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 $\alpha, \lambda, \rho$ need not be invertible

Introduced by Szlachányi (2012) in the context of bialgebroids
Skew monoidal categories

A version of monoidal categories: structural transformations $\alpha, \lambda, \rho$ 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

$T$ 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
  - \textit{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

$$Lt(T)X := A. T(1 + X \times A).$$
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
- \textit{Lifting} the operations from our old monad to the new one.

\textbf{Definition}

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

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
  \[\text{app}(\sigma, \tau)[x \mapsto \omega] = \text{app}(\sigma[x \mapsto \omega], \tau[x \mapsto \omega])\]
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
  \[
  \text{app}\left(\sigma, \tau\right)[x \mapsto \omega] = \text{app}\left(\sigma[x \mapsto \omega], \tau[x \mapsto \omega]\right). 
  \]

\textbf{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 $\leadsto$ $T$-list objects
This talk

list objects

- well-understood datatype

$\leadsto$

$T$-list objects

- extends datatype of lists
This talk

list objects  ~⇒~  $T$-list objects

- well-understood datatype
- are free monoids

- extends datatype of lists
- are free $T$-monoids
### This talk

<table>
<thead>
<tr>
<th>list objects</th>
<th>$\leadsto$</th>
<th>$T$-list objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>well-understood datatype</td>
<td></td>
<td>extends datatype of lists</td>
</tr>
<tr>
<td>are free monoids</td>
<td></td>
<td>are free $T$-monoids</td>
</tr>
<tr>
<td>described by $A.(I + XA)$</td>
<td></td>
<td>described by $A.T(I + XA)$</td>
</tr>
</tbody>
</table>
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 \textit{concrete} way to reason about free \( T \)-monoids.
This talk

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

- $T$-list objects
  - 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

\[1(X)X \times (X)\]
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

\[
1(X)X \times (X)
\]

describing an initial type. Given any \((1AX \times A)\), there exists a unique iterator

\[
\begin{array}{ccc}
1 & \longrightarrow & (X) & \longleftarrow & X \times (X) \\
& & \downarrow & & \downarrow X \times \text{it}(n,c) \\
1 & \longrightarrow & A & \longleftarrow & X \times A
\end{array}
\]
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

\[
\begin{align*}
P \xrightarrow{P} (X)P & \xleftarrow{P} X(X)P \\
\downarrow & \downarrow \text{it(,) } \downarrow X\text{it(n,c)} \\
P \rightarrow A & \leftarrow XA
\end{align*}
\]
List objects in a monoidal category $(, , )$

Remark

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

\[
\begin{array}{ccc}
A^P & \xrightarrow{\text{it}(,)} & XA^P \\
\downarrow & & \downarrow \text{it}(n,c) \\
(X) & \xleftarrow{\text{Xit}(,)} & X(X)
\end{array}
\]
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 \( \text{Cpo} \).
List objects are free monoids
List objects are free monoids

Definition

A monoid in a monoidal category $(\cdot, \cdot)$ is an object $()$ such that the multiplication $\cdot$ is associative and $()$ is a neutral element for this multiplication.
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* \(x \mapsto (x, \ast) \mapsto (x, []) \mapsto x :: [] = [x].\)
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*

\[
XXX(X)(X)
\]

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

We can reason concretely about free monoids by reasoning about lists.
List objects are initial algebras

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

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

Remark
This result relies on a general theory of parametrised initial algebras.
List objects are initial algebras

**Definition**

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

**Lemma**

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

**Remark**

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

Definition

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

Lemma

If $(, ,)$ is a monoidal category with finite coproducts $(0, +)$ and $\omega$-colimits, both preserved by all $(-)P$ for $P \in$, then the initial algebra of the functor $(+ X(-))$ 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 $(\cdot, \cdot)$ is a monoidal category with finite coproducts $(0, +)$ and $\omega$-colimits, both preserved by all $(-)P$ for $P \in$, then the initial algebra of the functor $(+ X(-))$ 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
list objects

- well-understood datatype
- are free monoids
- described by \( A.(I + XA) \).
Rest of this talk

list objects

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

$\leadsto$

$T$-list objects

(new work)

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

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

- **$T$-list objects**
  - (new work)
  - extends datatype of lists
  - are free $T$-monoids
  - described by $A.T(I + XA)$.

