Skew monoidal structures on categories of algebras

Marcelo Fiore and Philip Saville

University of Cambridge Dept. of Computer Science

11th July 2018

Skew monoidal categories

A version of monoidal categories: structural transformations α, λ, ρ need not be invertible

Introduced by Szlachányi (2012) in the context of *bialgebroids*

Skew monoidal categories

A version of monoidal categories: structural transformations α, λ, ρ need not be invertible

Introduced by Szlachányi (2012) in the context of *bialgebroids*

Recently studied in some detail: Uustalu (2014), Andrianopoulos (2017), — MFPS paper, Bourke & Lack (2017, 2018), Lack and Street (2014) ...

Captures some old examples (Alternkirch 2010) and can be better behaved than the monoidal case (Street 2013) monoidal

$\mathbb{T}$ monoidal

reflexive coequalizers in T + preservation conditions

The monadic list transformer

The monadic list transformer

We want to model effects as monads.

The monadic list transformer

We want to model effects as monads.

Problem: monads do not compose straightforwardly!

The monadic list transformer

We want to model effects as monads.

Problem: monads do not compose straightforwardly!

Want to

The monadic list transformer

We want to model effects as monads.

Problem: monads do not compose straightforwardly!

Want to

- Build new monads from old, while

The monadic list transformer

We want to model effects as monads.

Problem: monads do not compose straightforwardly!

Want to

- Build new monads from old, while
- *Lifting* the operations from our old monad to the new one.

The monadic list transformer

We want to model effects as monads.

Problem: monads do not compose straightforwardly!

Want to

- Build new monads from old, while
- *Lifting* the operations from our old monad to the new one.

Definition

The list transformer of Jaskelioff takes a monad T to the monad $\mathrm{Lt}(T)X := A.T(1 + X \times A)$.

The monadic list transformer

We want to model effects as monads.

Problem: monads do not compose straightforwardly!

Want to

- Build new monads from old, while
- *Lifting* the operations from our old monad to the new one.

Definition

The list transformer of Jaskelioff takes a monad T to the monad $\mathrm{Lt}(T)X := A.T(1 + X \times A)$.

Our contribution: universal description as a list object with algebraic structure.

Abstract syntax with binding and metavariables (Fiore)

To build the abstract syntax of a type system...

Abstract syntax with binding and metavariables (Fiore)

To build the abstract syntax of a type system...

Without binding: freely generate the terms from the rules and basic terms. Constructors modelled as algebras.

Abstract syntax with binding and metavariables (Fiore)

To build the abstract syntax of a type system...

Without binding: freely generate the terms from the rules and basic terms. Constructors modelled as algebras.

With binding: freely generate the algebra with

Abstract syntax with binding and metavariables (Fiore)

To build the abstract syntax of a type system...

Without binding: freely generate the terms from the rules and basic terms. Constructors modelled as algebras.

With binding: freely generate the algebra with

- A monoid structure modelling binding,

Abstract syntax with binding and metavariables (Fiore)

To build the abstract syntax of a type system...

Without binding: freely generate the terms from the rules and basic terms. Constructors modelled as algebras.

With binding: freely generate the algebra with

- A monoid structure modelling binding,
- A compatibility law between binding and constructors, so that $\mathrm{app}(\sigma, \tau)[x \mapsto \omega] = \mathrm{app}(\sigma[x \mapsto \omega], \tau[x \mapsto \omega])$.

Abstract syntax with binding and metavariables (Fiore)

To build the abstract syntax of a type system...

Without binding: freely generate the terms from the rules and basic terms. Constructors modelled as algebras.

With binding: freely generate the algebra with

- A monoid structure modelling binding,
- A compatibility law between binding and constructors, so that $\mathrm{app}(\sigma, \tau)[x \mapsto \omega] = \mathrm{app}(\sigma[x \mapsto \omega], \tau[x \mapsto \omega])$.

Abstract syntax = free such structure

= a list object with algebraic structure.

A unifying framework for many diverse examples of list objects with algebraic structure

- Notions of natural numbers in domain theory,
- The monadic list transformer,
- Abstract syntax with binding and metavariables,
- Algebraic operations,
- Instances of the Haskell MonadPlus type class,
- Higher-dimensional algebra.

This talk

This talk

list objects $\rightsquigarrow$ T-**list objects**

This talk

list objects $\rightsquigarrow$ T**-list objects**

- well-understood datatype
- extends datatype of lists

This talk

list objects $\rightsquigarrow$ T**-list objects**

- well-understood datatype
- are free monoids

- extends datatype of lists
- are free T-monoids

This talk

list objects $\rightsquigarrow$ **T-list objects**

- well-understood datatype
- are free monoids
- described by $A.(I + XA)$.

- extends datatype of lists
- are free T-monoids
- described by $A.T(I + XA)$.

This talk

list objects $\rightsquigarrow$ T-**list objects**

- well-understood datatype
- are free monoids
- described by $A.(I + XA)$.

- extends datatype of lists
- are free T-monoids
- described by $A.T(I + XA)$.

Gives a *concrete* way to reason about free T-monoids.

This talk

list objects $\rightsquigarrow$ T-**list objects**

- well-understood datatype
- are free monoids
- described by $A.(I + XA)$.

- extends datatype of lists
- are free T-monoids
- described by $A.T(I + XA)$.

Gives a *concrete* way to reason about free T-monoids.

Gives an algebraic structure for T-list objects.

Past work: list objects in CCCs (Joyal, Cockett)

A list object (X) on X consists of

Past work: list objects in CCCs (Joyal, Cockett)

A list object (X) on X consists of

$$1(X)$$

Past work: list objects in CCCs (Joyal, Cockett)

A list object (X) on X consists of

$$1(X)X \times (X)$$

Past work: list objects in CCCs (Joyal, Cockett)

A list object (X) on X consists of

$$1(X)X \times (X)$$

that is initial:

Past work: list objects in CCCs (Joyal, Cockett)

A list object (X) on X consists of

$$1(X)X \times (X)$$

that is initial: given any $(1AX \times A)$, there exists a unique iterator

$$\begin{tikzcd}
1 \arrow[r] \arrow[d, equal] & (X) \arrow[d, dashed, "\mathrm{it}(,)"] & X \times (X) \arrow[l] \arrow[d, "X \times \mathrm{it}(n,c)"] \\
1 \arrow[r] & A & X \times A \arrow[l]
\end{tikzcd}$$

List objects in a monoidal category $(, ,)$

List objects in a monoidal category (, ,)

A list object (X) on X consists of

$$I(X)X(X)$$

List objects in a monoidal category (, ,)

A list object (X) on X consists of

$$I(X)X(X)$$

that is parametrised initial:

List objects in a monoidal category (, ,)

A list object (X) on X consists of

$$I(X)X(X)$$

that is parametrised initial: given any $(PnAcXA)$, there exists a unique iterator

$$\begin{tikzcd}
P \arrow[r, "P"] \arrow[d] & (X)P \arrow[d, dashed, "\mathrm{it}(,)"] & X(X)P \arrow[l, "P"'] \arrow[d, "X\mathrm{it}(n{,}c)"] \\
P \arrow[r] & A & XA \arrow[l]
\end{tikzcd}$$

List objects in a monoidal category (, ,)

Remark

If each $(-)P$ has a right adjoint, parametrised initiality is equivalent to the non-parametrised version:

$$\begin{tikzcd}
{} \arrow[r] \arrow[d, equal] & (X) \arrow[d, dashed, "\mathrm{it}(,)"] & X(X) \arrow[l] \arrow[d, "X\mathrm{it}(n,c)"] \\
{} \arrow[r] & A^P & XA^P \arrow[l]
\end{tikzcd}$$

List objects in a monoidal category (, ,)

Connection to past work

- Closely connected to Kelly's notion of algebraically-free monoid in a monoidal category.
- The list object () is precisely a left natural numbers object in the sense of Paré and Román. *E.g.* the flat natural numbers $A.(1 + A)$ in **Cpo**.

List objects are free monoids

List objects are free monoids

Definition

A monoid in a monoidal category (, ,) is an object () such that the multiplication is associative and is a neutral element for this multiplication.

List objects are free monoids

Lemma

1. *Every list object (X) is a monoid.*

List objects are free monoids

Lemma

1. *Every list object* (X) *is a monoid.*
2. *This monoid is the free monoid on* X*, with universal map*

$$XXXX(X)(X)$$

taking $x \mapsto (x, *) \mapsto (x, []) \mapsto x :: [] = [x]$.

List objects are free monoids

Lemma

1. *Every list object (X) is a monoid.*
2. *This monoid is the free monoid on X, with universal map*

$$XXXX(X)(X)$$

taking $x \mapsto (x, *) \mapsto (x, []) \mapsto x :: [] = [x]$.

We can reason concretely about free monoids by reasoning about lists.

List objects are initial algebras

List objects are initial algebras

Definition

An algebra for a functor $F :\to$ is a pair $(A, \alpha : FA \to A)$.

List objects are initial algebras

Definition

An algebra for a functor $F :\to$ is a pair $(A, \alpha : FA \to A)$.

Lemma

If $(,,)$ *is a monoidal category with finite coproducts* $(0, +)$ *and* ω*-colimits, both preserved by all* $(-)P$ *for* $P \in$*, then the initial algebra of the functor* $\big(+ X(-)\big)$ *is a list object on* X*.*

List objects are initial algebras

Definition

An algebra for a functor $F :\to$ is a pair $(A, \alpha : FA \to A)$.

Lemma

If $(, ,)$ *is a monoidal category with finite coproducts* $(0, +)$ *and* ω*-colimits, both preserved by all* $(-)P$ *for* $P \in$*, then the initial algebra of the functor* $\big(+ X(-)\big)$ *is a list object on* X*.*

Remark

This result relies on a general theory of parametrised initial algebras.

The story so far

The story so far

list objects

The story so far

list objects

- well-understood datatype

The story so far

list objects

- well-understood datatype
- are free monoids

The story so far

list objects

- well-understood datatype
- are free monoids
- described by $A.(I + XA)$.

Rest of this talk

list objects $\rightsquigarrow$ ***T*-list objects** (new work)

- well-understood datatype
- are free monoids
- described by $A.(I + XA)$.

- extends datatype of lists
- are free T-monoids
- described by $A.T(I + XA)$.

Rest of this talk

list objects $\rightsquigarrow$ ***T*-list objects**

(new work)

- well-understood datatype
- are free monoids
- described by $A.(I + XA)$.

- extends datatype of lists
- are free T-monoids
- described by $A.T(I + XA)$.

...and instantiate this for applications

Compatible algebraic structure

Compatible algebraic structure

Definition

A monad on a category is a functor $T :\to$ equipped with a multiplication $\mu : T^2 \to T$ and a unit $\eta :_\to T$ satisfying associativity and unit laws.

Compatible algebraic structure

Definition

A monad on a category is a functor $T :\to$ equipped with a multiplication $\mu : T^2 \to T$ and a unit $\eta :_{\to} T$ satisfying associativity and unit laws.

Definition

An algebra for a monad (T, μ, η) is a pair $(A, \alpha : TA \to A)$ satisfying unit and associativity laws.

Compatible algebraic structure

Definition

A monad on a category is a functor $T :\to$ equipped with a multiplication $\mu : T^2 \to T$ and a unit $\eta :_\to T$ satisfying associativity and unit laws.

Definition

An algebra for a monad (T, μ, η) is a pair $(A, \alpha : TA \to A)$ satisfying unit and associativity laws.

Definition

A strong monad T is a monad on a monoidal category $(,)$ that is equipped with a natural transformation $_{A,B} : T(A)B \to T(AB)$ satisfying coherence laws.

List objects with algebraic structure

T-list objects

T-list objects

Let $(,)$ be a strong monad on a monoidal category $(,)$. A -list object (X) on X consists of

$$\longrightarrow () \longleftarrow ()$$

T-list objects

Let $(,)$ be a strong monad on a monoidal category $(,)$. A -list object (X) on X consists of

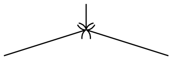
$$\begin{tikzcd}
 & (()) \arrow[d] & \\
{} \arrow[r] & () & () \arrow[l]
\end{tikzcd}$$

T-list objects

Let $(,)$ be a strong monad on a monoidal category $(,)$. A -list object (X) on X consists of

$$\begin{tikzcd}
 & (()) \arrow[d] & \\
 {} \arrow[r] & () & () \arrow[l]
\end{tikzcd}$$

such that for every structure

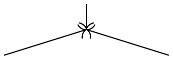


T-list objects

Let $(,)$ be a strong monad on a monoidal category $(,)$. A -list object (X) on X consists of

$$\begin{tikzcd}
 & (()) \arrow[d] & \\
 \arrow[r] & () & () \arrow[l]
\end{tikzcd}$$

such that for every structure



there exists a unique mediating map $(,,):() \rightarrow$

T-list objects

such that

$$\begin{tikzcd}
{} \arrow[r] \arrow[d, "\cong"'] & () & () \arrow[l] \arrow[d, "(,,)"] \\
{} \arrow[ur, no head] & & {} \arrow[ul, "(,,)"']
\end{tikzcd}$$

and

$$\begin{tikzcd}
(()) \arrow[r, "(),"] \arrow[d] & (()) \arrow[r, "((,,))"] & {} \arrow[dll] \\
() \arrow[urr, dashed, no head, "(,,)"'] & &
\end{tikzcd}$$

T-list objects

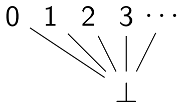
Remark

Every list object is a T-list object.

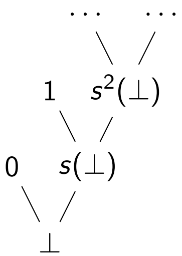
If every $(-)P$ has a right adjoint, the iterator $(,,)$ is a T-algebra homomorphism.

Natural numbers in **Cpo**, revisited

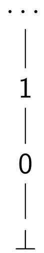
Flat natural numbers, $A.(1 + A)$:



Lazy natural numbers, $A.(1 + A)_\perp$:

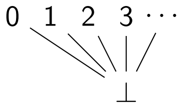


Strict natural numbers, $A.A_\perp$:

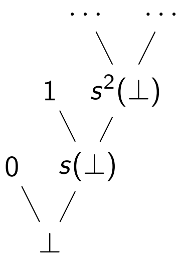


Natural numbers in **Cpo** as T-list objects on the unit

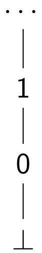
Flat natural numbers, $A.(1 + A)$:



Lazy natural numbers, $A.(1 + A)_\perp$:

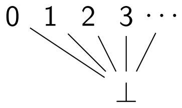


Strict natural numbers, $A.A_\perp$:

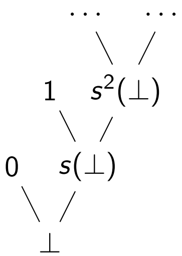


Natural numbers in **Cpo** as T-list objects on the unit

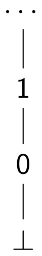
Flat natural numbers, $A.(1+A)$:



Lazy natural numbers, $A.(1+A)_\bot$:



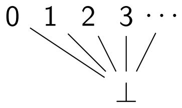
Strict natural numbers, $A.A_\bot$:



T-list object with $(\times, 1)$ structure and monad $T =$

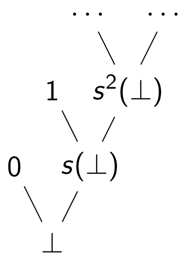
Natural numbers in **Cpo** as T-list objects on the unit

Flat natural numbers, $A.(1+A)$:



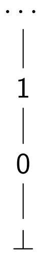
T-list object with $(\times, 1)$ structure and monad $T =$

Lazy natural numbers, $A.(1+A)_\perp$:



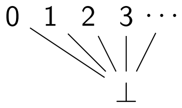
T-list object with $(\times, 1)$ structure and $T := (-)_\perp$ the lifting monad

Strict natural numbers, $A.A_\perp$:



