List objects with algebraic structure provide a unifying framework for many diverse examples
Notions of natural number in \( \text{Cpo} \)

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

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

Lazy natural numbers, \( \mu A.(1 + A) \downarrow \):

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

Strict natural numbers, \( \mu A.A \downarrow \):

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

Unifying idea: natural numbers objects with algebraic structure.
Notions of natural number in $\text{Cpo}$

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

```
 0 1 2 3 ···
```

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

```
0 1 \(s^2(\bot)\)
```

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

```
\ldots
```

Unifying idea: natural numbers objects with algebraic structure.
The monadic list transformer

Jaskelioff’s *list transformer* takes a monad $T$ to the monad

$L_t(T)X := \mu A. T(1 + X \times A)$. 
The monadic list transformer

Jaskelioff’s *list transformer* takes a monad $T$ to the monad
$$\text{Lt}(T)X := \mu A. T(1 + X \times A).$$

**Universal property:** as a list object with algebraic structure.
Abstract syntax with variable binding (Fiore et al.)

To build the abstract syntax of a type system...
Abstract syntax with variable binding (Fiore et al.)

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

- presheaves $X : \mathbb{F} \to \text{Set}$ where $X(n)$ the set of terms with $n$ free variables,
Abstract syntax with variable binding (Fiore et al.)

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

- presheaves $X : \mathbb{F} \to \text{Set}$ where $X(n)$ the set of terms with $n$ free variables,

- with an algebra structure $\Sigma X \to X$ giving constructors (e.g. $\text{app} : X \times X \to X$),
Abstract syntax with variable binding (Fiore et al.)

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

▶ presheaves $X : \mathbb{F} \rightarrow \text{Set}$ where $X(n)$ the set of terms with $n$ free variables,

▶ with an algebra structure $\Sigma X \rightarrow X$ giving constructors (e.g. $\text{app} : X \times X \rightarrow X$),

▶ and binding given by $X \bullet X$, consisting of sequences of form $(n, \sigma \in X(n), (\tau_1, \ldots, \tau_n) \in X(m)^n)$,
Abstract syntax with variable binding (Fiore et al.)

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

- presheaves $X : \mathbb{F} \to \text{Set}$ where $X(n)$ the set of terms with $n$ free variables,

- with an algebra structure $\Sigma X \to X$ giving constructors (e.g. $\text{app} : X \times X \to X$),

- and binding given by $X \bullet X$, consisting of sequences of form $(n, \sigma \in X(n), (\tau_1, \ldots, \tau_n) \in X(m)^n)$,

- giving a monoid structure $V \to X \leftarrow X \bullet X$, 

Abstract syntax with variable binding (Fiore et al.)

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

- presheaves $X : \mathbb{F} \to \text{Set}$ where $X(n)$ the set of terms with $n$ free variables,

- with an algebra structure $\Sigma X \to X$ giving constructors (e.g. $\text{app} : X \times X \to X$),

- and binding given by $X \bullet X$, consisting of sequences of form $(n, \sigma \in X(n), (\tau_1, \ldots, \tau_n) \in X(m)^n)$,

- giving a monoid structure $V \to X \leftarrow X \bullet X$,

- subject to a compatibility law stating that e.g. $\text{app}(\sigma, \tau)[x \mapsto \omega] = \text{app}(\sigma[x \mapsto \omega], \tau[x \mapsto \omega])$. 
Abstract syntax with variable binding (Fiore et al.)

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

▶ presheaves \( X : \mathbb{F} \to \text{Set} \) where \( X(n) \) the set of terms with \( n \) free variables,

▶ with an algebra structure \( \Sigma X \to X \) giving constructors (e.g. \( \text{app} : X \times X \to X \)),

▶ and binding given by \( X \bullet X \), consisting of sequences of form \( (n, \sigma \in X(n), (\tau_1, \ldots, \tau_n) \in X(m)^n) \),

▶ giving a monoid structure \( V \to X \leftarrow X \bullet X \),

▶ subject to a compatibility law stating that e.g. \( \text{app}(\sigma, \tau)[x \mapsto \omega] = \text{app}(\sigma[x \mapsto \omega], \tau[x \mapsto \omega]) \).

