Bird Meertens Formalism

$cdot\cdot$ = function composition; $K\,a\,b = a$ constant func; $[\cdot] : \alpha \to [\alpha], a \mapsto [a]$, $[]$ empty list; $([\alpha], +\!\!+, [])$ is a free monoid; $[\alpha]^+$ non-empty list. Homomorphism: between monoids, $h\,[] = id_\oplus, h\,[a] = f\,a, h(x +\!\!+ y) = h\,x \oplus h\,y$ (since it is free), h is uniquely determined by f and $\oplus$.
Map: $f * [a_1, \cdots, a_n] = [fa_1, \cdots, fa_n]$, $f* \in \mathrm{Hom}$. Reduce: $\otimes/[a_1, \cdots, a_n] = a_1 \otimes \cdots \otimes a_n$, $\otimes/ \in \mathrm{Hom}$. Any hom h can be written as $h = \otimes/ \cdot f*$.
Map distrib: $(f \cdot g)* = f * \cdot g*$; Map promotion: $f * \cdot +\!\!+ / = +\!\!+/ \cdot f * *$; Reduce promotion: $\otimes/ \cdot +\!\!+/ = \otimes/ \cdot (\otimes/)*$.
foldl: $\otimes \not\rightarrow_e [a_1, \cdots, a_n] = ((e \otimes a_1) \otimes \cdots) \otimes a_n$ and foldr: $\otimes \not\leftarrow_e [a_1, \cdots, a_n] = a_1 \otimes (\cdots \otimes (a_n \otimes e))$.
(Accumulation) scanl: $\oplus \mathbin{/\!\!\!\not\rightarrow}_e [a_1, \cdots, a_n] = [e, e \oplus a_1, \cdots, (e \oplus a_1) \oplus \cdots \oplus a_n]$ and scanr:
$\oplus \mathbin{/\!\!\!\not\leftarrow}_e [a_1, \cdots, a_n] = [a_1 \oplus \cdots \oplus (a_n \oplus e), \cdots, a_n \oplus e, e]$.
$\mathrm{inits} = (+\!\!+ \mathbin{/\!\!\!\not\rightarrow}_{[]}) \cdot ([\cdot])*$, $\mathrm{tails} = (+\!\!+ \mathbin{/\!\!\!\not\leftarrow}_{[]}) \cdot [\cdot]*$, $\mathrm{segs} = +\!\!+/ \cdot \mathrm{tails} * \cdot \mathrm{inits}$.
Accumulation lemma: $\oplus \mathbin{/\!\!\!\not\rightarrow} = (\oplus \not\rightarrow_e) * \cdot \mathrm{inits}$ (and $\oplus \mathbin{/\!\!\!\not\rightarrow} = (\oplus \not\rightarrow) * \cdot \mathrm{init}^+$).
Horner's Rule: If $\mathrm{distrib} : (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$ then $\oplus/ \cdot \otimes/ * \cdot \mathrm{tails} = \odot \not\rightarrow_e$ where $e = id_\otimes, a \odot b = (a \otimes b) \oplus e$.
Generalize: $\otimes/ \cdot (\otimes/ \cdot f*) * \cdot \mathrm{tails} = \odot \not\rightarrow_e$ where $e = id_\otimes, a \odot b = (a \otimes f\,b) \oplus e$.
MSS: $\mathrm{mss} = \uparrow / \cdot +/ * \cdot \mathrm{segs} = \uparrow / \cdot +/ * \cdot +\!\!+ / \cdot \mathrm{tails} * \cdot \mathrm{inits}$ (map promotion) $= \uparrow / \cdot +\!\!+/ \cdot +/ * * \cdot \mathrm{tails} * \cdot \mathrm{inits}$ (reduce promotion) $= \uparrow / \cdot \uparrow / * \cdot + / * * \cdot \mathrm{tails} * \cdot \mathrm{inits}$ (map distrib) $= \uparrow / \cdot (\uparrow / \cdot +/ * \cdot \mathrm{tails}) * \cdot \mathrm{inits}$ (Horner's rule $a \odot b = (a + b) \uparrow 0$) $= \uparrow / \cdot \odot \not\rightarrow_0 * \cdot \mathrm{inits}$ (accumulation lemma) $= \uparrow / \cdot \odot \mathbin{/\!\!\!\not\rightarrow}_0$.

