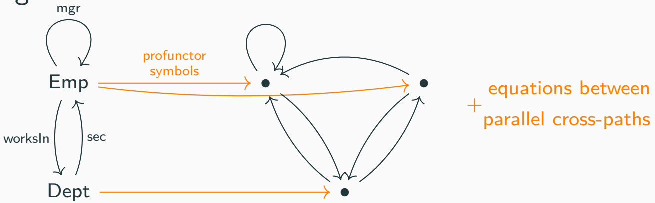
$$\mathsf{Eval}_{\mathcal{P}} \circ \mathsf{Eval}_{\mathcal{Q}} \cong \mathsf{Eval}_{\mathcal{P} \odot \mathcal{Q}}.$$

Examples are much easier with presentations, so let's get to that first.

Profunctor Presentations

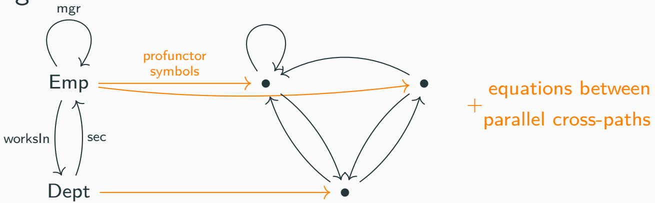
Uncurried Profunctor Presentations

Since instances on a category $\mathcal{C}$ are profunctors $\mathbf{1} \nrightarrow \mathcal{C}$, we can start from instance presentations and generalise. An **uncurried profunctor presentation** $C \to D$ is a category presentation simultaneously extending C and D:



Uncurried Profunctor Presentations

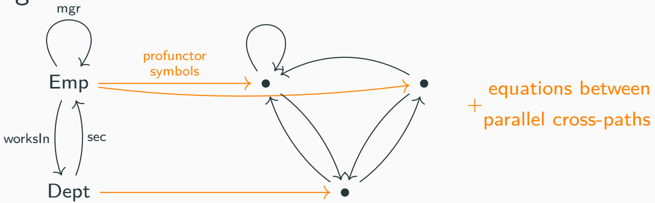
Since instances on a category $\mathcal{C}$ are profunctors $\mathbf{1} \nrightarrow \mathcal{C}$, we can start from instance presentations and generalise. An **uncurried profunctor presentation** $C \to D$ is a category presentation simultaneously extending C and D:



This notion turns out to be equivalent to $(C^{\mathrm{op}} \times D)$-instance presentations.

Uncurried Profunctor Presentations

Since instances on a category $\mathcal{C}$ are profunctors $\mathbf{1} \nrightarrow \mathcal{C}$, we can start from instance presentations and generalise. An **uncurried profunctor presentation** $C \to D$ is a category presentation simultaneously extending C and D:



This notion turns out to be equivalent to $(C^{\mathrm{op}} \times D)$-instance presentations.

By defining morphisms of uncurried presentations in a straightforward way, we obtain a category $\mathbf{UnCurr}(C, D)$ and a semantics functor $(\!|-|\!) : \mathbf{UnCurr}(C, D) \to \mathbf{Prof}((\!|C|\!), (\!|D|\!))$.

Problem: Failure of Compositionality in the Finite

Given finite category presentations C and D and a profunctor $\mathcal{P} : (\!|C|\!) \to (\!|D|\!)$, if it admits a finite profunctor presentation P (such that $(\!|P|\!) \cong \mathcal{P}$) then we say that it is **finitely uncurried presentable**

Problem: Failure of Compositionality in the Finite

Given finite category presentations C and D and a profunctor $\mathcal{P} : (\!|C|\!) \to (\!|D|\!)$, if it admits a finite profunctor presentation P (such that $(\!|P|\!) \cong \mathcal{P}$) then we say that it is **finitely uncurried presentable**

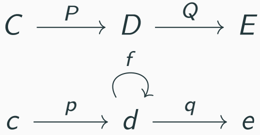
Theorem: The class of finitely uncurried presentable profunctors is not closed under composition.

