Containers for Effects and Contexts: Lecture 1

Tarmo Uustalu, Institute of Cybernetics, Tallinn

University of Oxford, 6–10 July 2015
This course

- We will think about computational effects and contexts as modelled with monads, comonads and related machinery.

- We will primarily be interested in questions like: Where do they come from? How to generate them? How many are they? And also: How to arrive at answers to such questions with as little work as possible?

- In other words, we will amuse ourselves with the combinatorics of monads etc.

- The main tool: Containers (possibly quotient containers). But not today.

- Today’s ambition: Monads, monad maps and distributive laws.
Useful prior knowledge

- This is not strictly needed, but will help.
- Basics of functional programming and the use of monads (and perhaps idioms, comonads) in functional programming.
- From category theory:
  - functors, natural transformations
  - adjunctions
  - symmetric monoidal (closed) categories
  - Cartesian (closed) categories, coproducts
  - initial algebra, final coalgebra of a functor
  - ...:-(

All examples however will be for **Set.** :-)

(But many generalize to any Cartesian (closed) or monoidal (closed) category.)
Monads
Monads

A monad on a category $\mathcal{C}$ is given by a

- a functor $T : \mathcal{C} \to \mathcal{C}$,
- a natural transformation $\eta : \text{Id}_\mathcal{C} \to T$ (the unit),
- a natural transformation $\mu : T \cdot T \to T$ (the multiplication)

such that

This definition says that monads are monoids in the monoidal category $([\mathcal{C}, \mathcal{C}], \text{Id}_\mathcal{C}, \cdot)$. 
An alternative formulation: Kleisli triples

- A more FP-friendly formulation is this.
  - A Kleisli triple is given by
    - an object mapping $T : |C| \to |C|$, for any object $A$, a map $\eta_A : A \to TA$, for any map $k : A \to TB$, a map $k^* : TA \to TB$ (the Kleisli extension operation)
    
    such that
    - if $k : A \to TB$, then $k^* \circ \eta_A = k$, $\eta^*_A = \text{id}_{TA}$,
    - if $k : A \to TB$, $\ell : B \to TC$, then $(\ell^* \circ k)^* = \ell^* \circ k^* : TA \to TC$.
    
    (Notice there are no explicit functoriality and naturality conditions.)
Monads = Kleisli triples

There is a bijection between monads and Kleisli triples.

Given $T$, $\eta$, $\mu$, one defines

if $k : A \to TB$, then $k^* = TA \xrightarrow{Tk} T(TB) \xrightarrow{\mu_B} TB$.

Given $T$ (on objects only), $\eta$ and $-^*$, one defines

if $f : A \to B$, then

$$Tf = \left( A \xrightarrow{f} B \xrightarrow{\eta_B} TB \right)^* : TA \to TB,$$

$$\mu_A = \left( TA \xrightarrow{id_{TA}} TA \right)^* : T(TA) \to TA.$$
Kleisli category of a monad

- A monad $T$ on a category $C$ induces a category $\text{Kl}(T)$ called the *Kleisli category* of $T$ defined by
  - an object is an object of $C$,
  - a map of from $A$ to $B$ is a map of $C$ from $A$ to $TB$,
  - $\text{id}_A^T = A \xrightarrow{\eta_A} TA$,
  - if $k : A \to^T B$, $\ell : B \to^T C$, then $\ell \circ^T k = A \xrightarrow{k} TB \xrightarrow{T\ell} T(TC) \xrightarrow{\mu_C} TC$

- From $C$ there is an identity-on-objects *inclusion* functor $J$ to $\text{Kl}(T)$, defined on maps by
  - if $f : A \to B$, then $Jf = A \xrightarrow{f} B \xrightarrow{\eta_B} TB = A \xrightarrow{\eta_A} TA \xrightarrow{Tf} TB$. 
Monad algebras

- An *algebra* of a monad \((T, \eta, \mu)\) is an object \(A\) with a map \(a : TA \to A\) such that

\[
\begin{array}{ccc}
A & \xrightarrow{\eta_A} & TA \\
\downarrow & & \downarrow \\
TA & \xrightarrow{a} & A
\end{array}
\]

\[
\begin{array}{ccc}
T(TA) & \xrightarrow{Ta} & TA \\
\downarrow & & \downarrow \\
TA & \xrightarrow{a} & A
\end{array}
\]

- A *map* between two algebras \((A, a)\) and \((B, b)\) is a map \(h\) such that

\[
\begin{array}{ccc}
TA & \xrightarrow{Th} & TB \\
\downarrow & \downarrow & \downarrow \\
A & \xrightarrow{h} & B
\end{array}
\]