Homomorphisms

$h \in \mathrm{Hom}((\alpha, \oplus, id_\oplus), (\beta, \otimes, id_\otimes))$ iff $h \cdot \oplus/ = \otimes/ \cdot h*$.
$\mathrm{all}\,p = \wedge/ \cdot p*, \mathrm{some}\,p = \vee/ \cdot p*$.
"All applied to" $^\circ$ is $[f, \cdots, h]^\circ a = [fa, \cdots, ha]$ and $[]^\circ a = []$.
$h\,x = \mathbf{if}\,p\,x\,\mathbf{then}\,f\,x\,\mathbf{else}\,g\,x$ is written as $h = (p \to f, g)$, then $h \cdot (p \to f, g) = (p \to h \cdot f, h \cdot g)$,
$(p \to f, g) \cdot h = (p \cdot h \to f \cdot h, g \cdot h)$, $(p \to f, f) = f$.
Filter: $p \triangleleft = +\!\!+/ \cdot (p \to [id]^\circ, []^\circ)*$ taking all that satisfy p, then there's filter promotion: $(p\triangleleft) \cdot +\!\!+/ = +\!\!+/ \cdot (p\triangleleft)*$ and map-filter swap: $(p\triangleleft) \cdot f* = f * \cdot (p \cdot f)\triangleleft$.
Cross product: $[a, b] X_\oplus [c, d, e] = [a \oplus c, b \oplus c, a \oplus d, b \oplus d, a \oplus e, b \oplus e]$ or $x X_\oplus [] = []$, $x X_\oplus [a] = (\oplus a) * x$,
$x X_\oplus (y +\!\!+ z) = (x X_\oplus y) +\!\!+ x X_\oplus z$. Then $[]$ is the zero of $X_{+\!\!+}$ and we have cross promotion: $f * * \cdot X_{+\!\!+}/ = X_{+\!\!+}/ \cdot f * **$,
$(\oplus/) * \cdot X_{+\!\!+}/ = X_\oplus/ \cdot (\oplus/) * *$.
Then $(\mathrm{all}\,p)\triangleleft = +\!\!+/ \cdot (X_{+\!\!+}/ \cdot (p \to [[id]^\circ]^\circ, []^\circ)*)*$, since $[]$ is the zero of $X_{+\!\!+}$.
cp: take list of list and return list of lists, one from each component. $cp\,[[a, b], [c], [d, e]] = [[a, c, d], [b, c, d], [a, c, e], [b, c, e]]$, then $cp = X_{+\!\!+}/ \cdot ([id]^\circ*)*$; subs: return all subsequences, $subs = X_{+\!\!+}/ \cdot [[id]^\circ, []^\circ]*$, or $subs = +\!\!+/ * \cdot +\!\!+ / * * \cdot cp \cdot [[id]^\circ, []^\circ]*$.
Then $\uparrow_\# / \cdot (\mathrm{all}\,p)\triangleleft \in \mathrm{Hom}$ since $= \uparrow_\# / \cdot +\!\!+/ \cdot (X_{+\!\!+}/ \cdot (p \to [[id]^\circ]^\circ, []^\circ)*)*$ by reduce promotion
$= \uparrow_\# / \cdot (\uparrow_\# / \cdot X_{+\!\!+}/ \cdot (p \to [[id]^\circ]^\circ, []^\circ)*)*$. By $\uparrow_\#, +\!\!+$ distribution $= \uparrow_\# / \cdot (+\!\!+/ \cdot \uparrow_\# / * \cdot (p \to [[id]^\circ]^\circ, []^\circ)*)*$, and
$\uparrow_\# / \cdot (p \to [[id]^\circ]^\circ, []^\circ) = (p \to [id]^\circ, K_\omega)$ where $\omega = \uparrow_\# /[]$ is the zero of $+\!\!+$.
Longest Segment: $lsp = \uparrow_\# / \cdot (\mathrm{all}\,p) \triangleleft \cdot \mathrm{segs}$. Then by segment decomposition,
$= \uparrow_\# / \cdot (\uparrow_\# / \cdot (\mathrm{all}\,p) \triangleleft \cdot \mathrm{tails}) * \cdot \mathrm{inits} = \uparrow_\# / \cdot (\uparrow_\# / \cdot (+\!\!+/ \cdot (p \to [id]^\circ, K_\omega)*) * \cdot \mathrm{tails}) * \cdot \mathrm{inits}$ by general Horner's rule,
$= \uparrow_\# / \cdot \odot \not\rightarrow_{[]} * \cdot \mathrm{inits}$ where $x \odot a = (x +\!\!+ (p\,a \to [a], \omega)) \uparrow_\# []$ and by accumulation lemma, $= \uparrow_\# / \cdot \odot \mathbin{/\!\!\!\not\rightarrow}_{[]}$ which is linear in calculations of p.
Homomorphism Theorem: $h \in \mathrm{Hom}$ iff $h\,v = h\,x \wedge h\,w = h\,y \implies h(v +\!\!+ w) = h(x +\!\!+ y)$ for all lists v, w, x, y.