Problem: Failure of Compositionality in the Finite

Given finite category presentations C and D and a profunctor $\mathcal{P} : (\!|C|\!) \to (\!|D|\!)$, if it admits a finite profunctor presentation P (such that $(\!|P|\!) \cong \mathcal{P}$) then we say that it is **finitely uncurried presentable**

Theorem: The class of finitely uncurried presentable profunctors is not closed under composition.

Proof: consider the following presentations (with no equations):

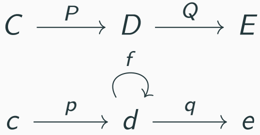


Problem: Failure of Compositionality in the Finite

Given finite category presentations C and D and a profunctor $\mathcal{P} : (\!|C|\!) \to (\!|D|\!)$, if it admits a finite profunctor presentation P (such that $(\!|P|\!) \cong \mathcal{P}$) then we say that it is **finitely uncurried presentable**

Theorem: The class of finitely uncurried presentable profunctors is not closed under composition.

Proof: consider the following presentations (with no equations):



Problem: Failure of Compositionality in the Finite

Given finite category presentations C and D and a profunctor $\mathcal{P} : (\!|C|\!) \to (\!|D|\!)$, if it admits a finite profunctor presentation P (such that $(\!|P|\!) \cong \mathcal{P}$) then we say that it is **finitely uncurried presentable**

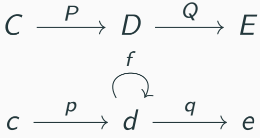
Theorem: The class of finitely uncurried presentable profunctors is not closed under composition.

Proof: consider the following presentations (with no equations):

$$(\!|P|\!)(c, d) = \{[p.f^k] \mid k \geq 0\}$$

$$(\!|Q|\!)(d, e) = \{[f^k.q] \mid k \geq 0\}$$

$$((\!|P|\!) \odot (\!|Q|\!))(c, e) = \{[p.f^k.q] \mid k \geq 0\}$$



But any uncurried presentation $R : C \to E$ such that $(\!|R|\!)(c, e)$ is infinite must have an infinite number of generating profunctor symbols. $\square$

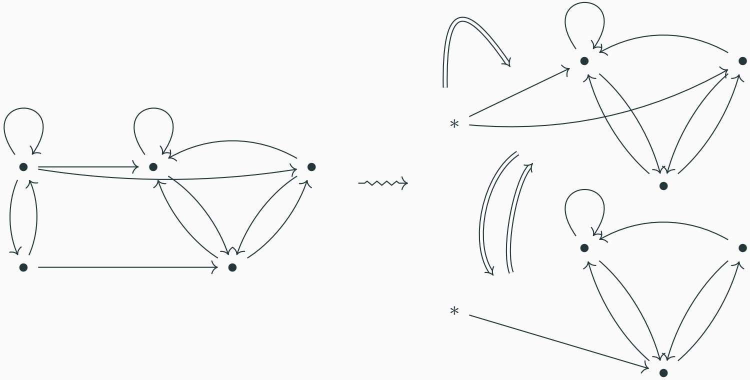
Curried Profunctor Presentations

Recall that semantically, $\mathbf{CAT}(\mathcal{C}^{\mathsf{op}} \times \mathcal{D}, \mathbf{Set}) \simeq \mathbf{CAT}(\mathcal{C}^{\mathsf{op}}, \mathbf{Set}^{\mathcal{D}})$.

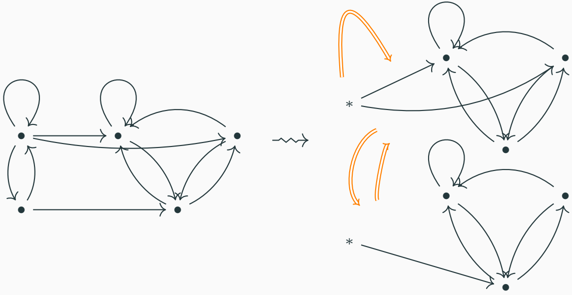
Curried Profunctor Presentations

Recall that semantically, $\mathbf{CAT}(\mathcal{C}^{\mathrm{op}} \times \mathcal{D}, \mathbf{Set}) \simeq \mathbf{CAT}(\mathcal{C}^{\mathrm{op}}, \mathbf{Set}^{\mathcal{D}})$.

