Doctrine-specific ur-algorithms

Mohamed Barakat

Berkeley Seminar @ Topos Institute
March 18, 2024

Universität Siegen

Joint work with Sebastian Posur, Kamal Saleh, Fabian Zickgraf

Motivating question: The connecting morphism

Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.

$$\begin{array}{ccccc} A & \xrightarrow{a} & B & & \\ & & {\scriptstyle b}\downarrow & & \\ & & C & \xrightarrow{c} & D \end{array}$$

Motivating question: The connecting morphism

Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.

$$\begin{array}{ccccc} \boldsymbol{A} & \xrightarrow{\boldsymbol{a}} & \boldsymbol{B} & \overset{d}{\twoheadrightarrow} & \operatorname{coker}(a) \\ & & \boldsymbol{b}\big\downarrow & & \\ & & \boldsymbol{C} & \xrightarrow{\boldsymbol{c}} & \boldsymbol{D} \end{array}$$

Motivating question: The connecting morphism

Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.

$$
\begin{array}{ccccc}
\boldsymbol{A} & \xrightarrow{\boldsymbol{a}} & \boldsymbol{B} & \overset{d}{\twoheadrightarrow} & \operatorname{coker}(a) \\
 & & \boldsymbol{b}\big\downarrow & & e\big\downarrow \\
 & & \boldsymbol{C} & \xrightarrow{\boldsymbol{c}} & \boldsymbol{D}
\end{array}
$$

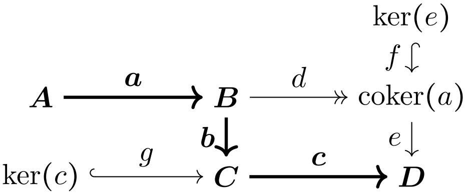
Motivating question: The connecting morphism

Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.

$$
\begin{array}{ccccc}
 & & & & \ker(e) \\
 & & & & \Big\downarrow f \\
\boldsymbol{A} & \xrightarrow{\boldsymbol{a}} & \boldsymbol{B} & \xrightarrow{d} & \operatorname{coker}(a) \\
 & & \boldsymbol{b}\Big\downarrow & & \Big\downarrow e \\
 & & \boldsymbol{C} & \xrightarrow{\boldsymbol{c}} & \boldsymbol{D}
\end{array}
$$

Motivating question: The connecting morphism

Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.



Motivating question: The connecting morphism

Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.

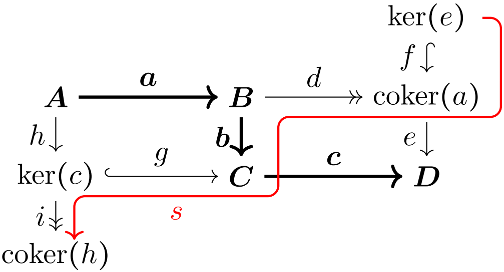
$$
\begin{array}{ccccc}
 & & & & \ker(e) \\
 & & & & {\scriptstyle f}\big\downarrow \\
\boldsymbol{A} & \xrightarrow{\boldsymbol{a}} & \boldsymbol{B} & \xrightarrow{d} & \operatorname{coker}(a) \\
{\scriptstyle h}\big\downarrow & & {\scriptstyle \boldsymbol{b}}\big\downarrow & & {\scriptstyle e}\big\downarrow \\
\ker(c) & \xrightarrow{g} & \boldsymbol{C} & \xrightarrow{\boldsymbol{c}} & \boldsymbol{D}
\end{array}
$$

Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.

$$
\begin{array}{ccccc}
 & & & & \ker(e) \\
 & & & & {\scriptstyle f}\downarrow \\
\boldsymbol{A} & \xrightarrow{\boldsymbol{a}} & \boldsymbol{B} & \overset{d}{\twoheadrightarrow} & \operatorname{coker}(a) \\
{\scriptstyle h}\downarrow & & {\scriptstyle \boldsymbol{b}}\boldsymbol{\downarrow} & & {\scriptstyle e}\downarrow \\
\ker(c) & \overset{g}{\hookrightarrow} & \boldsymbol{C} & \xrightarrow{\boldsymbol{c}} & \boldsymbol{D} \\
{\scriptstyle i}\downarrow & & & & \\
\operatorname{coker}(h) & & & &
\end{array}
$$