Fusion and Tupling and Unfold

Fusion lemma: $f(a \oplus r) = a \otimes f\,r \implies f \cdot \mathrm{foldr}\,(\oplus) e = \mathrm{foldr}\,(\otimes)(f\,e)$.
For $\max = \mathrm{head} \cdot \mathrm{foldr}\,\mathrm{insert}[]$ where $\mathrm{head}(\mathrm{insert}\,a\,r) = a \uparrow \mathrm{head}\,r$, we obtain $\max = \mathrm{foldr} \uparrow (-\infty)$.
Tupling: Some functions, like $\mathrm{average}, \mathrm{tailsum}$, can't be written as foldr/foldl. We can tuple them with folds like $\mathrm{sum}, \mathrm{length}$, and make the tupling a foldl/r. e.g.

```
average xs = let (s,l) = tup xs in s / l
    where tup = foldr (\a (s,l) -> (a+s,l+1)) (0,0)
```

Functions are said to be form a mutumorphism if $f_i, 1 \leqslant i \leqslant n$ can be written as $f_i(a : x) = a \oplus_i (f_1 x, \cdots, f_n x)$ where $\oplus_i$ are binary operators. Then $f = (f_1, \cdots, f_n)$ can be written as foldr.
Unfold: a pattern to generate lists, $\mathrm{unfold}^\infty = \mathrm{unfold}(\mathrm{const\ False})$.

```
unfold :: (b->Bool) -> (b->a) -> (b->b) -> b -> [a]
unfold p f g x = if p x then [] else f x : unfold p f g $ g x
```

$\mathrm{mults}\,n = [0, n, 2n, \cdots]$ isn't one unfold, since $\mathrm{tail} \cdot \mathrm{unfold}^\infty\,f\,g = \mathrm{unfold}^\infty\,f\,g \cdot g$.
A **Hylomorphism** is fold after unfold: $\mathrm{hylo}(\oplus, e)(p, f, g) = \mathrm{foldr} \oplus e\,(\mathrm{unfold}\,p\,f\,g)$.
A **Metamorphism** is unfold after fold:

```
reformat :: Int → [[Char]] → [[Char]]
reformat n = unfold (==[]) (take n) (drop n) . concat
```

Parallelization: If $ff^\circ = id$, then $f(x + y) = fx \odot fy$ where $a \odot b = f(f^\circ a + f^\circ b)$. Find a simple weak inverse e.g. $\mathrm{sum}^\circ a = [a]$, $f = \mathrm{mps} \otimes \mathrm{sum}$, $f^\circ(p, s) = [p, s - p]$.