- The algebras of the monad and maps between them form a category \(\text{EM}(T)\) with an obvious forgetful functor \(U : \text{EM}(T) \to C\).
Computational interpretation

- Think of $\mathcal{C}$ as the category of pure functions and of $TA$ as the type of effectful computations of values of a type $A$.
- $\eta_A : A \rightarrow TA$ is the identity function on $A$ viewed as trivially effectful.
- $Jf : A \rightarrow TB$ is a general pure function $f : A \rightarrow B$ viewed as trivially effectful.
- $\mu_A : T(TA) \rightarrow TA$ flattens an effectful computation of an effectful computation.
- $k^* : TA \rightarrow TB$ is an effectful function $k : A \rightarrow TB$ extended into one that can input an effectful computation.
- An algebra $(A, a : TA \rightarrow A)$ serves as a recipe for handling the effects in computations of values of type $A$. 
Kleisli adjunction

In the opposite direction of $J : C \to \text{KI}(T)$ there is a functor $R : \text{KI}(T) \to C$ defined by

- $RA = TA$,
- if $k : A \to^T B$, then $Rk = TA \xrightarrow{k^*} TB$.

$R$ is right adjoint to $J$.

\[
\begin{array}{ccc}
\text{KI}(T) & \xleftarrow{J} & C \\
\downarrow{R} & & \downarrow{
\begin{array}{c}
J A \\
A \to^T B \\
\end{array}
}
\end{array}
\]

Importantly, $R \cdot J = T$. Indeed,

- $R(JA) = TA$,
- if $f : A \to B$, then $R(Jf) = (\eta_B \circ f)^* = Tf$.

Moreover, the unit of the adjunction is $\eta$.

$J \dashv R$ is the initial adjunction factorizing $T$ in this way.
Eilenberg-Moore adjunction

In the opposite direction of $U : \mathbf{EM}(T) \to C$ there is a functor $L : C \to \mathbf{EM}(T)$ defined by

- $LA = (TA, \mu_A)$,
- if $f : A \to B$, then $Lk = Tf : (TA, \mu_A) \to (TB, \mu_B)$.

$L$ is left adjoint to $U$.

\[ \begin{array}{ccc}
\mathbf{EM}(T) & \xleftarrow{L} & C \\
\downarrow{U} & & \downarrow{U} \\
A & \to & B
\end{array} \]

$L \dashv U$ is the final adjunction factorizing $T$.

$U \cdot L = T$. Indeed,

- $U(LA) = U(TA, \mu_A) = TA$,
- if $f : A \to B$, then $U(Lf) = U(Tf) = Tf$.

The unit of the adjunction is $\eta$.

$L \dashv U$ is the final adjunction factorizing $T$. 

Exceptions monads

- The functor:
  \[ TA = E + A \] where \( E \) is some set (of exceptions)

- The monad structure:
  \[ \eta_A x = \text{inr} \ x, \]
  \[ \mu_A (\text{inl} \ e) = \text{inl} \ e, \]
  \[ \mu_A (\text{inr} \ (\text{inl} \ e)) = \text{inl} \ e, \]
  \[ \mu_A (\text{inr} \ (\text{inr} \ x)) = \text{inr} \ x. \]

- This is the only monad structure on this functor.

- (This example generalizes to any coCartesian category, in fact to any monoidal category with a given monoid. In a coCartesian category, any object \( E \) carries exactly one monoid structure defined by \( o = \exists_E : 0 \to E \) and \( \oplus = \nabla_E : E + E \to E \).)
Reader monads

- The functor:
  \[ TA = S \Rightarrow A \] where \( S \) is a set (of readable states)

- The monad structure:
  \[ \eta_A x = \lambda s \cdot x, \]
  \[ \mu_A f = \lambda s \cdot f s s. \]

- This is the only monad structure on this functor.

(This example generalizes to any monoidal closed category with a given comonoid. In a Cartesian closed category, any object \( S \) comes with a unique comonoid structure given by \( !_S : S \to 1, \Delta_S : S \to S \times S. \) )
Writer monads

- We are interested in this functor:
  \[ TA = P \times A \] where \( P \) is a set (of updates)

- The possible monad structures are:
  \[ \eta_A x = (o, x), \]
  \[ \mu_A (p, (p', x)) = (p \oplus p', x) \]
  where \((o, \oplus)\) is a monoid structure on \( P \) (trivial update, composition of updates)

- Monad structures on this functor are in a bijection with monoid structures on \( P \).