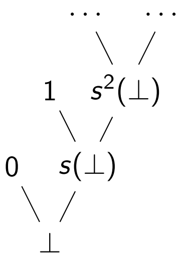
Natural numbers in **Cpo** as T-list objects on the unit

Flat natural numbers, $A.(1 + A)$:



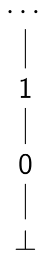
T-list object with $(\times, 1)$ structure and monad $T =$

Lazy natural numbers, $A.(1 + A)_\perp$:



T-list object with $(\times, 1)$ structure and $T := (-)_\perp$ the lifting monad

Strict natural numbers, $A.A_\perp$:



T-list object with $(+, 0)$ structure and $T := (-)_\perp$ the lifting monad

Monoids with compatible algebraic structure

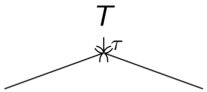
T-monoids

T-monoids

Let (,) be a strong monad on on a monoidal category (,). A -monoid (EM-monoid (Piróg)) is a monoid

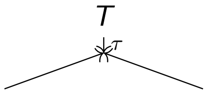
T-monoids

Let (,) be a strong monad on on a monoidal category (,). A -monoid (EM-monoid (Piróg)) is a monoid equipped with a T-algebra

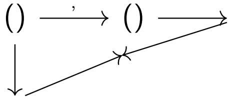


T-monoids

Let (,) be a strong monad on on a monoidal category (,). A -monoid (EM-monoid (Piróg)) is a monoid equipped with a T-algebra

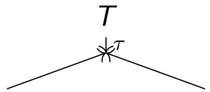


compatible in the sense that

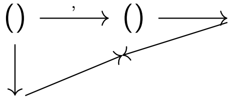


T-monoids

Let $(,)$ be a strong monad on on a monoidal category $(,)$. A -monoid (EM-monoid (Piróg)) is a monoid equipped with a T-algebra



compatible in the sense that



Remark

T-monoids generalise both monoids and T-algebras.

T-monoids

Remark

In the context of abstract syntax, T is freely generated from some theory, and T-monoids are models of this theory.

T-monoids

Remark

In the context of abstract syntax, T is freely generated from some theory, and T-monoids are models of this theory.

Lemma

For every monoid the endofunctor $T := (-)$ *is a monad, and* $T \simeq ()$.

T-monoids

Remark

In the context of abstract syntax, T is freely generated from some theory, and T-monoids are models of this theory.

Lemma

For every monoid the endofunctor $T := (-)$ *is a monad, and* $T \simeq ()$*.*

Example

In particular, a T-monoid for the endofunctor $T := S(-)$ is precisely an algebraic operation with signature S in the sense of Jaskelioff, and can be identified with a map $S\eta(S) \to$ interpreting S inside .

T-monoids

Remark

In the context of abstract syntax, T is freely generated from some theory, and T-monoids are models of this theory.

Lemma

For every monoid the endofunctor $T := (-)$ *is a monad, and* $T \simeq ()$.

Example

Thinking of a Lawvere theory as a monoid L in $\big('(1), \bullet\big)$, we can identify Lawvere theories extending L with T-monoids for $T := \bullet(-)$.

T-list objects are free T-monoids

T-list objects are free T-monoids

For a strong monad $(T,)$ on a monoidal category $(,)$,

T-list objects are free T-monoids

For a strong monad $(T,)$ on a monoidal category $(,)$,

Lemma

1. *Every T-list object (X) is a T-monoid.*

T-list objects are free T-monoids

For a strong monad $(T,)$ on a monoidal category $(,)$,

Lemma

1. *Every T-list object (X) is a T-monoid.*
2. *This T-monoid is the free T-monoid on X, with universal map*

$$XXXX(X)(X)$$

T-list objects are free T-monoids

For a strong monad $(T,)$ on a monoidal category $(,)$,

Lemma

1. *Every T-list object (X) is a T-monoid.*
2. *This T-monoid is the free T-monoid on X, with universal map*

$$XXXX(X)(X)$$

We can reason concretely about free T-monoids by reasoning about T-lists.

