Recursive Function Classes in Cartesian Categories

Joaquín Díaz Boïls

Universitat de València

MAP 2010, Logroño, November

Plan

Plan

- What do we need for recursion in CT

Plan

- **What do we need for recursion in CT**
  - natural numbers object diagrams
  - representation
  - various Cartesian Categories

Plan

- What do we need for recursion in CT
  - natural numbers object diagrams
  - representation
  - various Cartesian Categories
- From syntactic to semantic

Plan

- **What do we need for recursion in CT**
  - natural numbers object diagrams
  - representation
  - various Cartesian Categories
- **From syntactic to semantic**
  - free and syntactical structures
  - categories of Algorithms

Plan

- **What do we need for recursion in CT**
  - natural numbers object diagrams
  - representation
  - various Cartesian Categories
- **From syntactic to semantic**
  - free and syntactical structures
  - categories of Algorithms
- **Subrecursion**

Plan

- **What do we need for recursion in CT**
  - natural numbers object diagrams
  - representation
  - various Cartesian Categories
- **From syntactic to semantic**
  - free and syntactical structures
  - categories of Algorithms
- **Subrecursion**
  - translations to CT
  - Polarized Categories

The nno diagrams

In a Cartesian Category (*cc*) we have composition and a product

The nno diagrams

In a Cartesian Category (*cc*) we have composition and a product

Recursion operator is obtained by means of an *nno*

The nno diagrams

In a Cartesian Category (cc) we have composition and a product

Recursion operator is obtained by means of an *nno*

Definition

A *nno* in a category $\mathcal{C}$ with terminal object 1 is (N, z, s) with a commutative diagram

The nno diagrams

In a Cartesian Category (*cc*) we have composition and a product

Recursion operator is obtained by means of an *nno*

Definition

A *nno* in a category $\mathcal{C}$ with terminal object 1 is (N, z, s) with a commutative diagram

$$\begin{array}{ccccc} 1 & \xrightarrow{z} & N & \xrightarrow{s} & N \\ \downarrow & & \downarrow m & & \downarrow m \\ 1 & \xrightarrow{f} & A & \xrightarrow{g} & A \end{array}$$

where m is unique

The nno diagrams

Basic recursive structure is a *parametrized nno*

The nno diagrams

Basic recursive structure is a *parametrized nno*

Definition

A *parametrized nno* (pnno) is an *nno* (N, z, s) for which there exists a commutative diagram

The nno diagrams

Basic recursive structure is a *parametrized nno*

Definition

A *parametrized nno* (pnno) is an *nno* (N, z, s) for which there exists a commutative diagram

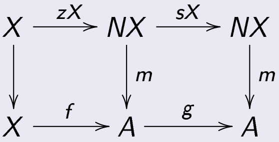
$$\begin{array}{ccccc} X & \xrightarrow{zX} & NX & \xrightarrow{sX} & NX \\ \downarrow & & \downarrow m & & \downarrow m \\ X & \xrightarrow{f} & A & \xrightarrow{g} & A \end{array}$$

The nno diagrams

Basic recursive structure is a *parametrized nno*

Definition

A *parametrized nno* (pnno) is an *nno* (N, z, s) for which there exists a commutative diagram



We can speak of a weak nno (wnno) if uniqueness is not required

The nno diagrams

These definitions can be generalized

The nno diagrams

These definitions can be generalized

Definition

A *left nno* (*lnno*) in a *Monoidal Category* $\mathcal{V} = (\mathcal{V}, \otimes, I, \alpha, \lambda, \varrho)$ **is** (N, z, s) **such that**

The nno diagrams

These definitions can be generalized

Definition

A *left nno* (*lnno*) in a *Monoidal Category* $\mathcal{V} = (\mathcal{V}, \otimes, I, \alpha, \lambda, \varrho)$ **is** (N, z, s) **such that**

$$\begin{array}{ccccc} I \otimes A & \xrightarrow{z \otimes A} & N \otimes A & \xrightarrow{s \otimes A} & N \otimes A \\ {\scriptstyle \lambda}\big\downarrow & & \big\downarrow{\scriptstyle m} & & \big\downarrow{\scriptstyle m} \\ A & \xrightarrow{\quad f \quad} & B & \xrightarrow{\quad g \quad} & B \end{array}$$