Solution: move from (C, D)-uncurried presentations to C^{op}-indexed families of D-instance presentations, with morphisms between them.

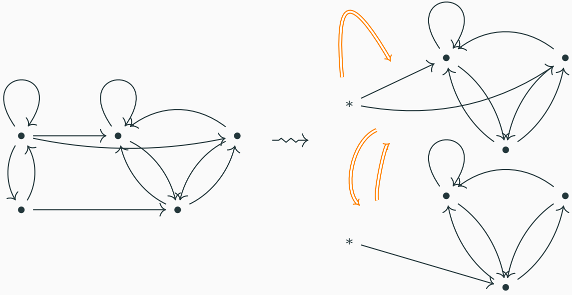


Curried Profunctor Presentations



The morphisms of instance presentations, when composed with each other, must satisfy the equations of C in a suitable sense (up to provable equality). We call these **curried profunctor presentations**.

Curried Profunctor Presentations



The morphisms of instance presentations, when composed with each other, must satisfy the equations of C in a suitable sense (up to provable equality). We call these **curried profunctor presentations**.

Morphisms are defined in a straightforward way. We obtain a category $\mathbf{Curr}(C, D)$ with semantics $(\!|-|\!) : \mathbf{Curr}(C, D) \to \mathbf{Prof}((\!|C|\!), (\!|D|\!))$.

Syntactic Composition of Curried Presentations

Given curried profunctor presentations $P : C \to D$ and $Q : D \to E$, there is a **composite curried presentation** $P \circledast Q : C \to E$. This is obtained by following an algorithm known as *sub-query unnesting* or *view unfolding* (as sketched for instance in [SW17]).

Syntactic Composition of Curried Presentations

Given curried profunctor presentations $P : C \to D$ and $Q : D \to E$, there is a **composite curried presentation** $P \circledast Q : C \to E$. This is obtained by following an algorithm known as *sub-query unnesting* or *view unfolding* (as sketched for instance in [SW17]).

Importantly, $P \circledast Q$ is finite if both P and Q are.

Syntactic Composition of Curried Presentations

Given curried profunctor presentations $P : C \nrightarrow D$ and $Q : D \nrightarrow E$, there is a **composite curried presentation** $P \circledast Q : C \nrightarrow E$. This is obtained by following an algorithm known as *sub-query unnesting* or *view unfolding* (as sketched for instance in [SW17]).

Importantly, $P \circledast Q$ is finite if both P and Q are.

Lemma: the construction extends to a functor

$$\circledast : \mathbf{Curr}(C, D) \times \mathbf{Curr}(D, E) \to \mathbf{Curr}(C, E).$$

Theorem: There is a natural isomorphism

$$\mu : (\!|-|\!) \odot (\!|=|\!) \xrightarrow{\cong} (\!|- \circledast =|\!)$$

i.e. $\circledast$ is correct with respect to profunctor composition.

Syntactic Composition of Curried Presentations

Given curried profunctor presentations $P : C \to D$ and $Q : D \to E$, there is a **composite curried presentation** $P \circledast Q : C \to E$. This is obtained by following an algorithm known as *sub-query unnesting* or *view unfolding* (as sketched for instance in [SW17]).

Importantly, $P \circledast Q$ is finite if both P and Q are.

Lemma: the construction extends to a functor

$$\circledast : \mathbf{Curr}(C, D) \times \mathbf{Curr}(D, E) \to \mathbf{Curr}(C, E).$$

Theorem: There is a natural isomorphism

$$\mu : (\!|-|\!) \odot (\!|=|\!) \xrightarrow{\cong} (\!|- \circledast =|\!)$$

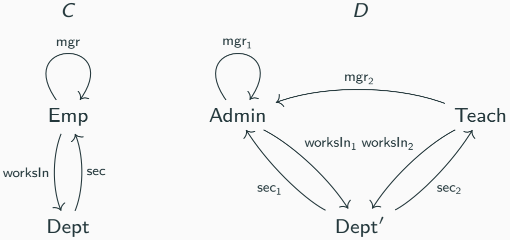
i.e. $\circledast$ is correct with respect to profunctor composition.