...and instantiate this for applications
Compatible algebraic structure

**Definition**

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

**Definition**

An algebra for a monad $(T, \mu, \eta)$ 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 $(\mathcal{C}, \otimes, I)$ that is equipped with a natural transformation $A, B : T(A) \otimes B \to T(A \otimes B)$ satisfying coherence laws.
Compatible algebraic structure

Definition

A monad on a category $\mathcal{C}$ is a functor $T : \mathcal{C} \to \mathcal{C}$ equipped with a multiplication $\mu : T^2 \to T$ and a unit $\eta : \mathbb{1} \to T$ satisfying associativity and unit laws.
Compatible algebraic structure

Definition

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

Definition

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

Definition

An algebra for a monad \( (T, \mu, \eta) \) 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 \( (\otimes, I) \) 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 $(\cdot,\cdot)$ be a strong monad on a monoidal category $(\cdot,\cdot)$. A $T$-list object $(X)$ on $X$ consists of

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

$$
\begin{CD}
(()) @>>> () \\
@VVV \\
() @<<< ()
\end{CD}
$$
$T$-list objects

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

\[
\begin{array}{c}
\vspace{1cm}
(())
\end{array}
\]

such that for every structure

\[
\begin{array}{c}
\vspace{1cm}
() \quad \quad \quad \quad (())
\end{array}
\]

\[
\begin{array}{c}
\vspace{1cm}
() \quad \quad \quad \quad ()
\end{array}
\]
$T$-list objects

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

\[
(()) 
\]

such that for every structure

\[
\begin{array}{ccc}
\ast & \rightarrow & () \\
\downarrow & & \\
(()) & & ()
\end{array}
\]

there exists a unique mediating map $(, ,) : () \rightarrow$
$T$-list objects

such that

and

\[(()) \rightarrow ((())) \rightarrow ((()))\]
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 \( \textbf{Cpo} \), revisited

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

\[
\begin{array}{ccccccccccc}
\ldots & 0 & 1 & 2 & 3 & \ldots
\end{array}
\]

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

\[
\begin{array}{ccccccccccc}
\ldots & 1 & s^2(\bot) & 0 & s(\bot) & \ldots
\end{array}
\]

Strict natural numbers, \( A.A_\bot \):

\[
\begin{array}{ccccccccccc}
\ldots & 1 & \ldots
\end{array}
\]

T-list object with \((+,0)\) structure and \( T := (\_\_\_\bot) \) the lifting monad.
Natural numbers in \textbf{Cpo} as $T$-list objects on the unit

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

\[ \ldots 0 1 2 3 \ldots \]

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

\[ \ldots \quad 1 \quad s^2(\bot) \]

\[ \quad 0 \quad s(\bot) \]

\[ \quad \bot \]

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

\[ \ldots \quad \bot \]

\[ \quad \bot \]

$T$-list object with $(+, 0)$ structure and $T := (-)$ the lifting monad.
Natural numbers in \( \mathbf{Cpo} \) as \( T \)-list objects on the unit

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

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
\downarrow & & & \\
\bot & & & \\
\end{array}
\]

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

\[
\begin{array}{cccc}
\cdots & \cdots & & \\
\downarrow & & & \\
1 & s^2(\bot) & & \\
\downarrow & & & \\
0 & s(\bot) & & \\
\downarrow & & & \\
\bot & & & \\
\end{array}
\]

Strict natural numbers, \( A.A_\bot \):

\[
\begin{array}{cc}
\cdots & \\
\downarrow & \\
1 & \\
\downarrow & \\
0 & \\
\downarrow & \\
\bot & \\
\end{array}
\]

\( T \)-list object with \( (\times, 1) \) structure and monad \( T = \)
Natural numbers in $\mathbf{Cpo}$ as $T$-list objects on the unit

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

```
0 1 2 3 ...
```

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

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

```
1 \downarrow / \downarrow /
0 \downarrow / \downarrow /
\perp
```

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

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

```
\ldots
```

```
1
\downarrow
0
\downarrow
\perp
```

$T$-list object with $(\times, 1)$ structure and monad $T = \{-\}$
Natural numbers in \( \mathbf{Cpo} \) as \( T \)-list objects on the unit

