The known

Alg. Theories

(mon, grp, ring, mod, ...)

[1, 5, 6]
The Unknown

\[
\begin{align*}
\text{Alg. Theories} & \quad \text{(mon, grp, ring, mod, \ldots)} \\
\text{Cat. Alg.} & \quad \text{Univ. Alg.} \quad \text{1st-order} \quad \text{Eq. Log.} \\
\end{align*}
\]

\[=\]

\[
\begin{align*}
\text{Type Theories} & \quad \text{(simply typed, dep. typed, polymorphic, linear, \ldots)} \\
\end{align*}
\]

\[\Rightarrow [1, 5, 6, 29].\]
Programme

mathematical models $\rightarrow$ meta-theories

Development of **algebraic meta-theories** for formal languages.

- **Semantics**
  - Model theory.

- **Syntax**
  - Initial-algebra semantics.
  - Structural induction and recursion.
  - Substitution.

Synthesis of *deduction systems* for equational reasoning and computation by rewriting.

[29], [26].
The Space

Multi-Sorted Univ. Alg

Univ. Alg

binding

Simple Type Theory

binding

simple types

binding

_sorts_

_simple types_

[1], [2]
The Space

Univ. Alg \rightarrow \text{Multi-Sorted Univ. Alg} \leftarrow \text{Dep.-Sorted Alg.}

binding

type dep.

binding

simple types

Simple Type Theory

Univ. Alg

Simple Type Theory

binding

Simple Type Theory

simple types

Univ. Alg

binding

Uni...
The Space

Univ. Alg ➔ Multi-Sorted Univ. Alg ➔ Dep.-Sorted Alg. ➔ Dependent Type Theory

Univ. Alg ➔ Binding Alg. ➔ Simple Type Theory

sorts ➔ binding ➔ type dep. ➔ binding ➔ simple types

◮ [1], [2], [14], [10]
The Space

Multi-Sorted Univ. Alg. ←→ binding

Dep.-Sorted Alg.

Simple Type Theory

dependent types

Polymorphic Type Theory

- [1], [2], [14], [10], [8, 12]
The Space

Univ. Alg.  

- Multi-Sorted Univ. Alg.  
  - sorts  
  - binding  
  - linearity  

- Binding Alg.  
  - simple types  

- Simple Type Theory  
  - binding  
  - explicit polymorphism  

- Dep.-Sorted Alg.  
  - type dep.  

- Dependent Type Theory  
  - type dep.  

Operads  

- [1], [2], [14], [10], [8, 12], [9, 11].
The Space

Univ. Alg

Multi-Sorted Univ. Alg

binding

binding

simple types

Dep.-Sorted Alg.

Simple Type Theory

binding

type dep.

Dependent Type Theory

Polymorphic Type Theory

Operads

linearity

The Talk

I  Modelling of simple type theories.

II  Modelling of dependent type theories.

III  Foundations.

◮  [1], [2], [14], [10], [8, 12], [9, 11].
I

Algebraic Modelling of Simple Type Theories
## Simple Type Theory

<table>
<thead>
<tr>
<th>Types</th>
<th>Algebraic theories</th>
<th>Simply-typed theories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terms</td>
<td>Algebraic</td>
<td>Algebraic with binding</td>
</tr>
</tbody>
</table>

The syntactic theory should account for:

- variables and meta-variables
- variable binding and $\alpha$-equivalence
- capture-avoiding and meta substitution
- mono and multi sorting

[19, 20, 22, 25].

Categories of Contexts

Def: An $S$-sorted context structure is given by

- a small category $\mathbb{C}$ with terminal object,
- objects $\langle \sigma \rangle \in \mathbb{C}$ for all $\sigma \in S$,
- product diagrams

\[
\begin{array}{ccc}
\Gamma & \xleftarrow{\sigma} & \langle \sigma \rangle \\
\downarrow & & \downarrow \\
\Gamma & \xrightarrow{} & \langle \sigma \rangle \\
\end{array}
\]

for all $\Gamma \in \mathbb{C}$ and $\sigma \in S$.

Example: $\text{FinSet}^{\text{op}}$ is the initial mono-sorted context structure.
Example: Untyped $\lambda$-calculus.

Syntax:

$$t ::= x \mid t'(t'') \mid \lambda x. t'$$
Example: Untyped λ-calculus.