Corollary: the class of finitely curried presentable profunctors is closed under composition.

An Example of Syntactic Composition (and Querying)

We explain the $\circledast$ construction through an example.

Example: consider the following two category presentations.



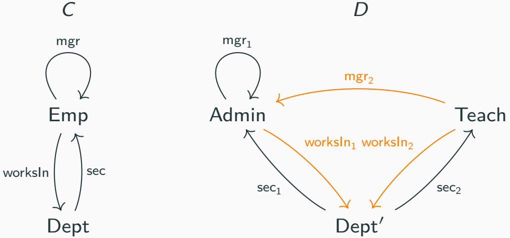
$$\mathsf{mgr}_1.\mathsf{worksIn}_1 = \mathsf{worksIn}_1$$
$$\mathsf{mgr}_2.\mathsf{worksIn}_1 = \mathsf{worksIn}_2$$
$$\mathsf{sec}_1.\mathsf{worksIn}_1 = 1_{\mathsf{Dept}'}$$
$$\mathsf{sec}_2.\mathsf{worksIn}_2 = 1_{\mathsf{Dept}'}$$

The equations of C are as before. The equations of D are a duplication of the ones of C, except for the variation $\mathsf{mgr}_2.\mathsf{worksIn}_1 = \mathsf{worksIn}_2$.

An Example of Syntactic Composition (and Querying)

We explain the $\circledast$ construction through an example.

Example: consider the following two category presentations.



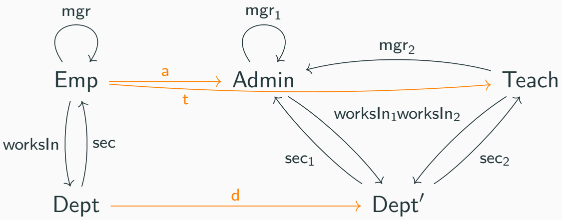
$$\mathsf{mgr}_1.\mathsf{worksIn}_1 = \mathsf{worksIn}_1$$
$$\mathsf{mgr}_2.\mathsf{worksIn}_1 = \mathsf{worksIn}_2$$
$$\mathsf{sec}_1.\mathsf{worksIn}_1 = 1_{\mathsf{Dept}'}$$
$$\mathsf{sec}_2.\mathsf{worksIn}_2 = 1_{\mathsf{Dept}'}$$

The equations of C are as before. The equations of D are a duplication of the ones of C, except for the variation $\mathsf{mgr}_2.\mathsf{worksIn}_1 = \mathsf{worksIn}_2$.

An Example of Syntactic Composition (and Querying)

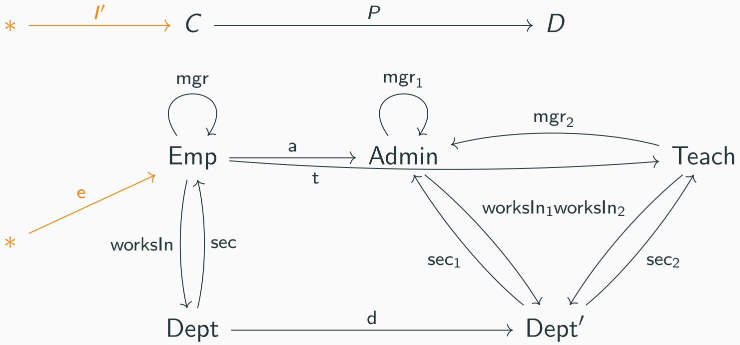
Now consider the following curried presentation $P : C \to D$:

$$
\begin{aligned}
P(\mathsf{Emp}) &:= \langle \mathsf{a} : \mathsf{Admin}, \mathsf{t} : \mathsf{Teach} \mid t.\mathsf{mgr}_2 = \mathsf{a} \rangle \\
P(\mathsf{Dept}) &:= \langle \mathsf{d} : \mathsf{Dept}' \mid \varnothing \rangle \\
P(\mathsf{mgr}) &:= \{\mathsf{a} \mapsto \mathsf{a.mgr}_1, \mathsf{t} \mapsto \mathsf{t}\} \\
P(\mathsf{sec}) &:= \{\mathsf{a} \mapsto \mathsf{d.sec}_1, \mathsf{t} \mapsto \mathsf{d.sec}_2\} \\
P(\mathsf{worksIn}) &:= \{\mathsf{d} \mapsto \mathsf{a.worksIn}_1\}
\end{aligned}
$$