- (This example generalizes to any monoidal category with a given monoid.)
State monads

- The monad:
  - \( T A = S \Rightarrow S \times A \) where \( S \) is a set (of readable/overwritable states),
  - \( \eta_A x = \lambda s. (s, x) \)
  - \( \mu_A f = \lambda s. \text{let } (s', g) = f s \text{ in } g (s', x) \)

- (This example works in any monoidal closed category.)
List monad and variations

- The list monad:
  - \( TA = \text{List} A, \)
  - \( \eta_A x = [x], \)
  - \( \mu_A xss = \text{concat} xss. \)

- Some variations:
  - \( TA = \{xs : A^* \mid \text{xs is square-free}\} \)
  - \( TA = \{xs : A^* \mid \text{xs is duplicate-free}\} \)
  - \( TA = 1 + A \times A \)
  - \( TA = M_f A \)
  - \( TA = P_f A \)
  - non-empty versions of the above

- Can you characterize the algebras of these monads?
Monad maps
Monad maps

- A *monad map* between monads $T$, $T'$ on a category $C$ is a natural transformation $\tau : T \rightarrow T'$ satisfying

$$
\begin{array}{ccc}
A & \xrightarrow{\eta_A} & A \\
\downarrow{\eta_A} & & \downarrow{\eta'_A} \\
T A & \xrightarrow{\tau_A} & T' A \\
\end{array}
\quad
\begin{array}{ccc}
T(TA) & \xrightarrow{\tau TA} & T'(TA) \\
\downarrow{\mu_A} & & \downarrow{\mu'_A} \\
T'(T'A) & & T'(T'A) \\
\end{array}

$$

- Monads on $C$ and maps between them form a category $\textbf{Monad}(C)$.
- Monad maps are monoid maps in the monoidal category $([C, C], \text{Id}_C, \cdot)$ and the category of monads is the category of monoids in $([C, C], \text{Id}_C, \cdot)$. 
Kleisli triple maps

- A map between two Kleisli triples $T$, $T'$ is, for any object $A$, a map $\tau_A : TA \rightarrow T'A$ such that
  - $\tau_A \circ \eta_A = \eta'_A$,
  - if $k : A \rightarrow TB$, then $\tau_B \circ k^* = (\tau_B \circ k)^* \circ \tau_A$.

  (No explicit naturality condition on $\tau$!)

- Kleisli triples on $C$ and maps between them form a category that is isomorphic to $\textbf{Monad}(C)$. 
Monad maps vs. functors between Kleisli categories

- There is a bijection between monad maps $\tau : T \to T'$ and functors $V : \text{Kl}(T) \to \text{Kl}(T')$ such that

\[
\begin{array}{c}
\text{Kl}(T) \\
\downarrow V \\
\text{Kl}(T')
\end{array}
\]

This is defined by

- $VA = A$,
- if $k : A \to TB$, then $Vk = A \xrightarrow{k} TB \xrightarrow{\tau_B} T'B$.

and

- $\tau_A = V(TA \xrightarrow{id_{TA}} TA) : TA \to T'A$. 
Monad maps vs. functors between E-M categories

- There is a bijection between monad maps \( \tau : T \rightarrow T' \) and functors \( V : EM(T') \rightarrow EM(T) \) such that

\[
\begin{array}{ccc}
EM(T') & \xrightarrow{V} & EM(T) \\
\downarrow{U'} & & \downarrow{U} \\
C & & \\
\end{array}
\]

(Note the reversed direction.)

- This is defined by
  - \( V(A, a) = (A, a \circ \tau_A) \),
  - if \( h : (A, a) \rightarrow (B, b) \), then
    \( Vh = h : (A, a \circ \tau_A) \rightarrow (B, b \circ \tau_B) \).

  and
  - \( \tau_A = \text{let } (T'A, a) \leftarrow V(T'A, \mu'_A) \text{ in } a \circ T\eta'_A. \)
Examples: Exceptions, reader, writer monads

- Monad maps between the exception monads for sets $E$, $E'$ are in a bijection with pairs of an element of $E' + 1$ and a function between $E$ and $E'$. (Why?)

- Monad maps between the reader monads for sets $S$, $S'$ are in a bijection with maps between $S'$, $S$.