Motivating question: The connecting morphism

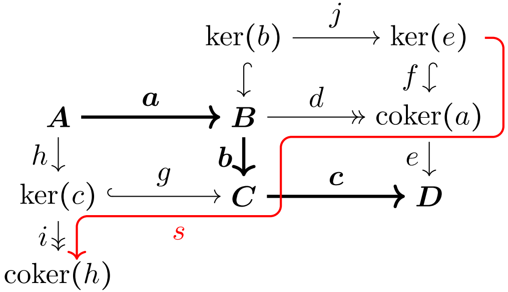
Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.



Then there exists a *natural* morphism $\ker(e) \xrightarrow{s} \operatorname{coker}(h)$

Motivating question: The connecting morphism

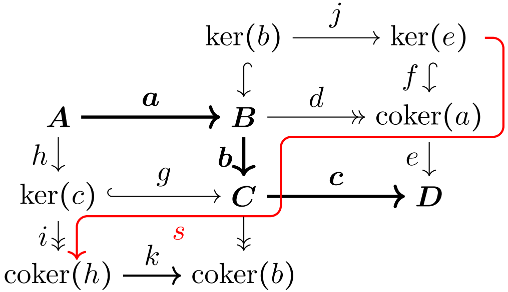
Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.



Then there exists a *natural* morphism $\ker(e) \xrightarrow{s} \operatorname{coker}(h)$

Motivating question: The connecting morphism

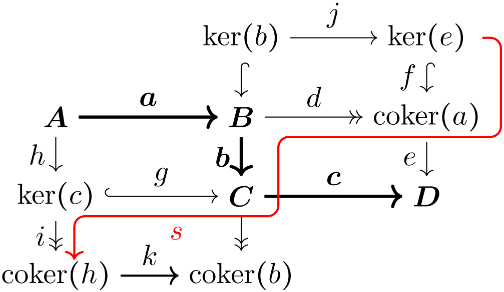
Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.



Then there exists a *natural* morphism $\ker(e) \xrightarrow{s} \operatorname{coker}(h)$

Motivating question: The connecting morphism

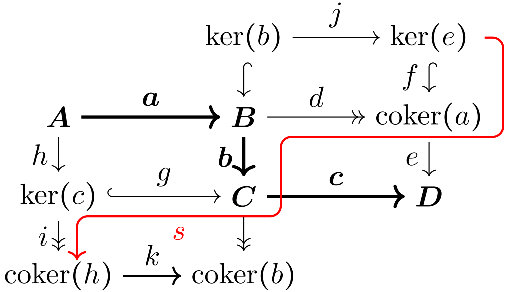
Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.



Then there exists a *natural* morphism $\ker(e) \xrightarrow{s} \operatorname{coker}(h)$ with $\ker(b) \xrightarrow{j} \ker(e) \xrightarrow{s} \operatorname{coker}(h) \xrightarrow{k} \operatorname{coker}(b)$ an exact sequence.

Motivating question: The connecting morphism

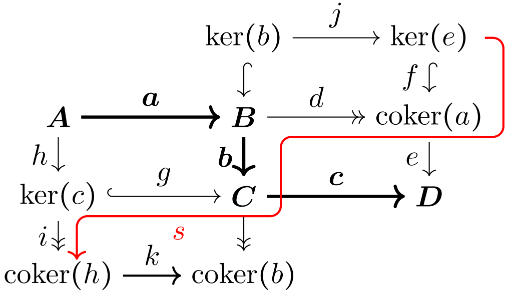
Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.