- Syntax:
  \[ t ::= x \mid t'(t'') \mid \lambda x. t' \]

- Algebras:
  \[
  \begin{align*}
  \text{var} & : 1 \to \mathcal{A}^y(s) \\
  \text{app} & : \mathcal{A}^2 \to \mathcal{A} \\
  \text{abs} & : \mathcal{A}^y(s) \to \mathcal{A}
  \end{align*}
  \]

\[ \hat{\mathcal{C}} \overset{\text{def}}{=} \text{Set}^{\mathcal{C}^{\text{op}}} \text{ and } y : \mathcal{C} \hookrightarrow \hat{\mathcal{C}} \text{ is the Yoneda embedding} \]

\[ \Rightarrow [3], [20, 22]. \]
**Example:** Untyped $\lambda$-calculus.

- **Syntax:**
  \[
  t ::= x \mid t'(t'') \mid \lambda x. t'
  \]

- **Algebras:**
  \[
  \begin{align*}
  \text{var} &: 1 \to A_{y^\langle s \rangle} \\
  \text{app} &: A^2 \to A \\
  \text{abs} &: A_{y^\langle s \rangle} \to A
  \end{align*}
  \]

\[\hat{C} = \text{Set}^{\text{op}} \text{ and } y : C \hookrightarrow \hat{C} \text{ is the Yoneda embedding}\]

**NB:**
\[
P_{y^\langle s \rangle}(\Gamma) = P(\Gamma, s)
\]

as

\[\begin{array}{c}
\text{C} \\ \downarrow \\
-_{,s} \end{array} \xrightarrow{\text{(-)x\text{y^\langle s \rangle}} \downarrow \text{(-)y^\langle s \rangle}} \begin{array}{c}
\hat{C} \\ \downarrow \\
\text{C}
\end{array}\]

- \cite{3}, \cite{20, 22}. 
Algebraic Models of Variable Binding in $\hat{CS}$

Example: Untyped $\lambda$-calculus.

- Syntax:
  \[
  t ::= x \mid t'(t'') \mid \lambda x. t'
  \]

- Algebras:
  \[
  \begin{align*}
  \text{var} & : y\langle s \rangle \to A \\
  \text{app} & : A^2 \to A \quad \text{in } \hat{C} \\
  \text{abs} & : A^{y\langle s \rangle} \to A \\
  \end{align*}
  \]

- Initial model:
  \[
  \Lambda \in \mathcal{S}\text{et}^{\text{FinSet}}
  \]
  with $\Lambda(n)$ the set of $\alpha$-equivalence classes of $\lambda$-terms with free variables amongst $x_1, \ldots, x_n$.

$[20, 22]$
Single-Variable Substitution

Substitution algebras:

\[ \text{subst} : A^{y^{(s)}} \times A \rightarrow A \]

satisfying

\[ \ldots \text{natural axioms} \ldots \]
Single-Variable Substitution

Substitution algebras:

\[
\text{subst} : \mathcal{A}^y \times \mathcal{A} \rightarrow \mathcal{A}
\]

satisfying

... natural axioms ...

Initial model:

[\mathcal{L} \in \text{Set}^{\text{FinSet}}]

with capture-avoiding single-variable substitution.

[20, 22].
Substitution Algebras

1. \( u : A \vdash \mathsf{var}(x)[u/x] = u \)

2. \( t : A, u : A \vdash t[u/x] = t \)

3. \( t : A^y(y^{(s)} \times y^{(s)}), x : y^{(s)} \)
   \( \vdash t(x, y)[\mathsf{var}(x)/y] = t(x, x) \)

4. \( t : A^y(y^{(s)} \times y^{(s)}), u : A^{y^{(s)}}, v : A \)
   \( \vdash (t(y, x)[u(x)/y])[v/x] = (t(y, x)[v/x])[u(x)[v/x]/y] \)

\[ \Rightarrow \ [20, 22] \]
Substitution Algebras

1. \( u : A \vdash \text{var}(x)^u_{/x} = u \)

2. \( t : A, u : A \vdash t^u_{/x} = t \)

3. \( t : A^y_{^{(s)} \times y_{^{(s)}}}, x : y^{(s)} \)
   \[ \vdash t(x, y)^{\text{var}(x)_{/y}} = t(x, x) \]

4. \( t : A^y_{^{(s)} \times y_{^{(s)}}}, u : A^y_{^{(s)}}, v : A \)
   \[ \vdash (t(y, x)^u_{/x})^{v_{/x}} = (t(y, x)^v_{/x})^{u_{/x}}_{/y} \]