- Monad maps between the writer monads for monoids $(P, o, \oplus)$ and $(P', o', \oplus')$ are in a bijection with homomorphisms between these monoids.
Examples: From exceptions to writer or vice versa

- There is no monad map $\tau$ from the exception monad for a set $E$ and the writer monad for a monoid $(P, o, \oplus)$ (unless $E = 0$). There is not even a natural transformation between the underlying functors: it is impossible to have a map $\tau_0 : 0 + E \rightarrow P \times 0$.

- Monad maps $\tau$ from the writer monad for $(P, o, \oplus)$ to the exception monad for $E$ are in a bijection between monoid homomorphisms between $(P, o, \oplus)$ and the free monoid on the left zero semigroup on $E$. (Can you simplify this condition further?) They can be written as

$$\tau_X = P \times X \rightarrow (E + 1) \times X \rightarrow E \times X + 1 \times X \rightarrow E + X$$
Examples: Reader and state monads

- The monad maps between the state monads for $S$ and $C$ are in a bijection with lenses, i.e., pairs of functions $\text{lkp} : C \rightarrow S$, $\text{upd} : C \times S \rightarrow C$ such that
  - $\text{lkp}(\text{upd}(c, s)) = s$,
  - $\text{upd}(c, \text{lkp} c) = c$,
  - $\text{upd}(\text{upd}(c, s), s') = \text{upd}(c, s')$.

- Can you characterize the monad maps from the reader monad for $S$ to the state monad for $C$? The other way around? (Be careful here!)
Examples: Nonempty lists and powerset

- How many monad maps are there from the nonempty list monad to itself?
  - Answer: 4, viz. the identity map, reverse, take only the first element, take only the last element.
  - Why does taking the 2nd element not qualify? Or taking the two first elements? (These are natural transformations, but...)

- How many monad maps are there from the nonempty list monad to the nonempty powerset monad? The other way around?
Compatible compositions of monads
Compatible compositions of monads

- A compatible composition of two monads \((T_0, \eta_0, \mu_0), (T_1, \eta_1, \mu_1)\) is a monad structure \((\eta, \mu)\) on \(T = T_0 \cdot T_1\) satisfying

\[
\begin{array}{c}
\eta_0 \cdot \eta_1 \\
\downarrow \\
\eta_0 \cdot \eta_1 \cdot \eta_0 \cdot \eta_1 \\
\downarrow \\
T_0 \cdot T_1 \\
\eta \\
\downarrow \\
T_0 \cdot T_1 \\
\end{array}
\]

- Conditions 1-3 say just that \(T_0 \cdot \eta_1\) and \(\eta_0 \cdot T_1\) are monad morphisms between \((T_0, \eta_0, \mu_0)\) resp. \((T_1, \eta_1, \mu_1)\) and \((T, \eta, \mu)\).
  Condition 1 fixes that \(\eta = \eta_0 \cdot \eta_1\); so the only freedom is about \(\mu\).
A distributive law of a monad \((T_1, \eta_1, \mu_1)\) over \((T_0, \eta_0, \mu_0)\) is a natural transformation \(\theta : T_1 \cdot T_0 \to T_0 \cdot T_1\) such that

\[
\begin{align*}
T_1 \cdot T_0 & \xrightarrow{\theta} T_0 \cdot T_1 \\
\eta_0 \cdot T_1 & \xrightarrow{} T_1 \\
T_0 \cdot \eta_0 & \xrightarrow{} T_1 \\
\end{align*}
\]
Compatible compositions = distributive laws

- Compatible compositions of \((T_0, \eta_0, \mu_0), (T_1, \eta_1, \mu_1)\) are in a bijection with distributive laws of \((T_1, \eta_1, \mu_1)\) over \((T_0, \eta_0, \mu_0)\).

- Given \(\mu\), one recovers \(\theta\) by

\[
\theta = T_1 \cdot T_0 \eta_0 \cdot T_1 \cdot T_0 \eta_1 \rightarrow T_0 \cdot T_1 \cdot T_0 \cdot T_1 \mu \rightarrow T_0 \cdot T_1
\]

- Given \(\theta\), \(\mu\) is defined by

\[
\mu = T_0 \cdot T_1 \cdot T_0 \cdot T_1 T_0 \cdot \theta \cdot T_1 T_0 \cdot T_0 \cdot T_1 \cdot T_1 \mu_0 \cdot \mu_1 \rightarrow T_0 \cdot T_1
\]
Algebras of compatible compositions

- Given a distributive law $\theta$, a $\theta$-pair of algebras is given by a set $A$ with a $(T_0, \eta_0, \mu_0)$-algebra structure $(A, a_0)$ and a $(T_1, \eta_1, \mu_1)$-algebra structure $(A, a_1)$ such that

\[ T_1A \xleftarrow{a_1} A \xrightarrow{a_0} T_0A \]

\[ T_1a_0 \downarrow \quad \quad \downarrow T_0a_1 \]

\[ T_1(T_0A) \xrightarrow{\theta_A} T_0(T_1A) \]

- Such pairs of algebras are in a bijection with $(T, \eta, \mu)$-algebras.