Then there exists a *natural* morphism $\ker(e) \xrightarrow{s} \operatorname{coker}(h)$ with $\ker(b) \xrightarrow{j} \ker(e) \xrightarrow{s} \operatorname{coker}(h) \xrightarrow{k} \operatorname{coker}(b)$ an exact sequence.

- Does "with" mean "such that" or "furthermore"?

Snake Lemma: Given three composable morphisms $A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D$ in an Abelian category with $abc = 0$.



Then there exists a *natural* morphism $\ker(e) \xrightarrow{s} \operatorname{coker}(h)$ with $\ker(b) \xrightarrow{j} \ker(e) \xrightarrow{s} \operatorname{coker}(h) \xrightarrow{k} \operatorname{coker}(b)$ an exact sequence.

- Does “with” mean “such that” or “furthermore”?
- In what sense is s unique? Give a construction algorithm.

An oracle for free Abelian categories

- We could answer the above questions if we would have an oracle for computing in free Abelian categories:

An oracle for free Abelian categories

- We could answer the above questions if we would have an oracle for computing in free Abelian categories:

Software demo

`https://homalg-project.github.io/nb/SnakeInFreeAbelian`

An oracle for free Abelian categories

- We could answer the above questions if we would have an oracle for computing in free Abelian categories:

Software demo

`https://homalg-project.github.io/nb/SnakeInFreeAbelian`

Exercise: Along the same lines treat spectral sequences of bicomplexes.

So we have a **proof by computation** of the snake lemma.

A categorical tower for `AbelianClosure`

So we have a **proof by computation** of the snake lemma. But

- How to build the category constructor `AbelianClosure`?

A categorical tower for `AbelianClosure`

So we have a **proof by computation** of the snake lemma. But

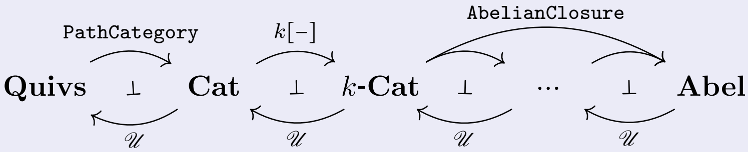
- How to build the category constructor `AbelianClosure`?
- What do we need to extract a **construction algorithm** for the connecting morphism s in any Abelian category?

A categorical tower for `AbelianClosure`

So we have a **proof by computation** of the snake lemma. But

- How to build the category constructor `AbelianClosure`?
- What do we need to extract a **construction algorithm** for the connecting morphism s in any Abelian category?

The answer is to build the category constructor `AbelianClosure` as a **categorical tower** of 2-*adjunctions*:

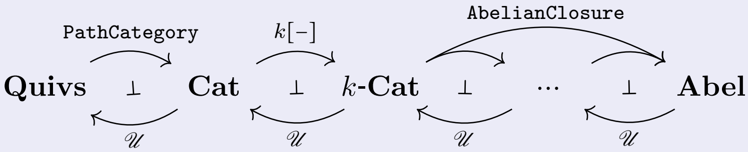


A categorical tower for `AbelianClosure`

So we have a **proof by computation** of the snake lemma. But

- How to build the category constructor `AbelianClosure`?
- What do we need to extract a **construction algorithm** for the connecting morphism s in any Abelian category?

The answer is to build the category constructor `AbelianClosure` as a **categorical tower** of 2-*adjunctions*:



The **co**unit of such a composed 2-adjunction will turn out to be the desired **ur-algorithm**, having the snake lemma, spectral sequences, and many more algorithms as special cases.

Free-forgetful 2-adjunctions

- The above tower of categorical constructors is typically composed of several free-forgetful 2-adjunctions

$$\mathcal{D} \underset{\mathscr{U}}{\overset{\mathscr{L}}{\underset{\longleftarrow}{\overset{\longrightarrow}{\perp}}}} \mathcal{E}$$

  between a 2-category $\mathcal{D}$ of categories (called **doctrine**) and another doctrine $\mathcal{E}$ of categories with extra structure.