The nno diagrams

These definitions can be generalized

Definition

A *left nno* (*lnno*) in a *Monoidal Category* $\mathcal{V} = (\mathcal{V}, \otimes, I, \alpha, \lambda, \varrho)$ **is** (N, z, s) **such that**

$$
\begin{array}{ccccc}
I \otimes A & \xrightarrow{z \otimes A} & N \otimes A & \xrightarrow{s \otimes A} & N \otimes A \\
\downarrow \lambda & & \downarrow m & & \downarrow m \\
A & \xrightarrow{f} & B & \xrightarrow{g} & B
\end{array}
$$

commutes

Representation

We define recursive function classes in $cc + nno$ by its *representation*

Representation

We define recursive function classes in $cc + nno$ by its *representation*

Definition

We say that $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is *representable* in a $cc + nno$ $\mathcal{C}$ if there exists $\overline{f} : N^k \longrightarrow N$ in $\mathcal{C}$ such that

$$\overline{f}\,\langle \sharp n_1, \ldots, \sharp n_k \rangle = \sharp f(n_1, \ldots, n_k)$$

The results

Given categorical structures we obtain recursive function classes:

The results

Given categorical structures we obtain recursive function classes:

For total function classes in the form $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ we have:

The results

Given categorical structures we obtain recursive function classes:

For total function classes in the form $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ we have:

1. $$\mathcal{PR} = \{\textit{Representables in } cc{+}wpnno\}$$

2. $$\mathcal{PR} \subseteq \{\textit{Representables in Topos+nno}\} \subset \textit{TotalRec}$$

The results

For partial functions we have:

The results

For partial functions we have:

$$\mathit{PartialRec} \subseteq \{\mathit{Representables}\ in\ cc{+}nno{+}equalizers\}$$

The results

For partial functions we have:

$$PartialRec \subseteq \{Representables\ in\ cc{+}nno{+}equalizers\}$$

And for a different kind of representation:

The results

For partial functions we have:

$$PartialRec \subseteq \{Representables\ in\ cc{+}nno{+}equalizers\}$$

And for a different kind of representation:

$$PartialRec = \{Numeralwise\ Representables\ in\ cc{+}wnno{+}equalizers\}$$

Free and syntactical structures

Two constructions

Free and syntactical structures

Two constructions

1. over *Grph*

Free and syntactical structures

Two constructions

1. over *Grph*

Peano-Lawvere Axiom in a category

A category satisfies PL Axiom (or it is PL) if every object has an nno

Free and syntactical structures

Two constructions

1. over *Grph*

Peano-Lawvere Axiom in a category

A category satisfies PL Axiom (or it is PL) if every object has an nno

PL categories are not in general cartesian nor are they endowed with a terminal object

Free and syntactical structures

Two constructions

1. over *Grph*

Peano-Lawvere Axiom in a category

A category satisfies PL Axiom (or it is PL) if every object has an nno

PL categories are not in general cartesian nor are they endowed with a terminal object

However $P = F_{PL}(\cdot)$ has both

Free and syntactical structures

Let be $P = F_{PL}(\cdot)$ the *Category of $\mathcal{PR}$-formal functions*

Free and syntactical structures

Let be $P = F_{PL}(\cdot)$ the *Category of $\mathcal{PR}$-formal functions*

Definition

Free and syntactical structures

Let be $P = F_{PL}(\cdot)$ the *Category of $\mathcal{PR}$-formal functions*

Definition

We define P' as the PL *precategory* generated $(\cdot)$

Free and syntactical structures

Let be $P = F_{PL}(\cdot)$ the *Category of $\mathcal{PR}$-formal functions*

Definition

We define P' as the PL *precategory* generated $(\cdot)$
it will be called *precategory of $\mathcal{PR}$-programs*

Free and syntactical structures

Lema

We call $\mathbb{P}$ *the image graph* $P \longrightarrow Set$

Free and syntactical structures

Lema

We call $\mathbb{P}$ *the image graph* $P \longrightarrow Set$

- *its objects are* $\mathbb{N}^p$

Free and syntactical structures

Lema

We call $\mathbb{P}$ *the image graph* $P \longrightarrow Set$

- *its objects are* $\mathbb{N}^p$
- *its morphisms* $f : \mathbb{N}^p \longrightarrow \mathbb{N}^q$ *such that* $f = (f_0, f_1, ..., f_{q-1})$ *where* $f_i \in \mathcal{PR}$

Free and syntactical structures

Lema

We call $\mathbb{P}$ *the image graph* $P \longrightarrow Set$

- *its objects are* $\mathbb{N}^p$
- *its morphisms* $f : \mathbb{N}^p \longrightarrow \mathbb{N}^q$ *such that* $f = (f_0, f_1, \ldots, f_{q-1})$ *where* $f_i \in \mathcal{PR}$

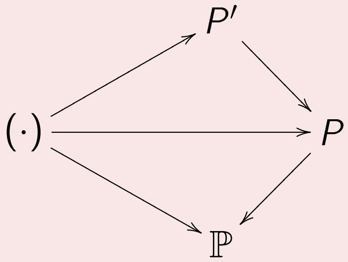
$\mathbb{P}$ can (only?) be characterized by equivalence relations in P

Free and syntactical structures

We can summarize as

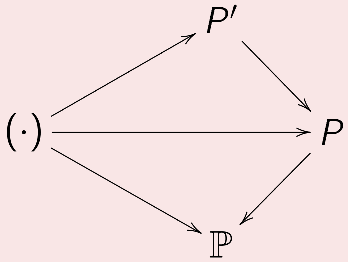
Free and syntactical structures

We can summarize as



Free and syntactical structures

We can summarize as



P' is a syntactical construction while $\mathbb{P}$ is a category with semantics

Categories of Algorithms

2. over *Set*

Categories of Algorithms

2. over *Set*

Concept of algorithm is bounded by

Categories of Algorithms

2. over *Set*

Concept of algorithm is bounded by

Categories of Algorithms

2. over *Set*

Concept of algorithm is bounded by

1. the implementations that we handle: programs

Categories of Algorithms

2. over *Set*

Concept of algorithm is bounded by

1. the implementations that we handle: programs
2. formalizations that are known: recursive functions

Categories of Algorithms

There is a tree for any $\mathcal{PR}$-program labelling the edges with

Categories of Algorithms

There is a tree for any $\mathcal{PR}$-program labelling the edges with

C (composition), **R** (recursion) and **B** (bracket)

Categories of Algorithms

There is a tree for any $\mathcal{PR}$-program labelling the edges with

C (composition), **R** (recursion) and **B** (bracket)

Categories of Algorithms

There is a tree for any $\mathcal{PR}$-program labelling the edges with

C (composition), **R** (recursion) and **B** (bracket)

- nodes are $\mathbb{N}^k$

Categories of Algorithms

There is a tree for any $\mathcal{PR}$-program labelling the edges with

C (composition), **R** (recursion) and **B** (bracket)

- nodes are $\mathbb{N}^k$
- edges are $\mathcal{PR}$-functions generated from z, s and π_i^k

Categories of Algorithms

There is a tree for any $\mathcal{PR}$-program labelling the edges with

C (composition), **R** (recursion) and **B** (bracket)

- nodes are $\mathbb{N}^k$
- edges are $\mathcal{PR}$-functions generated from z, s and π_i^k

Definition

We call this graph $\mathcal{PR}desc$

Categories of Algorithms

There is a tree for any $\mathcal{PR}$-program labelling the edges with

C (composition), **R** (recursion) and **B** (bracket)

- nodes are $\mathbb{N}^k$
- edges are $\mathcal{PR}$-functions generated from z, s and π_i^k

Definition

We call this graph $\mathcal{PR}desc$
it has all descriptions about how to compute all $\mathcal{PR}$-functions

Categories of Algorithms

Essentially equal $\sim$ in $\mathcal{PR}desc$ gives equivalence classes

Categories of Algorithms

Essentially equal $\sim$ in $\mathcal{PR}desc$ gives equivalence classes

We construct by *prunning*

Categories of Algorithms

Essentially equal $\sim$ in $\mathcal{PR}desc$ gives equivalence classes

We construct by *prunning*

$$\frac{\mathcal{PR}desc}{\sim} = \mathcal{PR}Alg$$

as the free initial category $F_{CatxN}(\emptyset)$ in $CatxN$

Categories of Algorithms

Reducing by $\approx$ *(run the same operation)* we have

Categories of Algorithms

Reducing by $\approx$ *(run the same operation)* we have

$$\frac{\mathcal{PR}Alg}{\approx} = \mathcal{PR}Func$$

Categories of Algorithms

Reducing by $\approx$ *(run the same operation)* we have

$$\frac{\mathcal{PR}Alg}{\approx} = \mathcal{PR}Func$$

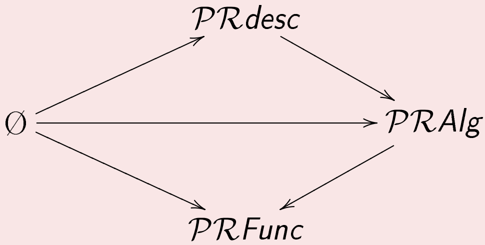
and the schema

Categories of Algorithms

Reducing by $\approx$ *(run the same operation)* we have

$$\frac{\mathcal{PR}Alg}{\approx} = \mathcal{PR}Func$$

and the schema



Free and syntactical structures

Similar constructions

Free and syntactical structures

Similar constructions

- initial PRU $\mathcal{E}$ (Pfender)

Free and syntactical structures

Similar constructions

- initial PRU $\mathcal{E}$ (Pfender)
- *Freyd Cover* $\mathcal{FC}$ from a cc $\mathcal{C}$ + *nno* (Román)

Free and syntactical structures

Similar constructions

- initial PRU $\mathcal{E}$ (Pfender)
- *Freyd Cover* $\mathcal{FC}$ from a cc $\mathcal{C}$ + *nno* (Román)
- *free Monoidal Category* + *lnno* $\Phi(\emptyset)$ (Román-Paré)

Translations to CT

Classes in *Gzregorzcyk Hierarchy* can be defined by *bounding arithmetics*

Translations to CT

Classes in *Gzregorzcyk Hierarchy* can be defined by *bounding arithmetics*

Consider the smallest derivations set

Translations to CT

Classes in *Gzregorzcyk Hierarchy* can be defined by *bounding arithmetics*

Consider the smallest derivations set

1. containing a derivation of every initial function 0, $Sx = x + 1$, $Px = max(0, x - 1)$ and *conditional* $C(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{else} \end{cases}$

Translations to CT

Classes in *Gzregorzcyk Hierarchy* can be defined by *bounding arithmetics*

Consider the smallest derivations set

1. containing a derivation of every initial function 0, $Sx = x + 1$, $Px = max(0, x - 1)$ and *conditional* $C(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{else} \end{cases}$
2. closed under the derivation rules

Translations to CT

Classes in *Gzregorzcyk Hierarchy* can be defined by *bounding arithmetics*

Consider the smallest derivations set

1. containing a derivation of every initial function 0, $Sx = x + 1$, $Px = max(0, x - 1)$ and *conditional* $C(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{else} \end{cases}$
2. closed under the derivation rules
    1. *full composition*: given derivations h and $g_1, ..., g_m$ we derive

$$f(\overline{x}) = h(g_1(\overline{x}^1), ..., g_m(\overline{x}^m))$$

Translations to CT

Classes in *Gzregorzcyk Hierarchy* can be defined by *bounding arithmetics*

Consider the smallest derivations set

1. containing a derivation of every initial function 0, $Sx = x + 1$, $Px = max(0, x - 1)$ and *conditional* $C(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{else} \end{cases}$
2. closed under the derivation rules
   1. *full composition*: given derivations h and $g_1, ..., g_m$ we derive
   $$f(\overline{x}) = h(g_1(\overline{x}^1), ..., g_m(\overline{x}^m))$$
   2. *full primitive recursiveness:* for g and h if $x \neq 0$ we derive
   $$f(x, \overline{y}) = h(x, \overline{y}, f(Px, \overline{y}))$$