Agda Semantics

```
∘: func composition; ≡: identity; ∷ list ctor (x ∷ xs); [_]: construct singleton list; ++: list concat; for
_⊕_, (_⊕ y) = λ x → x ⊕ y, (x ⊕_) = λ y → x ⊕ y
-- Proofs are functions:
+0 : (x : ℕ) → x + 0 ≡ x
+0 0 = refl; +0 (suc x) rewrite +0 x = refl -- rewrite p1 | p2: simplify by p1 & p2
||≡ff₂ : ∀ {b1 b2} → b1 || b2 ≡ ff → b2 ≡ ff
||≡ff₂ {tt} () -- absurd pattern (), if the condition is impossible
-- Equality Theory: begin, _≡⟨_⟩_, _≡⟨⟩_, ∎ (qed)
foldr-map-fusion : ∀ {A B C : Set} → (f : A → B) → (_⊕_ : B → C → C) → (e : C)
  → foldr _⊕_ e ∘ map f ≗ foldr (λ a r → f a ⊕ r) e
foldr-map-fusion {A} {B} {C} f _⊕_ e xs =
    begin -- begins proof
        foldr _⊕_ e (map f xs) -- LHS of theorem
    ≡⟨ cong h (map-is-foldr f xs) ⟩ -- _≡⟨ p ⟩_: same as rewrite p1
        h (foldr (λ x y → (f x) ∷ y) [] xs) -- cong f p: x = y ⇒ f x = f y
    ≡⟨ foldr-fusion h {F} {g} [] fuse xs ⟩ -- rewrite p1 | p2
        foldr g (h []) xs
    ≡⟨⟩ -- _≡⟨⟩_ means trivial reason, same as ≡⟨ refl ⟩
        foldr (λ a r → f a ⊕ r) e xs -- RHS of theorem
    ∎ -- QED, end of proof
        where -- some helper functions
-- if written in rewrite and refl, it is:
foldr-map-fusion f _⊕_ e xs rewrite (cong h (map-is-foldr f xs)) = foldr-fusion h {F} {g} [] fuse xs
cong : {A B : Set} (f : A → B) {x y : A} → x ≡ y → f x ≡ f y -- cong f prop
cong-app : {A B : Set} {f g : A → B} → f ≡ g → (x : A) → f x ≡ g x -- cong-app prop x
sym : {A : Set} {x y : A} → x ≡ y → y ≡ x -- sym prop
trans : {A : Set} {x y z : A} → x ≡ y → y ≡ z → x ≡ z -- trans prop1 prop2
subst : {A : Set} {x y : A} (P : A → Set) → x ≡ y → P x → P y -- subst P prop
-- Important functions:
(suc n) + m = suc (n + m) -- + and * all iterate on the left parameter
suc m * n = n + (m * n)
_++_ (x ∷ xs) ys = x ∷ (xs ++ ys)
map f (x ∷ xs) = f x ∷ map f xs
foldr : ∀ {A B : Set} → (A → B → B) → B → List A → B
foldr _⊗_ e [] = e
foldr _⊗_ e (x ∷ xs) = x ⊗ foldr _⊗_ e xs
foldl : ∀ {A B : Set} → (B → A → B) → B → List A → B
foldl _⊗_ e [] = e
foldl _⊗_ e (x ∷ xs) = foldl _⊗_ (e ⊗ x) xs
scanl _⊗_ e (x ∷ xs) = e ∷ scanl _⊗_ (e ⊗ x) xs
scanr _⊗_ e (x ∷ xs) with scanr _⊗_ (e ⊗ x) xs
... | y ∷ ys = (x ⊗ y) ∷ (y ∷ ys) -- use of “with”: pattern matching
... | [] = [] -- non-executable branch
extensionality : ∀ {A B : Set} {f g : A → B} → (∀ (x : A) → f x ≡ g x) → f ≡ g
reduce : ∀{A : Set} → (_⊕_ : A → A → A) → (e : A) → IsMonoid _⊕_ e → List A ⟶ A
reduce _⊕_ e _ [] = e
reduce _⊕_ e p (x ∷ xs) = x ⊕ reduce _⊕_ e p xs
-- Data Structures: definition and usage
record IsSemigroup {A : Set} (_⊕_ : A → A → A) : Set where
    field assoc : ∀ x y z → (x ⊕ y) ⊕ z ≡ x ⊕ (y ⊕ z)
proof-of-theorem _⊕_ p-is-semigroup x y = ⊕-assoc ...
    where -- for p-is-semigroup : IsSemigroup _⊗_, use this to apply assoc
        open IsSemigroup p-is-semigroup renaming (assoc to ⊕-assoc)
-- Miscellaneous
max = λ x y → if x ≤A y then y else x -- lambda calculus
foo x y with (predicate x) in prop
foo x y | true = -- puts the proof in prop : predicate x ≡ true
... | true = -- or one can simply write ...
```

Remember operators must be separated with spaces.