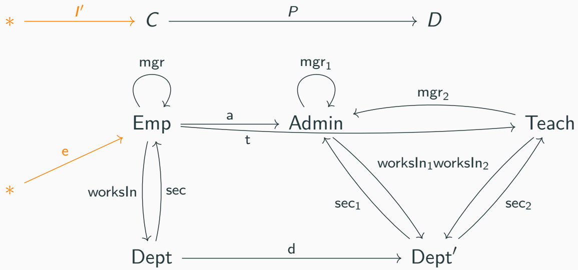


An Example of Syntactic Composition (and Querying)

Recall the instance presentation $I' = \langle \text{e} : \text{Emp} \mid \text{e.mgr} = \text{e} \rangle$ from the introduction, seen as a curried profunctor presentation $* \to C$. Diagrammatically, the situation is this:

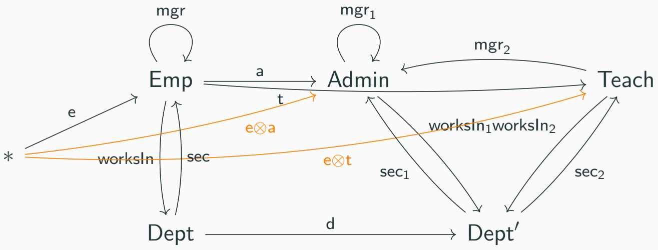


An Example of Syntactic Composition (and Querying)



To obtain the composite $I' \circledast P : * \to D$, we must define a unique D-instance presentation $(I' \circledast P)(*)$. To do it, look at all pairs of "composable" generators and pair them into new symbols.

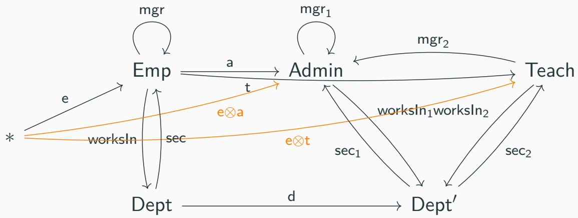
An Example of Syntactic Composition (and Querying)



To obtain the composite $I' \circledast P : * \to D$, we must define a unique D-instance presentation $(I' \circledast P)(*)$. To do it, look at all pairs of “composable” generators and pair them into new symbols.

We obtain generators $\mathsf{e} \otimes \mathsf{a} : \mathsf{Admin}$ and $\mathsf{e} \otimes \mathsf{t} : \mathsf{Teach}$.

An Example of Syntactic Composition (and Querying)



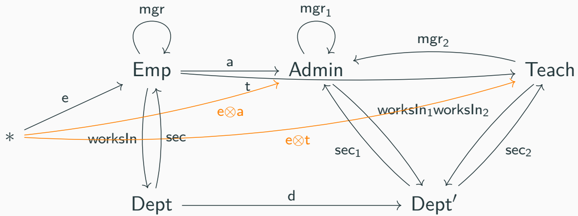
Then, to obtain the equations of the instance we take all equations from $I'(*)$, $P(\mathsf{Emp})$ and $P(\mathsf{Dept})$ and "tensor them" on the left and on the right all possible generators:

$$\mathsf{e.mgr} = \mathsf{e} \rightsquigarrow (\mathsf{e.mgr}) \otimes \mathsf{a} = \mathsf{e} \otimes \mathsf{a} \qquad \rightsquigarrow \mathsf{e} \otimes \mathsf{a.mgr}_1 = \mathsf{e} \otimes \mathsf{a}$$