Translations to CT

Complexity lower than $\mathcal{PR}$ can be modeled using *ramified recursion*

Translations to CT

Complexity lower than $\mathcal{PR}$ can be modeled using *ramified recursion*

it is based in *comprehension* or

Translations to CT

Complexity lower than $\mathcal{PR}$ can be modeled using *ramified recursion*

it is based in *comprehension* or

Axiom schema of specification

Translations to CT

Complexity lower than $\mathcal{PR}$ can be modeled using *ramified recursion*

it is based in *comprehension* or

Axiom schema of specification

Any definable subclass of a set is a set

Translations to CT

Complexity lower than $\mathcal{PR}$ can be modeled using *ramified recursion*

it is based in *comprehension* or

Axiom schema of specification

Any definable subclass of a set is a set

Instead of *bounding induction* we use weaker subsystems

Translations to CT

Complexity lower than $\mathcal{PR}$ can be modeled using *ramified recursion*

it is based in *comprehension* or

Axiom schema of specification

Any definable subclass of a set is a set

Instead of *bounding induction* we use weaker subsystems

We use two kinds of arguments: *normal* and *safe*

Translations to CT

Definition

Translations to CT

Definition

Subsets $\mathbb{N}_{k+1}$ that *make recursion* having $\mathbb{N}_o, ..., \mathbb{N}_k$ are called *tiers*

Translations to CT

Definition

Subsets $\mathbb{N}_{k+1}$ that *make recursion* having $\mathbb{N}_o, \ldots, \mathbb{N}_k$ are called *tiers*

We calculate *tier of a derivation* $f \in \mathcal{PR}$ as

Translations to CT

Definition

Subsets $\mathbb{N}_{k+1}$ that *make recursion* having $\mathbb{N}_o, ..., \mathbb{N}_k$ are called *tiers*

We calculate *tier of a derivation* $f \in \mathcal{PR}$ as

- $\rho(f) = 0$ if f is an initial function

Translations to CT

Definition

Subsets $\mathbb{N}_{k+1}$ that *make recursion* having $\mathbb{N}_o, ..., \mathbb{N}_k$ are called *tiers*

We calculate *tier of a derivation* $f \in \mathcal{PR}$ as

- $\rho(f) = 0$ if f is an initial function
- $\rho(f) = max\{\rho(h), \rho(g_1), ..., \rho(g_m)\}$ if f is defined by full composition of derivations h and $g_1, ..., g_m$

Translations to CT

Definition

Subsets $\mathbb{N}_{k+1}$ that *make recursion* having $\mathbb{N}_o, ..., \mathbb{N}_k$ are called *tiers*

We calculate *tier of a derivation* $f \in \mathcal{PR}$ as

- $\rho(f) = 0$ if f is an initial function
- $\rho(f) = max\{\rho(h), \rho(g_1), ..., \rho(g_m)\}$ if f is defined by full composition of derivations h and $g_1, ..., g_m$
- $\rho(f) = max\{\rho(g), 1 + \rho(h)\}$ if f is defined by full primitive recursion of derivations g and h

Translations to CT

ε_1, ε_2 and ε_3 of Gzregorzcyk have been constructed from

Translations to CT

ε_1, ε_2 and ε_3 of Gzregorzcyk have been constructed from

Doctrines with

Translations to CT

ε_1, ε_2 and ε_3 of Gzregorzcyk have been constructed from

Doctrines with

1. an *SM 2-Comprehension*

Translations to CT

ε_1, ε_2 and ε_3 of Gzregorzcyk have been constructed from

Doctrines with

1. an *SM 2-Comprehension*
2. two *tiers* N_0, N_1 of numerals with *dyadics*

$$I \xrightarrow{z} N_k \xrightarrow{s_k} N_k$$

Translations to CT

ε_1, ε_2 and ε_3 of Gzregorzcyk have been constructed from

Doctrines with

1. an *SM 2-Comprehension*
2. two *tiers* N_0, N_1 of numerals with *dyadics*

$$I \xrightarrow{z} N_k \xrightarrow{s_k} N_k$$

3. *ramified recursion*

Translations to CT

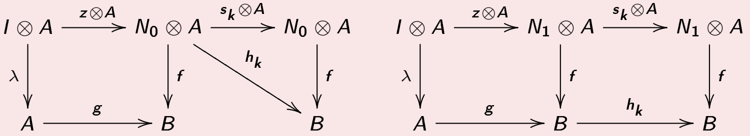
ε_1, ε_2 and ε_3 of Gzregorzcyk have been constructed from

Doctrines with

1. an *SM 2-Comprehension*
2. two *tiers* N_0, N_1 of numerals with *dyadics*

$$I \xrightarrow{z} N_k \xrightarrow{s_k} N_k$$

3. *ramified recursion*



Polarized Categories

Polarization is a way to see ramification in CT

Polarized Categories

Polarization is a way to see ramification in CT

Definition

A *Polarized Strong Category* (*SPolCat*) consists of

Polarized Categories

Polarization is a way to see ramification in CT

Definition

A *Polarized Strong Category* (*SPolCat*) consists of

- a module $M : \mathcal{C} \times \mathcal{D} \longrightarrow \mathcal{D}$ where

Polarized Categories

Polarization is a way to see ramification in CT

Definition

A *Polarized Strong Category* (*SPolCat*) consists of

- a module $M : \mathcal{C} \times \mathcal{D} \longrightarrow \mathcal{D}$ where

$\mathcal{C}$ is a *cc* (the *opponent*) and $\mathcal{D}$ a category (the *player*)

Polarized Categories

Polarization is a way to see ramification in CT

Definition

A *Polarized Strong Category* (*SPolCat*) consists of

- a module $M : \mathcal{C} \times \mathcal{D} \longrightarrow \mathcal{D}$ where

$\mathcal{C}$ is a *cc* (the *opponent*) and $\mathcal{D}$ a category (the *player*)

- endowed with a *strong composition*

$$\frac{(C_1, D_1) \xrightarrow{f} D_2 \quad (C_2, D_2) \xrightarrow{g} D_3}{(C_1 \times C_2, D_1) \xrightarrow{f;g} D_3}$$

Polarized Categories

Definition

Polarized Categories

Definition

A *Polarized Functor* of a *SPolCat* $\mathcal{C}$ consists of

Polarized Categories

Definition

A *Polarized Functor* of a *SPolCat* $\mathcal{C}$ consists of

- functors $F_p : \mathcal{D} \longrightarrow \mathcal{D}$ and $F_o : \mathcal{C} \longrightarrow \mathcal{C}$

Polarized Categories

Definition

A *Polarized Functor* of a *SPolCat* $\mathcal{C}$ consists of

- functors $F_p : \mathcal{D} \longrightarrow \mathcal{D}$ and $F_o : \mathcal{C} \longrightarrow \mathcal{C}$
- with F_{op} acting

Polarized Categories

Definition

A *Polarized Functor* of a *SPolCat* $\mathcal{C}$ consists of

- functors $F_p : \mathcal{D} \longrightarrow \mathcal{D}$ and $F_o : \mathcal{C} \longrightarrow \mathcal{C}$
- with F_{op} acting

$$\frac{(C,1) \stackrel{f}{\longrightarrow} D}{(F_o(C),1) \stackrel{F_{op}(f)}{\longrightarrow} F_p(D)}$$

Polarized Categories

Definition

A *Polarized Functor* of a *SPolCat* $\mathcal{C}$ consists of

- functors $F_p : \mathcal{D} \longrightarrow \mathcal{D}$ and $F_o : \mathcal{C} \longrightarrow \mathcal{C}$
- with F_{op} acting

$$\frac{(C,1) \xrightarrow{f} D}{(F_o(C),1) \xrightarrow{F_{op}(f)} F_p(D)}$$

Example

$\otimes$ for $\mathcal{D}$ and $\times$ for $\mathcal{C}$ form a polarized functor

Polarized Categories

We can do *comprehended recursion* on fixed points over F_{op}

Polarized Categories

We can do *comprehended recursion* on fixed points over F_{op}

Let $F^\star$ be a free algebra generated by

Polarized Categories

We can do *comprehended recursion* on fixed points over F_{op}

Let $F^{\star}$ be a free algebra generated by

- contexts $C = 1$ and $D = Nat$

Polarized Categories

We can do *comprehended recursion* on fixed points over F_{op}

Let $F^\star$ be a free algebra generated by

- contexts $C = 1$ and $D = Nat$
- constructors

$$Zero : 1 \longrightarrow Nat \text{ and } Succ : Nat \longrightarrow Nat$$

Polarized Categories

$F^{\star}$ is a fixed point for a polarized functor in the form

Polarized Categories

$F^\star$ is a fixed point for a polarized functor in the form

$$\sum_k F_k(Z)$$

Polarized Categories

$F^\star$ is a fixed point for a polarized functor in the form

$$\sum_k F_k(Z)$$

Can we construct a *SPolCat* from this to get a general form of ramified recursion in CT?