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

\[
\begin{array}{cccc}
\hline
0 & 1 & 2 & 3 \\
\hline
\end{array}
\]

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

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

\[
\begin{array}{c}
1 \\
\hline
0 \\
\hline
\perp
\end{array}
\]

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

Strict natural numbers, \( A.A_\perp \):

\[
\begin{array}{c}
\perp \\
\hline
0 \\
\hline
\perp
\end{array}
\]

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

Let $(\mathcal{C}, \otimes)$ be a strong monad on a monoidal category $(\mathcal{C}, \otimes)$. A $T$-monoid (or $T\text{-}EM$-monoid (Piróg)) is a monoid equipped with a $T$-algebra $T\tau$ compatible in the sense that

Remark $T$-monoids generalise both monoids and $T$-algebras.
$T$-monoids

Let $(\mathcal{C}, \otimes)$ be a strong monad on a monoidal category $(\mathcal{C}, \otimes)$. A $T$-monoid (Piróg) is a monoid equipped with a $T$-algebra $T\tau$ compatible in the sense that

[Equation]

Remark $T$-monoids generalise both monoids and $T$-algebras.
$T$-monoids

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

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

![Diagram](https://via.placeholder.com/150)

Remark $T$-monoids generalise both monoids and $T$-algebras.
$T$-monoids

Let $(\cdot, \cdot)$ be a strong monad on a monoidal category $(\cdot, \cdot)$. A $T$-monoid (EM-monoid (Piróg)) is a monoid equipped with a $T$-algebra compatible in the sense that

$$
\begin{array}{c}
T \\
\downarrow \\
() \\
\downarrow
\end{array} 
\xrightarrow{\cdot} 
\begin{array}{c}
() \\
\downarrow
\end{array} 
\xrightarrow{\cdot} 
\begin{array}{c}
()
\end{array}
$$

Remark

$T$-monoids generalise both monoids and $T$-algebras.
**$T$-monoids**

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

\[
T \xrightarrow{T} \xrightarrow{\cdot} \xrightarrow{(\cdot)} \xrightarrow{()}
\]

compatible in the sense that

\[
() \xrightarrow{\cdot} () \xrightarrow{\cdot} \xrightarrow{()}
\]

**Remark**

$T$-monoids generalise both monoids and $T$-algebras.
In the context of abstract syntax, $T$ is freely generated from some theory, and $T$-monoids are models of this theory.
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 $T \simeq ()$.***
**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 $T :\equiv (-)$ is a monad, and $T \cong ()$.

**Example**

In particular, a $T$-monoid for the endofunctor $T :\equiv S(-)$ 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.
**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 $T$ the endofunctor $T := (−)$ is a monad, and $T \simeq (\cdot)$.*

**Example**

Thinking of a Lawvere theory as a monoid $L$ in $(1, \bullet)$, we can identify Lawvere theories extending $L$ with $T$-monoids for $T := \bullet(−)$. 
$T$-list objects are free $T$-monoids

Lemma 1.

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

This $T$-monoid is the free $T$-monoid on $X$, with universal map $X \rightarrow X$. We can reason concretely about free $T$-monoids by reasoning about $T$-lists.
$T$-list objects are free $T$-monoids

For a strong monad $(T,)$ on a monoidal category $(\cdot, \cdot)$,
\textit{T-list} objects are free \textit{T}-monoids

For a strong monad \((T, \_\_)\) on a monoidal category \((\_, \_\_)\),

\textbf{Lemma}

1. \textit{Every T-list object} \((X)\) \textit{is a T-monoid}.
\textit{T-list objects are free $T$-monoids}

For a strong monad $(T, \mu)$ on a monoidal category $(\otimes, 1)$,

\textbf{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

\[ XXXXX(X)(X) \]
Every $\mathcal{T}$-list object $(X)$ is a $\mathcal{T}$-monoid.