$$\mathsf{e.mgr} = \mathsf{e} \rightsquigarrow (\mathsf{e.mgr}) \otimes \mathsf{t} = \mathsf{e} \otimes \mathsf{t} \qquad \rightsquigarrow \mathsf{e} \otimes \mathsf{t} = \mathsf{e} \otimes \mathsf{t}$$

$$\mathsf{t.mgr}_2 = \mathsf{a} \rightsquigarrow \mathsf{e} \otimes (\mathsf{t.mgr}_2) = \mathsf{e} \otimes \mathsf{a} \qquad \rightsquigarrow (\mathsf{e} \otimes \mathsf{t}).\mathsf{mgr}_2 = \mathsf{e} \otimes \mathsf{a}$$

An Example of Syntactic Composition (and Querying)

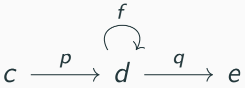


Since there are no arrow symbols in the domain presentation, we are done. $I' \circledast P$ is the conjunctive query I' *migrated along* P, given by the instance

$$\langle \mathsf{e}{\otimes}\mathsf{a} : \mathsf{Admin}, \mathsf{e}{\otimes}\mathsf{t} : \mathsf{Teach} \mid (\mathsf{e}{\otimes}\mathsf{a}).\mathsf{mgr}_1 = \mathsf{e}{\otimes}\mathsf{a}, (\mathsf{e}{\otimes}\mathsf{t}).\mathsf{mgr}_2 = \mathsf{e}{\otimes}\mathsf{a} \rangle.$$

In other words, it looks for all pairs of an admin A and a teacher T such that the manager of T is A and A is their own manager.

So... What failed here?



Understanding Curried Versus Uncurried

- We want to guarantee the existence of a finite set of generators for the composite.

Understanding Curried Versus Uncurried

- We want to guarantee the existence of a finite set of generators for the composite.
- In the definition of $\circledast$, we did this by pairing generators from P and Q into generators of the form $p \otimes q$.

Understanding Curried Versus Uncurried

- We want to guarantee the existence of a finite set of generators for the composite.
- In the definition of $\circledast$, we did this by pairing generators from P and Q into generators of the form $p \otimes q$.
- This worked because Q contains instance presentation morphisms $Q(f)$ for each f in D, which give you the information to turn a cross-path $f.q$ into some path $Q(f)(q) \equiv q.h_1.\ldots.h_\ell$ starting with a profunctor symbol of Q.

Understanding Curried Versus Uncurried

- We want to guarantee the existence of a finite set of generators for the composite.
- In the definition of $\circledast$, we did this by pairing generators from P and Q into generators of the form $p \otimes q$.
- This worked because Q contains instance presentation morphisms $Q(f)$ for each f in D, which give you the information to turn a cross-path $f.q$ into some path $Q(f)(q) \equiv q.h_1.\dots.h_\ell$ starting with a profunctor symbol of Q.
- Given an uncurried presentation Q, we don't have the morphisms $Q(f)$ anymore, but can still require that every cross-path in Q can be rewritten to start with a profunctor symbol. In this case we say that Q is **non-generative**.

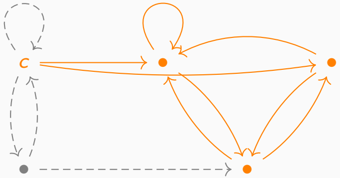
Curryable Profunctor Presentations

It turns out that we need another condition, so that the amount of *equations* in the composite is finite.

Curryable Profunctor Presentations

It turns out that we need another condition, so that the amount of *equations* in the composite is finite.

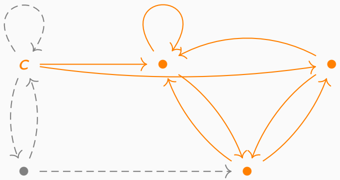
Given $c \in C$ let P^c denote the D-instance presentation obtained by “restriction”:



Curryable Profunctor Presentations

It turns out that we need another condition, so that the amount of *equations* in the composite is finite.

