List Objects with Algebraic Structure

Marcelo Fiore and Philip Saville

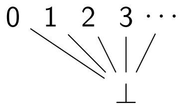
University of Cambridge Computer Laboratory

6th September 2017

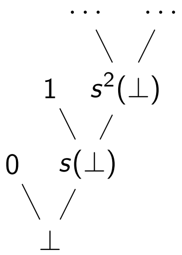
A unifying framework for many diverse examples of list objects with algebraic structure

Notions of natural number in **Cpo**

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



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



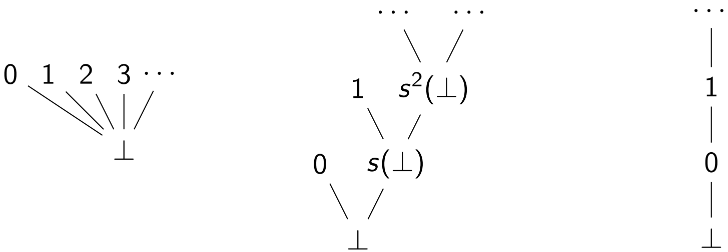
Strict natural numbers, $\mu A.A_\perp$:

Notions of natural number in **Cpo**

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

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

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



Our contribution: all these are natural numbers objects with algebraic structure.

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 $\mathrm{Lt}(T)X := \mu A. T(1 + X \times A)$.

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 $\mathrm{Lt}(T)X := \mu A. T(1 + X \times A)$.

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

Abstract syntax with binding and metavariables (Fiore *et al.*)

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

Abstract syntax with binding and metavariables (Fiore *et al.*)

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 *et al.*)

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 *et al.*)

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 *et al.*)

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 *e.g.* $\mathrm{app}(\sigma, \tau)[x \mapsto \omega] = \mathrm{app}(\sigma[x \mapsto \omega], \tau[x \mapsto \omega])$.

Abstract syntax with binding and metavariables (Fiore *et al.*)

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 *e.g.* $\mathrm{app}(\sigma, \tau)[x \mapsto \omega] = \mathrm{app}(\sigma[x \mapsto \omega], \tau[x \mapsto \omega])$.

$$\textbf{Abstract syntax} = \textbf{free such structure}$$
$$= \textbf{a list object with algebraic structure.}$$

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 $\mu A.(I + X \otimes A)$.

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

This talk

list objects $\rightsquigarrow$ **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 $\rightsquigarrow$ ***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.

Gives an algebraic structure for T-list objects.

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

A *list object* $\mathrm{L}(X)$ on X consists of

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

A *list object* $\mathrm{L}(X)$ on X consists of

$$1 \xrightarrow{nil} \mathrm{L}(X)$$

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

A *list object* $\mathrm{L}(X)$ on X consists of

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

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

A *list object* $\mathrm{L}(X)$ on X consists of

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

that is initial:

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

A *list object* $\mathrm{L}(X)$ on X consists of

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

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

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

List objects in a monoidal category $(\mathcal{C}, \otimes, I)$

List objects in a monoidal category $(\mathcal{C}, \otimes, I)$

A *list object* $\mathrm{L}(X)$ on X consists of

$$I \xrightarrow{nil} \mathrm{L}(X) \xleftarrow{cons} X \otimes \mathrm{L}(X)$$

List objects in a monoidal category $(\mathcal{C}, \otimes, I)$

A *list object* $\mathrm{L}(X)$ on X consists of

$$I \xrightarrow{nil} \mathrm{L}(X) \xleftarrow{cons} X \otimes \mathrm{L}(X)$$

that is *parametrised initial*:

