Functional Data Structures in Monoidal Categories

Zhixuan Yang

March, 2025

Outline

- Background: clever people have come up with efficient *purely functional* data structures

Outline

- Background: clever people have come up with efficient *purely functional* data structures
- Observation: many of them type check in *linear* languages

Outline

- Background: clever people have come up with efficient *purely functional* data structures
- Observation: many of them type check in *linear* languages
- Idea: we can interpret them in suitable *monoidal categories*, such as endofunctors with composition

Outline

- Background: clever people have come up with efficient *purely functional* data structures
- Observation: many of them type check in *linear* languages
- Idea: we can interpret them in suitable *monoidal categories*, such as endofunctors with composition
- Profit: *asymptotically* faster free monads (syntax trees supporting pattern matching and substitution), etc

Live Coding

Let's recap on these data structures [Okasaki 1998]!

	head / *tail*	*cons*	*snoc*	$xs ++ ys$
cons lists	$O(1)$	$O(1)$	$O(n)$	$O(\|xs\|)$
snoc lists	$O(n)$	$O(n)$	$O(1)$	$O(\|ys\|)$
queues*	$O(1)$	$O(1)$	$O(1)$	$O(\|ys\|)$
catenable lists*	$O(1)$	$O(1)$	$O(1)$	$O(1)$

* amortised complexity

(Now go to the accompanying code List.hs, SnocList.hs, Queue.hs, CList.hs).

Monoidal Languages

A *monoidal language* $\mathscr{L} := \langle \mathscr{B}, \mathscr{P} \rangle$ is parameterised by

1. a set $\mathscr{B}$ of *base types*;

Types of $\mathscr{L}$ are generated by

$$\frac{\alpha \in \mathscr{B}}{\vdash \alpha \text{ type}} \qquad \frac{}{\vdash I \text{ type}} \qquad \frac{\vdash A \text{ type} \qquad \vdash B \text{ type}}{\vdash A \,\square\, B \text{ type}}$$

Monoidal Languages

A *monoidal language* $\mathscr{L} := \langle \mathscr{B}, \mathscr{P} \rangle$ is parameterised by

1. a set $\mathscr{B}$ of *base types*;
2. a family of sets $\mathscr{P}(A, B)$ indexed by pairs of types A and B;

Every element $f \in \mathscr{P}(A, B)$ is called a *primitive operation*.

Monoidal Languages

Contexts are finite lists of variables and types.

Terms under contexts are generated by

$$\frac{}{x : A \vdash x : A} \qquad \frac{f \in \mathscr{P}(A, B) \qquad \Gamma \vdash t : A}{\Gamma \vdash f\ t : B} \qquad \frac{\Gamma_1 \vdash t_1 : A \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1, \Gamma_2 \vdash (t_1, t_2) : A \mathbin{\square} B}$$

$$\frac{\Gamma \vdash t_1 : A_1 \mathbin{\square} A_2 \qquad \Gamma_l, x_1 : A_1, x_2 : A_2, \Gamma_r \vdash t_2 : B}{\Gamma_l, \Gamma, \Gamma_r \vdash \mathit{let}\ (x_1, x_2) = t_1\ \mathit{in}\ t_2 : B} \qquad \ldots$$

Example

Given base types $\{Egg, Oil, Rice\}$ and operations

$$beat \in \mathscr{P}(Egg, Egg),\ fry \in \mathscr{P}(Oil \,\square\, Rice, Rice),\ mix \in \mathscr{P}(Egg \,\square\, Rice, Rice)$$

we have a term:

$$e : Egg, o : Oil, r : Rice \vdash mix\ (beat\ e,\ fry\ (o, r)) : Rice$$

Symmetric Monoidal Languages

Symmetric monoidal languages additionally allow variables in the context to be reordered:

$$\frac{\Gamma_1 \vdash t_1 : A \qquad \Gamma_2 \vdash t_2 : B}{\Gamma_1, \Gamma_2 \vdash (t_1, t_2) : A \mathbin{\square} B} \qquad \frac{\Gamma_1 \vdash t_1 : A \qquad \Gamma_2 \vdash t_2 : B \qquad \Gamma = \Gamma_1 \uplus \Gamma_2}{\Gamma \vdash (t_1, t_2) : A \mathbin{\square} B}$$

and similarly for all rules.