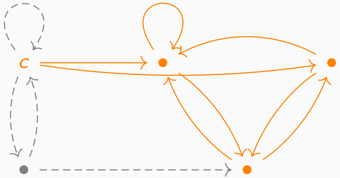
Given $c \in C$ let P^c denote the D-instance presentation obtained by "restriction":



Suppose that for every pair of paths t, t' in P^c starting at c, if $t \approx_P t'$, then $t \approx_{P^c} t'$. (i.e. P is a conservative extension of P^c in the sense of algebraic theories.) If this happens for all $c \in C$, we say that P is **conservative**.

It turns out that we need another condition, so that the amount of *equations* in the composite is finite.

Given $c \in C$ let P^c denote the D-instance presentation obtained by "restriction":

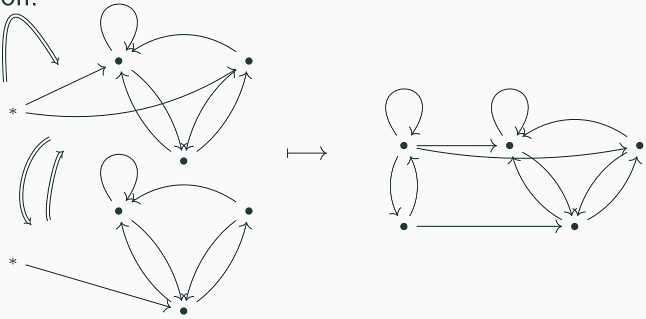


Suppose that for every pair of paths t, t' in P^c starting at c, if $t \approx_P t'$, then $t \approx_{P^c} t'$. (i.e. P is a conservative extension of P^c in the sense of algebraic theories.) If this happens for all $c \in C$, we say that P is **conservative**.

P is said to be **curryable** if it is conservative and nongenerative.

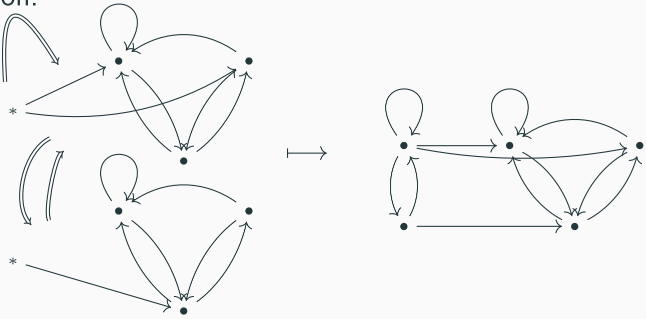
Equivalence between Curried and Curryable

To understand the relationship between curried and curryable, we begin by observing the existence of a straightforward uncurrying operation.



Equivalence between Curried and Curryable

To understand the relationship between curried and curryable, we begin by observing the existence of a straightforward uncurrying operation.



Thm: this construction determines a functor $\overline{(-)} : \mathbf{Curr}(C, D) \to \mathbf{UnCurr}(C, D)$ which restricts to finite presentations and preserves the semantics.

Equivalence between Curried and Curryable

Let $\mathbf{Crble}(C, D)$ be the non-full subcategory of $\mathbf{UnCurr}(C, D)$ spanned by curryable presentations and morphisms that send all cross-paths to right paths $(*)$.

Theorem: The functor $\overline{(-)} : \mathbf{Curr}(C, D) \to \mathbf{UnCurr}(C, D)$ co-restricts to an equivalence of categories

$$\overline{(-)} : \mathbf{Curr}(C, D) \xrightarrow{\simeq} \mathbf{Crble}(C, D).$$

This equivalence restricts to an equivalence between the subcategories of finite presentations.

Remark: The technical condition $(*)$ can be dropped by weakening equivalence to biequivalence (where the 2-cells of $\mathbf{Curr}(C, D)$ and $\mathbf{UnCurr}(C, D)$ are given by provable equality of presentations).

Thank you!

[Sch+17] Patrick Schultz et al. **"Algebraic Databases".** *Theory and Applications of Categories* 32.16 (2017), pp. 547–619.

[Spi12] David I Spivak. **"Functorial data migration".** *Information and Computation* 217 (2012), pp. 31–51.

[SW17] Patrick Schultz and Ryan Wisnesky. **"Algebraic data integration".** *Journal of Functional Programming* 27 (2017).