Free-forgetful 2-adjunctions

- The above tower of categorical constructors is typically composed of several free-forgetful 2-adjunctions

$$\mathcal{D} \underset{\mathscr{U}}{\overset{\mathscr{L}}{\underset{\bot}{\rightleftarrows}}} \mathcal{E}$$

  between a 2-category $\mathcal{D}$ of categories (called **doctrine**) and another doctrine $\mathcal{E}$ of categories with extra structure.

We will next see an instructive example of such a 2-adjunction.

Coproducts in categories

Coproducts are generalizations of joins in posets (e.g., lcm's):

$$D_0 \qquad \cdots \qquad D_{\ell-1}$$

Coproducts in categories

Coproducts are generalizations of joins in posets (e.g., lcm's):

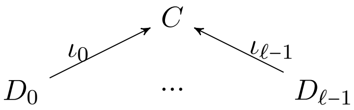
$$C$$

$$D_0 \qquad \cdots \qquad D_{\ell-1}$$

0 `Coproduct`

Coproducts in categories

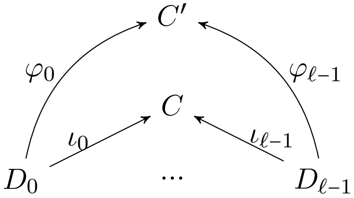
Coproducts are generalizations of joins in posets (e.g., lcm's):



0. `Coproduct`
1. `InjectionOfCofactorOfCoproduct`

Coproducts in categories

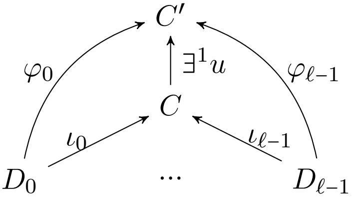
Coproducts are generalizations of joins in posets (e.g., lcm's):



0. `Coproduct`
1. `InjectionOfCofactorOfCoproduct`

Coproducts in categories

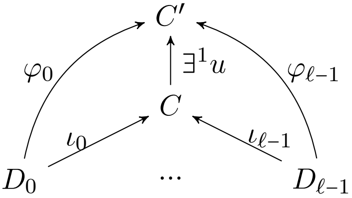
Coproducts are generalizations of joins in posets (e.g., lcm's):



0. `Coproduct`
1. `InjectionOfCofactorOfCoproduct`
2. `UniversalMorphismFromCoproduct`

Coproducts in categories

Coproducts are generalizations of joins in posets (e.g., lcm’s):



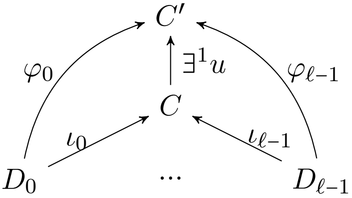
0. `Coproduct`
1. `InjectionOfCofactorOfCoproduct`
2. `UniversalMorphismFromCoproduct`

There is a bijection $(D_i \xrightarrow{\varphi_i} C')_{i=0}^{\ell-1} \quad \leftrightarrow \quad C \xrightarrow{u} C'$.

- 1’ `ComponentOfMorphismFromCoproduct` (analysis/elim.)
- 2 `UniversalMorphismFromCoproduct` (synthesis/intro.)

Coproducts in categories

Coproducts are generalizations of joins in posets (e.g., lcm's):



0. `Coproduct`
1. `InjectionOfCofactorOfCoproduct`
2. `UniversalMorphismFromCoproduct`

There is a bijection $(D_i \xrightarrow{\varphi_i} C')_{i=0}^{\ell-1} \quad \leftrightarrow \quad C \xrightarrow{u} C'$.

- 1' `ComponentOfMorphismFromCoproduct` (analysis/elim.)
- 2 `UniversalMorphismFromCoproduct` (synthesis/intro.)

Is there a way to package all 3 algorithms in one *ur-algorithm*?

The finite coproduct completion

Definition (Cocartesian (monoidal))

A **cocartesian** category is a category with all finite coproducts.

The finite coproduct completion

Definition (Cocartesian (monoidal))

A **cocartesian** category is a category with all finite coproducts.

Definition

Denote by $\mathbf{Cocart}$ the category of cocartesian categories (as objects) and coproduct preserving functors (as morphisms).

The finite coproduct completion

Definition (Cocartesian (monoidal))

A **cocartesian** category is a category with all finite coproducts.

Definition

Denote by $\mathbf{Cocart}$ the category of cocartesian categories (as objects) and coproduct preserving functors (as morphisms).

There exists a free-forgetful 2-adjunction

$$\mathscr{L} = \texttt{FiniteCoproductCompletion}$$

$$\mathbf{Cat} \underset{\mathscr{U}}{\overset{\mathscr{L}}{\rightleftarrows}} \mathbf{Cocart}, \quad \mathscr{L} \dashv \mathscr{U}$$

$$\mathscr{U} = \texttt{UnderlyingCategory}$$

FiniteStrictCoproductCompletion

`FiniteStrictCoproductCompletion`(**D**) is built syntactically

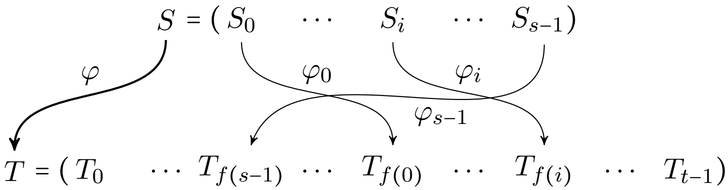
`FiniteStrictCoproductCompletion`(**D**) is built syntactically:

- An object is a finite list $D = (D_0, \ldots, D_{\ell-1})$ of objects in **D**.

FiniteStrictCoproductCompletion

`FiniteStrictCoproductCompletion`(**D**) is built syntactically:

- An object is a finite list $D = (D_0, \dots, D_{\ell-1})$ of objects in **D**.
- A morphism $\varphi : S \to T$ is a wiring diagram

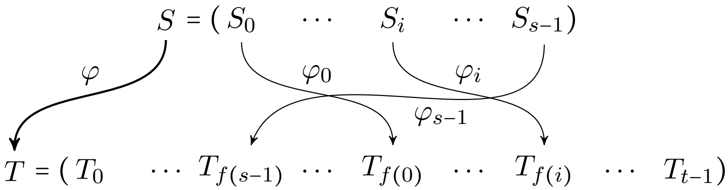


defined by a function $f : \{0, \dots, s-1\} \to \{0, \dots, t-1\}$ and labeled by a list of morphisms $(\varphi_i : S_i \to T_{f(i)})_{i=0}^{s-1} \in$ **D**.

FiniteStrictCoproductCompletion

`FiniteStrictCoproductCompletion`(**D**) is built syntactically:

- An object is a finite list $D = (D_0, \dots, D_{\ell-1})$ of objects in **D**.
- A morphism $\varphi : S \to T$ is a wiring diagram



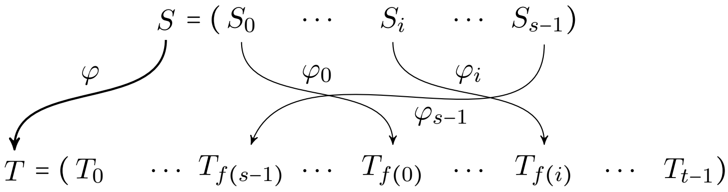
defined by a function $f : \{0, \dots, s-1\} \to \{0, \dots, t-1\}$ and labeled by a list of morphisms $(\varphi_i : S_i \to T_{f(i)})_{i=0}^{s-1} \in$ **D**.

```
SkeletalFinSets =
FiniteStrictCoproductCompletion(TerminalCategory)
```

FiniteStrictCoproductCompletion

`FiniteStrictCoproductCompletion`(**D**) is built syntactically:

- An object is a finite list $D = (D_0, \dots, D_{\ell-1})$ of objects in **D**.
- A morphism $\varphi : S \to T$ is a wiring diagram



defined by a function $f : \{0, \dots, s-1\} \to \{0, \dots, t-1\}$ and labeled by a list of morphisms $(\varphi_i : S_i \to T_{f(i)})_{i=0}^{s-1} \in$ **D**.

`SkeletalFinSets = FiniteStrictCoproductCompletion(TerminalCategory)`

The (finite) coproduct completion invents functions.

The 2-adjunction

For a strict cocartesian category **E** and a functor $F : \mathbf{D} \to \mathscr{U}(\mathbf{E})$ in $\mathbf{Cat}$ the adjunct functor

$$\widehat{F} \coloneqq \mathscr{L}(F)\varepsilon_{\mathbf{E}} : \mathtt{FiniteStrictCoproductCompletion}(\mathbf{D}) \to \mathbf{E}$$

in $\mathbf{Cocart}$ is given by

$$D = (D_0, \dots, D_{\ell-1}) \overset{\mathscr{L}(F)}{\mapsto} F(D) \coloneqq (F(D_0), \dots, F(D_{\ell-1})) \overset{\varepsilon_{\mathbf{E}}}{\mapsto} \mathtt{Coproduct}(F(D)) = \coprod_{i=0}^{\ell-1} F(D_i)$$

The 2-adjunction

For a strict cocartesian category **E** and a functor $F : \mathbf{D} \to \mathscr{U}(\mathbf{E})$ in $\mathbf{Cat}$ the adjunct functor

$$\widehat{F} := \mathscr{L}(F)\varepsilon_{\mathbf{E}} : \texttt{FiniteStrictCoproductCompletion}(\mathbf{D}) \to \mathbf{E}$$

in $\mathbf{Cocart}$ is given by

$$D = (D_0, \ldots, D_{\ell-1}) \overset{\mathscr{L}(F)}{\mapsto} F(D) := (F(D_0), \ldots, F(D_{\ell-1}))$$

$$\overset{\varepsilon_{\mathbf{E}}}{\mapsto} \texttt{Coproduct}(F(D)) = \coprod_{i=0}^{\ell-1} F(D_i)$$

For a morphism $\varphi : S \to T$ use

- $\texttt{InjectionOfCofactorOfCoproduct}$ to construct the compositions $F(S_i) \to F(T_{f(i)}) \xrightarrow{\iota_{f(i)}} \coprod_{j=0}^{t-1} F(T_j) =: \widehat{F}(T)$

The 2-adjunction

For a strict cocartesian category **E** and a functor $F : \mathbf{D} \to \mathscr{U}(\mathbf{E})$ in $\mathbf{Cat}$ the adjunct functor

$$\widehat{F} \coloneqq \mathscr{L}(F)\varepsilon_{\mathbf{E}} : \texttt{FiniteStrictCoproductCompletion}(\mathbf{D}) \to \mathbf{E}$$

in $\mathbf{Cocart}$ is given by

$$D = (D_0, \ldots, D_{\ell-1}) \overset{\mathscr{L}(F)}{\mapsto} F(D) \coloneqq (F(D_0), \ldots, F(D_{\ell-1}))$$
$$\overset{\varepsilon_{\mathbf{E}}}{\mapsto} \texttt{Coproduct}(F(D)) = \coprod_{i=0}^{\ell-1} F(D_i)$$

For a morphism $\varphi : S \to T$ use

- `InjectionOfCofactorOfCoproduct` to construct the compositions $F(S_i) \to F(T_{f(i)}) \xrightarrow{\iota_{f(i)}} \coprod_{j=0}^{t-1} F(T_j) \eqqcolon \widehat{F}(T)$
- `UniversalMorphismFromCoproduct` to construct the universal morphism $\widehat{F}(S) \xrightarrow{\widehat{F}(\varphi)} \widehat{F}(T)$