Optional Type Formers

Right linear function types

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.\, t : B/A} \qquad \frac{\Gamma_1 \vdash t_1 : B/A \qquad \Gamma_2 \vdash t_2 : A}{\Gamma_1, \Gamma_2 \vdash t_1\ t_2 : B}$$

Optional Type Formers

Cartesian product types

$$\frac{\Gamma \vdash t_1 : A_1 \qquad \Gamma \vdash t_2 : A_2}{\Gamma \vdash \langle t_1, t_2 \rangle : A_1 \times A_2}$$

$$\frac{\Gamma \vdash t_1 : A_1 \times A_2 \qquad \Gamma_l, x : A_i, \Gamma_r \vdash t_2 : B}{\Gamma_l, \Gamma, \Gamma_r \vdash t_2[\pi_i\ t_1/x] : B}\ i \in \{1, 2\}$$

*not needed in this talk

Optional Type Formers

Coproduct types

$$\frac{\Gamma \vdash t : A_i}{\Gamma \vdash \iota_i\ t : A_1 + A_2}\ i \in \{1, 2\}$$

$$\frac{\Gamma \vdash t : A_1 + A_2 \qquad x_i : A_i \vdash t_i : C, i \in \{1, 2\}}{\Gamma \vdash \mathit{case}\ t\ \mathit{of}\ \{\iota_1\ x_i \mapsto t_1;\ \iota_2\ x_2 \mapsto t_2\} : C}$$

Optional Type Formers

Inductive types

$$\frac{\Theta, \alpha \vdash T \text{ type} \qquad \alpha \text{ occurs strictly positively in } T}{\Theta \vdash \mu\alpha.\, T \text{ type}}$$

$$\frac{\Gamma \vdash t : T[\mu\alpha.\, T/\alpha]}{\Gamma \vdash \mathit{cons}\ t : \mu\alpha.\, T} \qquad \frac{\Gamma \vdash i : \mu\alpha.\, T \qquad x : T[A/\alpha] \vdash a : A}{\Gamma \vdash \mathit{fold}\ a\ i : A}$$

and similarly for inductive *nested* types (such as *CList*).

Observation

The clever implementations of lists can be implemented in the monoidal language (with functions, inductive types, and coproducts).

E.g., for every type A, define $L\ A := \mu X.\ I + A \mathbin{\square} X$. Concatenation is like

$$\frac{a := \big(x : I + A \mathbin{\square} (L\ A)/(L\ A) \vdash \cdots : (L\ A)/(L\ A)\big)}{i : L\ A \vdash \mathit{fold}\ a\ i : (L\ A)/(L\ A)}$$

Recap

A *category* $\mathscr{C}$ consists of

1. (*objects*) a set $\textsc{Obj}\,\mathscr{C}$,
2. (*morphisms*) a family of sets $\mathscr{C}(A, B)$ for every $A, B \in \textsc{Obj}\,\mathscr{C}$,
3. (*identity*) an element $id_A \in \mathscr{C}(A, A)$ for every $A \in \textsc{Obj}\,\mathscr{C}$,
4. (*composition*) an element $g \cdot f \in \mathscr{C}(A, C)$ for every $f \in \mathscr{C}(A, B)$, $g \in \mathscr{C}(B, C)$,

subject to ….

Recap

A *functor* $F : \mathscr{C} \longrightarrow \mathscr{D}$ consists of

1. (*object mapping*) a function $F_0 : \mathscr{C} \longrightarrow \mathscr{D}$,
2. (*morphism mapping*) a family of functions for all $A, B \in \text{Obj}\, C$

$$F_1 : \mathscr{C}(A, B) \longrightarrow \mathscr{D}(F_0 A, F_0 B),$$

such that F_1 preserves identities and composition.

Recap

Let $F, G : \mathscr{C} \to \mathscr{D}$. A *natural transformation* $\alpha : F \to G$ is a family of morphisms $\alpha_A \in \mathscr{D}(FA, GA)$, for all $A \in \mathscr{C}$, such that

$$\begin{array}{ccc} FA & \xrightarrow{Ff} & FB \\ {\scriptstyle \alpha_A}\downarrow & & \downarrow{\scriptstyle \alpha_B} \\ GA & \xrightarrow[Gf]{} & GB \end{array}$$

commutes for all $A, B \in \mathscr{C}, f \in \mathscr{C}(A, B)$.

Functors and nat. transformations form a category $\mathscr{D}^{\mathscr{C}}$.