5. \( t, t' : A^y_{^{(s)}}, u : A \)
   \[ \vdash \text{app}(t(x), t'(x))^u_{/x} = \text{app}(t(x)^u_{/x}, t'(x)^u_{/x}) \]

6. \( t : A^y_{^{(s)} \times y_{^{(s)}}}, u : A \)
   \[ \vdash \text{abs}(\lambda y. t(y, x))^u_{/x} = \text{abs}(\lambda y. t(y, x)^u_{/x}) \]

\[ \rightarrow [20, 22]. \]
Free Constructions

\[
\begin{array}{c}
\text{SubstAlg} \\
\downarrow \downarrow \\
\text{Set}^{\text{FinSet}}
\end{array}
\]

- Explicit description of syntax with variable binding.
- Induction principle for syntax with variable binding.
- Definition of capture-avoiding substitution by structural recursion.
Free Constructions

\[
\begin{array}{c}
\text{SubstAlg} \\
\uparrow \\
\downarrow \\
\text{Set} \quad \text{FinSet}
\end{array}
\]

- Explicit description of syntax with variable binding.
- Induction principle for syntax with variable binding.
- Definition of capture-avoiding substitution by structural recursion.
- Mathematical foundations for metavariabes.

\[\text{[23, 25].}\]
Metavariables

\[
\begin{array}{c}
\text{SubstAlg} \\
\downarrow \\
\text{Set} \\
\cup
\end{array}
\begin{array}{c}
\text{FinSet} \\
\downarrow \\
M
\end{array}
\]

Kleisli maps

\[ y(n) \rightarrow M( \bigsqcup_i y(m_i)) \]

are in bijective correspondence with terms

\[ t ::= x \mid t'(t'') \mid \lambda x. t' \]

\[ \mid M_i[t_1, \ldots, t_{m_i}] \]

with free variables amongst \( x_1, \ldots, x_n \).

Definition of \textit{meta-substitution} by structural recursion:

\[ M(X) \times Y^X \rightarrow M(Y) \]

\[ \textbf{13, 23, 25}. \]
Second-Order Equational Presentations

Example: Untyped λ-calculus.

(β) \( M : 1 \triangleright x \vdash (\lambda x. M[x]) \ x = M[x] \)

(η) \( M : 0 \triangleright \cdot \vdash \lambda x. M[\cdot] (x) = M[\cdot] \)
Second-Order Equational Presentations

Example: Untyped $\lambda$-calculus.

\[(\beta) \; M : 1 \triangleright x \vdash (\lambda x. M[x]) \; x = M[x]\]
\[(\eta) \; M : 0 \triangleright \cdot \vdash \lambda x. M[\cdot] (x) = M[\cdot]\]

Second-Order Equational Logic

(Extended Metasubstitution Rule)

\[\begin{align*}
M_1 : m_1, \ldots, M_k : m_k \triangleright \Gamma \vdash s \equiv t \\
\Theta \triangleright \Delta, x_{i,1}, \ldots, x_{i,m_i} \vdash s_i \equiv t_i \quad (1 \leq i \leq k)
\end{align*}\]

\[\Theta \triangleright \Gamma, \Delta\]

\[\vdash s\{M_i := (\vec{x}_i)s_i\}_{1 \leq i \leq k} \equiv t\{M_i := (\vec{x}_i)t_i\}_{1 \leq i \leq k}\]

\[\Rightarrow \; [13], [31].\]
Results

Extension of the mathematical theory of (first-order) algebraic structure to simple type theory:

- Conservativity of Second-Order Equational Logic over Birkhoff’s (first-order) Equational Logic.
- Soundness and completeness of Second-Order Equational Logic.
- Soundness and completeness of (bidirectional) Second-Order Term Rewriting.
- Presentation/theory correspondence via classifying categories and internal languages.
- Universal-algebra/categorical-algebra correspondence.
- Theory of syntactic algebraic translations.