The 2-adjunction

For a strict cocartesian category **E** and a functor $F : \mathbf{D} \to \mathscr{U}(\mathbf{E})$ in $\mathbf{Cat}$ the adjunct functor

$$\widehat{F} \coloneqq \mathscr{L}(F)\varepsilon_{\mathbf{E}} : \texttt{FiniteStrictCoproductCompletion}(\mathbf{D}) \to \mathbf{E}$$

in $\mathbf{Cocart}$ is given by

$$D = (D_0, \ldots, D_{\ell-1}) \overset{\mathscr{L}(F)}{\mapsto} F(D) \coloneqq (F(D_0), \ldots, F(D_{\ell-1}))$$
$$\overset{\varepsilon_{\mathbf{E}}}{\mapsto} \texttt{Coproduct}(F(D)) = \coprod_{i=0}^{\ell-1} F(D_i)$$

For a morphism $\varphi : S \to T$ use

- `InjectionOfCofactorOfCoproduct` to construct the compositions $F(S_i) \to F(T_{f(i)}) \xrightarrow{\iota_{f(i)}} \coprod_{j=0}^{t-1} F(T_j) =: \widehat{F}(T)$
- `UniversalMorphismFromCoproduct` to construct the universal morphism $\widehat{F}(S) \xrightarrow{\widehat{F}(\varphi)} \widehat{F}(T)$

The counit is the *ur-algorithm*, evaluating syntax into semantics!

Polynomial functors

The dual category construction is also a 2-adjunction on each doctrine

$$\mathcal{D} \underset{\mathscr{R}\,=\,\texttt{Opposite}}{\overset{\mathscr{L}\,=\,\texttt{Opposite}}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathcal{D}^{\mathrm{co-dual}}$$

Polynomial functors

The dual category construction is also a 2-adjunction on each doctrine

$$\mathscr{L} = \texttt{Opposite}$$

$$\mathcal{D} \underset{\longleftarrow}{\overset{\longrightarrow}{\perp}} \mathcal{D}^{\text{co}-\text{dual}}$$

$$\mathscr{R} = \texttt{Opposite}$$

Implementing `Opposite` requires a lot of meta programming.

Polynomial functors

The dual category construction is also a 2-adjunction on each doctrine

$$\mathcal{D} \underset{\mathscr{R} = \texttt{Opposite}}{\overset{\mathscr{L} = \texttt{Opposite}}{\rightleftarrows}} \mathcal{D}^{\mathrm{co-dual}} \qquad (\perp)$$

Implementing `Opposite` requires a lot of meta programming.

Define:

- $\texttt{ProducCompletion} :=$ $\texttt{Opposite} \circ \texttt{CoproducCompletion} \circ \texttt{Opposite}$

Polynomial functors

The dual category construction is also a 2-adjunction on each doctrine

$$\mathcal{D} \underset{\mathscr{R} = \texttt{Opposite}}{\overset{\mathscr{L} = \texttt{Opposite}}{\rightleftarrows}} \mathcal{D}^{\mathrm{co-dual}} \qquad (\mathscr{L} \dashv \mathscr{R})$$

Implementing `Opposite` requires a lot of meta programming.

Define:

- $\texttt{ProducCompletion} :=$ $\texttt{Opposite} \circ \texttt{CoproducCompletion} \circ \texttt{Opposite}$
- $\texttt{DistributiveCompletion} :=$ $\texttt{CoproducCompletion} \circ \texttt{ProducCompletion}$

Polynomial functors

The dual category construction is also a 2-adjunction on each doctrine

$$\mathcal{D} \underset{\mathscr{R} = \texttt{Opposite}}{\overset{\mathscr{L} = \texttt{Opposite}}{\rightleftarrows}} \mathcal{D}^{\mathrm{co-dual}} \qquad (\mathscr{L} \dashv \mathscr{R})$$

Implementing `Opposite` requires a lot of meta programming.

Define:

- $\texttt{ProducCompletion} \coloneqq \texttt{Opposite} \circ \texttt{CoproducCompletion} \circ \texttt{Opposite}$
- $\texttt{DistributiveCompletion} \coloneqq \texttt{CoproducCompletion} \circ \texttt{ProducCompletion}$
- $\texttt{Poly} \coloneqq \texttt{DistributiveCompletion}(\texttt{TerminalCategory})$

The 2-adjunctions

- The left 2-adjoint $\mathscr{L}(\mathbf{D})$ is the free category in $\mathcal{E}$ (of type $\mathcal{E}$) generated by $\mathbf{D} \in \mathcal{D}$. The data structures of the free model $\mathscr{L}(\mathbf{D})$ are purely **syntactic**.

1

The 2-adjunctions

- The left 2-adjoint $\mathscr{L}(\mathbf{D})$ is the free category in $\mathcal{E}$ (of type $\mathcal{E}$) generated by $\mathbf{D} \in \mathcal{D}$. The data structures of the free model $\mathscr{L}(\mathbf{D})$ are purely **syntactic**.
- The **co**unit $\varepsilon_{\mathbf{E}} : \mathscr{L}(\mathscr{U}(\mathbf{E})) \to \mathrm{Id}_{\mathbf{E}}$ evaluates[1] syntax into **semantics**.

[1] it is sometimes called the evaluation morphism for other reasons

The 2-adjunctions

- The left 2-adjoint $\mathscr{L}(\mathbf{D})$ is the free category in $\mathcal{E}$ (of type $\mathcal{E}$) generated by $\mathbf{D} \in \mathcal{D}$. The data structures of the free model $\mathscr{L}(\mathbf{D})$ are purely **syntactic**.
- The **co**unit $\varepsilon_{\mathbf{E}} : \mathscr{L}(\mathscr{U}(\mathbf{E})) \to \mathrm{Id}_{\mathbf{E}}$ evaluates[1] syntax into **semantics**.
- The proof is computed in the syntactic model $\mathscr{L}(\mathbf{D})$.

[1] it is sometimes called the evaluation morphism for other reasons

The 2-adjunctions

- The left 2-adjoint $\mathscr{L}(\mathbf{D})$ is the free category in $\mathcal{E}$ (of type $\mathcal{E}$) generated by $\mathbf{D} \in \mathcal{D}$. The data structures of the free model $\mathscr{L}(\mathbf{D})$ are purely **syntactic**.
- The **co**unit $\varepsilon_{\mathbf{E}} : \mathscr{L}(\mathscr{U}(\mathbf{E})) \to \mathrm{Id}_{\mathbf{E}}$ evaluates[1] syntax into **semantics**.
- The proof is computed in the syntactic model $\mathscr{L}(\mathbf{D})$.
- The evaluation into semantics is the **program extraction** (our explicit version of the Curry-Howard correspondence).

[1] it is sometimes called the evaluation morphism for other reasons

Extracting the snake lemma program

Having constructed the connecting morphism s in the *syntacticly* free model

$$\mathscr{L}(\mathbf{D}) = \texttt{AbelianClosure}(\underbrace{\texttt{Algebroid}_{\mathbb{Q}}(A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{c} D)/abc}_{\mathbf{D}})$$

we can now apply our evaluating counit

$$\varepsilon_{\mathscr{L}(\mathbf{D})} : \mathscr{L}(\mathscr{U}(\mathscr{L}(\mathbf{D}))) \to \mathscr{L}(\mathbf{D})$$

to the syntactic s an extract the program

$$\begin{aligned}
&\texttt{ConnectingMorphism}(a, b, c) :=\\
&\quad \texttt{CokernelColift}(\\
&\qquad \texttt{KernelLift}(b \cdot c, a),\\
&\qquad \texttt{KernelLift}(c, \texttt{KernelEmbedding}(b \cdot c) \cdot b) \cdot\\
&\qquad\quad \texttt{CokernelProjection}(\texttt{KernelLift}(c, a \cdot b)))
\end{aligned}$$

(up to some rewriting rules in $\texttt{AbelianClosure}(\mathbf{D})$).

Thank you