Strict Monoidal Categories

A *strict monoidal category* $\langle \mathscr{C}, \Box, I \rangle$ is a category $\mathscr{C}$ and a functor $\Box : \mathscr{C} \longrightarrow \mathscr{C}^{\mathscr{C}}$ (i.e. $\mathscr{C} \times \mathscr{C} \longrightarrow \mathscr{C}$) and an object $I \in \mathscr{C}$ such that

- $I \Box A = A = A \Box I$,
- $(A \Box B) \Box C = A \Box (B \Box C)$

for all $A, B, C \in \textsc{Obj}\,\mathscr{C}$, and similarly for morphisms:

- $id_A \Box f = f = f \Box id_A$,
- $(f \Box g) \Box h = f \Box (g \Box h)$.

Strict Monoidal Categories

A *strict monoidal category* $\langle \mathscr{C}, \Box, I \rangle$ is a category $\mathscr{C}$ and a functor $\Box : \mathscr{C} \to \mathscr{C}^{\mathscr{C}}$ (i.e. $\mathscr{C} \times \mathscr{C} \to \mathscr{C}$) and an object $I \in \mathscr{C}$ such that

$\dots$

For every $\mathscr{C}$, $\langle \mathscr{C}^{\mathscr{C}}, \circ, \mathrm{Id} \rangle$ is a strict monoidal category, where $\circ : \mathscr{C}^{\mathscr{C}} \times \mathscr{C}^{\mathscr{C}} \to \mathscr{C}^{\mathscr{C}}$ is functor composition, and $\mathrm{Id} \in \mathscr{C}^{\mathscr{C}}$ is the identity functor.

Interpretation

An *interpretation* of $\mathscr{L} = \langle \mathscr{B}, \mathscr{P} \rangle$ in a monoidal category $\mathscr{E}$ is

1. an assignment of $\mathscr{E}$-objects $[\![\alpha]\!]$ to each base type $\alpha \in \mathscr{B}$, which induces the interpretation of all types and contexts:

$$[\![I]\!] = I_{\mathscr{E}} \qquad [\![A \mathbin{\square} B]\!] = [\![A]\!] \mathbin{\square_{\mathscr{E}}} [\![B]\!]$$

$$[\![\cdot]\!] = I_{\mathscr{E}} \qquad [\![\Gamma, x : A]\!] = [\![\Gamma]\!] \mathbin{\square_{\mathscr{E}}} [\![A]\!]$$

2. …

Interpretation

An *interpretation* of $\mathscr{L} = \langle \mathscr{B}, \mathscr{P} \rangle$ in a monoidal category $\mathscr{E}$ is

1. an assignment of $\mathscr{E}$-objects $[\![\alpha]\!]$ to each base type $\alpha \in \mathscr{B}$,
2. an assignment of $\mathscr{E}$-morphisms $[\![f]\!] : [\![A]\!] \longrightarrow [\![B]\!]$ to each primitive operation $f \in \mathscr{P}(A, B)$, which determines the interpretation of all terms:

$$[\![x]\!] = id \qquad [\![f\ t]\!] = [\![f]\!] \cdot [\![t]\!] \qquad [\![(t_1, t_2)]\!] = [\![t_1]\!] \square_{\mathscr{E}} [\![t_2]\!]$$

$$[\![let\ (x_1, x_2) = t_1\ in\ t_2]\!] = [\![t_2]\!] \cdot ([\![\Gamma_l]\!] \square_{\mathscr{E}} [\![t_1]\!] \square_{\mathscr{E}} [\![\Gamma_r]\!])$$

Interpretation

To interpret the optional type formers

$$B/A \qquad A \times B \qquad A + B \qquad \mu\alpha.\, T$$

the monoidal category $\mathscr{E}$ needs to satisfy some properties.

Under some condition on $\mathscr{C}$, the monoidal category $\langle \mathscr{C}^{\mathscr{C}}, \circ, \mathrm{Id} \rangle$ does have these type formers.

Data Structure in Monoidal Categories

The clever functional data structures can be interpreted in the monoidal category $\langle \mathscr{C}^{\mathscr{C}}, \circ, \mathrm{Id} \rangle$.

For now, let's do the interpretation manually.

(Now go to the accompanying code FastFree.hs and LamPHOAS.hs).

Next Steps

- Prove the complexity of *CListF* rigorously, or even mechanically using CALF?
- Compare the complexity of substitution-based λ-normaliser using *CListF* with *normalising by evaluation*
- Explore other algorithms and data structures
- Make the translation automatic. Bernardy and Spiwack's LINEAR-SMC looks promising

Thank You

Find structural similarity between simple and complex things, so complex things become simple.

(old Chinese proverb, circa 2025)