T-list objects are initial algebras

T-list objects are initial algebras

For a strong monad $(T,)$ on a monoidal category $(,)$,

Lemma

If every $(-)P$ preserves binary coproducts, and the initial algebra exists, then $A.T(I + XA)$ is a T-list object on X.

Theorem

Let be a strong monad on a monoidal category $(,,)$ with binary coproducts $(+)$. If

1. *for every $\in$, the endofunctor $(-)$ preserves binary coproducts, and*
2. *for every $X \in$, the initial algebra of $T(I + X-)$ exists*

Then has all -list objects and, thereby, the free -monoid monad .

Theorem

Let be a strong monad on a monoidal category $(,,)$ with binary coproducts $(+)$. If

1. *for every $\in$, the endofunctor $(-)$ preserves binary coproducts, and*
2. *for every $X \in$, the initial algebra of $T(I + X-)$ exists*

Then has all -list objects and, thereby, the free -monoid monad .

Remark

Thinking in terms of T-list objects makes the proof straightforward!

Technical contribution

Technical contribution

$$A.(I + XA) \rightsquigarrow \text{list object} \rightsquigarrow \text{free monoid}$$

Technical contribution

$$A.(I + XA) \rightsquigarrow \text{list object} \rightsquigarrow \text{free monoid}$$

T-list object

Technical contribution

$A.(I + XA) \rightsquigarrow$ list object $\rightsquigarrow$ free monoid

T-list object $\rightsquigarrow$ free T-monoid

Technical contribution

$$A.(I + XA) \rightsquigarrow \text{list object} \rightsquigarrow \text{free monoid}$$

$$A.T(I + XA) \rightsquigarrow T\text{-list object} \rightsquigarrow \text{free } T\text{-monoid}$$

Technical contribution

$$A.(I + XA) \rightsquigarrow \text{list object} \rightsquigarrow \text{free monoid}$$

$$A.T(I + XA) \rightsquigarrow T\text{-list object} \rightsquigarrow \text{free } T\text{-monoid}$$

Remark

A natural extension: algebraic structure encapsulated by Lawvere theories or operads. This gives rise to a notion of near-semiring category, which underlies many of the applications.

Applications

Applications

T-NNOs

In a a monoidal category $(,)$:

$$\text{NNO} = \text{list object on}$$
$$T\text{-NNO} = T\text{-list object on}$$

In **Cpo**: gives rise to the *flat*-, *lazy*- and *strict* natural numbers.

Applications

Functional programming

- In the bicartesian closed setting: Jaskelioff's monadic list transformer $\mathrm{Lt}(T)X := A.T(1 + X \times A)$ is just the free T-monoid monad.

Applications

Functional programming

- In the bicartesian closed setting: Jaskelioff's monadic list transformer $\mathrm{Lt}(T)X := A.T(1 + X \times A)$ is just the free T-monoid monad.
- In the category of endofunctors over a cartesian category: the $\mathrm{MonadPlus}$ type class $\mathrm{Mp}(F)X := A.\mathrm{List}(X + FA)$ of Rivas is a List-list object.

Applications

Functional programming

- In the bicartesian closed setting: Jaskelioff's monadic list transformer $\mathrm{Lt}(T)X := A.T(1 + X \times A)$ is just the free T-monoid monad.
- In the category of endofunctors over a cartesian category: the $\mathrm{MonadPlus}$ type class $\mathrm{Mp}(F)X := A.\mathrm{List}(X + FA)$ of Rivas is a List-list object.
- In the category of endofunctors over a cartesian category: the datatype

$$\mathrm{Bun}(F)X := A.(1 + X \times A + F(A) \times A + A \times A)$$

is an instance of Spivey's Bunch type class that is a T-list object for T the extension of the theory of monoids with a unary operator.

Applications

Functional programming

- In the bicartesian closed setting: Jaskelioff's monadic list transformer $\mathrm{Lt}(T)X := A.T(1 + X \times A)$ is just the free T-monoid monad.
- In an nsr-category: the $\mathrm{MonadPlus}$ type class $\mathrm{Mp}(F)X := A.\mathrm{List}_*(X + FA)$ is a List_*-list object.
- In an nsr-category:

$$\mathrm{Bun}(F)X := A.\big(J + (I + XA + A) * A\big)$$

is an instance of Spivey's Bunch type class that is a T-list object for T the extension of the theory of monoids with a unary operator.

Applications

Abstract syntax and variable binding (Fiore)

In the category of presheaves with substitution tensor product

$$(P \bullet Q)(n) = \int^{m\in} (Pm) \times (Qn)^m$$

Applications

Abstract syntax and variable binding (Fiore)

In the category of presheaves with substitution tensor product

$$(P \bullet Q)(n) = \int^{m \in} (Pm) \times (Qn)^m$$

we get

$$\textbf{abstract syntax} = \textbf{free } T\textbf{-monoid on variables}$$
$$= A.T(V + X \bullet A)$$

Applications

Abstract syntax and variable binding (Fiore)

In the category of presheaves with substitution tensor product

$$(P \bullet Q)(n) = \int^{m\in} (Pm) \times (Qn)^m$$

we get

$$\begin{aligned}\textbf{abstract syntax} &= \textbf{free } T\textbf{-monoid on variables} \\ &= A.T(V + X \bullet A)\end{aligned}$$

abstract syntax is a list object with algebraic structure

Applications

Abstract syntax and variable binding (Fiore)

In the category of presheaves with substitution tensor product

$$(P \bullet Q)(n) = \int^{m \in} (Pm) \times (Qn)^m$$

we get

$$\textbf{abstract syntax} = \textbf{free } T\textbf{-monoid on variables}$$
$$= A.T(V + X \bullet A)$$

Remark

This relies on a slightly more general theory, in which the strength $_{X,I \to P} : T(X)P \to T(XP)$ only acts on pointed objects.

Applications

Higher-dimensional algebra

The web monoid in Szawiel and Zawadowski's construction of opetopes is a T-list object in an nsr-category.

Summary: *List objects with algebraic structure*

Summary: *List objects with algebraic structure*

$$A.(I + XA) \rightsquigarrow \text{list object} \rightsquigarrow \text{free monoid}$$
$$A.T(I + XA) \rightsquigarrow T\text{-list object} \rightsquigarrow \text{free } T\text{-monoid}$$

Summary: *List objects with algebraic structure*

$$A.(I + XA) \rightsquigarrow \text{list object} \rightsquigarrow \text{free monoid}$$
$$A.T(I + XA) \rightsquigarrow T\text{-list object} \rightsquigarrow \text{free } T\text{-monoid}$$

Framework unifying a wide range of examples.

Summary: *List objects with algebraic structure*

$$A.(I + XA) \rightsquigarrow \text{list object} \rightsquigarrow \text{free monoid}$$
$$A.T(I + XA) \rightsquigarrow T\text{-list object} \rightsquigarrow \text{free } T\text{-monoid}$$

Framework unifying a wide range of examples.

Algebraic structure $\rightsquigarrow$ list-style datatype. Simpler proofs!
(abstract syntax, opetopes?)

Summary: *List objects with algebraic structure*

$$A.(I + XA) \rightsquigarrow \text{list object} \rightsquigarrow \text{free monoid}$$
$$A.T(I + XA) \rightsquigarrow T\text{-list object} \rightsquigarrow \text{free } T\text{-monoid}$$

Framework unifying a wide range of examples.

Algebraic structure $\rightsquigarrow$ list-style datatype. Simpler proofs!
(abstract syntax, opetopes?)

Initial algebra definition $\rightsquigarrow$ universal property.
(monadic list transformer, MonadPlus)

Summary: *List objects with algebraic structure*

$$A.(I + XA) \rightsquigarrow \text{list object} \rightsquigarrow \text{free monoid}$$
$$A.T(I + XA) \rightsquigarrow T\text{-list object} \rightsquigarrow \text{free } T\text{-monoid}$$

Framework unifying a wide range of examples.

Algebraic structure $\rightsquigarrow$ list-style datatype. Simpler proofs!
(abstract syntax, opetopes?)

Initial algebra definition $\rightsquigarrow$ universal property.
(monadic list transformer, MonadPlus)

A journal-length version is in preparation.