- [31, 32]
II
Algebraic Modelling of Dependent Type Theories
Dependent Type Theory

<table>
<thead>
<tr>
<th>types</th>
<th>simply-typed theories</th>
<th>dependently-typed theories</th>
</tr>
</thead>
<tbody>
<tr>
<td>terms</td>
<td>algebraic</td>
<td>algebraic with binding</td>
</tr>
<tr>
<td></td>
<td>algebraic with binding</td>
<td></td>
</tr>
</tbody>
</table>

The syntactic theory should account for:

- type dependency
- variable binding and $\alpha$-equivalence
- term and type substitution
Dependency.

\[
\vdash C_0 \\
\vdash C_0(x, y) \\
\]

Binding.

\[
\frac{\Gamma, x : \sigma \vdash \tau}{\Gamma \vdash \Pi x : \sigma. \tau}
\]

\[
\frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x : \sigma.t : \Pi x : \sigma. \tau}
\]

Substitution.

\[
\frac{\Gamma \vdash t : \Pi x : \sigma.\tau \quad \Gamma \vdash u : \sigma}{\Gamma \vdash t(u) : \tau[u/x]}
\]

\[
(\lambda x : \sigma.t)(u) = t[u/x]
\]
Def: A \textit{dependently-typed context structure} is given by

- a small category $\mathcal{C}$ with terminal object,
- a presheaf $S \in \hat{\mathcal{C}}$,
- a functorial assignment of pullbacks for all $\Gamma \vdash \sigma$ and $f : \Delta \to \Gamma$ in $\mathcal{C}$.
Local context-extension lemma.

For $\Gamma \vdash \sigma$ and $P \in \widehat{\mathbb{C}}_{/\Gamma}$,

$$P y (\pi_{\Gamma \vdash \sigma}) (\Delta \xrightarrow{f} \Gamma) \cong P (\Delta, \sigma[f] \rightarrow \Gamma) .$$
Type-Dependent Binding

- Local context-extension lemma.

For $\Gamma \vdash \sigma$ and $P \in \widehat{C}_{/\Gamma}$,

$$P_{\gamma}(\pi_{\Gamma \vdash \sigma})(\Delta \xrightarrow{f} \Gamma) \cong P(\Delta, \sigma[f] \rightarrow \Gamma).$$

- Type-dependent binding operators.

$$\Pi_{\sigma} : S_{\Gamma}^{\gamma}(\pi_{\Gamma \vdash \sigma}) \rightarrow S_{\Gamma}$$ in $\widehat{C}_{/\Gamma}$. 
Decomposition of Binding Arities

For $\Gamma \vdash \sigma$, consider the adjunction

$$
\begin{array}{c}
\mathcal{C}_{/\Gamma} \\
\uparrow \quad \rlap{$\pi^*$} \\
\downarrow \\
\mathcal{C}_{/\pi}
\end{array}
\cong
\begin{array}{c}
\mathcal{C}_{/\Gamma,\sigma}
\end{array}
$$
Decomposition of Binding Arities

For $\Gamma \vdash \sigma$, consider the adjunction

\[ \begin{array}{c}
\mathbb{C}/\Gamma \xrightarrow{\pi^*} \mathbb{C}_{/\pi} \\
\mathbb{C}_{/\Gamma} \xleftarrow{\pi^*} \mathbb{C}_{/\Gamma,\sigma}
\end{array} \]

It induces the adjoint situation:

\[ \begin{array}{c}
\mathbb{C}/\Gamma \xrightarrow{\delta_\sigma} \mathbb{C}_{/\Gamma,\sigma} \\
\mathbb{C}_{/\Gamma} \xleftarrow{\epsilon_\sigma} \mathbb{C}_{/\Gamma,\sigma}
\end{array} \]

Binding-arity decomposition lemma.

For $\Gamma \vdash \sigma$, the monad $(-)_{\mathbb{C}/\Gamma}$ on $\mathbb{C}_{/\Gamma}$ is induced by the adjunction $\epsilon_\sigma \dashv \delta_\sigma$. 
We thus obtain the following situation:

\[ \hat{C}/\Gamma \cong \hat{C}/\gamma(\Gamma) \]
Binding Arities

We thus obtain the following situation

\[ \widehat{\mathcal{C}}_{/\Gamma} \cong \widehat{\mathcal{C}}_{/y(\Gamma)} \]

\[ \Pi_{\sigma} : \prod_{\sigma} (y(\Gamma, \sigma)^* S) \rightarrow S \quad \text{in} \quad \widehat{\mathcal{C}} \]
We thus obtain the following situation

\[
\hat{C}_{/\Gamma} \cong \hat{C}_{/y(\Gamma)} \quad \overset{T}{\longrightarrow} \quad \pi \Gamma \vdash \sigma \\
\hat{C}_{/y(\Gamma,\sigma)} \cong \hat{C}_{/\Gamma,\sigma} \quad \overset{T}{\longleftarrow} \quad \sigma \leftarrow \sum \sigma
\]

**Type-Dependent Binding**

- Type-dependent binding operators.

\[\Pi_\sigma : \prod_\sigma (y(\Gamma,\sigma)^*S) \rightarrow S \text{ in } \hat{C}\]

**Term Binding**

- Term binding operators.