This $\mathcal{T}$-monoid is the free $\mathcal{T}$-monoid on $X$, with universal map $X XXX (X)(X)$

We can reason concretely about free $\mathcal{T}$-monoids by reasoning about $\mathcal{T}$-lists.
$T$-list objects are initial algebras
Lemma

If every \((-\_)_P\) preserves binary coproducts, and the initial algebra exists, then \(A \cdot T(I + XA)\) 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 \((\cdot,\cdot)\) with binary coproducts \((+\cdot)\). If

1. for every \(\in\), the endofunctor \((-\cdot)\) preserves binary coproducts,

and

2. for every \(X \in\), the initial algebra of \(T(I + X-\cdot)\) 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
A.\((I+XA)\) $\rightsquigarrow$ list object $\rightsquigarrow$ free monoid
Technical contribution

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

\( T \)-list object

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.
Technical contribution

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

$T$-list object $\leadsto \text{free } T$-monoid
Technical contribution

\[ A.(I + XA) \leadsto \text{list object} \leadsto \text{free monoid} \]

\[ A.T(I + XA) \leadsto T\text{-list object} \leadsto \text{free } T\text{-monoid} \]
Technical contribution

\[ A.(I + XA) \leadsto \text{list object} \leadsto \text{free monoid} \]

\[ A.\ T(I + XA) \leadsto \text{T-list object} \leadsto \text{free T-monoid} \]

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

\textbf{T-NNOs}

In a monoidal category \((\cdot,\cdot)\):

\[
\text{NNO} = \text{list object on} \quad T\text{-NNO} = T\text{-list object on}
\]

In \(\mathbf{Cpo}\): gives rise to the \textit{flat-}, \textit{lazy-} and \textit{strict} natural numbers.
Applications

Functional programming

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

Functional programming

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

- In the category of endofunctors over a cartesian category: the MonadPlus type class $Mp(F)X := A.\text{List}(X + FA)$ of Rivas is a List-list object.
Applications

Functional programming

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

- In the category of endofunctors over a cartesian category: the MonadPlus type class $\text{Mp}(F)X := A. \text{List}(X + FA)$ of Rivas is a List-list object.

- In the category of endofunctors over a cartesian category: the datatype $\text{Bun}(F)X := A.(1 + X \times A + F(A) \times A + A \times A)$ 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 \( \text{Lt}(T)X := A.T(1 + X \times A) \) is just the free \( T \)-monoid monad.

- In an nsr-category: the \text{MonadPlus} type class \( \text{Mp}(F)X := A.\text{List}_\ast(X + FA) \) is a \text{List}_\ast-list object.

- In an nsr-category:

\[
\text{Bun}(F)X := A.(J + (I + XA + A) \ast A)
\]

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.
Abstract syntax and variable binding (Fiore)

In the category of presheaves with substitution tensor product

\[(P \bullet Q)(n) = \int_{m \in \mathbb{N}} (Pm) \times (Qn)^m\]
Applications

Abstract syntax and variable binding (Fiore)

In the category of presheaves with substitution tensor product

\[(P \bullet Q)(n) = \int_{m \in \mathbb{N}} (Pm) \times (Qn)^m\]

we get

abstract syntax = free $T$-monoid on variables

= $A.T(V + X \bullet A)$
Abstract syntax and variable binding (Fiore)

In the category of presheaves with substitution tensor product

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

we get

abstract syntax = free $T$-monoid on variables

$$= A.T(V + X \bullet A)$$

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

\[(P \bullet Q)(n) = \int_{m \in \mathbb{P}} (P_m) \times (Q_n)^m\]

we get

abstract syntax = free \( T \)-monoid on variables

\[= A. T(V + X \bullet A)\]

Remark

This relies on a slightly more general theory, in which the strength \( \chi, l \rightarrow P : T(X)P \rightarrow T(XP) \) 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*

\[ \mathcal{A}(I + XA) \leadsto \text{list object} \leadsto \text{free monoid} \]
\[ \mathcal{A}.T(I + XA) \leadsto T\text{-list object} \leadsto \text{free } T\text{-monoid} \]

Framework unifying a wide range of examples.
Summary: *List objects with algebraic structure*

\[ A.(I + XA) \leadsto \text{list object} \leadsto \text{free monoid} \]
\[ A.T(I + XA) \leadsto T\text{-list object} \leadsto \text{free } T\text{-monoid} \]

**Framework unifying a wide range of examples.**

Algebraic structure \(\leadsto\) 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.