Abstract syntax = free such structure

= a list object with algebraic structure.
This talk

list objects ▶ well-understood datatype ▶ are free monoids ▶ described by \( \mu_A \). (\( I + X \otimes A \)).

c\( \Rightarrow \) T-list objects ▶ extends datatype of lists ▶ are free \( T \)-monoids ▶ described by \( \mu_A \). T(I + X \otimes A).

Gives a concrete way to reason about free \( T \)-monoids.
Gives an algebraic structure for \( T \)-list objects.
This talk

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

- **list objects**
  - well-understood datatype

- \( \sim \)

- **\( T \)-list objects**
  - extends datatype of lists

Gives a concrete way to reason about free \( T \)-monoids.
Gives an algebraic structure for \( T \)-list objects.
This talk

<table>
<thead>
<tr>
<th>list objects</th>
<th>$\sim\rightarrow$</th>
<th>$T$-list objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>- well-understood datatype</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- are free monoids</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Gives a concrete way to reason about free $T$-monoids.
Gives an algebraic structure for $T$-list objects.
This talk

### list objects

- well-understood datatype
- are free monoids
- described by $\mu A. (I + X \otimes A)$.

$\leadsto$

### $T$-list objects

- extends datatype of lists
- are free $T$-monoids
- described by $\mu A. T(I + X \otimes A)$. 

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

Gives an algebraic structure for $T$-list objects.
This talk

list objects $\leadsto T$-list objects

- well-understood datatype
- are free monoids
- described by $\mu A. (I + X \otimes A)$.

- extends datatype of lists
- are free $T$-monoids
- described by $\mu A. T(I + X \otimes A)$.

Gives a concrete way to reason about free $T$-monoids.
This talk

**list objects** ▶ well-understood datatype
▶ are free monoids
▶ described by

\[ \mu A. (I + X \otimes A). \]

**T-list objects** ▶ extends datatype of lists
▶ are free T-monoids
▶ described by

\[ \mu A. T(I + X \otimes A). \]

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

Gives an algebraic structure for T-list objects.
Past work: list objects in $(\times, 1)$ (Cockett [Coc90])
Past work: list objects in $(\times, 1)$ (Cockett [Coc90])

A list object $L(X)$ on $X$ consists of
Past work: list objects in $(\times, 1)$ (Cockett [Coc90])

A *list object* $L(X)$ on $X$ consists of

$$1 \xrightarrow{\text{nil}} L(X)$$
Past work: list objects in $\times, 1$ (Cockett [Coc90])

A list object $L(X)$ on $X$ consists of

$$1 \xrightarrow{\text{nil}} L(X) \leftrightarrow_{\text{cons}} X \times L(X)$$
Past work: list objects in $(\times, 1)$ (Cockett [Coc90])

A list object $L(X)$ on $X$ consists of

$$1 \xrightarrow{nil} L(X) \xleftarrow{\text{cons}} X \times L(X)$$

that is initial:
A list object \( L(X) \) on \( X \) consists of

\[
1 \xrightarrow{\text{nil}} L(X) \xleftarrow{\text{cons}} X \times L(X)
\]

that is initial: given any \( (1 \xrightarrow{n} A \xleftarrow{c} X \times A) \), there exists a unique iterator

\[
1 \xrightarrow{\text{nil}} L(X) \xleftarrow{\text{cons}} X \times L(X)
\]

\[
\downarrow\quad \downarrow\quad \downarrow
\]

\[
\text{it}(n,c) \quad X \times \text{it}(n,c)
\]

\[
1 \xrightarrow{n} A \xleftarrow{c} X \times A
\]
List objects in a monoidal category \((\mathcal{C}, \otimes, I)\)
List objects in a monoidal category \((\mathcal{C}, \otimes, I)\)

A list object \(L(X)\) on \(X\) consists of

\[
I \xrightarrow{nil} L(X) \xleftarrow{cons} X \otimes L(X)
\]
List objects in a monoidal category \((\mathcal{C}, \otimes, I)\)

A \textit{list object} \(L(X)\) on \(X\) consists of

\[
I \xrightarrow{\text{nil}} L(X) \xleftarrow{\text{cons}} X \otimes L(X)
\]

that is \textit{parametrised initial}: 
List objects in a monoidal category \((\mathcal{C}, \otimes, I)\)

A list object \(L(X)\) on \(X\) consists of

\[
I \xrightarrow{\text{nil}} L(X) \xleftarrow{\text{cons}} X \otimes L(X)
\]

that is **parametrised initial**: given any \((P \xrightarrow{n} A \xleftarrow{c} X \otimes A)\), there exists a unique iterator

\[
I \otimes P \xrightarrow{\text{nil} \otimes P} L(X) \otimes P \xleftarrow{\text{cons} \otimes P} X \otimes L(X) \otimes P
\]

\[
P \xrightarrow{n} A \xleftarrow{c} X \otimes A
\]
List objects in a monoidal category \((\mathcal{C}, \otimes, I)\)

Remark

- If each \((-) \otimes P\) has a right adjoint (e.g. in the cartesian closed case), parametrised initiality is equivalent to the non-parametrised version:

\[
\begin{array}{ccc}
  I & \xrightarrow{\text{nil}} & L(X) & \xleftarrow{\text{cons}} & X \otimes L(X) \\
    & & \downarrow \text{it}(n,c) & & \downarrow X \otimes \text{it}(n,c) \\
  I & \xrightarrow{n} & A^P & \xleftarrow{c} & X \otimes A^P
\end{array}
\]
List objects in a monoidal category \((\mathcal{C}, \otimes, I)\)

Remark

- If each \((-) \otimes P\) has a right adjoint (e.g. in the cartesian closed case), parametrised initiality is equivalent to the non-parametrised version:

  \[
  \begin{array}{c}
  \begin{array}{ccc}
  I & \xrightarrow{\text{nil}} & L(X) & \xleftarrow{\text{cons}} & X \otimes L(X) \\
  & & \downarrow \text{it}(n,c) & & \downarrow X \otimes \text{it}(n,c) \\
  I & \xrightarrow{n} & A^P & \xleftarrow{c} & X \otimes A^P
  \end{array}
  \end{array}
  \]

- The list object \(L(I)\) is precisely a natural numbers object: e.g. the flat natural numbers \(\mu A. (1 + A)\) in \(\mathbf{Cpo}\).
List objects are free monoids
List objects are free monoids

Definition

A *monoid* in a monoidal category \((\mathcal{C}, \otimes, I)\) is an object \((I \xrightarrow{e} M \xleftarrow{m} M \otimes M)\) such that the multiplication \(m\) is associative and \(e\) is a neutral element for this multiplication.
List objects are free monoids

Lemma

1. Every list object \( L(X) \) is a monoid.
List objects are free monoids

Lemma

1. Every list object \( L(X) \) is a monoid.

2. This monoid is the free monoid on \( X \), with universal map

\[
X \cong X \otimes I \xrightarrow{X \otimes \text{nil}} X \otimes L(X) \xrightarrow{\text{cons}} L(X)
\]

Taking \( x \mapsto (x, \ast) \mapsto (x, []) \mapsto x :: [] = [x] \).
List objects are free monoids

Lemma

1. Every list object $L(X)$ is a monoid.

2. This monoid is the free monoid on $X$, with universal map

$$X \cong X \otimes I \xrightarrow{X \otimes \text{nil}} X \otimes L(X) \xrightarrow{\text{cons}} L(X)$$

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 $F : C \to C$ is a pair $(A, \alpha : FA \to A)$. 

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

Definition

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

Lemma

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

Definition

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

Lemma

If $(C, \otimes, I)$ is a monoidal category with finite coproducts $(0, +)$ and $\omega$-colimits, both preserved by all $(-) \otimes P$ for $P \in C$, then the initial algebra of the functor $(I + X \otimes (-))$ 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
  \[ \mu A. (I + X \otimes A). \]
Rest of this talk

list objects

- well-understood datatype
- are free monoids
- described by $\mu A. (I + X \otimes A)$.

$\leadsto$  

$T$-list objects (new work)

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

list objects

- well-understood datatype
- are free monoids
- described by $\mu A.(I + X \otimes A)$.

$\Rightarrow$

$T$-list objects (new work)

- extends datatype of lists
- are free $T$-monoids
- described by $\mu A. T(I + X \otimes A)$.

...and instantiate this for applications
Compatible algebraic structure

Definition
A monad on a category $C$ is a functor $T : C \to C$ equipped with a multiplication $\mu : T \times T \to T$ and a unit $\eta : Id_C \to T$ 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 $st_{A, B} : T(A \otimes B) \to T(A) \otimes T(B)$ satisfying coherence laws.
Compatible algebraic structure

Definition

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

Definition

A *monad* on a category $C$ is a functor $T : C \to C$ equipped with a multiplication $\mu : T^2 \to T$ and a unit $\eta : \text{Id}_C \to T$ 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.
Compatible algebraic structure

Definition

A *monad* on a category $\mathcal{C}$ is a functor $T : \mathcal{C} \rightarrow \mathcal{C}$ equipped with a multiplication $\mu : T^2 \rightarrow T$ and a unit $\eta : \text{Id}_\mathcal{C} \rightarrow T$ 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.

Definition

A *strong monad* $T$ is a monad on a monoidal category $(\otimes, I)$ that is equipped with a natural transformation $st_{A,B} : T(A) \otimes B \rightarrow T(A \otimes B)$ satisfying coherence laws.
List objects with algebraic structure
$T$-list objects
Let \((T, st)\) be a strong monad on a monoidal category \((\otimes, I)\). A \(T\)-list object \(M(X)\) on \(X\) consists of

\[
\begin{align*}
I & \xrightarrow{\text{nil}} M(X) & X \otimes M(X) & \xleftarrow{\text{cons}} M(X)
\end{align*}
\]
**T-list objects**

Let \((T, \text{st})\) be a strong monad on a monoidal category \((\otimes, I)\). A \textit{T-list object} \(M(X)\) on \(X\) consists of

\[
\begin{align*}
T(M(X)) \\
\downarrow^\tau \\
I \xrightarrow{\text{nil}} M(X) \xleftarrow{\text{cons}} X \otimes M(X)
\end{align*}
\]
**T-list objects**

Let \((T, st)\) be a strong monad on a monoidal category \((\otimes, I)\). A **T-list object** \(M(X)\) on \(X\) consists of

\[
\begin{align*}
T(M(X)) & \quad \downarrow \tau \\
I & \xrightarrow{\text{nil}} M(X) & \xleftarrow{\text{cons}} X \otimes M(X)
\end{align*}
\]

such that for every structure

\[
\begin{align*}
TA & \quad \downarrow \alpha \\
P & \xrightarrow{n} A & \xleftarrow{c} X \otimes A
\end{align*}
\]
$T$-list objects

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

\[
\begin{array}{c}
T(M(X)) \\
\downarrow \tau \\
I \xrightarrow{\text{nil}} M(X) \xleftarrow{\text{cons}} X \otimes M(X)
\end{array}
\]

such that for every structure

\[
\begin{array}{c}
TA \\
\downarrow \alpha \\
P \xrightarrow{n} A \xleftarrow{c} X \otimes A
\end{array}
\]

there exists a unique mediating map $\text{it}(n, c, \alpha) : M(X) \otimes P \rightarrow A$
\( T \)-list objects

such that

\[
I \otimes P \xrightarrow{\text{nil} \otimes P} M(X) \otimes P \xleftarrow{\text{cons} \otimes P} X \otimes M(X) \otimes P
\]

\[
P \xrightarrow{n} A \xleftarrow{c} X \otimes A
\]
such that

\[
\begin{array}{c}
I \otimes P \xrightarrow{\text{nil} \otimes P} M(X) \otimes P \xleftarrow{\text{cons} \otimes P} X \otimes M(X) \otimes P \\
\downarrow \quad \downarrow \text{it}(n,c,\alpha) \quad \downarrow \text{it}(n,c,\alpha) \\
P \quad n \quad A \quad c \quad X \otimes A
\end{array}
\]

and

\[
\begin{array}{c}
T(M(X)) \otimes P \xrightarrow{\text{st}_{M(X),P}} T(M(X) \otimes P) \xrightarrow{T(\text{it}(n,c,\alpha))} TA \\
\downarrow \quad \downarrow \alpha \\
M(X) \otimes P \xrightarrow{\text{it}(n,c,\alpha)} A
\end{array}
\]
\( T \)-list objects

Remark

\( T \)-list objects extend list objects.

If the tensor \( \otimes \) is closed, the iterator \( \text{it}(n, c, \alpha) \) is a \( T \)-algebra homomorphism.
Natural numbers in \textbf{Cpo}, revisited

\textit{Flat natural numbers}, \(\mu A.(1 + A)\):

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

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

\[\ldots \quad \ldots\]

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

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

\[\downarrow \quad \downarrow\]

\[\bot \quad \bot\]

\textit{Strict natural numbers}, \(\mu A.A\)\(_\bot\):

\[\ldots\]

\[1\]

\[0\]

\[\bot\]
Natural numbers in \textbf{Cpo} as $T$-list objects on the unit

\textit{Flat natural numbers, $\mu A.(1 + A)$:}

\begin{align*}
0 & \quad 1 & \quad 2 & \quad 3 & \quad \ldots \\
\downarrow & & & & \\
\downarrow & & & & \\
\downarrow & & & & \\
\downarrow & & & & \\
\downarrow & & & & \\
\downarrow & & & & \\
\downarrow & & & & \\
& \ldots & \ldots & \ldots & \ldots
\end{align*}

\textit{Lazy natural numbers, $\mu A.(1 + A)\perp$:}

\begin{align*}
1 & \quad s^2(\perp) & \\
\downarrow & & \\
0 & \quad s(\perp) & \\
\downarrow & & \\
\downarrow & & \\
\downarrow & & \\
\downarrow & & \\
& \ldots & \ldots & \ldots & \ldots
\end{align*}

\textit{Strict natural numbers, $\mu A.A\perp$:}

\begin{align*}
\cdots & \quad \cdots & \\
\downarrow & & \\
0 & \quad 1 & \\
\downarrow & & \\
\downarrow & & \\
\downarrow & & \\
\downarrow & & \\
& \perp & \perp & \perp & \perp
\end{align*}
Natural numbers in \( \textbf{Cpo} \) as \( T \)-list objects on the unit

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

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & \cdots \\
\end{array}
\]

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

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

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

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

\[
\begin{array}{cccc}
1 \\
0 \\
\end{array}
\]
Natural numbers in \( \mathbf{Cpo} \) as \( T \)-list objects on the unit

**Flat natural numbers,**
\[ \mu A. (1 + A): \]
\[ \begin{array}{cccc}
0 & 1 & 2 & 3 \\
\downarrow & & & \\
\downarrow & & & \\
\downarrow & & & \\
\downarrow & & & \\
\downarrow & & & \\
\end{array} \]

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

**Lazy natural numbers,**
\[ \mu A. (1 + A)_\perp: \]
\[ \begin{array}{cc}
1 & s^2(\perp) \\
\downarrow & \downarrow \\
0 & s(\perp) \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\downarrow & \downarrow \\
\end{array} \]

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

**Strict natural numbers,**
\[ \mu A. A_\perp: \]
\[ \begin{array}{c}
\ldots \\
\downarrow \\
1 \\
\downarrow \\
0 \\
\downarrow \\
\downarrow \\
\downarrow \\
\end{array} \]

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

\textit{Flat natural numbers,} $\mu A. (1 + A)$:

\begin{center}
\begin{tikzpicture}
\node (0) at (0,0) {$0$};
\node (1) at (1,0) {$1$};
\node (2) at (2,0) {$2$};
\node (3) at (3,0) {$3$};
\node (down) at (3,-1) {$\bot$};
\node (4) at (4,0) {$\cdots$};
\path (0) edge (1);
\path (1) edge (2);
\path (2) edge (3);
\path (3) edge (down);
\end{tikzpicture}
\end{center}

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

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

\begin{center}
\begin{tikzpicture}
\node (0) at (0,0) {$0$};
\node (1) at (1,0) {$1$};
\node (2) at (2,0) {$s(\bot)$};
\node (down) at (2,-1) {$\bot$};
\node (3) at (3,0) {$s^2(\bot)$};
\node (4) at (4,0) {$\cdots$};
\path (0) edge (1);
\path (1) edge (2);
\path (2) edge (down);
\path (3) edge (0);
\end{tikzpicture}
\end{center}

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

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

\begin{center}
\begin{tikzpicture}
\node (0) at (0,0) {$0$};
\node (1) at (1,0) {$1$};
\node (down) at (1,-1) {$\bot$};
\node (4) at (2,0) {$\cdots$};
\path (0) edge (1);
\path (1) edge (down);
\end{tikzpicture}
\end{center}

$T$-list object with $(+, 0)$ structure and $T$ the lifting monad
Monoids with compatible algebraic structure
$T$-monoids

Let $(T, \tau)$ be a strong monad on a monoidal category $(\otimes, I)$. A $T$-monoid (also called $EM$-monoid [Pir16]) is a monoid equipped with a $T$-algebra $I : M \to TM \otimes M$ compatible in the sense that $T(C) \otimes C \cong TC \otimes C$.

Remark: $T$-monoids extend both monoids and $T$-algebras.
$T$-monoids

Let $(T, st)$ be a strong monad on a monoidal category $(\otimes, I)$. A $T$-monoid ($EM$-monoid [Pir16]) is a monoid

$$
I \longrightarrow M \leftarrow M \otimes M
$$
$T$-monoids

Let $(T, st)$ be a strong monad on a monoidal category $(\otimes, I)$. A $T$-monoid (EM-monoid [Pir16]) is a monoid equipped with a $T$-algebra

\[
\begin{array}{c}
TM \\
\downarrow \tau \\
I & \longrightarrow & M & \longleftarrow & M \otimes M
\end{array}
\]
T-monoids

Let \((T, \text{st})\) be a strong monad on a monoidal category \((\otimes, I)\). A \textit{T-monoid} (\textit{EM-monoid} [Pir16]) is a monoid equipped with a \textit{T-algebra}

\[
\begin{array}{c}
T \times \text{M} \\
\downarrow \tau \\
M \\
\end{array}
\]

\[I \rightarrow M \leftarrow M \otimes M\]

compatible in the sense that

\[
\begin{array}{c}
T(C) \otimes C \xrightarrow{\text{st}_{C,C}} T(C \otimes C) \xrightarrow{Tm} TC \\
C \otimes C \xrightarrow{m} C
\end{array}
\]

\[c \otimes c \downarrow \]

\[c \]

Remark \text{ T-monoids extend both monoids and T-algebras.}
**$T$-monoids**

Let $(T, st)$ be a strong monad on on a monoidal category $(\otimes, I)$. A $T$-monoid ($EM$-monoid [Pir16]) is a monoid equipped with a $T$-algebra

\[
\begin{array}{ccc}
TM & \Downarrow\tau \\
I & \rightarrow & M \\
& & \leftarrow M \otimes M
\end{array}
\]

compatible in the sense that

\[
\begin{array}{ccc}
T(C) \otimes C & \xrightarrow{st_{C,C}} & T(C \otimes C) \\
C \otimes C & \xrightarrow{m} & C
\end{array}
\]

\[
\begin{array}{ccc}
T(C) \otimes C & \xrightarrow{Tm} & TC
\end{array}
\]

**Remark**

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

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 $M$ the endofunctor $M \otimes (\_)$ is a monad, and $\left(M \otimes (\_)\right)\text{-Mon} (C) \simeq \left(M / \text{Mon}(C)\right)$. 
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 $M$ the endofunctor $M \otimes (\_)$ is a monad, and $(M \otimes (\_))\text{-Mon}(C) \simeq (M/\text{Mon}(C))$.

Example

In particular, for $S$ a signature on a monoid $M$ in the sense of [Jas09], an $(S \otimes \_)$-monoid for the endofunctor $S \otimes (\_)$ is precisely an algebraic operation with signature $S$ can be identified with a map $S \xrightarrow{\eta} L(S) \rightarrow M$ interpreting $S$ inside $M$. 
$T$-list objects are free $T$-monoids
$T$-list objects are free $T$-monoids

For a strong monad $(T, st)$ on a monoidal category $(\otimes, I)$,
$T$-list objects are free $T$-monoids

For a strong monad $(T, st)$ on a monoidal category $(\otimes, I)$,

**Lemma**

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


\textbf{Lemma}

1. Every \( T \)-\textit{list object} \( M(X) \) is a \( T \)-\textit{monoid}.

2. This \( T \)-\textit{monoid} is the free \( T \)-\textit{monoid} on \( X \), with universal map

\[
\begin{align*}
X & \xrightarrow{\rho} X \otimes I \\
& \xrightarrow{X \otimes \text{nil}} X \otimes M(X) \\
& \xrightarrow{\text{cons}} M(X)
\end{align*}
\]
**T-list objects are free T-monoids**

For a strong monad \((T, st)\) on a monoidal category \((\otimes, I)\),

**Lemma**

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

2. *This T-monoid is the free T-monoid on* \(X\), *with universal map*

\[
\begin{align*}
X & \xrightarrow{\eta} X \otimes I \\
& \xrightarrow{X \otimes \text{nil}} X \otimes M(X) \\
& \xrightarrow{\text{cons}} M(X)
\end{align*}
\]

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 $(T, st)$ on a monoidal category $(\otimes, I)$,

**Lemma**

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

Let $T$ be a strong monad on a monoidal category $(C, I, \otimes)$ with binary coproducts $(+)$. If

1. for every $P \in C$, the endofunctor $(-) \otimes P$ preserves binary coproducts, and

2. for every $X \in C$, the initial algebra of $T(I + X \otimes -)$ exists

Then $C$ has all $T$-list objects and, thereby, the free $T$-monoid monad $M_T$. 

Remark

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

Let $T$ be a strong monad on a monoidal category $(\mathcal{C}, I, \otimes)$ with binary coproducts $(+)$. If

1. for every $P \in \mathcal{C}$, the endofunctor $(-) \otimes P$ preserves binary coproducts, and

2. for every $X \in \mathcal{C}$, the initial algebra of $T(I + X \otimes -)$ exists

Then $\mathcal{C}$ has all $T$-list objects and, thereby, the free $T$-monoid monad $M_T$.

Remark

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

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

\[ \mu A. (I + X \otimes A) \rightsquigarrow \text{list object} \rightsquigarrow \text{free monoid} \]
Technical contribution

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

\[ T\text{-list object} \]
Technical contribution

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

\[ T\text{-list object} \rightsquigarrow \text{free } T\text{-monoid} \]
Technical contribution

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

\[ \mu A. T(I + X \otimes A) \rightsquigarrow T\text{-list object} \rightsquigarrow \text{free } T\text{-monoid} \]
Technical contribution

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

$$\mu A. T(I + X \otimes A) \leadsto T\text{-list object } \leadsto \text{free } T\text{-monoid}$$

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 monoidal category \((\otimes, I)\):

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

In \textbf{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 $\text{Lt}(T)X := \mu A. T(1 + X \times A)$ is just the free $T$-monoid monad.
Functional programming

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

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

Functional programming

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

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

- In the category of endofunctors over a cartesian category: Spivey’s Bunch type class

  \[
  \text{Bun}(F)X := \mu A.(1 + X \times A + F(A) \times A + A \times A)
  \]

  is a \( T \)-list object for \( T \) an extension of the theory of monoids.
Applications

Functional programming

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

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

- In an nsr-category:

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

is a $T$-list object for $T$ an extension of the theory of monoids.
Applications

Abstract syntax and variable binding (Fiore et al.)

In the category of presheaves $\mathbf{Set}^F$ with substitution tensor product

$$(P \bullet Q)(n) = \int_{m \in F} (Pn) \times (Qm)^n$$
Applications

Abstract syntax and variable binding (Fiore et al.)

In the category of presheaves $\text{Set}^F$ with substitution tensor product

$$(P \bullet Q)(n) = \int_{m \in F} (Pn) \times (Qm)^n$$

we get

abstract syntax = free $T$-monoid on variables

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

In the category of presheaves $\mathbf{Set}^F$ with substitution tensor product

$$(P \bullet Q)(n) = \int_{m \in F} (Pn) \times (Qm)^n$$

we get

abstract syntax = free $T$-monoid on variables

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

abstract syntax is a list object with algebraic structure
Abstract syntax and variable binding (Fiore et al.)

In the category of presheaves $\text{Set}^\mathbb{F}$ with substitution tensor product

$$(P \bullet Q)(n) = \int_{m \in \mathbb{F}} (Pn) \times (Qm)^n$$

we get

abstract syntax = free $T$-monoid on variables

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

Remark

This relies on a slightly more general theory, in which the strength $st_{X,l \to P} : T(X) \otimes P \to T(X \otimes P)$ 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*

\[ \mu A. (I + X \otimes A) \leadsto \text{list object} \leadsto \text{free monoid} \]
\[ \mu A. T(I + X \otimes A) \leadsto T\text{-list object} \leadsto \text{free } T\text{-monoid} \]
Summary: *List objects with algebraic structure*

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

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

\[ \mu A. (I + X \otimes A) \rightsquigarrow \text{list object} \rightsquigarrow \text{free monoid} \]
\[ \mu A. T(I + X \otimes A) \rightsquigarrow T\text{-list object} \rightsquigarrow \text{free } T\text{-monoid} \]

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

Algebraic structure \(\rightsquigarrow\) list-style datatype. Simpler proofs!
(e.g. abstract syntax, opetopes?)
Summary: *List objects with algebraic structure*

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

\[
\mu A. T(I + X \otimes A) \rightsquigarrow T\text{-list object} \rightsquigarrow \text{free } T\text{-monoid}
\]

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

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

Initial algebra definition \(\rightsquigarrow\) universal property.  
(*e.g. monadic list transformer, MonadPlus*)
Summary: *List objects with algebraic structure*

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

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

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

Algebraic structure \( \leadsto \) list-style datatype. Simpler proofs! (e.g. abstract syntax, opetopes?)

Initial algebra definition \( \leadsto \) universal property. (e.g. monadic list transformer, MonadPlus)

A journal-length version is in preparation.