For \(\Gamma \vdash \sigma\) and \(\Gamma, \sigma \vdash \tau\),

\[
\Pi_\sigma (T_\tau) \xrightarrow{\lambda} T \\
\downarrow \quad \quad \quad \downarrow \\
y(\Gamma) \xrightarrow{y(\Pi_\sigma(\tau))} S
\]

in \(\hat{C}\)
This is to give a term $\Gamma, \sigma \vdash p : \sigma[\pi_{\Gamma \vdash \sigma}]$. 
Variables

\[
\begin{align*}
\gamma(\Gamma, \sigma) & \rightarrow \Gamma_{\sigma} \\
\gamma(\pi_{\Gamma \vdash \sigma}) & \downarrow \\
\gamma(\Gamma) & \downarrow
\end{align*}
\]

This is to give a term $\Gamma, \sigma \vdash p : \sigma[\pi_{\Gamma \vdash \sigma}]$.

**NB:** **Categories with attributes/families**

The condition

\[
\begin{align*}
\gamma(\Gamma, \sigma) & \xrightarrow{\simeq} \Gamma_{\sigma} \\
\gamma(\pi_{\Gamma \vdash \sigma}) & \downarrow \\
\gamma(\Gamma) & \downarrow
\end{align*}
\]

is equivalent to **Dybjer's context comprehension** property: For all maps $f : \Delta \rightarrow \Gamma$ in $\mathbb{C}$ and terms $\Delta \vdash t : \sigma[f]$ there exists a unique map $\langle f, t \rangle : \Delta \rightarrow (\Gamma, \sigma)$ in $\mathbb{C}$ such that $p[\langle f, t \rangle] = t$ and $\pi_{\Gamma \vdash \sigma} \circ \langle f, t \rangle = f$.

The initial model is the *classifying category*.

\[ [14, 18] \]
Substitution Structure

For $\Gamma \vdash \sigma$,

\[ T_{\Gamma, y}(\pi_{\Gamma \vdash \sigma}) \times T_{\Gamma \vdash \sigma} \longrightarrow T_{\Gamma} \]

\[ S_{\Gamma, y}(\pi_{\Gamma \vdash \sigma}) \times T_{\Gamma \vdash \sigma} \longrightarrow S_{\Gamma} \]

in $\hat{\mathcal{C}}_{/y}(\Gamma)$

subject to axioms.
II

Algebraic Foundations
Kan Extensions

Every

\[ f : X \to Y \]

induces

\[ PX \leftarrow f^* \rightarrow PY \]

where

\[ \mathcal{P}C = \text{def } \text{Set}^C \]

and

\[ f_* P y = \text{Ran}_f P y = \int_{x \in X} [Y(y, f(x)) \Rightarrow P_x] \]

\[ f^* Q x = Q(f(x)) \]

\[ f_! P y = \text{Lan}_f P y = \int_{x \in X} Y(f(x), y) \times P_x \]
Generalised Dependent Polynomial Functors

The class of

generalised dependent polynomial functors

is the closure under natural isomorphism of the functors

\[ \mathcal{P}A \to \mathcal{P}B \]

arising as composites

\[ \mathcal{P}A \xrightarrow{s^*} \mathcal{P}I \xrightarrow{f^*} \mathcal{P}J \xrightarrow{t_!} \mathcal{P}B \]

from diagrams

\[
\begin{array}{ccc}
A & \xleftarrow{s} & I \\
\downarrow{f} & & \downarrow{t} \\
J & \to & B
\end{array}
\]

in \textbf{Cat}.

\[
t_! f_* s^* A b = \int_{j \in J} B(tj, b) \times \int_{i \in I} [J(j, fi) \Rightarrow A(si)]
\]
Examples:

- Dependent polynomial functors (aka indexed containers) between slices of $\text{Set}$ are isomorphic to generalised dependent polynomial functors.