- Given $a_0, a_1$, one constructs $a$ as
  - $a = T_0(T_1A) \xrightarrow{T_0a_1} T_0A \xrightarrow{a_0} A$.

- Given $a, a_0$ and $a_1$ are defined by
  - $a_0 = T_0A \xrightarrow{T_0\eta_1} T_0(T_1A) \xrightarrow{a} A$,
  - $a_1 = T_1A \xrightarrow{T_1\eta_0} T_1(T_0A) \xrightarrow{\theta_A} T_0(T_1A) \xrightarrow{a} A$. 
Any monad and an exceptions monad

- The exceptions monad for $E$ distributes in a unique way over any monad $(T_0, \eta_0, \mu_0)$.

  $$\theta : E + T_0A \to T_0(E + A)$$
  $$\theta_A \text{ (inl } e) = \eta_0 \text{ (inl } e),$$
  $$\theta_A \text{ (inr } c) = T_0 \text{ inr}$$

- So we have a unique monad structure on $TA = T_0(E + A)$ that is compatible with $(T_0, \eta_0, \mu_0)$.

- (This generalizes to any coCartesian category, also to any monoidal category with a comonoid.)
Any monad and a writer monad

- There is a unique distributive law of the writer monad for $(P, o, \oplus)$ over any monad $(T_0, \eta_0, \mu_0)$.
  \[ \theta : P \times T_0A \rightarrow T_0(P \times A) \]
  \[ \theta_A (p, c) = T_0(\lambda x. (p, x)) c. \]
  \[ (\theta \text{ is nothing but the unique strength of } T_0!) \]
- So monad structures on $TA = T_0(P \times A)$ compatible with $(T_0, \eta_0, \mu_0)$ are in a bijection with monoid structures on $P$.

- (This generalizes to any Cartesian category and any monoidal category in the form of a bijection between strengths and distributive laws.)
Monoid actions

- A right action of a monoid \((P, o, \oplus)\) on a set \(S\) is a map \(\downarrow : S \times P \to S\) satisfying

\[
\begin{align*}
  s \downarrow o &= s \\
  s \downarrow (p \oplus p') &= (s \downarrow p) \downarrow p'
\end{align*}
\]
Reader and writer monads

- Distributive laws of the writer monad for \((P, o, \oplus)\) over the reader monad for \(S\) are in a bijective correspondence with right actions of \((P, o, \oplus)\) on \(S\).
- The compatible composition of the two monads determined by a right action \(\downarrow\) is

\[
T A = S \Rightarrow P \times A
\]

\[
\eta x = \lambda s. (o, x)
\]

\[
\mu f = \lambda s. \text{let } (p, g) = f s \quad (p', x) = g (s \downarrow p)
\]

in \((p \oplus p', x)\)

—the update monad for \(S\), \((P, o, \oplus), \downarrow\).
State logging

- Take $S$ to be some set (of states).
- Take $P = \text{List } S$, $o = []$, $\oplus = ++$ (state logs).
- $s \downarrow [] = s$
  
  $s \downarrow (s' :: ss) = s' \downarrow ss$

  (so $s \downarrow ss$ is the last element of $(s :: ss)$)
Reading a stack and popping

- Take $S = \text{List } E$ (states of a stack of elements drawn from a set $E$).
- Take $P = \text{Nat}$, $o = 0$, $\oplus = +$ (possible numbers of elements to pop).
- $es \downarrow n = \text{removelast } n \ es$. 
Reading a stack and pushing

- Take again $S = \text{List } E$ (states of a stack of elements drawn from a set $E$).
- Take $P = \text{List } E$, $o = []$, $\oplus = ++$ (lists of elements to push on the stack).
- $es \downarrow es' = es ++ es'$.
- (So here we choose $(S, \downarrow)$ to be the initial $(P, o, \oplus)$-set—which is always a possibility.)
Matching pairs of monoid actions