List objects in a monoidal category $(\mathcal{C}, \otimes, I)$

A *list object* $\mathrm{L}(X)$ on X consists of

$$I \xrightarrow{nil} \mathrm{L}(X) \xleftarrow{cons} X \otimes \mathrm{L}(X)$$

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

$$\begin{tikzcd}
I \otimes P \arrow[r, "nil\otimes P"] \arrow[d, "\cong"'] & \mathrm{L}(X) \otimes P \arrow[d, dashed, "\mathrm{it}(n,c)"] & X \otimes \mathrm{L}(X) \otimes P \arrow[l, "cons\otimes P"'] \arrow[d, "X\otimes\mathrm{it}(n,c)"] \\
P \arrow[r, "n"'] & A & X \otimes A \arrow[l, "c"]
\end{tikzcd}$$

List objects in a monoidal category $(\mathcal{C}, \otimes, I)$

Remark

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

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

List objects in a monoidal category $(\mathcal{C}, \otimes, I)$

Connection to past work

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

List objects are free monoids

Lemma

1. *Every list object* $\mathrm{L}(X)$ *is a monoid.*
2. *This monoid is the free monoid on* X*, with universal map*

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

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

List objects are free monoids

Lemma

1. *Every list object $\mathrm{L}(X)$ is a monoid.*
2. *This monoid is the free monoid on X, with universal map*

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

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

List objects are initial algebras

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

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

List objects are initial algebras

Definition

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

Lemma

If $(\mathcal{C}, \otimes, I)$ *is a monoidal category with finite coproducts* $(0, +)$ *and* ω*-colimits, both preserved by all* $(-) \otimes P$ *for* $P \in \mathcal{C}$*, then the initial algebra of the functor* $\big(I + X \otimes (-)\big)$ *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
  $\mu A.(I + X \otimes A)$.

Rest of this talk

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

(new work)

list objects

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

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

(new work)

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

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

...and instantiate this for applications

Compatible algebraic structure

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 $\eta : \mathrm{Id}_{\mathcal{C}} \to T$ satisfying associativity and unit 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 $\eta : \mathrm{Id}_{\mathcal{C}} \to T$ 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 $\mathcal{C}$ is a functor $T : \mathcal{C} \to \mathcal{C}$ equipped with a multiplication $\mu : T^2 \to T$ and a unit $\eta : \mathrm{Id}_{\mathcal{C}} \to T$ 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 $(\otimes, I)$ that is equipped with a natural transformation $st_{A,B} : T(A) \otimes B \to T(A \otimes B)$ satisfying coherence laws.

List objects with algebraic structure

T-list objects

T-list objects

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

$$I \xrightarrow[nil]{} \mathrm{M}(X) \xleftarrow[cons]{} X \otimes \mathrm{M}(X)$$

T-list objects

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

$$\begin{tikzcd}
 & T(\mathrm{M}(X)) \arrow[d, "\tau"] & \\
I \arrow[r, "nil"'] & \mathrm{M}(X) & X \otimes \mathrm{M}(X) \arrow[l, "cons"']
\end{tikzcd}$$

T-list objects

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

$$\begin{tikzcd} & T(\mathrm{M}(X)) \arrow[d, "\tau"] & \\ I \arrow[r, "nil"'] & \mathrm{M}(X) & X \otimes \mathrm{M}(X) \arrow[l, "cons"] \end{tikzcd}$$

such that for every structure

$$\begin{tikzcd} & TA \arrow[d, "\alpha"] & \\ P \arrow[r, "n"'] & A & X \otimes A \arrow[l, "c"] \end{tikzcd}$$

T-list objects

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

$$
\begin{tikzcd}
 & T(\mathrm{M}(X)) \arrow[d, "\tau"] & \\
I \arrow[r, "nil"'] & \mathrm{M}(X) & X \otimes \mathrm{M}(X) \arrow[l, "cons"]
\end{tikzcd}
$$

such that for every structure

$$
\begin{tikzcd}
 & TA \arrow[d, "\alpha"] & \\
P \arrow[r, "n"'] & A & X \otimes A \arrow[l, "c"]
\end{tikzcd}
$$

there exists a unique mediating map $\mathrm{it}(n, c, \alpha) : \mathrm{M}(X) \otimes P \to A$

T-list objects

such that

$$\begin{tikzcd}
I \otimes P \arrow[r, "nil \otimes P"] \arrow[d, "\cong"'] & \mathrm{M}(X) \otimes P \arrow[d, dashed, "\mathrm{it}(n,c,\alpha)"] & X \otimes \mathrm{M}(X) \otimes P \arrow[l, "cons \otimes P"'] \arrow[d, "X \otimes \mathrm{it}(n,c,\alpha)"] \\
P \arrow[r, "n"'] & A & X \otimes A \arrow[l, "c"]
\end{tikzcd}$$

and

$$\begin{tikzcd}
T\big(\mathrm{M}(X)\big) \otimes P \arrow[r, "st_{\mathrm{M}(X),P}"] \arrow[d, "\tau \otimes P"'] & T(\mathrm{M}(X) \otimes P) \arrow[r, "T(\mathrm{it}(n,c,\alpha))"] & TA \arrow[d, "\alpha"] \\
\mathrm{M}(X) \otimes P \arrow[rr, dashed, "\mathrm{it}(n,c,\alpha)"'] & & A
\end{tikzcd}$$

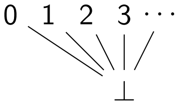
T-list objects

Remark

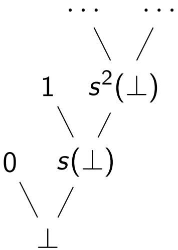
Every list object is a T-list object.
If every $(-) \otimes P$ has a right adjoint, the iterator $\mathrm{it}(n, c, \alpha)$ is a T-algebra homomorphism.

Natural numbers in **Cpo**, revisited

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



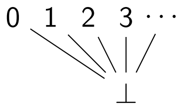
Lazy natural numbers, $\mu A.(1 + A)_\perp$:



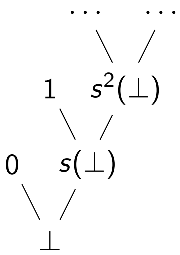
Strict natural numbers, $\mu A.A_\perp$:

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

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



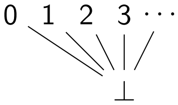
Lazy natural numbers, $\mu A.(1 + A)_\perp$:



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

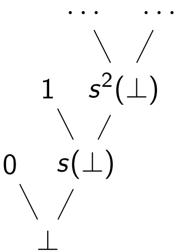
Natural numbers in **Cpo** as T-list objects on the unit

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



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

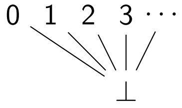
Lazy natural numbers, $\mu A.(1 + A)_\perp$:



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

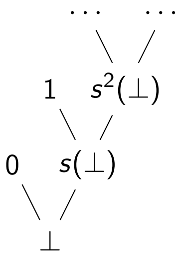
Natural numbers in **Cpo** as T-list objects on the unit

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



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

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

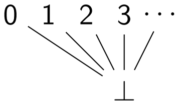


T-list object with $(\times, 1)$ structure and $T := (-)_\perp$ the lifting monad

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

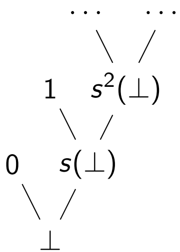
Natural numbers in **Cpo** as T-list objects on the unit

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



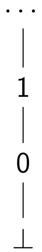
T-list object with $(\times, 1)$ structure and monad $T = \mathrm{Id}$

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



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

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



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

Monoids with compatible algebraic structure

T-monoids

T-monoids

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

$$I \longrightarrow M \longleftarrow M \otimes M$$

T-monoids

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

$$\begin{tikzcd} & TM \arrow[d, "\tau"] & \\ I \arrow[r] & M & M \otimes M \arrow[l] \end{tikzcd}$$

T-monoids

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

$$\begin{tikzcd} & TM \arrow[d, "\tau"] & \\ I \arrow[r] & M & M \otimes M \arrow[l] \end{tikzcd}$$

compatible in the sense that

$$\begin{tikzcd} T(C) \otimes C \arrow[r, "st_{C,C}"] \arrow[d, "c \otimes C"'] & T(C \otimes C) \arrow[r, "Tm"] & TC \arrow[d, "c"] \\ C \otimes C \arrow[rr, "m"'] & & C \end{tikzcd}$$

T-monoids

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

$$\begin{tikzcd} & TM \arrow[d, "\tau"] & \\ I \arrow[r] & M & M \otimes M \arrow[l] \end{tikzcd}$$

compatible in the sense that

$$\begin{tikzcd} T(C) \otimes C \arrow[r, "st_{C,C}"] \arrow[d, "c\otimes C"'] & T(C \otimes C) \arrow[r, "Tm"] & TC \arrow[d, "c"] \\ C \otimes C \arrow[rr, "m"'] & & C \end{tikzcd}$$

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 M the endofunctor $T := M \otimes (-)$ is a monad, and T-$Mon(\mathcal{C}) \simeq \big(M/Mon(\mathcal{C})\big)$.

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 M the endofunctor $T := M \otimes (-)$ is a monad, and T-Mon$(\mathcal{C}) \simeq \big(M/Mon(\mathcal{C})\big)$.

Example

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

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 M the endofunctor $T := M \otimes (-)$ is a monad, and $T\text{-}Mon(\mathcal{C}) \simeq \big(M/Mon(\mathcal{C})\big)$.

Example

Thinking of a Lawvere theory as a monoid L_M in $\big(\mathbf{Set}^{\mathbb{F}}, \mathbf{y}(1), \bullet\big)$, we can identify Lawvere theories extending L_M with T-monoids for $T := M \bullet (-)$.

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 $\mathrm{M}(X)$ is a T-monoid.*

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 $\mathrm{M}(X)$ is a T-monoid.*
2. *This T-monoid is the free T-monoid on X, with universal map*

$$X \xrightarrow{\cong} X \otimes I \xrightarrow{X \otimes nil} X \otimes \mathrm{M}(X) \xrightarrow{cons} \mathrm{M}(X)$$

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 $\mathrm{M}(X)$ is a T-monoid.*
2. *This T-monoid is the free T-monoid on X, with universal map*

$$X \xrightarrow{\cong} X \otimes I \xrightarrow{X \otimes nil} X \otimes \mathrm{M}(X) \xrightarrow{cons} \mathrm{M}(X)$$

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, 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 $(\mathcal{C}, I, \otimes)$ 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.

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

Technical contribution

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

Technical contribution

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

T-list object

Technical contribution

$\mu A.(I + X \otimes A) \rightsquigarrow$ list object $\rightsquigarrow$ free monoid

T-list object $\rightsquigarrow$ free T-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

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

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

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 $(\otimes, I)$:

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

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

Applications

Functional programming

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

Applications

Functional programming

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

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

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 $\mathrm{Lt}(T)X := \mu A.T(1 + X \times A)$ is just the free T-monoid monad.
- In an nsr-category: the $\mathrm{MonadPlus}$ type class $\mathrm{Mp}(F)X := \mu A.\mathrm{List}_*(X + F \otimes A)$ is a List_*-list object.
- In an nsr-category:

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

  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 *et al.*)

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

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

Applications

Abstract syntax and variable binding (Fiore *et al.*)

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

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

we get

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

Applications

Abstract syntax and variable binding (Fiore *et al.*)

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

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

we get

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

abstract syntax is a list object with algebraic structure

Applications

Abstract syntax and variable binding (Fiore *et al.*)

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

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

we get

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

Remark

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

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

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.

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, $\mathrm{MonadPlus}$)

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, $\mathrm{MonadPlus}$)

A journal-length version is in preparation.