- [21, 24].
Untyped abstract syntax

1. The rule

\[
\begin{array}{c}
\Gamma \vdash t \\
\Gamma \vdash t' \\
\hline
\Gamma \vdash t(t')
\end{array}
\]

has associated the generalised dependent polynomial endofunctor represented by

\[
\text{FinSet} \xleftarrow{\nabla_2} 2 \cdot \text{FinSet} \xrightarrow{\nabla_2} \text{FinSet} \xrightarrow{\text{id}} \text{FinSet}
\]
Untyped abstract syntax

1. The rule

\[
\frac{\Gamma \vdash t \quad \Gamma \vdash t'}{\Gamma \vdash t(t')}
\]

has associated the generalised dependent polynomial endofunctor represented by

\[\text{FinSet} \overset{\nabla_2}{\leftrightarrow} 2 \cdot \text{FinSet} \overset{\nabla_2}{\rightarrow} \text{FinSet} \overset{\text{id}}{\rightarrow} \text{FinSet}\]

2. The rule

\[
\frac{\Gamma, x \vdash t}{\Gamma \vdash \lambda x. t}
\]

has associated the generalised dependent polynomial endofunctor represented by

\[\text{FinSet} \overset{+1}{\leftrightarrow} \text{FinSet} \overset{\text{id}}{\rightarrow} \text{FinSet} \overset{\text{id}}{\rightarrow} \text{FinSet}\]
**Simply typed abstract syntax**

Let $S$ be the set of simple types and write $C$ for the category $\text{FinSet}_{/S}$ of $S$-sorted contexts.

1. The rule

\[
\Gamma \vdash t : \tau' \Rightarrow \tau \quad \Gamma \vdash t' : \tau' \\
\overline{\Gamma \vdash t(t') : \tau}
\]

has associated the generalised dependent polynomial endofunctor represented by

\[
C \times S \xrightarrow{[\text{id} \times \Rightarrow, \text{id} \times \pi_1]} 2 \cdot (C \times S^2) \xrightarrow{\nabla_2} C \times S^2 \xrightarrow{\text{id} \times \pi_2} C \times S
\]
Simply typed abstract syntax

Let $S$ be the set of simple types and write $C$ for the category $\text{FinSet}_S$ of $S$-sorted contexts.

1. The rule

\[ \Gamma \vdash t : \tau' \Rightarrow \tau \quad \Gamma \vdash t' : \tau' \]

\[ \Gamma \vdash t(t') : \tau \]

has associated the generalised dependent polynomial endofunctor represented by

\[ C \times S \xrightarrow{[\text{id} \times \Rightarrow, \text{id} \times \pi_1]} 2 \cdot (C \times S^2) \xrightarrow{\nabla_2} C \times S^2 \xrightarrow{\text{id} \times \pi_2} C \times S \]

2. The rule

\[ \Gamma, x : \sigma \vdash t : \tau \]

\[ \Gamma \vdash \lambda x. t : \sigma \Rightarrow \tau \]

has associated the generalised dependent polynomial endofunctor represented by

\[ C \times S \xleftarrow{\leftarrow \times \text{id}} C \times S \times S \xrightarrow{\text{id}} C \times S \times S \xrightarrow{\text{id} \times \Rightarrow} C \times S \]
NB: The association of generalised dependent polynomial functors to rules extends to *polymorphic languages*. In this context, the last component of the representation plays a crucial role as a pattern-matching constructor.
Convolution monoidal closed structure

1. *Day’s convolution tensor product* is isomorphic to a generalised dependent polynomial functor.

2. *Exponentiation to a representable* with respect to the closed structure associated to the convolution monoidal structure is a generalised polynomial functor.
Generalised Inductive Dependent Polynomial Functors

The class of generalised dependent polynomial functors represented by diagrams of the form

\[
A \leftarrow \coprod_{k \in K} L_k \cdot J_k \xrightarrow{\coprod_{k \in K} \nabla_{L_k}} \coprod_{k \in K} J_k \rightarrow B
\]

where $L_k$ is finite for all $k \in K$,

- is closed under constants, identities, coproducts, finite products, and composition; and
- admits a (cartesian) differential calculus.

These functors

- are inductive (viz. finitary and preserve epis); and
- admit inductively-defined free algebras for equational systems.
Application Areas

- Data types.
  (e.g. reasoning)

- Type theory.
  (e.g. formalisation)

- Logical frameworks.
  (e.g. synthesis)

- Dependently-typed programming.
  (e.g. zippers)

- Concurrency theory.
  (e.g. models)
Pointers


manuscript, 2008.