A matching pair of actions of two monoids \((P_0, o_0, \oplus_0)\) and \((P_1, o_1, \oplus_1)\) on each other is pair of maps \(\downarrow : P_1 \times P_0 \rightarrow P_0\) and \(\uparrow : P_1 \times P_0 \rightarrow P_1\) such that

\[
\begin{align*}
o_1 \downarrow p_0 &= p_0 \\
(p_1 \oplus_1 p'_1) \downarrow p_0 &= p_1 \downarrow (p'_1 \downarrow p_0) \\
p_1 \downarrow o_0 &= o_0 \\
p_1 \downarrow (p_0 \oplus_0 p'_0) &= (p_1 \downarrow p_0) \oplus_0 ((p_1 \uparrow p_0) \downarrow p'_0)
\end{align*}
\]

\[
\begin{align*}
p_1 \uparrow o_0 &= p_1 \\
p_1 \uparrow (p_0 \oplus_0 p'_0) &= (p_1 \uparrow p_0) \uparrow p'_0 \\
o_1 \uparrow p_0 &= o_1 \\
(p_1 \oplus_1 p'_1) \uparrow p_0 &= (p_1 \uparrow (p'_1 \downarrow p_0)) \oplus_1 (p'_1 \uparrow p_0)
\end{align*}
\]
Zappa-Szép product of monoids

- A Zappa-Szép product (aka knit product, bicrossed product, bilateral semidirect product) of two monoids \((P_0, o_0, \oplus_0)\) and \((P_1, o_1, \oplus_1)\) is a monoid structure \((o, \oplus)\) on \(P = P_0 \times P_1\) such that

\[
\begin{align*}
o &= (o_0, o_1) \\
(p, o_1) \oplus (p', o_1) &= (p \oplus_0 p', o_1) \\
(o_0, p) \oplus (o_0, p') &= (o_0, p \oplus_1 p') \\
(p, o_1) \oplus (o_0, p') &= (p, p')
\end{align*}
\]

- Zappa-Szép products of \((P_0, o_0, \oplus_0)\) and \((P_1, o_1, \oplus_1)\) are in a bijective correspondence with matching pairs of actions of \((P_0, o_0, \oplus_0)\) and \((P_1, o_1, \oplus_1)\).

- Given \(\oplus\), one constructs \(\downarrow\) and \(\uparrow\) by

  - \((p_1 \downarrow p_0, p_1 \uparrow p_0) = (o_0, p_1) \oplus (p_0, o_1)\)

- Given \(\downarrow\) and \(\uparrow\), \(\oplus\) is defined by

  - \((p_0, p_1) \oplus (p'_0, p'_1) = (p_0 \oplus_0 (p_1 \uparrow p'_0), (p_1 \downarrow p'_0) \oplus_1 p'_1)\)
Two writer monads

- Compatible compositions of writer monads for $(P_0, o_0, \oplus_0)$ and $(P_1, o_1, \oplus_1)$ are in a bijection with matching pairs of actions of the two monoids.
- They are isomorphic to writer monads for the corresponding Zappa-Szép products.
Combining popping and pushing

- Take \((P_0, o_0, \oplus_0) = (\text{Nat}, 0, +),\) 
  \((P_1, o_1, \oplus_1) = (\text{List } E, [], ++)\) where \(E\) is some set.

- \(es \downarrow n = n - \text{length } es,\)
  \(es \uparrow n = \text{removelast } n \ es.\)

- \((n, es) \oplus (n', es') = (n + (n' - \text{length } es'), (\text{removelast } n' \ es) ++ es')\)

- Pairs \((n, es)\) represent net effects of sequences of pop, push instructions on a stack: some number of elements is removed from and some new specific elements are added to the stack.
Combining reading, popping, pushing

How do I now show that

\[ TA = \text{List } E \Rightarrow \text{Nat} \times (\text{List } E \times A) \]

is a monad?

This is of the form \( T_0 \cdot T_1 \cdot T_2 \) where

\[ T_0A = \text{List } E \Rightarrow A \]
\[ T_1A = \text{Nat} \times A \]
\[ T_2A = \text{List } E \times A \]

We already know that

\[ T_{01} = T_0 \cdot T_1 \]
\[ T_{02} = T_0 \cdot T_2 \]
\[ T_{12} = T_1 \cdot T_2 \]

are compatible compositions of monads.

We want to be sure that \((T_0 \cdot T_1) \cdot T_2\) and \(T_0 \cdot (T_1 \cdot T_2)\) are compatible compositions of monads.

Moreover they’d better be the same monad!
In terms of distributive laws, this only requires checking the Yang-Baxter equation: