Relation Algebra

§1. Notation

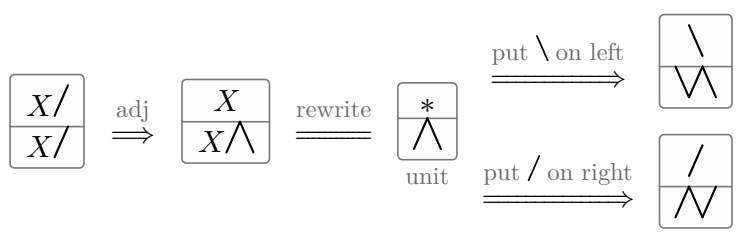
Traditionaly Adjunction F -| G is defined as $F(x) \leq y$ iff $x \leq G(y)$, and you derive laws like this:

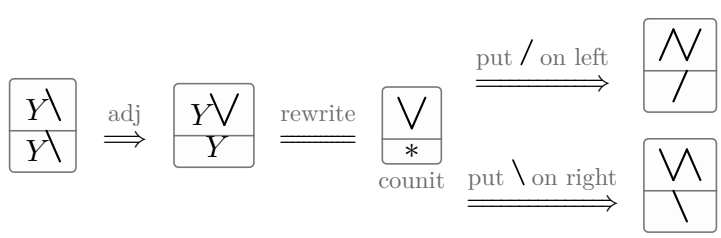
$$\begin{aligned} F(x) \leq F(x) \quad &\Longrightarrow \quad x \leq G(F(x)) \quad \Longrightarrow \quad F(x) \leq F(G(F(x))) \\ G(y) \leq G(y) \quad &\Longrightarrow \quad F(G(y)) \leq y \quad \Longrightarrow \quad G(F(G(y))) \leq G(y) \end{aligned}$$

By define $X : * \mapsto x$, $Y : * \mapsto y$ and use diagram order, application can be replaced by composition: $XF \leq Y$ iff $X \leq YG$

- write F as $/$, G as $\backslash$ (1a)
- render $a \leq b$ as $\begin{array}{|c|}\hline a \\ \hline b \\ \hline\end{array}$
- **rewrite** $X \leq XF$ as $1 \leq F$

$$\begin{array}{|c|}\hline X/ \\ \hline Y \\ \hline\end{array} = \begin{array}{|c|}\hline X \\ \hline Y\backslash \\ \hline\end{array} \qquad \text{adj}$$





now we have $/\!\!\backslash\!\!/ = /$ and $\backslash\!\!/\!\!\backslash = \backslash$, and you just discovered string diagrams.

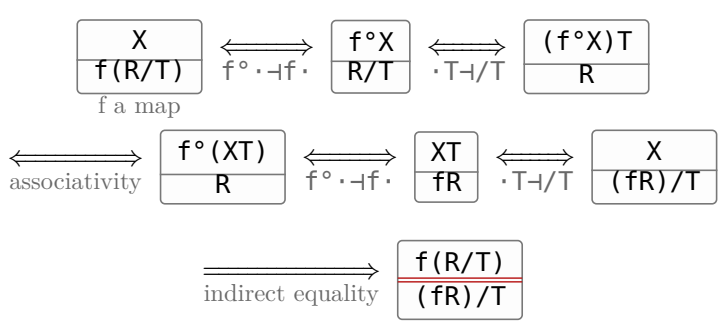
§1.1. Adjunctions

(1.1a)

F⊣G	F's type	monad FG	𝟙⊑FG	GF⊑𝟙	F preserves ∪	G preserves ∩	FGF=F	GFG=G
◁⊣▷	A→ A⊗A	◁▷=𝟙	𝟙⊑◁▷	▷◁⊑𝟙	(R ∪ S)◁= R◁ ∪ S◁	(R∩S)▷= R▷∩S▷	◁▷◁=◁	▷◁▷=▷
⊸⊣⊸	A→𝕀	⊸⊸=T	𝟙⊑⊸⊸	⊸⊸⊑𝟙	(R ∪ S)⊸= R⊸ ∪ S⊸	(R∩S)⊸= R⊸∩S⊸	⊸⊸⊸=⊸	⊸⊸⊸=⊸
°⊣°	(A→B)→ (B→A)	°°=𝟙	R=R°°	R=R°°	(R ∪ S)°= R° ∪ S°	(R∩S)°= R°∩S°	R°°°=R°	R°°°=R°
⊸◁⊣▷⊸	(X⊗A→Y)→ (X→Y⊗A)	𝟙	(⊸◁⊗𝟙) (𝟙⊗▷⊸)=𝟙	(𝟙⊗⊸◁) (▷⊸⊗𝟙)=𝟙	=	=	=	=
▷⊸⊣⊸◁	(X→Y⊗A)→ (X⊗A→Y)	𝟙	(𝟙⊗⊸◁) (▷⊸⊗𝟙)=𝟙	(⊸◁⊗𝟙) (𝟙⊗▷⊸)=𝟙	=	=	=	=
Δ⊣∩	(A→B)→ (A→B)²	R↦R∩R=R	R⊑R∩R	R∩S⊑R	Δ(R ∪ S)= ΔR ∪ ΔS	(R∩T)∩(S∩U)= (R∩S)∩(T∩U)	R∩R=R	R∩R=R
∪⊣Δ	(A→B)²→ (A→B)	(R,S)↦ (R ∪ S,R ∪ S)	R⊑R ∪ S	R ∪ R⊑R	(R ∪ T) ∪ (S ∪ U)= (R ∪ S) ∪ (T ∪ U)	Δ(R∩S)= ΔR∩ΔS	R ∪ R=R	R ∪ R=R
⊥⊣!	{*}→ (A→B)			⊥⊑R				
·S⊣/S	(A→B)→ (A→C)	S/S	R⊑(RS)/S	(T/S)S⊑T	(R ∪ T)S= RS ∪ TS	(R∩T)/S= R/S∩T/S	((RS)/S)S =RS	((T/S)S)/S =T/S
S·⊣S\	(B→C)→ (A→C)	S\S	R⊑S\(SR)	S(S\T)⊑T	S(R ∪ T)= SR ∪ ST	S\(R∩T)= S\R∩S\T	S(S\(SR)) =SR	S\(S(S\T)) =S\T
R∩⊣R⇒	(A→B)→ (A→B)	X↦R⇒(X∩R)	X⊑R⇒(X∩R)	R∩(R⇒Y)⊑Y	R∩(X ∪ Y)= (R∩X) ∪ (R∩Y)	R⇒(X∩Y)= (R⇒X)∩(R⇒Y)	R∩(R⇒(X∩R)) =X∩R	R⇒(R∩(R⇒Y)) =R⇒Y
𝒟⊣·T	(A→B)→ Cor A	R↦(𝒟R)T	R⊑(𝒟R)T	𝒟(AT)⊑A	𝒟(R ∪ S)= 𝒟R ∪ 𝒟S	(A∩B)T= AT∩BT	𝒟((𝒟R)T) =𝒟R	(𝒟(AT))T =AT
ℛ⊣T·	(A→B)→ Cor B	R↦T(ℛR)	R⊑T(ℛR)	ℛ(TA)⊑A	ℛ(R ∪ S)= ℛR ∪ ℛS	T(A∩B)= TA∩TB	ℛ(T(ℛR)) =ℛR	T(ℛ(TA)) =TA
·f⊣·f°	(A→B)→ (A→C)	ff°	𝟙⊑ff°	f°f⊑𝟙	(R ∪ S)f= Rf ∪ Sf	(R∩S)f°= Rf°∩Sf°	ff°f=f	f°ff°=f°
f°·⊣f·	(A→C)→ (B→C)	ff°	𝟙⊑ff°	f°f⊑𝟙	f°(X ∪ Y)= f°X ∪ f°Y	f(X∩Y)= fX∩fY	ff°f=f	f°ff°=f°
i⊣E	Map↪Rel	E	$\frac{1}{∋}$:A→EA	∋:EB→B	—	—	$\frac{∋}{∋}$=𝟙	$\frac{R}{∋}$∋=R
·∋⊣ $\overline{∋}$	Map(A,EB)→ (A→B)	𝟙	$\frac{f∋}{∋}$=f	$\frac{R}{∋}$∋=R	—	—	$\frac{f∋}{∋}$∋ =f∋	$\frac{\frac{R}{∋}∋}{∋}$ =$\frac{R}{∋}$
$\overline{∋}$ ⊣·∋	(A→B)→ Map(A,EB)	𝟙	$\frac{R}{∋}$∋=R	$\frac{f∋}{∋}$=f	—	—	$\frac{\frac{R}{∋}∋}{∋}$ =$\frac{R}{∋}$	$\frac{f∋}{∋}$∋ =f∋

§1.2. Composing adjunctions

(1.2a)



(1.2b)

	·T	T·	·g	g°·	T∩	°	⊶◁	Δ
·S	R/(ST)= (R/T)/S	T\(R/S)= (T\R)/S	R/(Sg)= (Rg°)/S	g(R/S)= (gR)/S	—	(Y/S)°= S°\(Y°)	(RS)^= R^(S⊗𝟙)	(T₁∩T₂)/S= T₁/S∩T₂/S
S·	S\(R/T)= (S\R)/T	(TS)\R= S\(T\R)	S\(Rg°)= (S\R)g°	(g°S)\R= S\(gR)	—	(S\Y)°= Y°/S°	((S⊗𝟙)R)^= SR^	S\(T₁∩T₂)= S\T₁∩S\T₂
·f	R/(fT)= (R/T)f°	T\(Rf°)= (T\R)f°	(fg)°=g°f°			(fY)°=Y°f°		(T₁∩T₂)f°= T₁f°∩T₂f°
f°·	f(R/T)= (fR)/T	(Tf°)\R= f(T\R)	—	(fg)°=g°f°		(fY)°=Y°f°		f(T₁∩T₂)= fT₁∩fT₂
R∩	—	—	—	—	(R∩T)⇒Y= R⇒(T⇒Y)	(R⇒Y)°= R°⇒(Y°)	—	R⇒(T₁∩T₂)= (R⇒T₁)∩(R⇒T₂)
°	(Y/T)°= T°\(Y°)	(T\Y)°= Y°/T°	(gY)°=Y°g°	(gY)°=Y°g°	(T⇒Y)°= T°⇒(Y°)	Y°°=Y	(R^)°= (R°⊗𝟙) (𝟙⊗▷⊸) both bends: the ° section's display	(T₁∩T₂)°= T₁°∩T₂°
⊶◁	(RT)^= R^(T⊗𝟙)	((T⊗𝟙)R)^= TR^	—	—	—	(R^)°= (R°⊗𝟙) (𝟙⊗▷⊸) both bends: the ° section's display	—	—
∪	(X₁ ∪ X₂)T= X₁T ∪ X₂T	T(X₁ ∪ X₂)= TX₁ ∪ TX₂	(X₁ ∪ X₂)g= X₁g ∪ X₂g	g°(X₁ ∪ X₂)= g°X₁ ∪ g°X₂	T∩(X₁ ∪ X₂)= (T∩X₁) ∪ (T∩X₂)	(X₁ ∪ X₂)°= X₁° ∪ X₂°		
⊥	⊥T=⊥	T⊥=⊥	⊥g=⊥	g°⊥=⊥	T∩⊥=⊥	⊥°=⊥		

§1.3. Adjoint triples

Beyond ∪⊣Δ⊣∩, which §1.1 already carries as two rows, the table holds one adjoint triple with content: ∃_f⊣f*⊣∀_f along a map f, once on each side of the composite. f : A→B throughout this subsection.

(1.3a)

$$\cdot f \dashv \cdot f^\circ \dashv /f^\circ$$
$$f^\circ\cdot \dashv f\cdot \dashv f\backslash$$

The first two links of each chain are the map shunting rules, and the third is §1.1′s division row read at a chosen divisor: ·S⊣/S at S:=f°, and S·⊣S\ at S:=f.

The third link is not the first over again. For a map f, /f collapses to ·f°; being a map is exactly what that collapse spends R/(fS)=(R/S)f°. — f° is not a map, so /f° does not collapse in turn, and the three operators stay distinct. On Rel(𝕀,A)=EA they are the image triple:

(1.3b)

operator	acts by	name
·f	S↦f[S]	direct image, ∃_f
·f°	T↦f⁻¹[T]	inverse image, f*
/f°	S↦{b : f⁻¹(b)⊆S}	∀_f

§1.1′s 𝒟⊣·⊤ is this same chain along the projection A⊗B→A, read through Rel(A,B)=E(A⊗B): 𝒟R={(a,a) : ∃b. aRb}, A⊤ is the relation that ignores b altogether, and the third link is R↦𝟙∩R/⊤={(a,a) : ∀b. aRb}.

(1.3c)

$$\mathcal{D}\dashv\cdot\top\dashv\mathbb{1}\cap\cdot/\top$$

Three is where it stops, and one map breaks both ends. Take A={a₁,a₂} and B={b}: f[{a₁}]∩f[{a₂}]={b} while f[{a₁}∩{a₂}]=∅, so ·f does not preserve meets and has no left adjoint, and the same two subsets give ∀_f({a₁} ∪ {a₂})={b} against ∀_f{a₁} ∪ ∀_f{a₂}=∅, so /f° does not preserve joins and has no right adjoint. A monic f restores the binary case and no more — ⊤f still reaches only im f, and ∀_f⊥=B∖im f is still not ⊥. The chain extends only when f° is a map as well, and then /f°=·f°°=·f and it repeats forever. For an S with neither S nor S° a map there is no triple at all: ·S⊣/S is two links and stops.

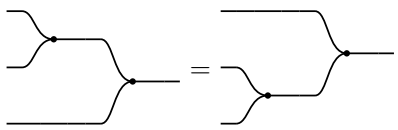
The rows that chain forever say nothing by it: ° is an order-isomorphism of hom-posets, so it is adjoint to itself on both sides and °⊣°⊣° has no end, and §1.1′s other row of that kind, ⊶◁⊣▷⊸, goes the same way.

§2. Relations

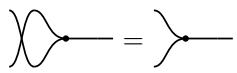
Rel is a poset-enriched category with $(\otimes, {}^\circ)$ where $\otimes$ is commutative cartisian product, and converse $\circ\dashv\circ$, and (2a)

C: $(\triangleright : A\otimes A\to A, \multimap : \mathbb{I}\to A)$ a commutative monoid with $C^\circ\dashv C$. We write C° as $(\triangleleft : A\to A\otimes A, \multimapinv : A\to\mathbb{I})$

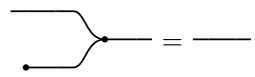
(2b)



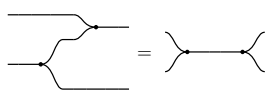
$\triangleright$ associative



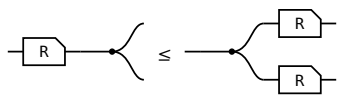
$\triangleright$ commutative



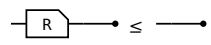
$\multimapinv$ is its unit



Frobenius, one half — the other is its $^\circ$

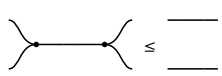


$R\triangleleft\leq\triangleleft(R\otimes R)$

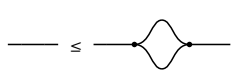


$R\multimapinv\leq\multimapinv$

§2.1. ∀ object A, $(A,\triangleleft,\multimapinv)\dashv(A,\triangleright,\multimapinv)$



$\triangleright\triangleleft\leq\mathbb{1}$



$\mathbb{1}\leq\triangleleft\triangleright$

$\multimapinv\multimap\leq\mathbb{1}$

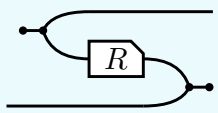
$\mathbb{1}\leq\multimap\multimapinv$ (2.1a)

The last of these is the only one that makes a picture **bigger**, and it is worth a name: **a wire is below the cut wire**. In `Rel` it reads `{(a,a)}⊆A×A` — cut a wire and its two ends stop having to agree, so cutting can only add pairs. It is the one weakening this calculus gives away for free.

§3. $\circ : \mathcal{C}^{\mathsf{op}} \longrightarrow \mathcal{C}$ is a 2 functor

(3a)

$\circ$ is primitive, part of the data the first section lists. It turns both of R's wires round, and the Frobenius structure DRAWS that — the picture and the formula below are R°, not its definition:



$$R^\circ = (\multimap\!\triangleleft \otimes \mathbb{1})(\mathbb{1} \otimes R \otimes \mathbb{1})(\mathbb{1} \otimes \triangleright\!\multimap)$$

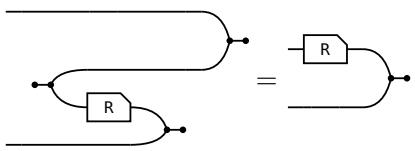
where $\multimap\!\triangleleft : \mathbb{I} \to a \otimes a$ opens a pair of wires out of nothing and $\triangleright\!\multimap : b \otimes b \to \mathbb{I}$ closes one, so the input of R° is where the output of R was. Taking that formula as the definition would now be circular: $\triangleleft$ and $\multimap$ are $\triangleright^\circ$ and $\multimap^\circ$. The laws of $\circ$ are that it is a contravariant 2-functor $\circ : \mathcal{C}^{op} \to \mathcal{C}$:

(i) $\mathbb{1}^\circ = \mathbb{1}$ (ii) $(RS)^\circ = S^\circ R^\circ$ (iii) $(R \otimes S)^\circ = R^\circ \otimes S^\circ$ (iv) $R \leq S$ implies $R^\circ \leq S^\circ$

§3.1. The slide

The one rule (ii) and (iii) use, and each of them uses it twice. A converse facing a merge on the lower strand is the box itself, upright, on the upper one:

(3.1a)

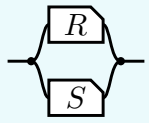


R° is DEFINED as the bending of $(R \otimes \mathbb{1}) \triangleright\!\multimap$, so the slide claims only that unbending it gives that back — and unbending undoes bending for every arrow. That is the snake above with a passenger: the $\multimap\!\triangleleft$ bends the a strand down and the $\triangleright\!\multimap$ brings it back up, while the b strand rides through untouched. Nothing else is spent below.

§4. ∩ is a commutative idempotent monoid on every hom-set

(4a)

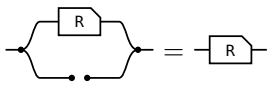
The **meet** of R,S : a→b, the paper's **convolution**, is R∩S:=◁(R⊗S)▷ — copy the input, run R and S on the two copies, merge the results — so what comes out is what both of them do.



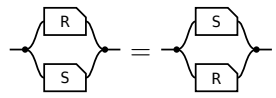
transcribed: a definition has no statement to export

On every hom-set it is associative, commutative and idempotent , , with unit the maximal arrow T=⊸∘⊸ (the paper's Lemma 4.11).

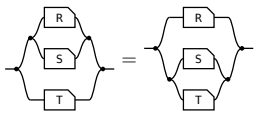
(4b)



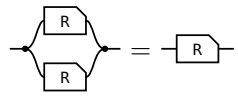
unit: one half of T per end — the merge's unit law absorbs the ⊸, the copy's counit law the ⊸



commutative: σ crosses R⊗S by naturality and is absorbed by cocommutativity and commutativity



associative: coassociativity and associativity; ⊗ re-brackets for nothing, being strict here

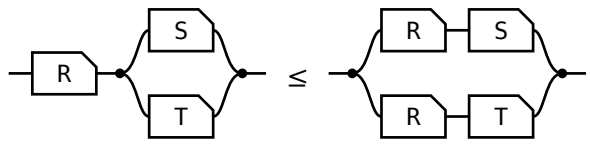


idempotent: the one that is not bookkeeping — the lax copy law is the whole of it, worked in allegory2

So ≤ is the order this monoid induces. R∩S≤R comes from the unit, and idempotency turns anything under both S and T into something under S∩T, since R=R∩R≤S∩T.

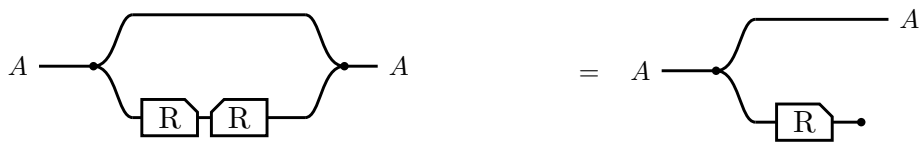
And one law relating ∩ to composition, which is **not** an equation:

(4c)

semi-distributivity, and what supplies it	picture
R (S∩T)⊑RS∩RT — the lax copy law. Equality exactly when R is single valued: the Maps section's F(R∩S)=FR∩FS.	

§5. Domain and range

The **domain** `Dom(R)≜𝟙∩RR°` and the **range** `Ran R≜Dom(R°)`. (5a)



(5b)

▷ lands the return leg back on the value ◁ handed out,
so R°▷ cuts to ⊸ — Frobenius

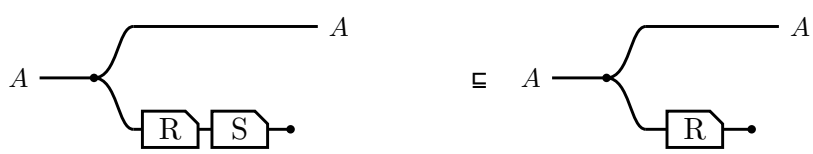
Running `R` and throwing the result away leaves only the fact that `R` could fire, and `Ran R` is the same picture with the box mirrored. In `Rel` both steps are `{(a,a) : ∃b. a R b}`.

(5c)

`Dom(R)≜𝟙∩RR°`
`Dom(R)⊑A⟺R⊑AR`, for `A` coreflexive
`Dom(RS)⊑Dom(R)`
`Dom(R∩S)=𝟙∩SR°`
`R` entire `⟺Dom(R)=𝟙⟺𝟙⊑RR°`
`R` simple `⟺R°R⊑𝟙`
`R` a map `⟺R` entire and simple
`R,S` entire `⟹RS` entire — likewise simple, likewise maps ,
`RS` entire `⟹R` entire

§5.1. Sliding the discard

`Dom(RS)⊑Dom(R)`, and a single glyph for `Dom` would have nothing to slide: with the box and the discard drawn apart, the law is one dot walking back along the lower strand.



(5.1a)

`S⊸⊑⊸`, the lax axiom for ⊸ in the first section
— the discard slides back past `S`

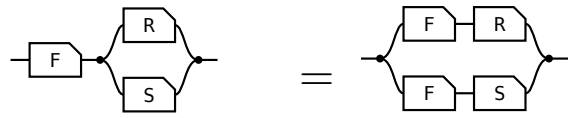
Equality is `S` entire, which is the same picture read as `Dom(R)=𝟙⟺R` entire.

§6. Maps

(6a)

surjective $\mathbb{1}\sqsubseteq R^\circ R$, that is, R° entire.	**single valued** $R^\circ R\sqsubseteq\mathbb{1}$
entire $\mathbb{1}\sqsubseteq RR^\circ$. With **single valued**, a **map**.	**injective** $RR^\circ\sqsubseteq\mathbb{1}$

(6b)

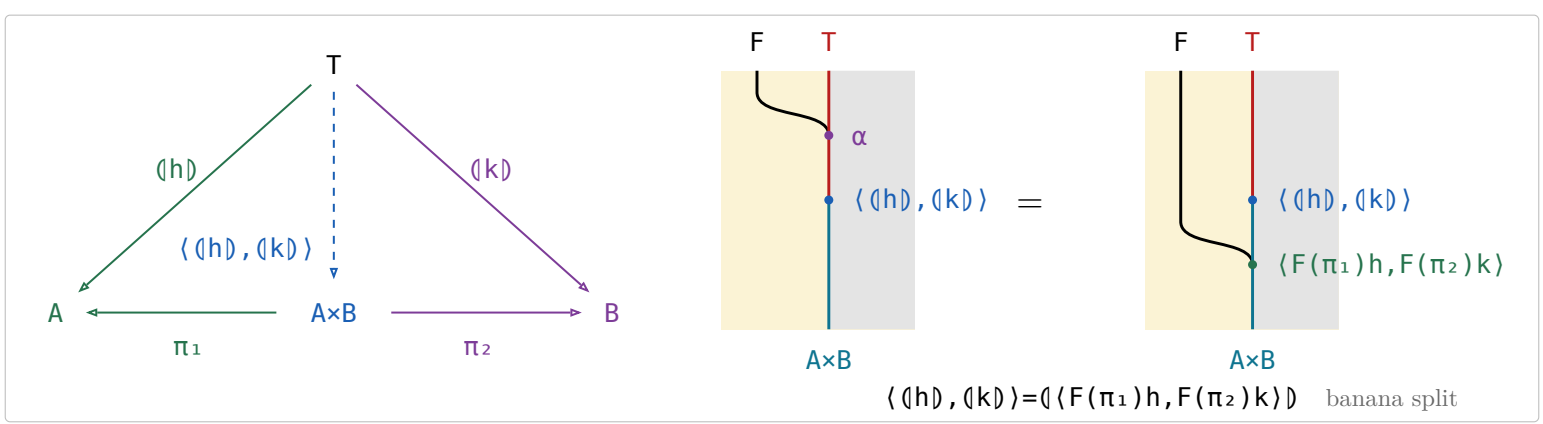
law	picture
$F(R\cap S)=FR\cap FS$ F single valued	 one person who admires both — a at A, b at B

§7. Reduce in $\mathcal{S}$et

(7a)

definition	type	note
`listr A::=nil|cons(A,listr A)`	$\mathcal{S}$et→$\mathcal{S}$et	The cons-lists over `A`, the datatype every row below folds.
`sum≜⦇[zero,plus]⦈`	`listr Nat→Nat`	`plus(a,b)=a+b`.
`length≜⦇[zero,π₂ succ]⦈`	`listr A→Nat`	`π₂` drops the head and keeps the count of the tail, `succ` adds one for the head.
`average≜⟨sum,length⟩ div`	`listr Nat→Real`	`div(m,n)=m/n`, with `div(0,0)=0` so `average` is total. Traverses the list twice.
banana-split law `⟨⦇h⦈,⦇k⦈⟩=⦇⟨F(π₁)h,F(π₂)k⟩⦈`	`T→A×B`	Any fork of folds is a single fold, hence one traversal — `F` the base functor.
what it reduces to `α⟨⦇h⦈,⦇k⦈⟩=F(⟨⦇h⦈,⦇k⦈⟩)⟨F(π₁)h,F(π₂)k⟩`	`FT→A×B`	All that ≜ `reduce` leaves to check: the fork satisfies the defining equation.
the instance `⟨sum,length⟩=⦇[zeros,pluss]⦈`	`listr Nat→Nat×Nat`	`pluss(a,(b,n))=(a+b,n+1)`, so `average` runs in one pass.
`preds n=[n,n−1,…,1]`	`Nat→[Nat]`	apply the earlier special case to write `preds` as `⦇k⦈π₁`.

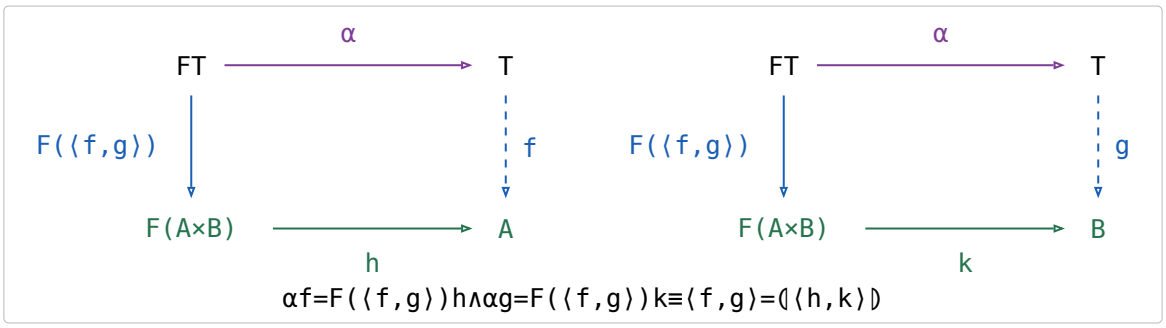
(7b)



§7.1. Fokkinga's mutual recursion theorem

- `F-Alg(𝒜)` — the algebras and their homomorphisms — has binary products whenever 𝒜 does, and the forgetful `U : F-Alg(𝒜)→𝒜` **creates** them: the product of `h : FA→A` and `k : FB→B` is carried by `A×B`, with structure `⟨F(π₁)h,F(π₂)k⟩ : F(A×B)→A×B`.
- `π₁` and `π₂` are homomorphisms out of it, and `⟨p,q⟩ : X→A×B` is a homomorphism **iff** `p` and `q` both are — substitute the structure and compare the two forks component by component.
- The banana-split law of (7a) IS that product's universal property read at the initial algebra: `⦇h⦈` and `⦇k⦈` are the unique homomorphisms to `(A,h)` and `(B,k)`, so their fork is the unique homomorphism into the product, which is `⦇⟨F(π₁)h,F(π₂)k⟩⦈`.
- Fokkinga's theorem is **strictly more general** and is not that product: in mutual recursion the two arrows leave `F(A×B)`, not `FA` and `FB`, so each sees BOTH components. The product is the case where they factor as `F(π₁)h` and `F(π₂)k`. What it says: **any algebra on a product carrier is folded by a fork**.

(7.1a)



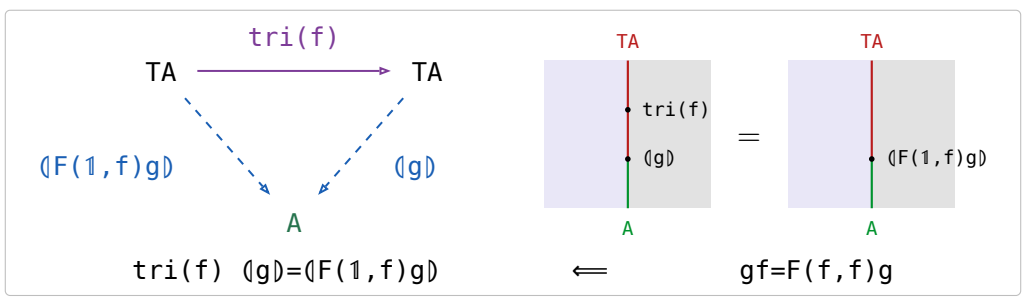
§7.2. Ruby triangles

(7.2a)

stage	definition
informally	`tri(f)[a₀,a₁,…,aᵢ,…,aₙ]=[a₀,f(a₁),…,fⁱ(aᵢ),…,fⁿ(aₙ)]`
for cons-lists	`tri(f)=⦇[nil,(𝟙×listr(f)) cons]⦈`
with the base functor named	`F(A,B)=1+A×B`, `α=[nil,cons]`, so `tri(f)=⦇F(𝟙,listr(f))α⦈`
abstractly	`F` a bifunctor with initial type `(α,T)`: `tri(f)=⦇F(𝟙,T(f))α⦈`

For the definition to make sense `f : A→A` is required, and then `tri(f) : TA→TA`.

(7.2b)



Horner's rule is Horner's rule, generalised from cons-lists to any initial type. Fusion reduces it to `F(𝟙,T(f))α⦇g⦈=F(𝟙,⦇g⦈)F(𝟙,f)g`.

§7.3. Depth of a tree

(7.3a)

the datatype	`tree A::=tip(A)|bin(tree A,tree A)`, base functor `F(A,B)=A+B×B`, initial type `([tip,bin],tree)`
the fold	`⦇[g,h]⦈` is the unique `f` with `f(tip(a))=g(a)` and `f(bin(x,y))=h(f(x),f(y))`
`tree(f)`	`tree(f)=⦇F(f,𝟙) [tip,bin]⦈`; pointwise `tree(f)(tip(a))=tip(f(a))` and `tree(f)(bin(x,y))=bin(tree(f)(x),tree(f)(y))`
`max`	`max=⦇[𝟙,bmax]⦈`, where `bmax(a,b)` is the larger of `a` and `b`
`depths`	`depths=tree(zero) tri(succ)` — replaces every tip by its depth in the tree, `zero` the constant function returning 0 and `succ` the successor
`depth`	`depth=depths max`

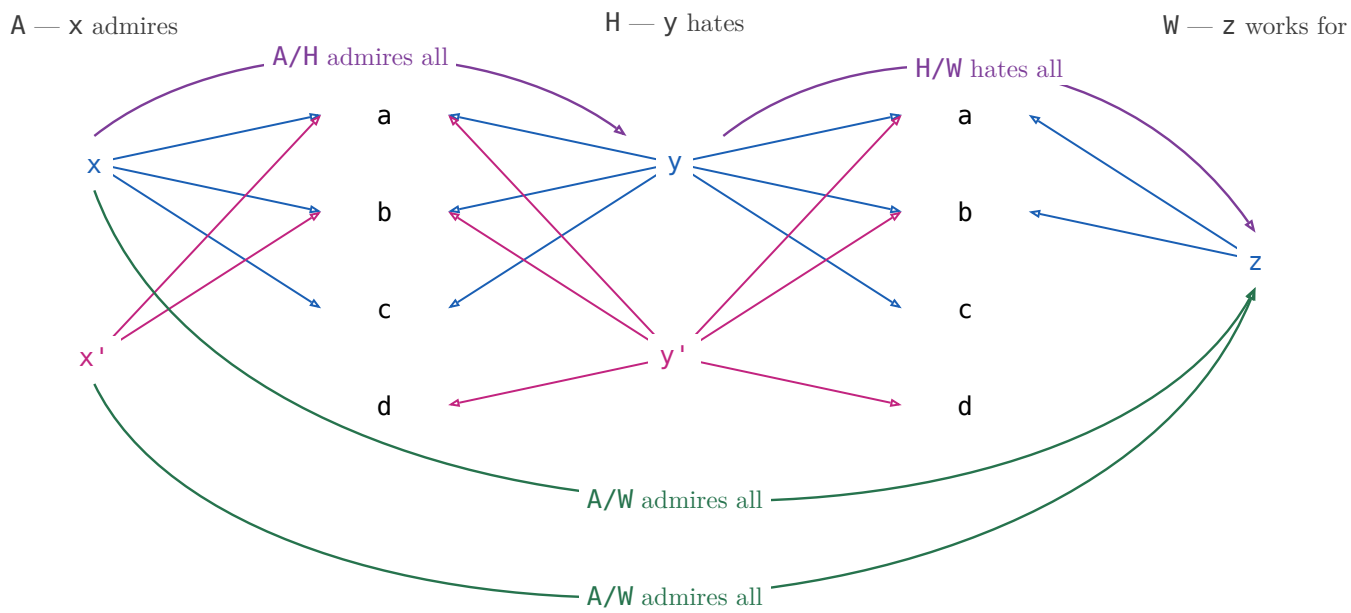
§8. / is all of

(8a)

```
x (R/S) y⟺∀p. y S p→x R p
x (S\R) y⟺∀p. p S x→p R y
/compares images: S(y)⊆R(x).   \ compares preimages: S°(x)⊆R°(y).
example: A admires,H hates,W works for
x (A/H) y — x admires everyone y hates.
x (H\A) y — everyone who hates x admires y.
```

§8.1. (R/S)(S/W)⊑R/W

(8.1a)

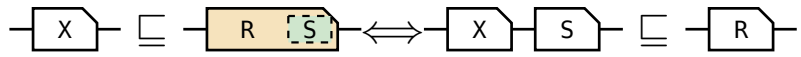
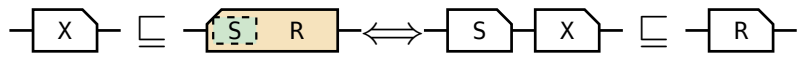
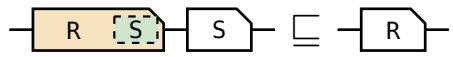
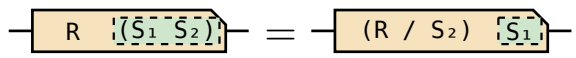
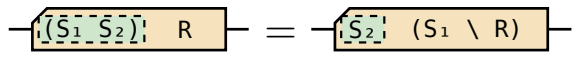
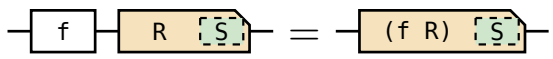
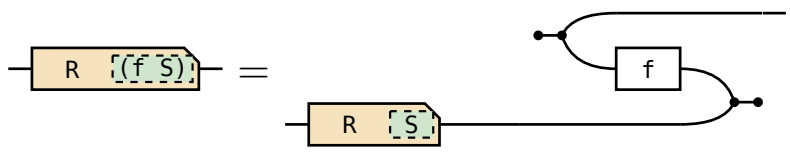
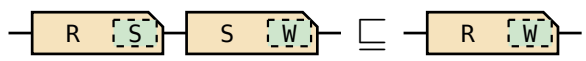
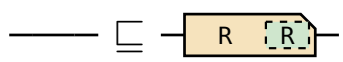
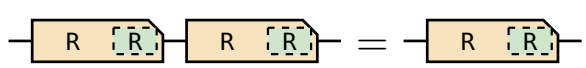
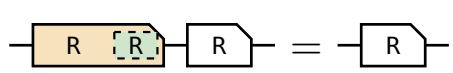
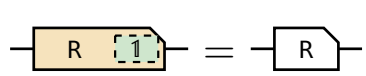
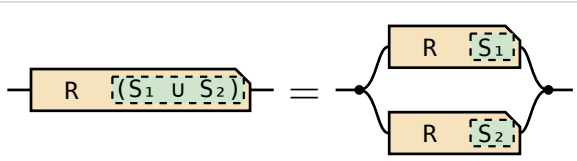
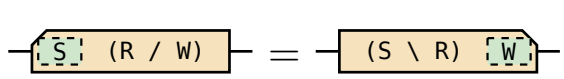


(8.1b)

x (A/H) y — x admires everyone y hates
y (H/W) z — y hates everyone z works for
x (A/W) z — x admires everyone z works for

(A/H)(H/W) is a path: x → y → z, and that is all of it. A/W also holds of x', who admires everyone z works for — but x' does not admire everyone anybody hates, so nothing composes to it. The missing path is exactly the strictness of (R/S)(S/W)⊑R/W.

(8.1c)

$X \sqsubseteq R/S \iff XS \sqsubseteq R$ X is any x-to-y pairing; one that only pairs x with a y such that x admires everyone y hates lies inside R/S, and R/S is the largest such.	
$X \sqsubseteq S\backslash R \iff SX \sqsubseteq R$ The mirror — divide on the left when x comes first.	
$(R/S)S \sqsubseteq R$ There is a y such that x admires everyone y hates, and p is one of the people y hates — then x admires p too. Strict at $S=\varnothing$: R/S is everyone, $(R/S)S=\varnothing$.	
$S\ (S\backslash R) \sqsubseteq R$ The mirror.	
associate: $R/(S_1S_2)=(R/S_2)/S_1$ **A friend's enemy** is two hops: divide by the far end first.	
$(S_1S_2)\backslash R=S_2\backslash(S_1\backslash R)$ The mirror.	
maps: $f\ (R/S)=(fR)/S$ Rename x before or after dividing — the licence to write fR/S.	
$R/(fS)=(R/S)f^\circ$ Rename y: a map leaves a denominator as f° outside the box.	
$(R/S)(S/W) \sqsubseteq R/W$ Someone who admires all of a hate-set that already covers everyone z works for admires those people too.	
$\mathbb{1} \sqsubseteq R/R$ R/R runs admirer to admirer: each admires everyone they admire. Strict: two people who each admire only a and b admire each other's idols too, and still stay two people.	
$(R/R)(R/R)=R/R$ R/R is the preorder **admires at least as much as**, and a preorder is idempotent. Freyd writes $\sqsubseteq$; with $\mathbb{1} \sqsubseteq R/R$ above it is an equality.	
$(R/R)R=R$ Reaching p through someone whose idols x fully admires is reaching p directly, since x admires their own idols.	
$R/\mathbb{1}=R$ Dividing by $\mathbb{1}$: p's set is just $\{p\}$, so admiring all of it is admiring p.	
$R/(S_1 \cup S_2)=R/S_1 \cap R/S_2$ Admiring a combined hate-set is admiring each set in full.	
$S\backslash(R/W)=(S\backslash R)/W$ Which is why $S\backslash R/W$ needs no bracket.	

Fifteen laws, fifteen pictures, and not one shows a generator: $\cap$, $\cup$, $\circ$ and composition are what the Frobenius generators build, and $/$ is none of those — it is posited, with nothing to unfold.

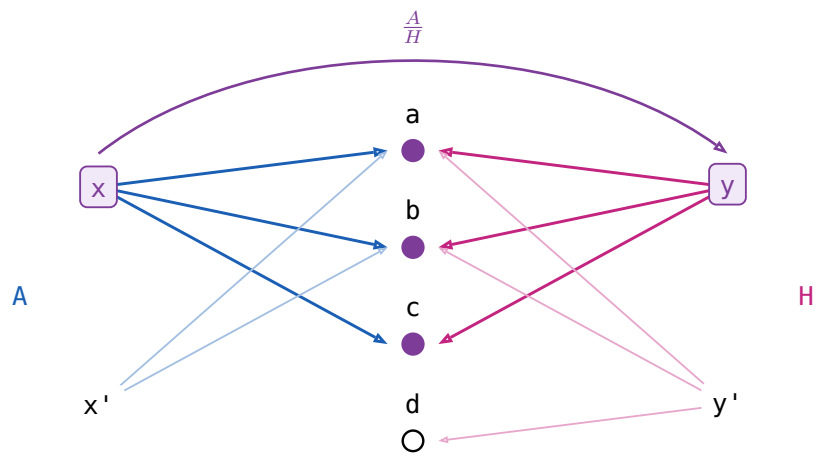
§9. $\frac{R}{S}$

$\frac{R}{S}$ ≜`(R/S)∩(S/R)°` . In `Rel` `x` and `y` has the same image: `∀p. (x R p⟺y S p)` (9a)

`x (A/H) y` — `x` admires everyone `y` hates (9b)
`x ((H/A)°) y` — `x` admires only people `y` hates
`x ((A/H)∩(H/A)°) y` — `x` admires only and all whom `y` hates

`x` admires exactly whom `y` hates: `/` is **all of**, $\frac{R}{S}$ is **only all of**.

(9c)



(9d)

law	the reading
$X \sqsubseteq \frac{R}{S} \iff XS \sqsubseteq R$ and `X°R⊑S`	`X` may pair `x` with `y` only when `x` admires exactly whom `y` hates. Both halves must typecheck, so the operation is **partial**.
$\left(\frac{R}{S}\right)^\circ = \frac{S}{R}$	Matching is symmetric.
$\frac{R}{S}\frac{S}{W} \sqsubseteq \frac{R}{W}$	And transitive.
$\frac{R}{S}S \sqsubseteq R$	$\left(\exists y.\, x\left(\frac{R}{S}\right)y \wedge ySp\right) \to xRp$ `x only admires whom y hates` $\frac{R}{S}S = \mathrm{Dom}\left(\frac{R}{S}\right)R$

$\frac{R}{R}R = R$	$\left(\exists y.\, x\left(\frac{R}{R}\right)y \wedge yRp\right) \iff xRp$
	`x and y admire the same people`
	`y=x always qualifies (𝟙⊑R%R below)`

$\mathbb{1} \sqsubseteq \frac{R}{R}$	$x\left(\frac{R}{R}\right)y$ if `x` and `y` admires the same peoples.
$\left(\frac{R}{R}\right)^2 = \frac{R}{R}$	So the relation **admires the same people** is an equivalence relation.
$X \sqsubseteq \frac{R}{R} \iff XR \sqsubseteq R$, for symmetric `X`	The largest symmetric arrow that leaves `R` alone.
$\frac{R}{\mathbb{1}}$ is the **simple part** of `R`	The people who admire exactly one person and nobody else. It equals `R` only when `R` is simple, unlike `R/𝟙=R`.
`Dom` $\frac{R}{S}$ `=𝟙∩(R/S)(S/R)`	Its domain is the **domain of simplicity** of `R`.

§10. Freyd's Power allegories

(10a)

A **power allegory** is a division allegory with one unary operation on arrows, ∋ **epsiloff**, subject to

$$\ni_{\mathrm{R}}\square = R\square,\quad \ni_{\mathrm{R}} = \ni_{\mathrm{R}\,\square}$$
$$\mathbb{1} \sqsubseteq (R/\ni_{\mathrm{R}})(\ni_{\mathrm{R}}/R) \qquad \ni_{\mathrm{R}} \text{ is } \textbf{thick}$$
$$\tfrac{\ni_{\mathrm{R}}}{\ni_{\mathrm{R}}} = \mathbb{1} \qquad \ni_{\mathrm{R}} \text{ is } \textbf{straight}$$

R□ is R's target, an identity arrow. For `R : A→B` write `∋ : EB→B`, dropping the subscript.

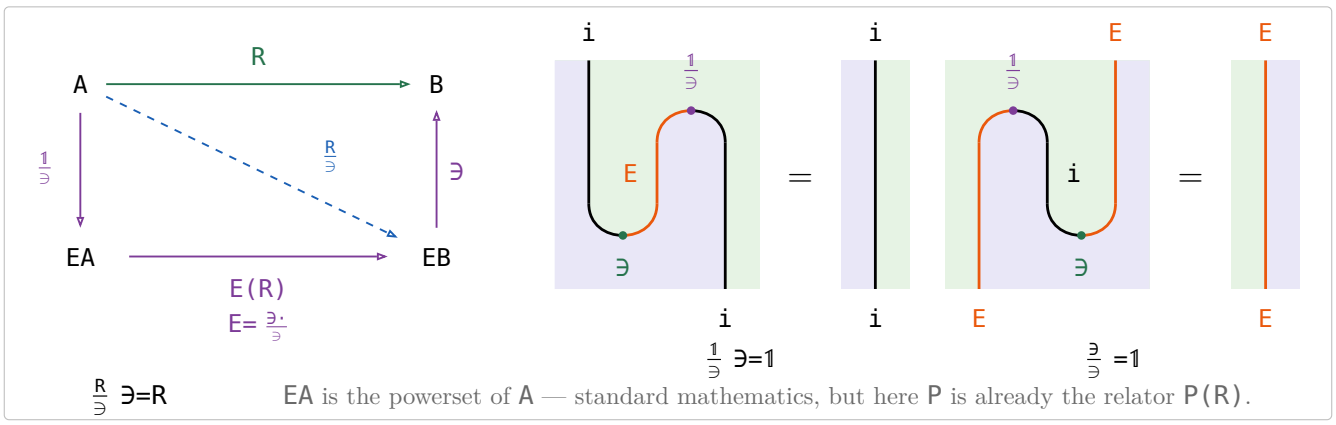
`∈≜∋° : A→EA`

(10b)

law	the reading
1 $\ni_{\mathrm{R}}\square = R\square,\ \ni_{\mathrm{R}} = \ni_{\mathrm{R}\,\square}$	One ∋ per object, not per arrow.
2 ∋ is **thick**	**Comprehension**: every x has a set of exactly the people x admires.
3 $\frac{R}{\ni}$: A→EB, for R : A→B	convert a relation to a function. a $\frac{R}{\ni}$ ={b|a R b}
4 $\frac{\mathsf{R}}{\ni}$ is a **map**	
5 $\frac{\mathsf{R}}{\ni}$ ∋=R	
6 F⊑ $\frac{\mathsf{F}\ni}{\ni}$, F simple	A partial choice of sets is inside the total one.
7 $\frac{\mathbb{1}}{\ni}$, the **singleton map**, monic	The one-person set.
8 $\frac{\ni}{\ni}$ =𝟙	Make the set of a set, then read it back one level down.
9 **fusion:** $\frac{\mathsf{fR}}{\ni}$ =f $\frac{\mathsf{R}}{\ni}$, f a map	Naturality of the unit, f $\frac{\mathbb{1}}{\ni}$ = $\frac{\mathbb{1}}{\ni}$ (E(f)).
10 $\frac{\mathsf{f}}{\ni}$ =f $\frac{\mathbb{1}}{\ni}$, f a map	Rename first or take singletons first — the fusion row above at R=𝟙.
11 $\frac{R}{S} = \frac{R}{\ni}\left(\frac{S}{\ni}\right)^{\circ}$	x and y match when R sends x and S sends y to the same set.
12 E(R)≜ $\frac{\ni\mathsf{R}}{\ni}$, E≜ $\frac{\ni\cdot}{\ni}$	E(R): EA→EB, image of a set of A
13 $\frac{\mathsf{R}}{\ni}$ = $\frac{\mathbb{1}}{\ni}$ E(R)	$\frac{\mathbb{1}}{\ni}$: x↦{x}
14 $\frac{\mathsf{S}}{\ni}$ E(R)= $\frac{\mathsf{S}}{\ni}$ $\frac{\ni\mathsf{R}}{\ni}$ = $\frac{\mathsf{SR}}{\ni}$	absorption — the monad's composition law, $\frac{\mathsf{S}}{\ni}$ ⋄ $\frac{\mathsf{R}}{\ni}$ = $\frac{\mathsf{SR}}{\ni}$, §10.2
15 subset≜∋/∋ : EA→EA	xs subset ys⟺∀a. ys∋a→xs∋a, that is ys⊆xs, not xs⊆ys.

§10.1. i⊣E Power Allegory defined as adjunction

(10.1a)



EA is the powerset of A — standard mathematics, but here P is already the relator P(R).

i is the inclusion Map(𝒜)→𝒜, doing nothing, so E is both the functor and the monad iE.

§10.2. $\mathcal{A}$≅Kleisli(E)

$\mathtt{E(R)} \triangleq \frac{\ni\mathtt{R}}{\ni}$, $\frac{\mathbb{1}}{\ni}$: A→EA, union=E(∋)= $\frac{\ni\ni}{\ni}$: E(EA)→EA (10.2a)

f⋄g≜f E(g) union : A→EC, for f : A→EB and g : B→EC

the monad is on Map($\mathcal{A}$), not on the allegory: E is a relator on all relations, but $\frac{\mathbb{1}}{\ni}$, union and f E(g) union are maps, and the Kleisli construction happens where they live.

$$\boxed{\frac{\mathtt{S}}{\ni} \diamond \frac{\mathtt{R}}{\ni}} \;\underset{\text{definition of } \diamond,\ \mathtt{union=E(\ni)}}{=\!=\!=\!=} \; \boxed{\frac{\mathtt{S}}{\ni}\ \mathtt{E}(\tfrac{\mathtt{R}}{\ni})\mathtt{E}(\ni)} \;\underset{\mathtt{E}\text{ a functor, }\mathtt{E(X)E(Y)=E(XY)}}{=\!=\!=\!=} \; \boxed{\frac{\mathtt{S}}{\ni}\ \mathtt{E}(\tfrac{\mathtt{R}}{\ni}\ni)} \tag{10.2b}$$

$$\underset{(10b),\ \frac{\mathtt{R}}{\ni}\ni=\mathtt{R}}{=\!=\!=\!=} \; \boxed{\frac{\mathtt{S}}{\ni}\ \mathtt{E(R)}} \;\underset{(10b),\ \text{absorption}}{=\!=\!=\!=} \; \boxed{\frac{\mathtt{SR}}{\ni}}$$

the first three steps only unfold ⋄ — E is a functor, so $\mathtt{E}(\frac{\mathtt{R}}{\ni})\mathtt{union}=\mathtt{E}(\frac{\mathtt{R}}{\ni}\ni)=\mathtt{E(R)}$ and the union is gone. Absorption, the row of (10b), is the whole law, and it is the functoriality of $\frac{\cdot}{\ni}$.

$\mathbb{1}$ goes to $\frac{\mathbb{1}}{\ni}$, the Kleisli identity, by definition — so $\frac{\cdot}{\ni}$ is an isomorphism of categories $\mathcal{A}$≅Kleisli(E), and §10.1's i⊣E is the Kleisli adjunction. i is the identity on objects, so every object of the allegory is free.

§11. Relator

Every hom-set of an allegory is a poset, so an allegory is a **locally posetal 2-category**: the 2-cell from R to S IS R⊑S. A **relator** F : $\mathcal{C}$→$\mathcal{D}$ is a 2-functor between allegories: (11a)

$$\mathtt{F(\mathbb{1})=\mathbb{1}} \qquad \mathtt{F(RS)=F(R)F(S)} \qquad \mathtt{R\sqsubseteq S \Longrightarrow F(R)\sqsubseteq F(S)}$$

Preserving ° is **not** asked for — ° is an identity-on-objects involution $\mathcal{C}^{op}$→$\mathcal{C}$, no part of the 2-category.

the statement
For f a map, F(f) is a map and F(f°)=F(f)°.
Over a **tabular** allegory a functor is a relator ⟺ it preserves °.
F(R°)=F(R)° for every R, so F(R)° needs no bracket.
Two relators agreeing on maps are equal.
F(X∩Y)=F(X)∩F(Y) for X,Y coreflexive.
F(R∩S)⊑F(R)∩F(S), and strictly.
F(Dom(R))=Dom(F(R)) for F preserving °.

(11b)

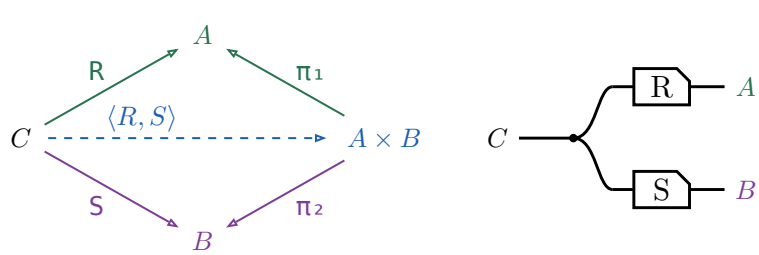
The **power relator** P — xs P(R) ys⟺(∀a∈xs. ∃b∈ys. a R b)∧(∀b∈ys. ∃a∈xs. a R b) — is where the fourth is strict: for R={(a_1,b_1),(a_2,b_2)} and S={(a_1,b_2),(a_2,b_1)} the pair ({a_1,a_2},{b_1,b_2}) is in P(R)∩P(S), while R∩S=∅.

§11.1. Fork ⟨R,S⟩

The **fork** of R : C→A and S : C→B is ⟨R,S⟩≜Rπ_1°∩Sπ_2° , where (π_1,π_2) is the tabulation of ⊤ . (11.1a)

⟨R,S⟩π_1=Dom(S)R ⟨R,S⟩π_2=Dom(R)S (11.1b)

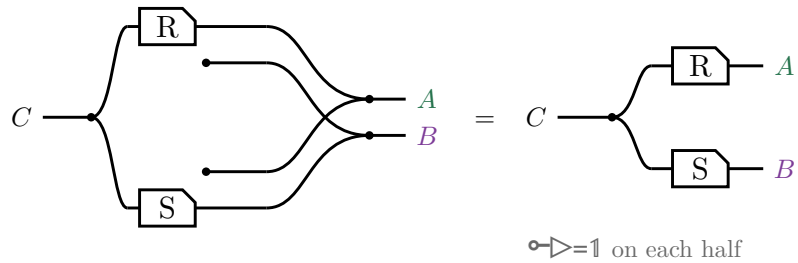
(11.1c)



A domain is coreflexive, so ⟨R,S⟩π_1⊑R, with equality exactly when S is entire; for maps both triangles commute and ⟨f,g⟩ is unique. In Rel, c ⟨R,S⟩ (a,b) iff c R a and c S b — copy c, then R on one strand and S on the other, which is ◁(R⊗S) on the right.

No ° survives the translation. π_1=$\mathbb{1}$⊗⊸ discards the second component, so π_1°=$\mathbb{1}$⊗⊸ **creates** one out of nothing, and ∩ is copy, run both, merge. Draw that and the created strands — the two dots with no left end, and the crossing they force — are merged against real ones, which is the monoid's unit law:

(11.1d)

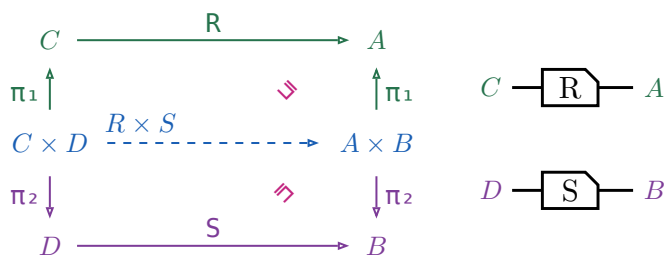


§11.1.1. Relational product R×S

(11.1.1a)

R×S≜⟨π₁R,π₂S⟩ , a relator in each argument but no longer a categorical product.

(11.1.1b)

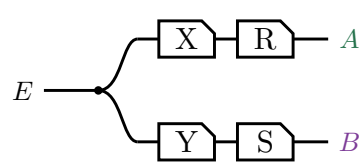


Right-then-up is (R×S)π₁, up-then-right is π₁R, and (R×S)π₁⊑π₁R, equality when S is entire. In Rel, (c,d) (R×S) (a,b) iff c R a and d S b — two strands side by side, no copy dot: R×S=R⊗S.

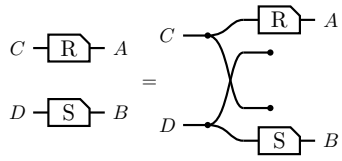
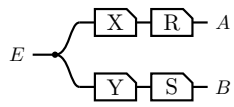
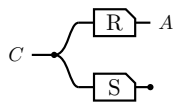
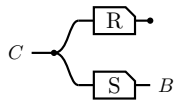
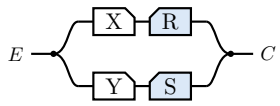
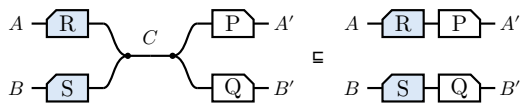
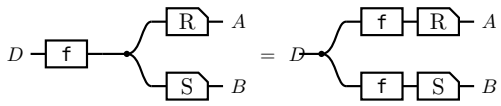
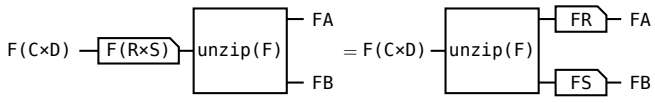
§11.1.2. Absorption

For X : E→C and Y : E→D, ⟨X,Y⟩(R×S)=⟨XR,YS⟩. Both sides are this picture:

(11.1.2a)

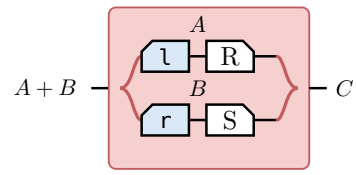
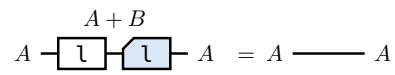
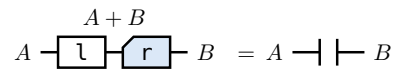
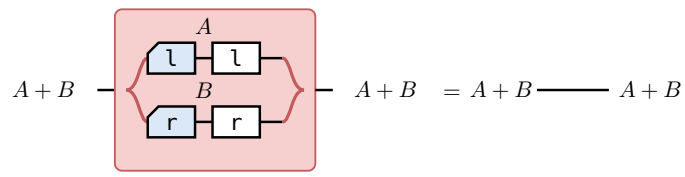


(11.1.2b)

	the law	picture
1	R×S=⟨π₁R,π₂S⟩	
2	⟨X,Y⟩(R×S)=⟨XR,YS⟩ both sides are the same strokes — absorption. Its ⊑ half is that half at S:=𝟙 and at R:=𝟙 — stages of their proof of this row; the corollary is the (R×S)(U×V)=(RU)×(SV) it yields, at R:=𝟙 and V:=𝟙.	
3	⟨R,S⟩π₁=Dom(S)R (11.1b)	
4	⟨R,S⟩π₂=Dom(R)S (11.1b)	
5	⟨X,Y⟩⟨R,S⟩°=(XR°)∩(YS°)	
6	⟨R,S⟩°⟨P,Q⟩⊑(R°P)×(S°Q)	
7	f⟨R,S⟩=⟨fR,fS⟩ f a map; it fails for an arbitrary arrow;	
8	F(R×S)unzip(F)=unzip(F)(F(R)×F(S)) unzip(F)≜⟨F(π₁),F(π₂)⟩, a map	
9	g=curry(f)⟺(g×𝟙)eval=f reading × as the relational product, does Rel have exponentials?	

§11.2. Coproduct [R,S] : A+B→C

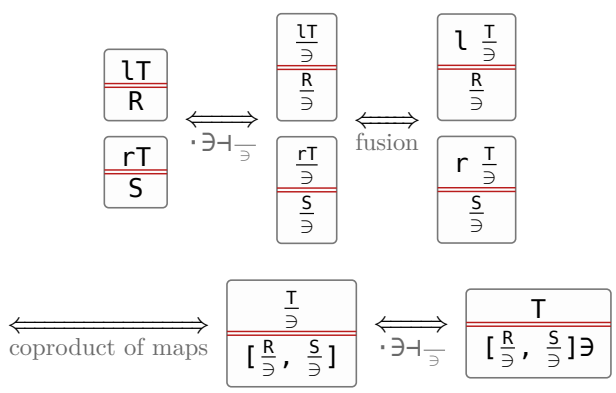
(11.2a)

the statement	picture
`[R,S]≜l°R ∪ r°S` The tape is the union — a particle entering at `A+B` takes exactly one branch — and the two mirrored boxes are what makes the branches disjoint.	
`[R,S]=[R/∋, S/∋]∋`	
`R+S≜[Rl,Sr]`	
`l[R,S]=R`, `r[R,S]=S`, and `[R,S]` is the only such arrow ,	
`ll°=𝟙=rr°`	
`lr°=⊥=rl°`	
`l°l ∪ r°r=𝟙`	
`[U,V]°[R,S]=U°R ∪ V°S`	

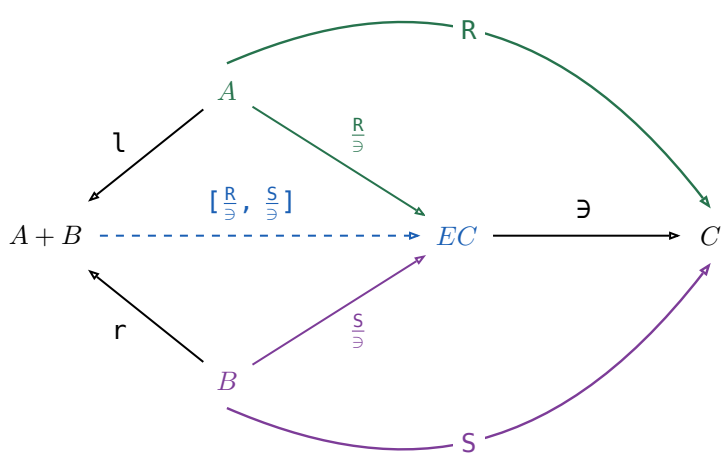
§11.2.1. `[R,S]≜[R/∋, S/∋]∋`

The universal-property row is not free: `l,r` were only ever asked to be a coproduct of **maps**. They stay one once every arrow is allowed because $\frac{\cdot}{\ni}$ sends an arrow `A→C` to a map `A→EC` reversibly, so the map coproduct can be applied underneath it. For any `T : A+B→C`,

(11.2.1a)

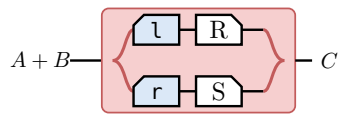
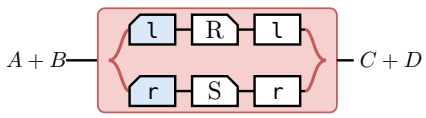
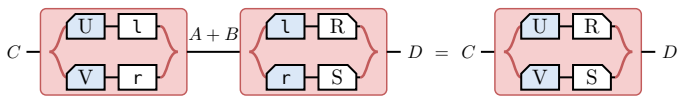
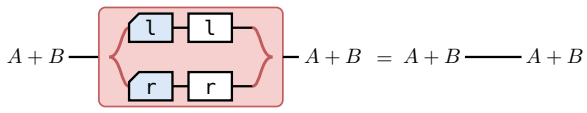


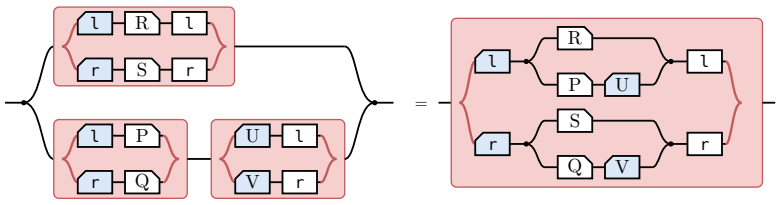
(11.2.1b)



Nothing here holds only up to `⊑`: every triangle commutes on the nose, which is the difference from the fork above. The border spells `[R,S]=[R/∋, S/∋]∋`, and pushing `∋` into the union that `[·,·]` on maps already is turns that back into the definition, `(l° R/∋ ∪r° S/∋)∋=l°R ∪ r°S`.

(11.2.1c)

the law	picture
`[R,S]=(l°R) ∪ (r°S)` (11.2a)'s first row	
`R+S=[Rl,Sr]`	
`[U,V]°[R,S]=(U°R) ∪ (V°S)` (11.2a)'s last row;	
`X≜[𝟙,⊥]=l°` and `Y≜[⊥,𝟙]=r°`, so `(Xl) ∪ (Yr)=[l,r]=𝟙` which is (5.9)	
prove (5.11), and say why duality does not carry it over from the product law	

`(R+S)∩([P,Q][U,V]°)`
`=(R∩(PU°))+(S∩(QV°))`
`[P,Q][U,V]°` is a full 2×2 of composites; `R+S` is diagonal, so the meet cuts the two off-diagonal branches
	

§11.3. The power relator P(R)

(11.3a)

For R : A→B,

$$P(R) \triangleq ((\ni R)/\ni) \cap ((\ni R^\circ)/\ni)^\circ : EA \to EB$$

$$E(R) \triangleq \frac{\ni R}{\ni} = ((\ni R)/\ni) \cap (\ni/(\ni R))^\circ$$

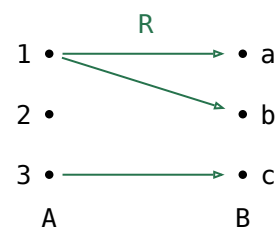
xs P(R) ys⟺(∀a∈xs. ∃b∈ys. a R b)∧(∀b∈ys. ∃a∈xs. a R b)

Every element of xs is related by R to some element of ys, and conversely.

(11.3b)

	in Rel	in words	im(xs)	partner
(∋R)/∋	∀y∈ys. ∃x∈xs. x R y	∀y. some xs R y	ys⊆im(xs)	every y has a partner
(∋/(∋R))°	∀y. (∃x∈xs. x R y)→y∈ys	every y with some xs R y is in ys	ys⊇im(xs)	ys leaves out no partner
E(R)	ys={y|∃x∈xs. x R y}	ys = every y with some xs R y	ys=im(xs)	every y has a partner, and none is left out
((∋R°)/∋)°	∀x∈xs. ∃y∈ys. x R y	∀x. x R some ys	—	every x has a partner
P(R)	∀y∈ys. ∃x∈xs. x R y and ∀x∈xs. ∃y∈ys. x R y	∀y. some xs R y and ∀x. x R some ys	—	every x and every y has a partner

(11.3c)



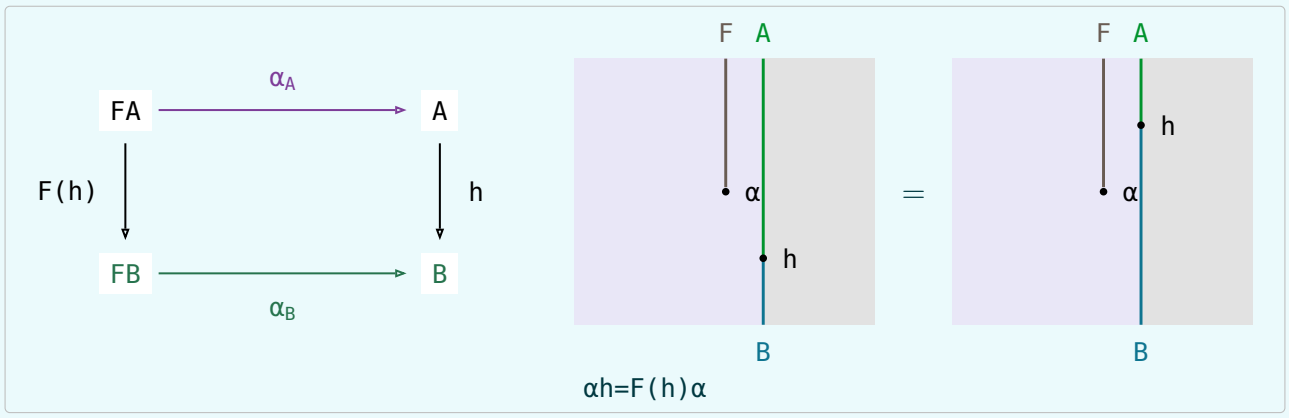
xs	(∋R)/∋	(∋/(∋R))°	E(R)	((∋R°)/∋)°	P(R)
∅	∅	all 8 subsets of abc	∅	all 8 subsets of abc	∅
{1}	∅, a, b, ab	ab, abc	ab	a, b, ab, ac, bc, abc	a, b, ab
{2}	∅	all 8 subsets of abc	∅	none	none
{3}	∅, c	c, ac, bc, abc	c	c, ac, bc, abc	c
{1,2}	∅, a, b, ab	ab, abc	ab	none	none
{1,3}	all 8 subsets of abc	abc	abc	ac, bc, abc	ac, bc, abc
{2,3}	∅, c	c, ac, bc, abc	c	none	none
{1,2,3}	all 8 subsets of abc	abc	abc	none	none

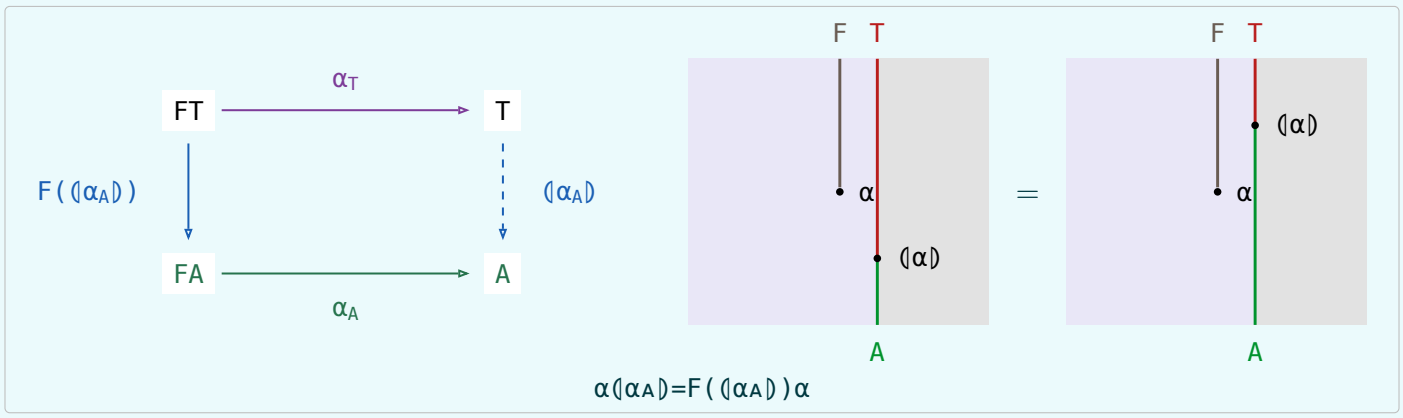
(11.3d)

law	the reading
X⊑P(R)⟺X∋⊑∋R and X°∋⊑∋R°	One containment, and the same one at R° — which is the definition read off the two divisions. Hence P(R°)=P(R)°, and R⊑S⟹P(R)⊑P(S) .
P(𝟙)= $\frac{\ni}{\ni}$ =𝟙	The straightness axiom verbatim: extensionality **is** P's unit law.
P(f)= $\frac{\ni f}{\ni}$, for f a map	In Rel, xs P(f) ys⟺ys={f(a)|a∈xs}. The half at f° says every a∈xs has its f(a) on ys; f has just the one image per a, so that already says ys contains everything xs reaches, which is the fraction's second half. For a map the two definitions coincide.
P(RS)=P(R)P(S)	⊒ is the division cancellation laws. ⊑ is the one law in this section that is not a calculation: it needs a tabulation of P(RS).

§11.4. Initial algebra

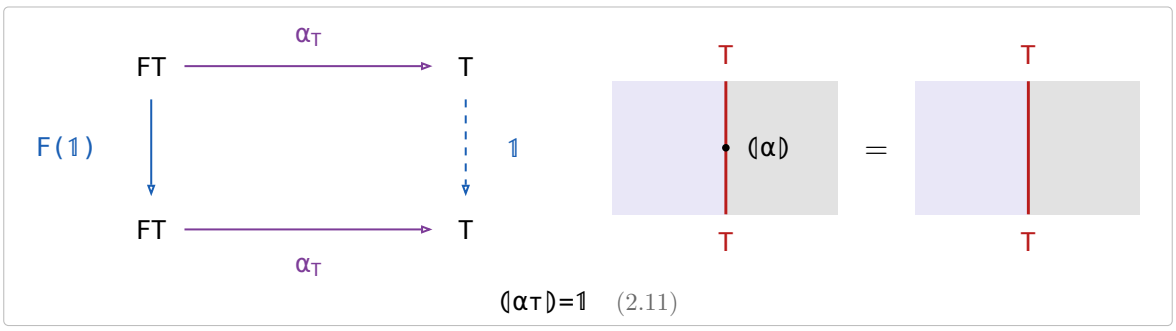
An **F-algebra** on A is a map $\alpha_A : FA \to A$. An **F-homomorphism** from α_A to α_B is a map $h : A \to B$ with $\alpha_A\ h = F(h)\alpha_B$. The **initial algebra** $\alpha_T : FT \to T$ is the F-algebra with exactly one F-homomorphism $(\!|\alpha_A|\!) : T \to A$ to every F-algebra α_A. (11.4a)





§11.4.1. Reflection

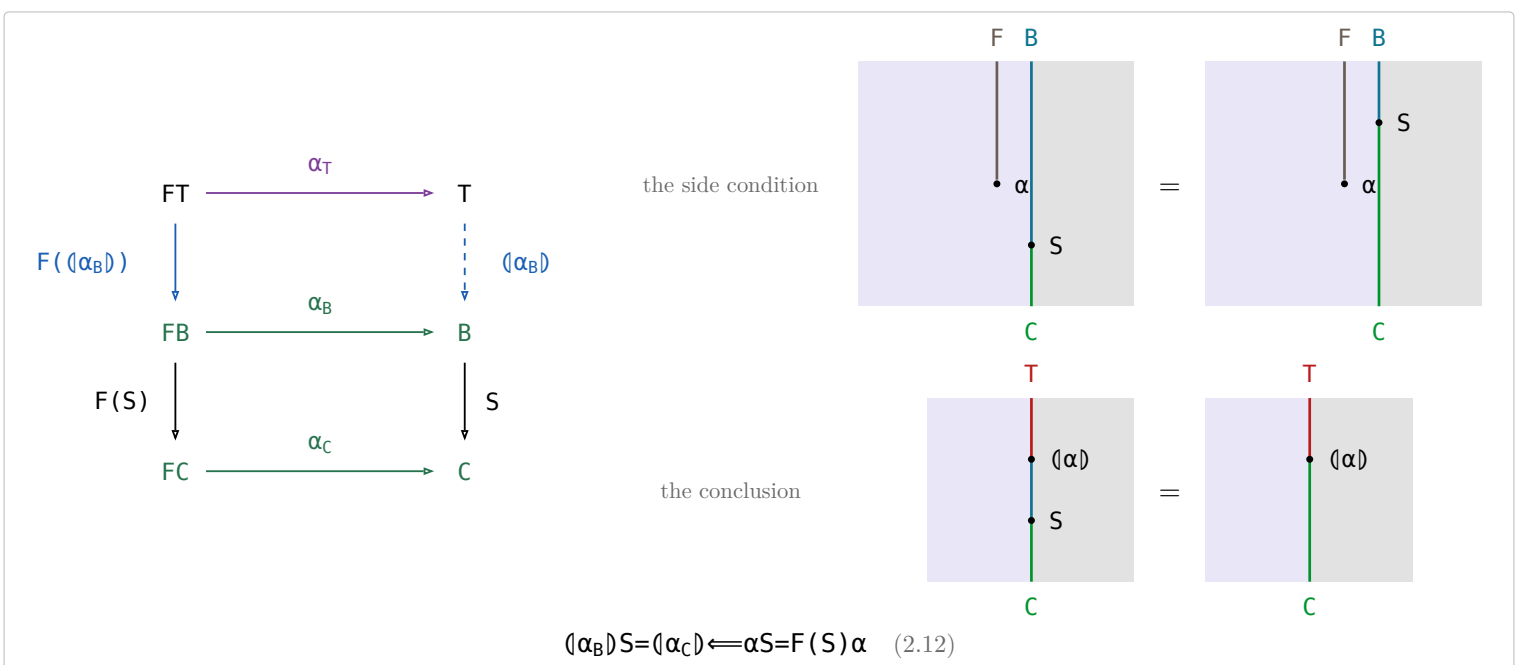
(11.4.1a)



Taking a value apart with α_T and putting it straight back is doing nothing.

§11.4.2. Fusion

When $S : B \to C$ is an F-homomorphism from α_B to α_C, folding with α_B and then applying S is folding with α_C. (11.4.2a)



§11.5. Type relator

(11.5a)

Let F be a binary relator with initial type (α,T), so T is a type functor. F(R,S) is its action on a pair, and F(X) abbreviates F(𝟙,X), the F of the reduce section. For every object A the initial algebra is α : F(A,TA)→TA, among the maps. T acts on an arrow R : A→B by

$$T(R)=(\!|F(R,\mathbb{1})\alpha|\!) : TA\to TB$$

(11.5b)

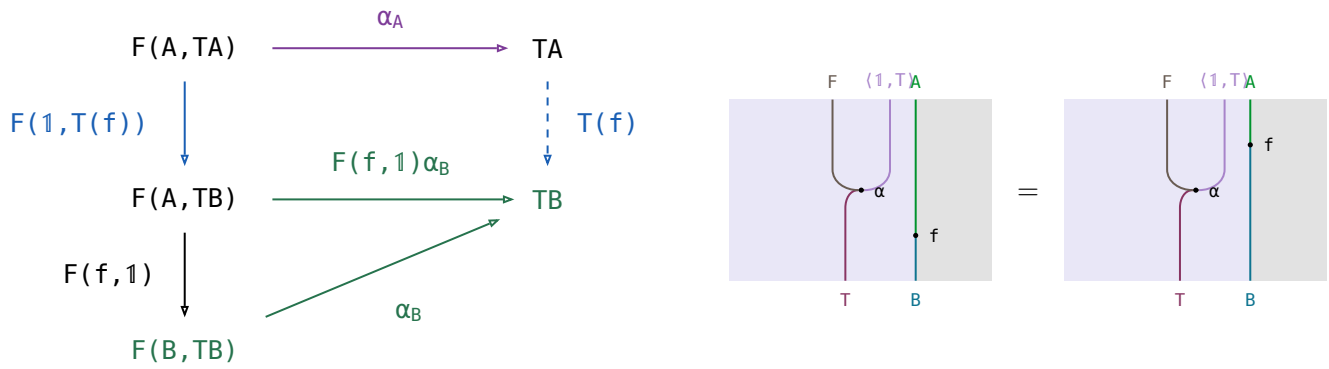
name	law	what it says
the defining equation	T(R)=(|F(R,𝟙)α|)	Rebuild the structure with α, applying R to the parameter on the way.
functor	T(𝟙)=𝟙 and T(R)T(S)=T(RS)	Acting by the identity changes nothing, and two actions in a row are one action.
type functor fusion	T(R)(|Q|)=(|F(R,𝟙)Q|)	A relator action followed by a fold is a single fold — the intermediate structure is never built. The side condition holds because F is a bifunctor — F(R,𝟙)F(𝟙,(|Q|))=F(R,(|Q|))=F(𝟙,(|Q|))F(R,𝟙).
naturality of α	αT(R)=F(R,T(R))α	Building and then mapping is the same as mapping the parts and then building, so α is natural from G(R)=F(R,T(R)) to T.
type relator	T(R)°=T(R°), for F preserving °	A datatype acts on relations, not only on maps — the map of the converse is the converse of the map.

§11.5.1. Type functor

Let F be a bifunctor taking both the parameter A and the recursive position TA, with an initial algebra α_A : F(A,TA)→TA for every object A. Then T is a functor, acting on a map f : A→B by (11.5.1a)

$$T(f) \triangleq (\!| F(f,\mathbb{1})\alpha_B |\!) : TA \to TB$$

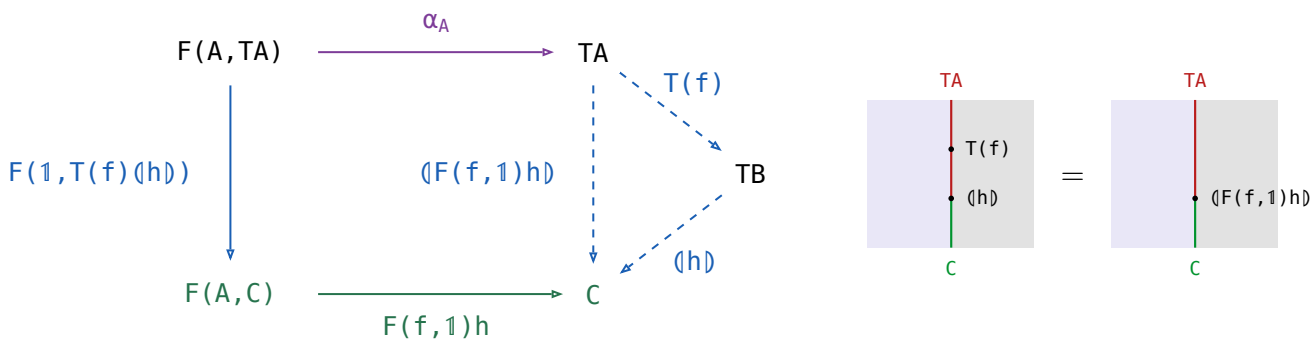
(11.5.1b)



α_A T(f)=F(f,T(f))α_B

- F : $\mathcal{A}\times\mathcal{A}\to\mathcal{A}$ is a bifunctor and a wire is a unary functor, so the two arguments are packed first: ⟨𝟙,T⟩ : $\mathcal{A}\to\mathcal{A}\times\mathcal{A}$ sends A to (A,TA), and F(⟨𝟙,T⟩(A)) is F(A,TA).
- The picture is three wires — F, ⟨𝟙,T⟩, and the object — and the region between the first two is $\mathcal{A}\times\mathcal{A}$.
- α is then an ordinary natural transformation F∘⟨𝟙,T⟩⇒T: its bead eats the F and ⟨𝟙,T⟩ wires, and the T wire is born under it.
- F(f,T(f)) costs no notation. It is the bead f on the object wire with ⟨𝟙,T⟩ and F running past: ⟨𝟙,T⟩ is what turns f into the pair (f,T(f)), and F is what applies it.
- The law is the naturality of α, which is exactly the freedom to slide that f bead past it.

(11.5.1c)



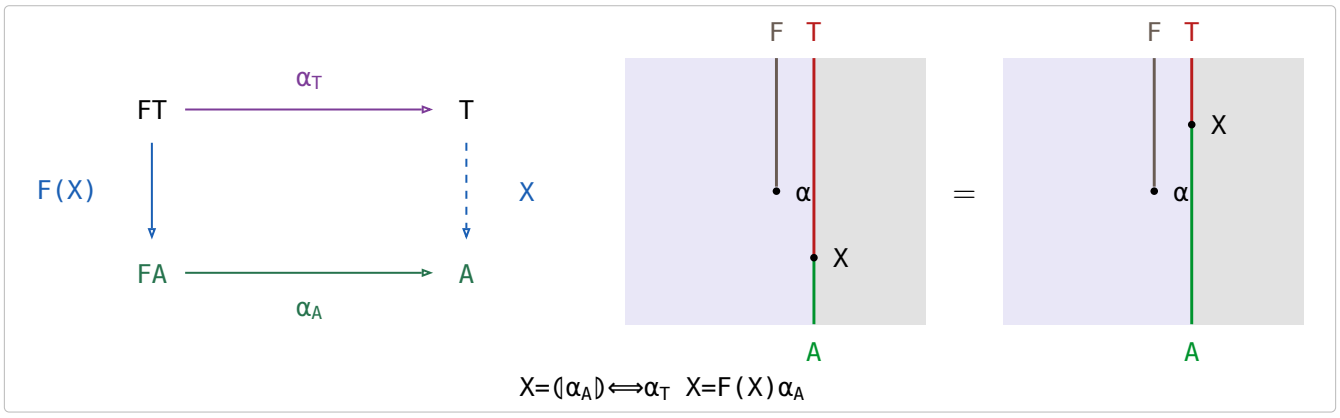
T(f)(|h|)=(|F(f,𝟙)h|)

§11.6. Reduce

let F be a relator and has **initial algebra** α_T : FT→T in the subcategory of functions. α_T is also initial in the allegory: (11.6a)

§11.6.1. The defining equation

(11.6.1a)



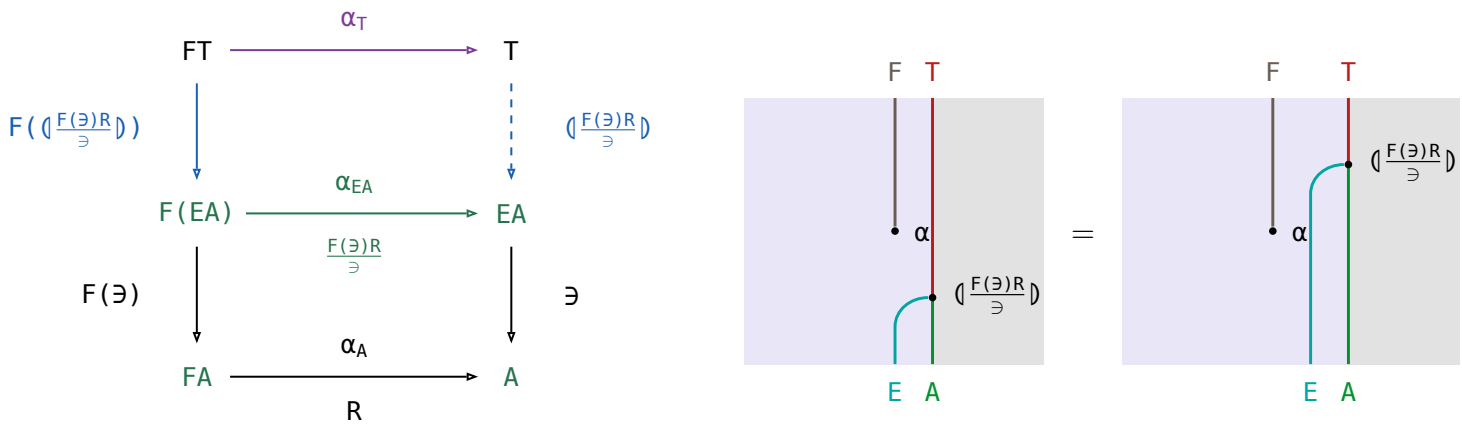
An F-algebra is a **weakened** natural transformation. A transformation F⇒𝟙 on $\mathcal{A}$ would need a component FX→X at every object and a commuting square at every arrow, but F-Algebra only need it works on T and A.

(11.6.1b)

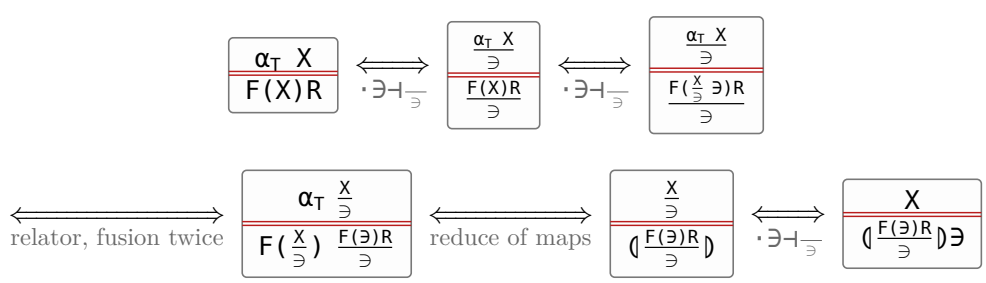
⦇[c,f]⦈=reduce(c,f) catamorphism: arrows of the form ⦇f⦈ are called catamorphisms, and these two examples reveal the notion to be a familiar idea in abstract clothing		
part	**Nat**	**[A]**
datatype	Nat::=zero|succ Nat	[A]::=nil|cons(A,[A])
base functor F	F(X)=1+X	F(X)=1+A×X
initial algebra α	α=[zero,succ] : 1+Nat→Nat	α=[nil,cons] : 1+A×[A]→[A]
the fold, pointwise	⦇[c,f]⦈(zero)=c ⦇[c,f]⦈(succ(n))=f(⦇[c,f]⦈(n))	⦇[c,f]⦈(nil)=c ⦇[c,f]⦈(cons(a,x))=f(a,⦇[c,f]⦈(x))

§11.6.2. ⦇R⦈=⦇$\frac{F(\ni)R}{\ni}$⦈∋

(11.6.2a)



(11.6.2b)



the last two rows at X:=⦇R⦈:

§11.6.3. φ(Y)⊑Y⟹(μX : φ(X))⊑Y

φ a **monotonic** mapping of the hom-set A→B into itself: X⊑Y⟹φ(X)⊑φ(Y) . (11.6.3a)

(μX : φ(X)) the least X : A→B with φ(X)⊑X .

(11.6.3b)

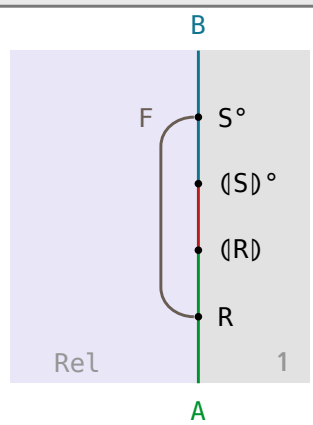
the law	what it says
φ(Y)⊑Y⟹(μX : φ(X))⊑Y	to bound (μX : φ(X)) above, exhibit one Y the body does not grow past — the half §11.6.4 and every chapter after it uses
φ((μX : φ(X)))=(μX : φ(X))	**Knaster–Tarski**: the least solution of φ(X)⊑X already solves φ(X)=X, so the least prefix point and the least fixed point are one relation

§11.6.4. ⦇S⦈°⦇R⦈=(μX : S°F(X)R)

(11.6.4a)

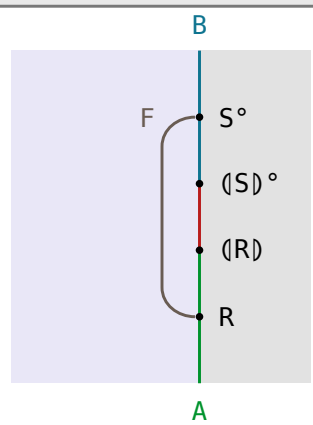
S°F(⦇S⦈°⦇R⦈)R=⦇S⦈°⦇R⦈

hylomorphism theorem: a prototypical 'divide and conquer' scheme — the term **S°** represents the decomposition stage, **F(⦇S⦈°⦇R⦈)** the stage of solving the subproblems recursively, and **R** the recombination stage; **R : FA→A**, **S : FB→B**, **α : FT→T** initial



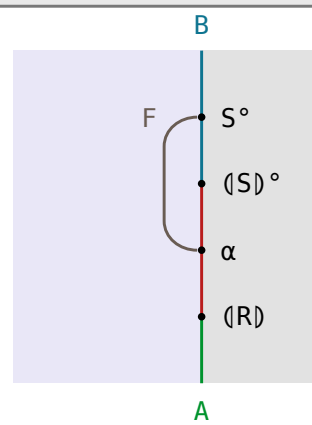
the body at ⦇S⦈°⦇R⦈

=



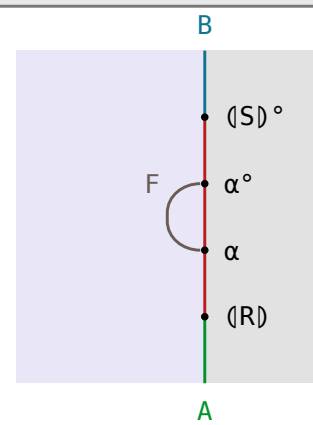
F(RS)=F(R)F(S) — ≜ relator

=



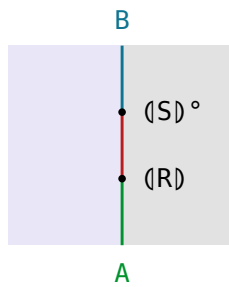
≜ reduce at R: F(⦇R⦈)R=α⦇R⦈

=



≜ reduce at S conversed: ⦇S⦈°α°=S°F(⦇S⦈)°, and F(⦇S⦈)°=F(⦇S⦈°) — (11b)

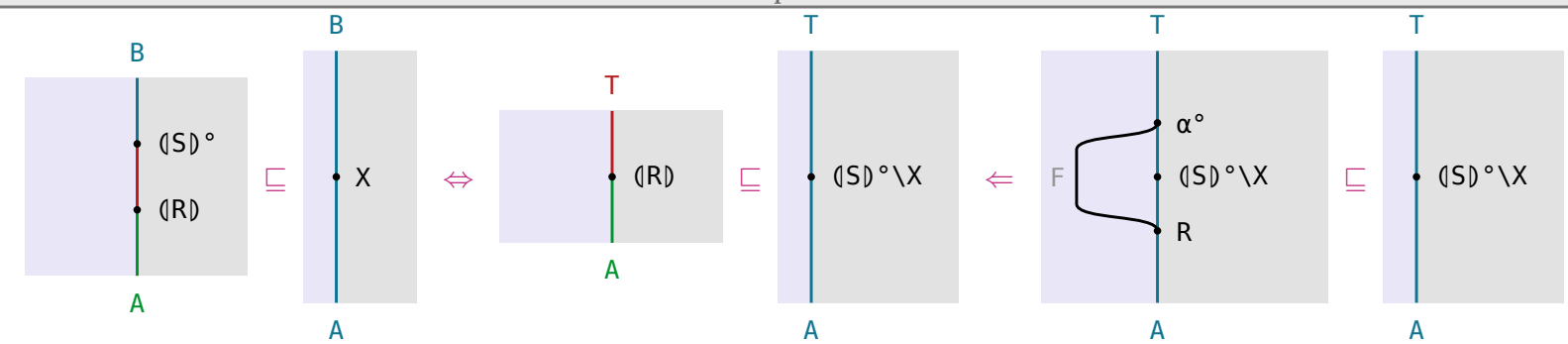
=



α°α=𝟙, Lambek

$S°F(X)R⊑X \Longrightarrow ⦇S⦈°⦇R⦈⊑X$ (11.6.4b)

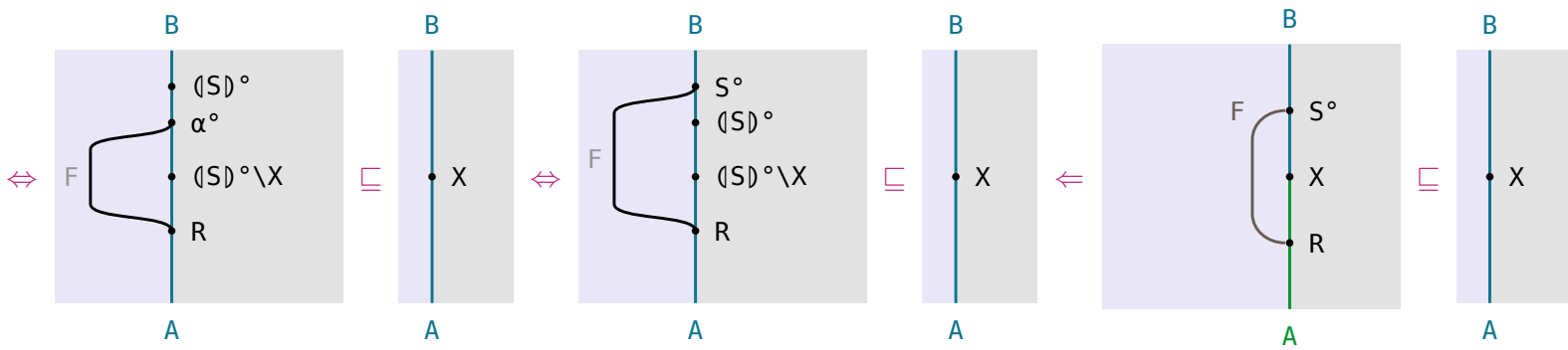
hylomorphism theorem: by Knaster–Tarski, the hylomorphism $⦇S⦈°⦇R⦈$ is included in X if X satisfies the associated recursion inequation



the conclusion

(1.1a)'s $S·⊣S\backslash$ at $⦇S⦈°$

(6.2) $⦇R⦈=(\mu X : \alpha°F(X)R)$ — $\triangleq$ reduce and (11.6.3b);



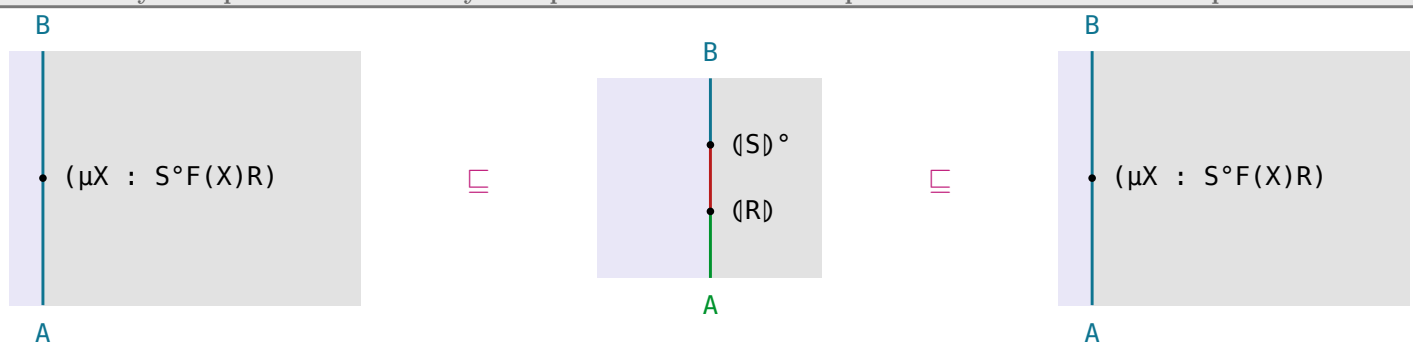
(1.1a)'s $S·⊣S\backslash$ at $⦇S⦈°$

$⦇S⦈°α°=S°F(⦇S⦈°)$ — (11.6.4a)

$F(RS)=F(R)F(S)$ — $\triangleq$ relator — and $⦇S⦈°(⦇S⦈°\backslash X)⊑X$ — (1.1a)

$⦇S⦈°⦇R⦈=(\mu X : S°F(X)R)$ (11.6.4c)

hylomorphism theorem: a hylomorphism is the least fixed point of a certain recursion equation



$\triangleq$ μ at $\varphi(X):=S°F(X)R$

(11.6.3b)'s $\varphi(Y)⊑Y \Longrightarrow (\mu X : \varphi(X))⊑Y$ at $Y:=⦇S⦈°⦇R⦈$, whose $S°F(⦇S⦈°⦇R⦈)R=⦇S⦈°⦇R⦈$ is (11.6.4a)

(11.6.4b) at $X:=(\mu X : S°F(X)R)$, whose $S°F((\mu X : S°F(X)R))R⊑(\mu X : S°F(X)R)$ is (11.6.3b)'s $\varphi((\mu X : \varphi(X)))=(\mu X : \varphi(X))$;

§12. Combinatorial functions

(12a)

definition	type	note
`[A]::=nil|cons(A,[A])`	$\mathcal{A}\to\mathcal{A}$	The list type, under the short name it keeps.
`list(R)≜⦇[nil,(R⊗𝟙) cons]⦈`	`[A]→[B]`	The relator's action on `R : A→B`: one `R` per element, the shape untouched.
`subseq≜⦇[nil,cons ∪ π₂]⦈`	`[A]→[A]`	`xs subseq ys`: `ys` is `xs` with elements dropped — `cons` keeps the head, `π₂` drops it.
`prefix≜⦇[nil,nil ∪ cons]⦈ =cat° π₁=init*`	`[A]→[A]`	`ys` is an initial segment of `xs`; the first `nil` is where it stops early. `init≜snoc° π₁`.
`suffix≜cat° π₂=tail*`	`[A]→[A]`	The dual, `tail≜cons° π₂`; as a reduce it needs snoc-lists.
`segment≜suffix prefix`	`[A]→[A]`	A contiguous stretch of `xs`: a suffix, then a prefix of that.
`partition≜concat°`	`[A]→[[A]⁺]`	This `cat` is restricted to `[A]⁺×[A]→[A]`, so `ys` is a list of non-empty segments of `xs`.
`concat≜⦇[nil,cat]⦈`	`[[A]]→[A]`	Joins the segments back up, which is why its converse splits a list.
`inits`	`[A]→[[A]]`	Implements $\frac{\texttt{prefix}}{\ni}$, listing the prefixes by increasing length.
`tails`	`[A]→[[A]]`	Implements $\frac{\texttt{suffix}}{\ni}$ by decreasing length — the opposite order.
`filter(p)≜` $\frac{\texttt{subseq list(p)}}{\ni}$ `est(R°)`	`[A]→[A]`	The longest subsequence of `xs` whose every element passes `p`. `est(R°)` is ≜ `est`
`R≜length≤length°`	`[A]→[A]`	The preorder `filter` and `takewhile` maximise over: the longer list wins. `≥≜≤°`
`takewhile(p)≜` $\frac{\texttt{prefix list(p)}}{\ni}$ `est(R°)`	`[A]→[A]`	The same with `prefix` for `subseq`: the longest prefix whose every element passes `p`.
`mss≜` $\frac{\texttt{segment sum}}{\ni}$ `est(≥)`	`[Int]→Int`	Maximum segment sum. `segment=suffix prefix` splits it into $\frac{\texttt{prefix sum}}{\ni}$ `est(≥)` on each suffix.

§12.1. $\frac{\texttt{subseq}}{\ni}$ =⦇[nil $\frac{1}{\ni}$,⟨$\frac{\mathbb{1}\times\ni}{\ni}$ E(cons),π₂⟩ cup]⦈

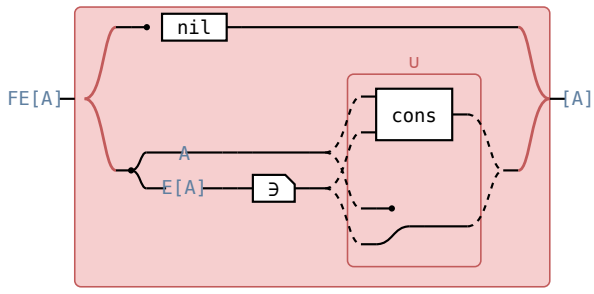
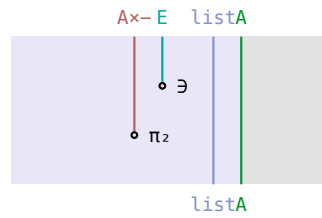
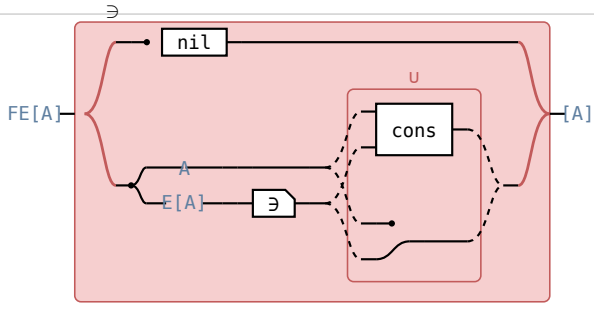
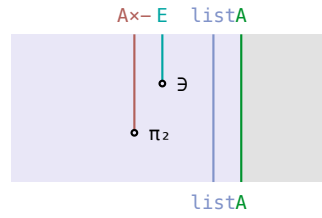
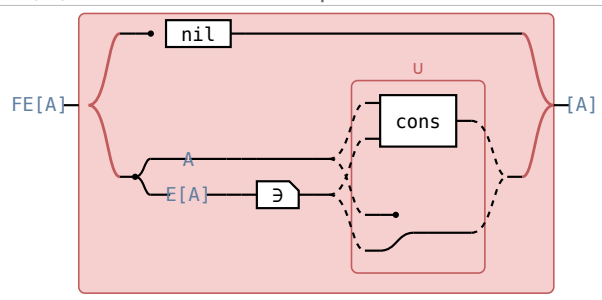
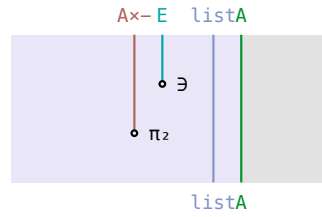
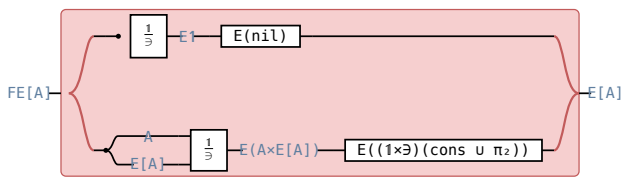
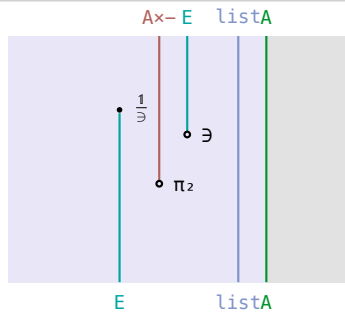
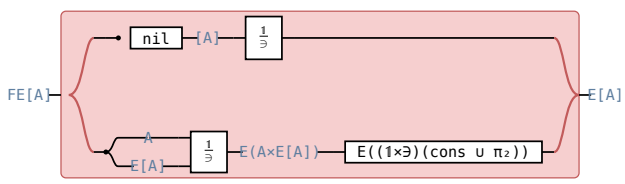
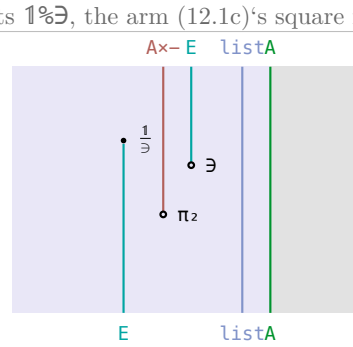
(12.1a)

`cup≜` $\frac{\pi_1\ni \cup \pi_2\ni}{\ni}$ `: EA×EA→EA`, so $\frac{R \cup S}{\ni}$ =⟨$\frac{R}{\ni}$, $\frac{S}{\ni}$⟩ cup.

(12.1b)

$$\frac{F(\ni)[nil,cons \cup \pi_2]}{\ni} = [nil\ \frac{\mathbb{1}}{\ni}, \frac{(\mathbb{1}\times\ni)(cons \cup \pi_2)}{\ni}]$$

the set of lists the algebra builds is, from nothing, just **nil**, and from a head and a set of tails, every tail in the set with the head put on or left off — (11.6.2b) at **subseq=⦇[nil,cons ∪ π₂]⦈**, ≜ **subseq**.

circuit — the fork is F([A])=1+A×[A]: nil above, the pair below	**Hinze–Marsden**
 $\frac{F(\ni)[nil,cons \cup \pi_2]}{\ni}$	 the π_2 operand of cons ∪ π_2 under the $\mathbb{1}\times\ni$ summand of F(∋), i.e. $(\mathbb{1}\times\ni)\pi_2$
=  $\frac{(\mathbb{1}+\mathbb{1}\times\ni)[nil,cons \cup \pi_2]}{\ni}$ F(X)=1+A×X — ≜ subseq	 the same operand under $\mathbb{1}+\mathbb{1}\times\ni$, whose $\mathbb{1}\times\ni$ summand it sits in
=  $\frac{[nil,(\mathbb{1}\times\ni)(cons \cup \pi_2)]}{\ni}$ R+S≜[Rl,Sr], l[R,S]=R, r[R,S]=S — (11.2a)	 the π_2 operand of the second arm $(\mathbb{1}\times\ni)(cons \cup \pi_2)$
=  $[\frac{nil}{\ni}, \frac{(\mathbb{1}\times\ni)(cons \cup \pi_2)}{\ni}]$ (11.2.1a) at T:=[nil,(1×∋)(cons ∪ π₂)]	 the π_2 operand under its $\mathbb{1}\%\ni$, the arm (12.1c)'s square rewrites, $(\mathbb{1}\times\ni)\pi_2=\pi_2\ni$
=  $[nil\ \frac{\mathbb{1}}{\ni}, \frac{(\mathbb{1}\times\ni)(cons \cup \pi_2)}{\ni}]$ (10b), $\frac{f}{\ni} = f\ \frac{\mathbb{1}}{\ni}$ for f a map, at f:=nil	 the same operand; the two rows differ only in the nil arm

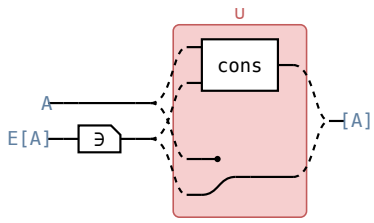
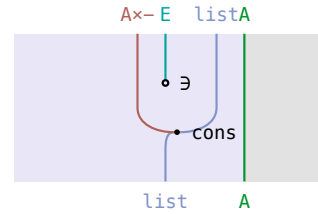
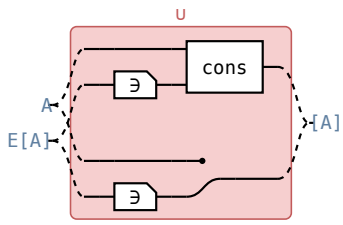
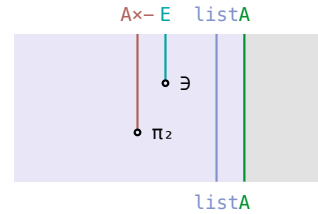
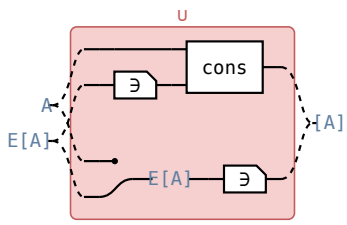
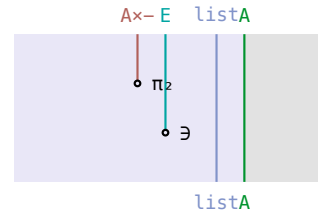
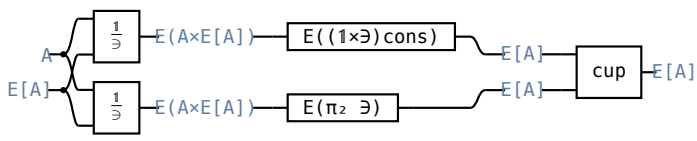
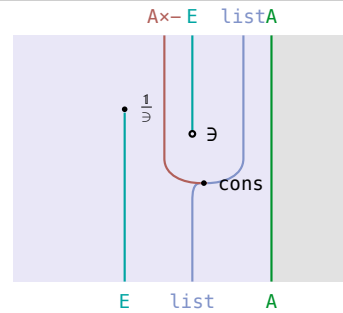
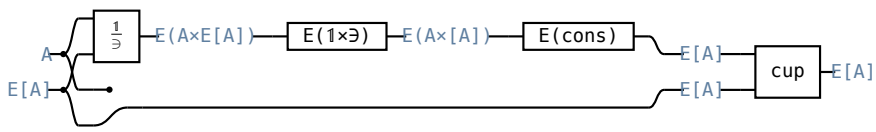
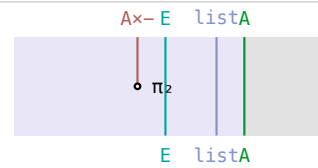
(12.1c)

$$\begin{tikzcd} A\times E[A] \arrow[r, "\pi_2"] \arrow[d, "\mathbb{1}\times\ni"'] & E[A] \arrow[d, "\ni"] \\ A\times[A] \arrow[r, "\pi_2"'] & {[A]} \end{tikzcd}$$

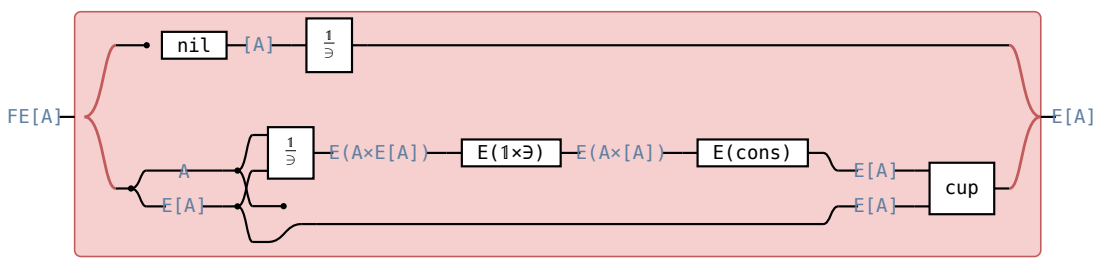
(12.1d)

$\frac{(1\times\ni)(\mathsf{cons} \cup \pi_2)}{\ni} = \langle \frac{1\times\ni}{\ni}\ \mathsf{E}(\mathsf{cons}), \pi_2 \rangle\ \mathsf{cup}$

power transpose of join: the power transpose of the join of two relations is $\langle \frac{R}{\ni}, \frac{S}{\ni} \rangle\ \mathsf{cup}$, where **cup** is the function that returns the union of two sets

circuit	Hinze–Marsden
 $\frac{(1\times\ni)(\mathsf{cons} \cup \pi_2)}{\ni}$ (12.1b)'s second branch	 the cons operand of $\mathsf{cons} \cup \pi_2$
=  $\frac{(1\times\ni)\mathsf{cons} \cup (1\times\ni)\pi_2}{\ni}$ $\mathsf{T}(X_1 \cup X_2) = \mathsf{T}X_1 \cup \mathsf{T}X_2$ — (1.2b)	 the π_2 operand of $(1\times\ni)\mathsf{cons} \cup (1\times\ni)\pi_2$
=  $\frac{(1\times\ni)\mathsf{cons} \cup \pi_2\ni}{\ni}$ $(1\times\ni)\pi_2 = \pi_2\ni$ — (11.1.1b) at π_2, an equality because 1 is entire	 the π_2 operand of $(1\times\ni)\mathsf{cons} \cup \pi_2\ni$
=  $\langle \frac{(1\times\ni)\mathsf{cons}}{\ni}, \frac{\pi_2\ni}{\ni} \rangle\ \mathsf{cup}$ $\frac{R \cup S}{\ni} = \langle \frac{R}{\ni}, \frac{S}{\ni} \rangle\ \mathsf{cup}$ — $\triangleq \mathsf{cup}$	 the cons operand under its $1\%\ni$
=  $\langle \frac{1\times\ni}{\ni}\ \mathsf{E}(\mathsf{cons}), \pi_2 \rangle\ \mathsf{cup}$ (10b), absorption $\frac{S}{\ni}\ \mathsf{E}(R) = \frac{SR}{\ni}$ at $S := 1\times\ni$, $R := \mathsf{cons}$; fusion and $\frac{\ni}{\ni} = 1$ on the π_2 operand	 the π_2 operand, bare π_2

(12.1e)



$[\frac{\mathsf{nil}}{\ni}, \langle \frac{1\times\ni}{\ni}\ \mathsf{E}(\mathsf{cons}), \pi_2 \rangle\ \mathsf{cup}]$

which writes $\mathsf{P}\mathsf{cons}$; cons is a map, and there $\mathsf{P}(\mathsf{cons}) = \mathsf{E}(\mathsf{cons})$ — (11.3d).

§13. Optimisation Problems

§13.1. est(R)≜∋∩(∈\R°)

(13.1a)

For R : A→A, est(R)≜∋∩(∈\R°) : EA→A .

xs (est(R)) x⟺x∈xs∧(∀y∈xs. x R y) the same predicate under the same letter, min R, so est(R)=min(R°) once R is an arrow;

xs (est(R)) x⟺(x in xs) and all x R\: xs in q

est(R)=∋∩all R° all R≜∈\R, q's all; the chains below keep it written ∈\

E(R)≜ $\frac{\ni R}{\ni}$: EA→EB, xs E(R) ys⟺ys={y|∃x∈xs. x R y} the image of xs, (10b)

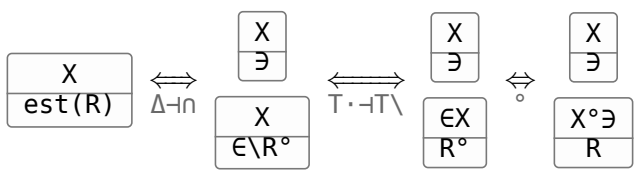
P(R) : EA→EB, xs P(R) ys⟺(∀x∈xs. ∃y∈ys. x R y)∧(∀y∈ys. ∃x∈xs. x R y) every x and every y has a partner, (11.3b)

(13.1b)

the law	what it says
X⊑est(R)⟺X⊑∋ and X°∋⊑R	in the set, and below every element of it
$\frac{1}{\ni}$ (∈\R)=R	bounding a singleton is bounding its element
$\frac{S}{\ni}$ (∈\R)=S°\R	bound S's image without building the set
union≜ $\frac{\ni\ni}{\ni}$: E(EA)→EA	flattens a set of sets
union (∈\R)=∈\(∈\R)	bound a union by bounding each member set
$\frac{1}{\ni}$ est(R)=𝟙∩R°	a singleton's minimum is its element, where R is reflexive $\frac{S}{\ni}$ est(R) at S:=𝟙
$\frac{S}{\ni}$ est(R)=S∩(S°\R°)	an S-value that points to every S-value
$\frac{S}{\ni}$ est(R)= $\frac{S}{\ni}$ est(R∩S°S)	only R between values S gives one argument counts — context
E(S) est(R)=(∋S)∩((∋S)°\R°)	the same for the image of a set $\frac{S}{\ni}$ est(R) at S:=∋S
P(f) est(R)=est(fRf°) f	shunt a function through a minimum
P(S) est(R)=(∋S)∩(∈\(SR°)) R reflexive	fusion with the power relator ∃ is the only proof here that tabulates
P(S) est(R)⊑(∋S)∩(∈\(SR°))	the half of the row above that costs nothing
P(est(R)) est(R)⊑union est(R) R transitive	a minimum in each set, then a minimum of those
P(est(R)) est(R)=P(Dom(est(R))) union est(R) R transitive	the same as an equality, once empty sets are dropped

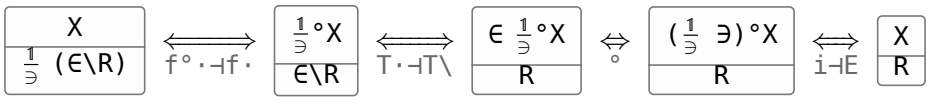
§13.1.1. X⊑est(R)⟺X⊑∋ and X°∋⊑R

(13.1.1a)



§13.1.2. $\frac{1}{\ni}$ (∈\R)=R

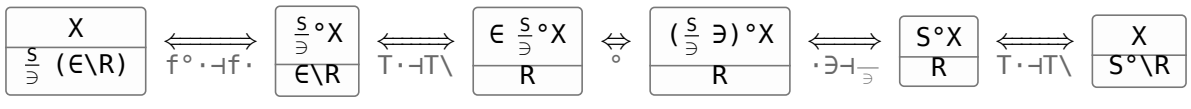
(13.1.2a)



§13.1.3. $\frac{S}{\ni}$ (∈\R)=S°\R

$\frac{S}{\ni}$ gathers the S-image of a point into one set and ∈\R asks that every member of that set be R-related to the target, so the set cancels and S°\R asks it of the S-image directly.

(13.1.3a)



§13.1.4. union (∈\R)=∈\(∈\R)

$$\begin{array}{c}X\\ \hline \mathtt{union}\ (\in\backslash R)\end{array} \underset{f^\circ\cdot\dashv f\cdot}{\Longleftrightarrow} \begin{array}{c}\mathtt{union}^\circ\ X\\ \hline \in\backslash R\end{array} \underset{T\cdot\dashv T\backslash}{\Longleftrightarrow} \begin{array}{c}\in\mathtt{union}^\circ\ X\\ \hline R\end{array} \underset{\circ}{\Leftrightarrow} \begin{array}{c}(\mathtt{union}\ni)^\circ X\\ \hline R\end{array} \underset{\cdot\ni\dashv\frac{}{\ni}}{\Longleftrightarrow} \begin{array}{c}(\ni\ni)^\circ X\\ \hline R\end{array} \underset{\circ,\ T\cdot\dashv T\backslash}{\Longleftrightarrow} \begin{array}{c}X\\ \hline \in\backslash(\in\backslash R)\end{array} \tag{13.1.4a}$$

§13.1.5. $\frac{S}{\ni}$ est(R)=S∩(S°\R°)

$$\begin{array}{c}X\\ \hline \frac{S}{\ni}\ \mathtt{est}(R)\end{array} \underset{f^\circ\cdot\dashv f\cdot}{\Longleftrightarrow} \begin{array}{c}\frac{S}{\ni}{}^\circ X\\ \hline \mathtt{est}(R)\end{array} \underset{\Delta\dashv\cap}{\Longleftrightarrow} \begin{array}{c}\frac{S}{\ni}{}^\circ X\\ \hline \ni\end{array}\ \begin{array}{c}\frac{S}{\ni}{}^\circ X\\ \hline \in\backslash R^\circ\end{array} \underset{f^\circ\cdot\dashv f\cdot}{\Longleftrightarrow} \begin{array}{c}X\\ \hline \frac{S}{\ni}\ \ni\end{array}\ \begin{array}{c}X\\ \hline \frac{S}{\ni}\ (\in\backslash R^\circ)\end{array} \underset{\cdot\ni\dashv\frac{}{\ni},\ (13.1.3a)}{\Longleftrightarrow} \begin{array}{c}X\\ \hline S\end{array}\ \begin{array}{c}X\\ \hline S^\circ\backslash R^\circ\end{array} \underset{\Delta\dashv\cap}{\Longleftrightarrow} \begin{array}{c}X\\ \hline S\cap(S^\circ\backslash R^\circ)\end{array} \tag{13.1.5a}$$

§13.1.6. $\frac{S}{\ni}$ est(R)= $\frac{S}{\ni}$ est(R∩S°S)

$$\begin{array}{c}X\\ \hline \frac{S}{\ni}\ \mathtt{est}(R\cap S^\circ S)\end{array} \underset{(13.1.5a),\ \circ}{\Longleftrightarrow} \begin{array}{c}X\\ \hline S\cap(S^\circ\backslash(R^\circ\cap S^\circ S))\end{array} \underset{\Delta\dashv\cap,\ T\cdot\dashv T\backslash}{\Longleftrightarrow} \begin{array}{c}X\\ \hline S\end{array}\ \begin{array}{c}S^\circ X\\ \hline R^\circ\cap S^\circ S\end{array} \tag{13.1.6a}$$

$$\underset{\Delta\dashv\cap,\ S^\circ\cdot\ \text{monotone}}{\Longleftrightarrow} \begin{array}{c}X\\ \hline S\end{array}\ \begin{array}{c}S^\circ X\\ \hline R^\circ\end{array} \underset{T\cdot\dashv T\backslash,\ \Delta\dashv\cap}{\Longleftrightarrow} \begin{array}{c}X\\ \hline S\cap(S^\circ\backslash R^\circ)\end{array} \underset{(13.1.5a)}{\Longleftrightarrow} \begin{array}{c}X\\ \hline \frac{S}{\ni}\ \mathtt{est}(R)\end{array}$$

§13.1.7. P(f) est(R)=est(fRf°) f

$$\begin{array}{c}\mathtt{P}(f)\ \mathtt{est}(R)\\ f\ \text{a map}\end{array} \overline{\overline{\mathsf{P{=}E}\ \text{on maps},\ (13.1.5a)}} \ (\ni f)\cap((\ni f)^\circ\backslash R^\circ) \overline{\overline{\circ,\ T\cdot\dashv T\backslash,\ f^\circ\cdot\dashv f\cdot}} \ (\ni f)\cap(\in\backslash(fR^\circ)) \tag{13.1.7a}$$

$$\overline{\overline{\text{modular law},\ f\ \text{simple}}} \ (\ni\cap((\in\backslash(fR^\circ))f^\circ))f \overline{\overline{\cdot f\dashv\cdot f^\circ,\ \circ,\ \mathtt{est}}} \ \mathtt{est}(fRf^\circ)\ f$$

§13.1.8. P(S) est(R)⊑(∋S)∩(∈\(SR°))

$$\begin{array}{c}\mathtt{P}(S)\ \mathtt{est}(R)\\ \hline (\ni S)\cap(\in\backslash(SR^\circ))\end{array} \overline{\overline{\Delta\dashv\cap,\ T\cdot\dashv T\backslash}} \begin{array}{c}\mathtt{P}(S)\ \mathtt{est}(R)\\ \hline \ni S\end{array}\ \begin{array}{c}\in\mathtt{P}(S)\ \mathtt{est}(R)\\ \hline SR^\circ\end{array} \overline{\overline{\text{UP of}\ \mathtt{est}}} \begin{array}{c}\mathtt{P}(S)\ni\\ \hline \ni S\end{array}\ \begin{array}{c}\in\mathtt{P}(S)\\ \hline S\in\end{array} \tag{13.1.8a}$$

§13.1.9. P(est(R)) est(R)⊑union est(R)

$$\begin{array}{c}\mathtt{P}(\mathtt{est}(R))\ \mathtt{est}(R)\\ \hline \mathtt{union}\ \mathtt{est}(R)\\ R\ \text{transitive}\end{array} \underset{(13.1.5a)}{\Longleftrightarrow} \begin{array}{c}\mathtt{P}(\mathtt{est}(R))\ \mathtt{est}(R)\\ \hline (\ni\ni)\cap((\ni\ni)^\circ\backslash R^\circ)\end{array} \overline{\overline{\circ,\ \Delta\dashv\cap,\ T\cdot\dashv T\backslash}} \begin{array}{c}\mathtt{P}(\mathtt{est}(R))\ \mathtt{est}(R)\\ \hline \ni\ni\end{array}\ \begin{array}{c}\in\in\mathtt{P}(\mathtt{est}(R))\\ \mathtt{est}(R)\\ \hline R^\circ\end{array} \overline{\overline{\text{UP of}\ \mathtt{est},\ R\ \text{transitive}}} \begin{array}{c}\mathtt{P}(\mathtt{est}(R))\ni\\ \hline \ni\mathtt{est}(R)\end{array}\ \begin{array}{c}\in\mathtt{P}(\mathtt{est}(R))\\ \hline \mathtt{est}(R)\in\end{array} \tag{13.1.9a}$$

, R transitive

§13.2. Lax natural transformations (LaT)

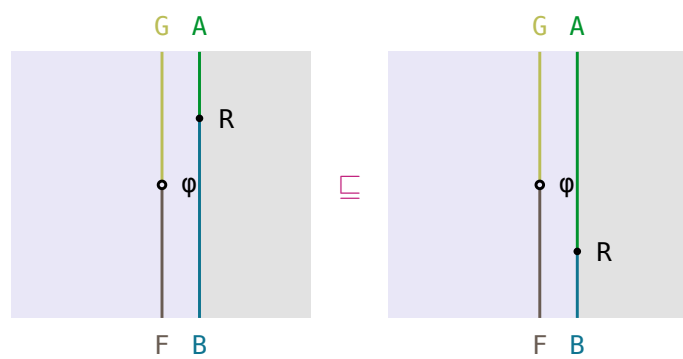
For relators $G,F : \mathcal{C}\to\mathcal{D}$ and components $\varphi_A : GA\to FA$, φ is **lax at** $R : A\to B$ when $G(R)\varphi_B\sqsubseteq\varphi_A F(R)$ — both sides $GA\to FB$, the left through the component at B, the right through the one at A. (13.2a)

$\varphi : G\Rightarrow F$ is a **lax natural transformation** (LaT) when it is lax at **every** R. (5.13)

Lax at every **map** already gives LaT, and at a map the inequation is an equality $G(f)\varphi=\varphi F(f)$: laxness is about relations only.

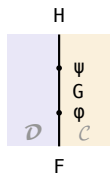
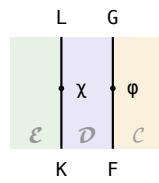
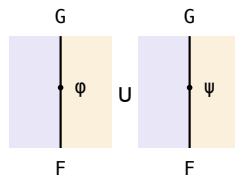
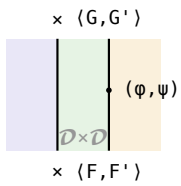
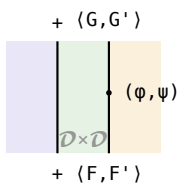
(13.2b)

$$\begin{tikzcd} GA \arrow[r, "\varphi_A"] \arrow[d, "G(R)"'] \arrow[dr, phantom, "\Leftarrow"] & FA \arrow[d, "F(R)"] \\ GB \arrow[r, "\varphi_B"'] & FB \end{tikzcd}$$



$G(R)\varphi_B\sqsubseteq\varphi_A F(R)$

(13.2c)

closed under	commutative diagram	string diagram
composition $\psi\varphi$	$\begin{tikzcd} HA \arrow[r, "\psi_A"] \arrow[d, dashed, "H(R)"'] \arrow[dr, phantom, "\Leftarrow"] & GA \arrow[r, "\varphi_A"] \arrow[d, dashed, "G(R)"'] \arrow[dr, phantom, "\Leftarrow"] & FA \arrow[d, dashed, "F(R)"'] \\ HB \arrow[r, "\psi_B"'] & GB \arrow[r, "\varphi_B"'] & FB \end{tikzcd}$ $H(R)\psi_B \sqsubseteq \psi_A G(R)$ and $G(R)\varphi_B \sqsubseteq \varphi_A F(R)$ give $H(R)(\psi_B\varphi_B) \sqsubseteq (\psi_A\varphi_A)F(R)$	
horizontal composition $\chi\circ\varphi$	$\begin{tikzcd} L(GA) \arrow[r, "\chi_{GA}K(\varphi_A)"] \arrow[d, dashed, "L(G(R))"'] \arrow[dr, phantom, "\Leftarrow"] & K(FA) \arrow[d, dashed, "K(F(R))"'] \\ L(GB) \arrow[r, "\chi_{GB}K(\varphi_B)"'] & K(FB) \end{tikzcd}$ $\varphi : G \Rightarrow F$ with $G,F : \mathcal{C} \to \mathcal{D}$ and $\chi : L \Rightarrow K$ with $L,K : \mathcal{D} \to \mathcal{E}$ give $\chi\circ\varphi : L\circ G \Rightarrow K\circ F$, the family $A \mapsto \chi_{GA}K(\varphi_A)$ $L(G(R))\chi_{GB} \sqsubseteq \chi_{GA}K(G(R))$ is χ lax at $G(R)$; then $K(G(R)\varphi_B) \sqsubseteq K(\varphi_A F(R))$ is K applied to φ's own inequation; $\chi := \mathbb{1}_K$ gives $K(\varphi) : K\circ G \Rightarrow K\circ F$, K composed on the outside; $\varphi := \mathbb{1}_G$ gives $\chi G : L\circ G \Rightarrow K\circ G$, G composed on the inside two candidates, $\chi_{GA}K(\varphi_A)$ and $L(\varphi_A)\chi_{FA}$, ordered by $\sqsubseteq$; this row is the first	
union $\varphi \cup \psi$	$\begin{tikzcd} GA \arrow[r, "\varphi_A"] \arrow[d, dashed, "G(R)"'] \arrow[dr, phantom, "\Leftarrow"] & FA \arrow[d, dashed, "F(R)"'] \\ GB \arrow[r, "\varphi_B"'] & FB \end{tikzcd} \cup \begin{tikzcd} GA \arrow[r, "\psi_A"] \arrow[d, dashed, "G(R)"'] \arrow[dr, phantom, "\Leftarrow"] & FA \arrow[d, dashed, "F(R)"'] \\ GB \arrow[r, "\psi_B"'] & FB \end{tikzcd}$ $G(R)\varphi_B \sqsubseteq \varphi_A F(R)$ and $G(R)\psi_B \sqsubseteq \psi_A F(R)$ give $G(R)(\varphi_B \cup \psi_B) \sqsubseteq (\varphi_A \cup \psi_A)F(R)$	
a relator K $K(\varphi)$	$\begin{tikzcd} K(GA) \arrow[r, "K(\varphi_A)"] \arrow[d, dashed, "K(G(R))"'] \arrow[dr, phantom, "\Leftarrow"] & K(FA) \arrow[d, dashed, "K(F(R))"'] \\ K(GB) \arrow[r, "K(\varphi_B)"'] & K(FB) \end{tikzcd}$ $G(R)\varphi_B \sqsubseteq \varphi_A F(R)$ gives $K(G(R))K(\varphi_B) \sqsubseteq K(\varphi_A)K(F(R))$	
product $\varphi\times\psi$	$\begin{tikzcd} GA\times G'A \arrow[r, "\varphi_A\times\psi_A"] \arrow[d, dashed, "G(R)\times G'(R)"'] \arrow[dr, phantom, "\Leftarrow"] & FA\times F'A \arrow[d, dashed, "F(R)\times F'(R)"'] \\ GB\times G'B \arrow[r, "\varphi_B\times\psi_B"'] & FB\times F'B \end{tikzcd}$ $\varphi : G \Rightarrow F$ and $\psi : G' \Rightarrow F'$ give $\varphi\times\psi : G\times G' \Rightarrow F\times F'$ $(R\times S)(U\times V) = (RU)\times(SV)$ and monotonicity in both slots — the row above's K applied to the inequation, run on a bifunctor; the fork is the derived case $\langle\varphi,\psi\rangle = \lhd(\varphi\times\psi)$, and its ONE cost is the lax copy law $R\lhd \sqsubseteq \lhd(R\times R)$ — (2b)	
coproduct $\varphi+\psi$	$\begin{tikzcd} GA+G'A \arrow[r, "\varphi_A+\psi_A"] \arrow[d, dashed, "G(R)+G'(R)"'] \arrow[dr, phantom, "\Leftarrow"] & FA+F'A \arrow[d, dashed, "F(R)+F'(R)"'] \\ GB+G'B \arrow[r, "\varphi_B+\psi_B"'] & FB+F'B \end{tikzcd}$ $\varphi : G \Rightarrow F$ and $\psi : G' \Rightarrow F'$ give $\varphi+\psi : G+G' \Rightarrow F+F'$ $(R+S)(U+V) = (RU)+(SV)$ and monotonicity in both slots; the co-fork is the derived case $[\varphi,\psi] = (\varphi+\psi)\triangledown$, and $\triangledown$ costs nothing	
meet — **fails** $\varphi\cap\psi$	the step it would need is $(\varphi_A F(R)) \cap (\psi_A F(R)) \sqsubseteq (\varphi\cap\psi)_A F(R)$, the wrong direction of (4c) a counterexample: (13.2.1a)	

Each row is the general slide $R_A X \sqsubseteq X' R_B$ and $R_B Y \sqsubseteq Y' R_C$ give $R_A(XY) \sqsubseteq (X'Y')R_C$ at $R_A, R_B := G(R), F(R)$ and $X, X' := \varphi_B, \varphi_A$, quantified over R; at $X' = X$ with R_A, R_B endorelations it is the monotonicity reading instead — the $\forall R$ sits outside the law, and is the only difference

§13.2.1. meet is not closed in LaT

counterexample — $\varphi,\psi : G\Rightarrow F$ lax natural does NOT give $\varphi\cap\psi$ lax natural (13.2.1a)

$$\begin{tikzcd}
(0,1) & & \varnothing \\
A\times A \arrow[rr, "\pi_1\cap\pi_2"] \arrow[d, "R\times R"'] \arrow[drr, phantom, "\not\Downarrow"] & & A \arrow[d, "R"] \\
B\times B \arrow[rr, "\pi_1\cap\pi_2"'] & & B \\
(0,0) & & \{0\} \not\subseteq \varnothing
\end{tikzcd}$$

$A=B\triangleq\{0,1\}$, $R\triangleq\{(0,0),(1,0)\}$, $\varphi\triangleq\pi_1\cap\pi_2 : \Delta\Rightarrow Id$

$\pi_1,\pi_2 : \Delta\Rightarrow Id$ are both LaTs (13.4.3c)'s g row and $\pi_1\cap\pi_2=\{((x,x),x)\}$

§13.2.2. Two stacked towers

(13.2.2a)

	lower — inside ONE allegory	**upper — between allegories**
0-cell	an object A	an allegory $\mathcal{C}$ — **a whole lower tower**
1-cell	a relation $R : A\to B$	a relator $F : \mathcal{C}\to\mathcal{D}$
2-cell	$R\sqsubseteq S$	a LaT $\varphi : G\Rightarrow F$ $\triangleq$ lax
order	$R\sqsubseteq S$	$\varphi\sqsubseteq\psi$ componentwise
union	$R \cup S$	$\varphi \cup \psi$, **survives** (13.2c)
product	$R\times S$ $\triangleq \times$	$\varphi\times\psi$, **survives**; a TENSOR, not a categorical product; SYMMETRIC (11.1b) (13.2c)
coproduct	$R+S$	$\varphi+\psi$, **survives**; a BIPRODUCT (13.2c)
meet	$R\cap S$	$\varphi\cap\psi$ componentwise, **fails** (13.2.1a)
converse	R°	φ°, **fails**, oplax: $\varphi_A{}^\circ G(R)\sqsubseteq F(R)\varphi_B{}^\circ$
zero object	z with $\mathbb{1}\ z=\mathbb{0}$	the constant relator at such a z, initial and terminal

§13.2.3. $R\sqsubseteq S$ is a special case of LaT

(13.2.3a)

$$\begin{tikzcd}
GX \arrow[r, "\varphi_X"] \arrow[d, dotted, "G(R)"'] \arrow[dr, phantom, "\Downarrow"] & FX \arrow[d, dotted, "F(R)"'] \\
GY \arrow[r, "\varphi_Y"'] & FY
\end{tikzcd}
\Longrightarrow
\begin{tikzcd}
A \arrow[r, "\varphi_X"] \arrow[d, dotted, "\mathbb{1}_A"'] \arrow[dr, phantom, "\Downarrow"] & B \arrow[d, dotted, "\mathbb{1}_B"'] \\
A \arrow[r, "\varphi_Y"'] & B
\end{tikzcd}
\Longrightarrow
\begin{tikzcd}
A \arrow[r, bend left, "\varphi_X"] \arrow[r, bend right, "\varphi_Y"'] \arrow[r, phantom, "\Downarrow"] & B
\end{tikzcd}$$

$G\triangleq const\ A$, $F\triangleq const\ B$ — the relators $X\mapsto A$ and $X\mapsto B$, each sending every arrow to $\mathbb{1}$ $\triangleq$ lax

with $\mathcal{C}$ two objects and one non-identity arrow $X\to Y$, a LaT $const\ A\Rightarrow const\ B$ is the pair $\varphi_X,\varphi_Y : A\to B$ with $\varphi_Y\sqsubseteq\varphi_X$ — the datum $R\sqsubseteq S$

not every LaT is constant: $\ni : P\Rightarrow Id$ (11.3d) $\pi_2 : (A\times-)\Rightarrow Id$ (12.1c) $\lhd : Id\Rightarrow\Delta$ (2b)

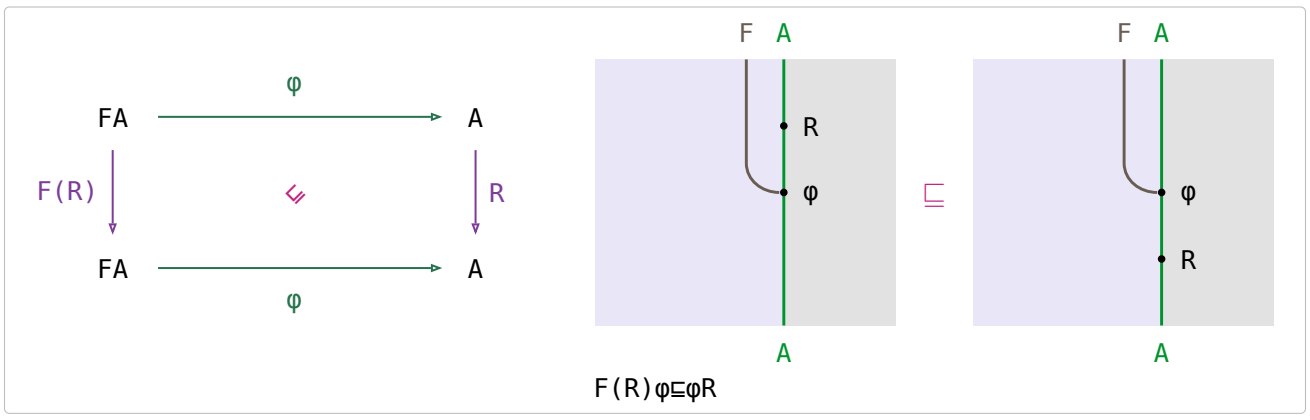
§13.3. Monotonic algebras

An F-algebra $\varphi : FA \to A$ is **monotonic on** $R : A \to A$ when it is lax at R at $G:=F$, $F:=Id$: $F(R)\varphi \sqsubseteq \varphi R$. R is an **endorelation**, so $B=A$ and the two components φ_A, φ_B are the one arrow φ — an algebra, not a family. (13.3a)

For a map $f : FA \to A$ that is $f^\circ F(R) f \sqsubseteq R$ (1.1a)'s $f^\circ \cdot \dashv f \cdot$ at $X:=F(R)f$, $Y:=R$, equivalently $F(R) \sqsubseteq f R f^\circ \cdot f \dashv \cdot f^\circ$ then $f^\circ \cdot \dashv f \cdot$, .

$(\le \times \le) + \sqsubseteq + \le$ — addition on Nat is monotonic on $\le$, which at the point level reads $c=a+b \wedge a \le a' \wedge b \le b' \Longrightarrow c \le a'+b'$.

(13.3b)

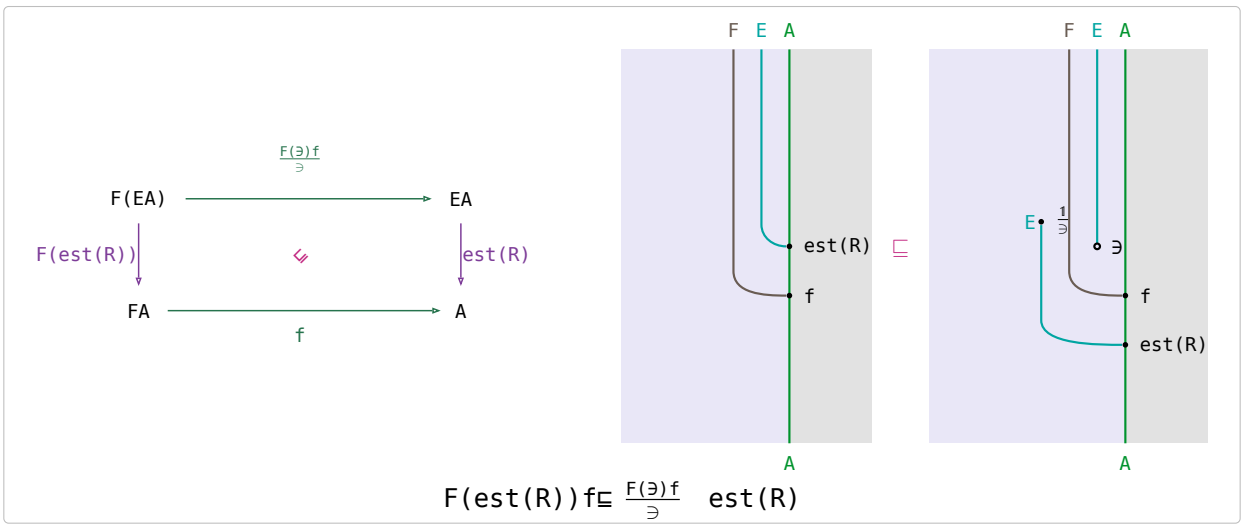


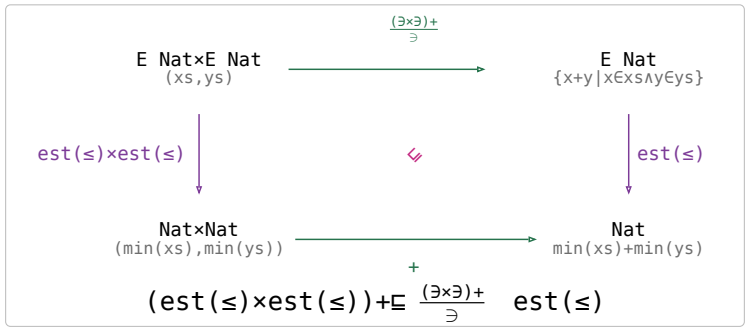
§13.3.1. Function f is monotonic on R iff it distributes over R

$f : FA \to A$ **distributes over** R if $F(est(R))f \sqsubseteq \frac{F(\ni)f}{\ni} est(R)$. (13.3.1a)

$+$ distributes over $\le$, at the point level $min(xs)+min(ys)=min\{x+y \mid x \in xs \wedge y \in ys\}$ for xs, ys non-empty and $min \triangleq est(\le)$.

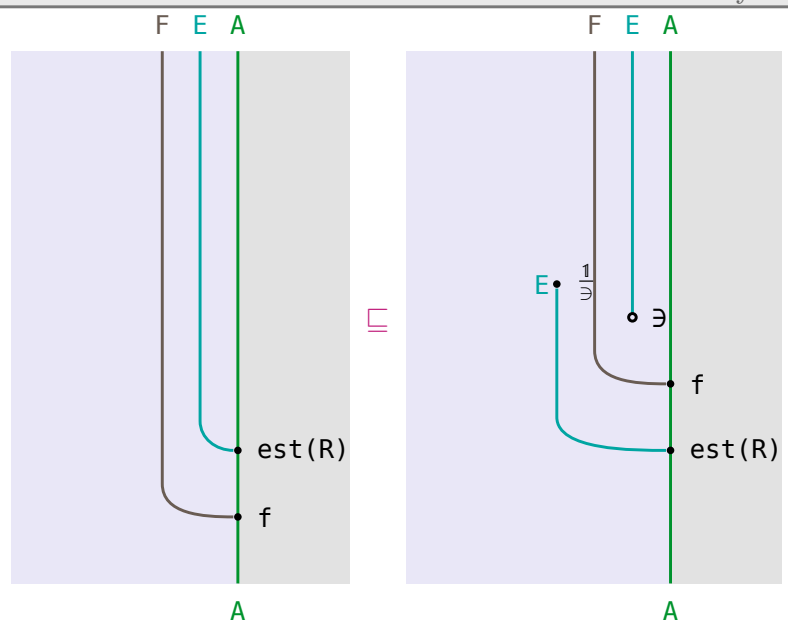
(13.3.1b)



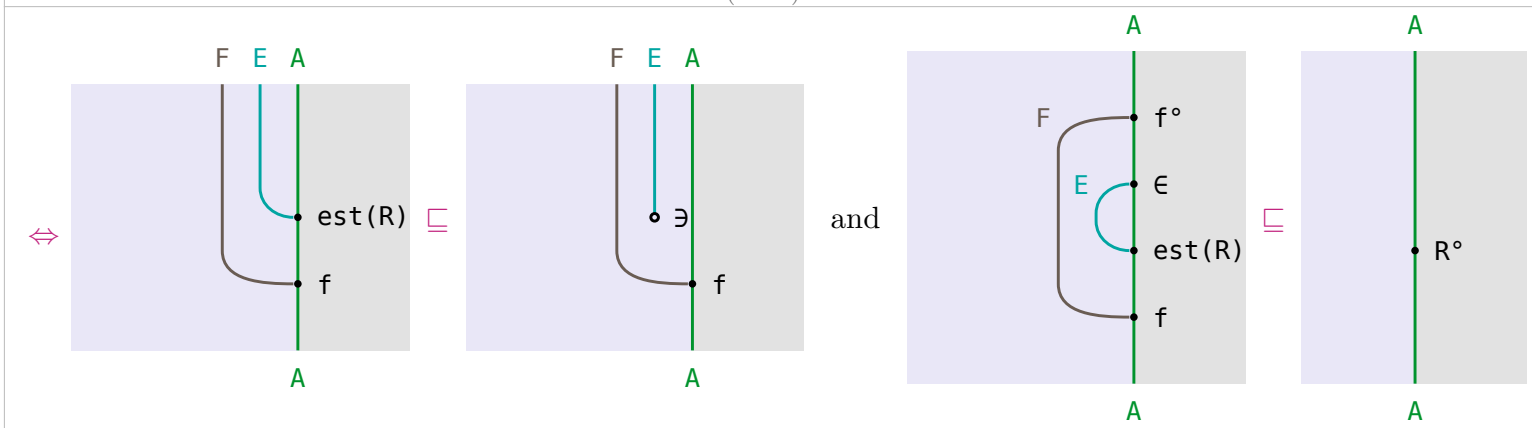


$$f°F(R)f⊑R \Longleftrightarrow F(est(R))f⊑ \frac{F(∋)f}{∋} \quad est(R)$$ (13.3.1c)

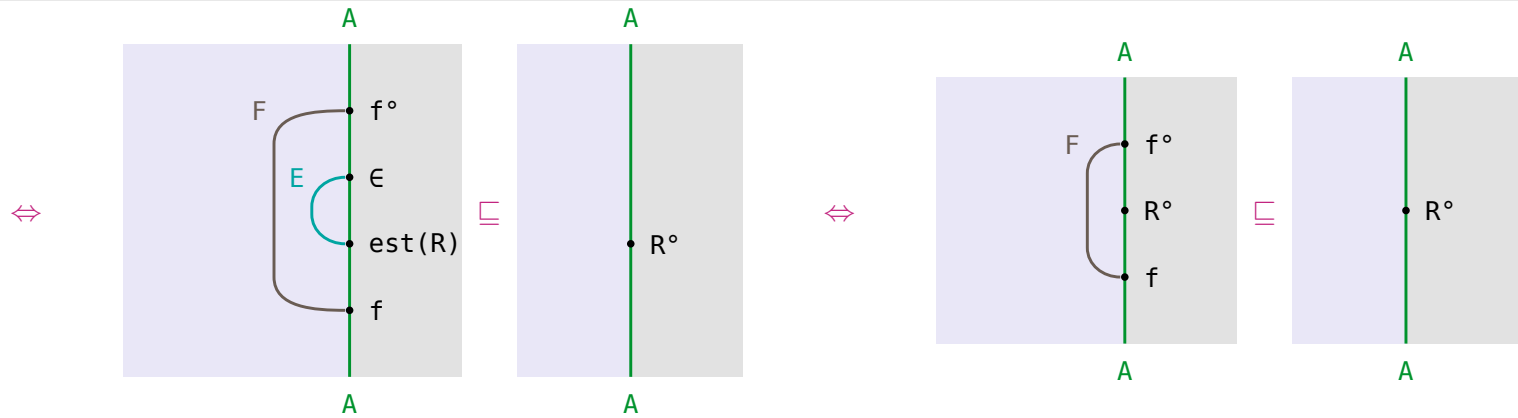
function f is monotonic over R if and only if it distributes over R; f a map, R reflexive



f distributes over R — ≜ distributes — the fraction bent as (10.1a)

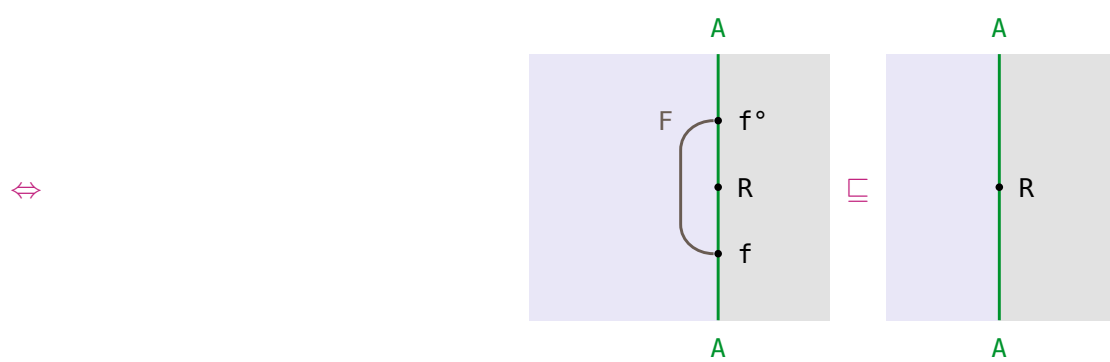


(13.1.5a) splits the bound in two, (8.1c) moving (F(∋)f)° across



est(R)⊑∋ — ≜ est — so the first conjunct drops

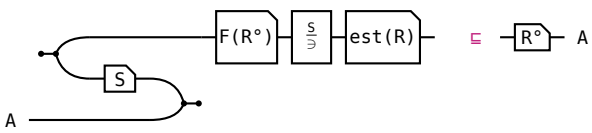
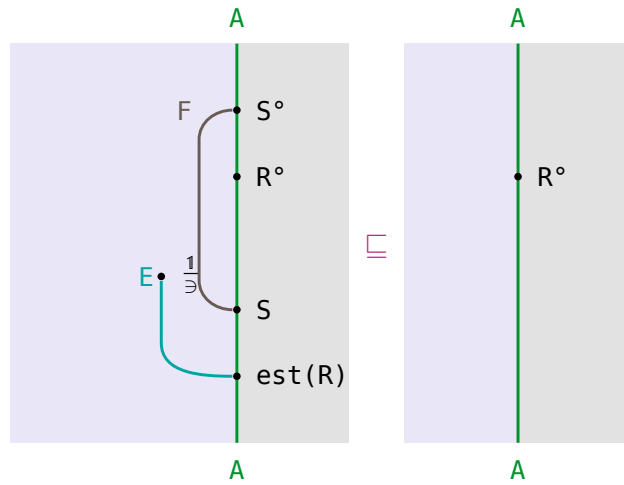
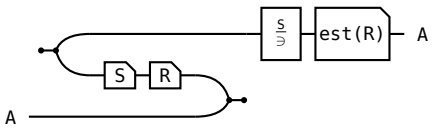
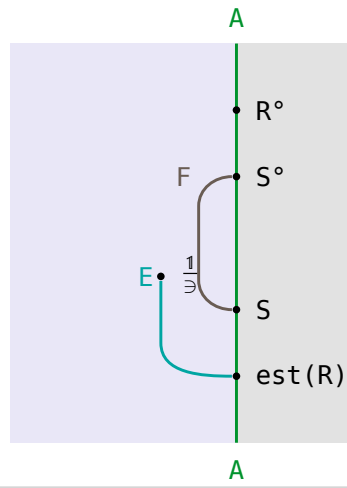
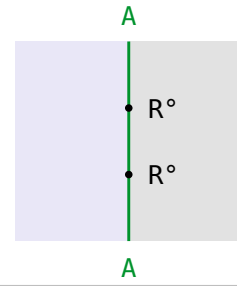
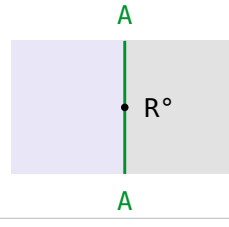
(F(∋)f)°=f°F(∈) — ≜ ° — and ∈ est(R)=R° — ≜ est, R reflexive



both sides conversed — F(R°)°=F(R), (11b)

§13.3.2. Greedy Theorem: ⦇$\frac{S}{\ni}$ est(R)⦈⊑⦇$\frac{S}{\ni}$⦈ est(R), given S monotoic on R, F preserving °, and R transitive

(13.3.2a)

circuit	Hinze–Marsden
⦇$\frac{S}{\ni}$ est(R)⦈⊑⦇$\frac{S}{\ni}$⦈ est(R) the conclusion: one minimum kept at each step is below every result collected and one minimum taken at the end	
⦇$\frac{S}{\ni}$ est(R)⦈ ⊑ ⦇S⦈ ⇔ ⦇S⦈°⦇$\frac{S}{\ni}$ est(R)⦈ ⊑ R° and (13.1.5a) at ⦇S⦈; ⦇$\frac{S}{\ni}$ est(R)⦈⊑⦇S⦈°\R°⟺⦇S⦈°⦇$\frac{S}{\ni}$ est(R)⦈⊑R° — (8.1c)	
⇐ $\frac{S}{\ni}$ est(R)⊑S the left conjunct; (13.1.5a), and ⦇−⦈ is monotone	
 ⇐ the right conjunct; ⦇S⦈°⦇$\frac{S}{\ni}$ est(R)⦈ is the least X with X=S°F(X)($\frac{S}{\ni}$ est(R)) hylomorphism theorem — so Knaster–Tarski leaves this one inequation $\frac{S}{\ni}$ = $\frac{1}{\ni}$ E(S) — (10.1a)	
 ⊑ S°F(R°)⊑R°S° — ≜ monotonic at S, conserved; F(R)°=F(R°) — (11b)	
 ⊑ S°($\frac{S}{\ni}$ est(R))⊑S°(S°\R°)⊑R° — (13.1.5a), (8.1c)	
 ⊑ R transitive	

§13.3.3. takewhile(p)=⦇[nil,(π₁p→cons,⊸ nil)]⦈

(13.3.3a)

name	definition	type	example	in words
F	FX=1+A×X	$\mathcal{A}\to\mathcal{A}$		nothing, or a head and a tail
nil, cons	[A]::=nil|cons(A,[A]) ≜ subseq	1→[A], A×[A]→[A]	cons(3, [1,2])=[3,1,2]	the empty list; a head onto a tail
α	[nil,cons]	F([A])→[A]		both constructors as one map
p	a coreflexive	A→A	p≜even — 2 p 2, and 3∉Dom(p)	{(a,a)|a passes the test}
R	length≤length°, a preorder	[A]→[A]	[1] R [1,2]	xs R ys⟺length(xs)≤length(ys)
⊸ nil	the constant nil — the second nil of prefix	A×[A]→[A]	(⊸ nil)(3, [1,2])=nil	drop the pair, return nil
prefix	⦇[nil,⊸ nil ∪ cons]⦈ ≜ subseq	[A]→[A]	[3,1,2] prefix [3,1]	xs prefix ys⟺∃zs. xs=ys#zs — at each cons, stop or keep the head
S	[nil,⊸ nil ∪ (p×1) cons]	F([A])→[A]	(4,[2]) S [4,2], and (3,[2]) S nil only	prefix's algebra with one extra p — stop, or keep a head that passes p
(g→X,Y)	X where g is defined and Y where it is not	A→B at X,Y : A→B		the test picks the branch

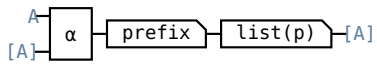
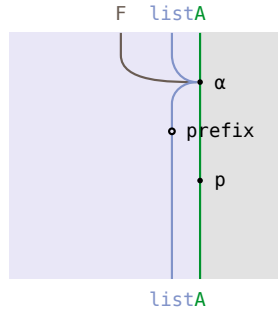
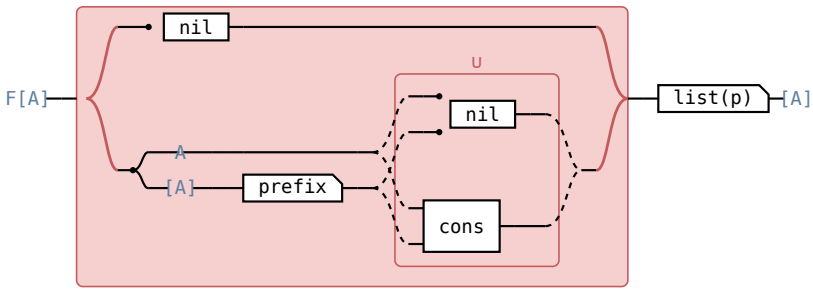
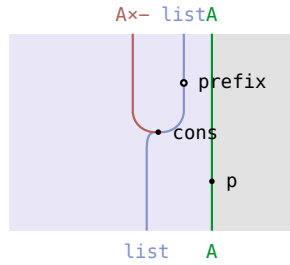
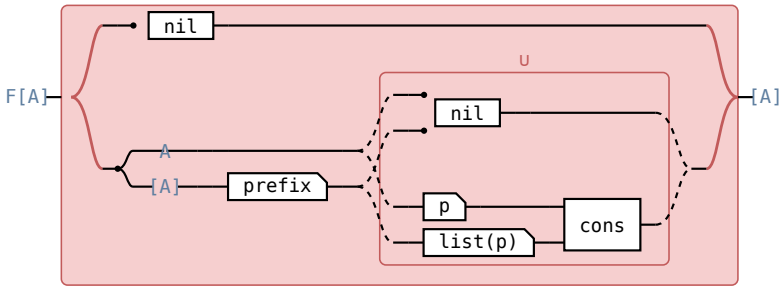
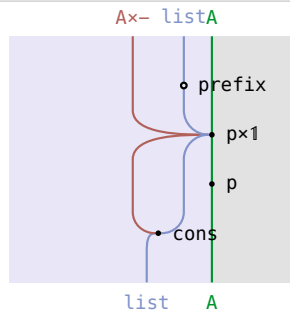
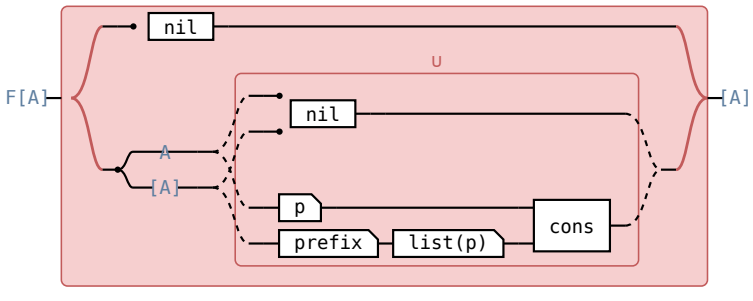
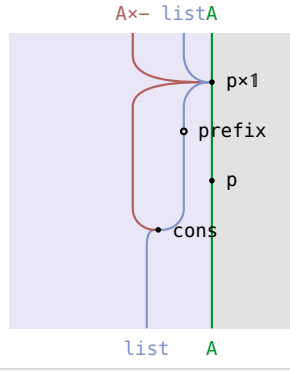
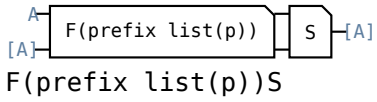
nil R=⊤, nil R°=nil nil is the shortest list — below every list, and above only itself, so it loses every est(R°)

(3.3.3b)

α prefix list(p)=F(prefix list(p))S

building the list and then keeping a **p**-passing prefix of it is keeping one of the tail first, and then building with **S**

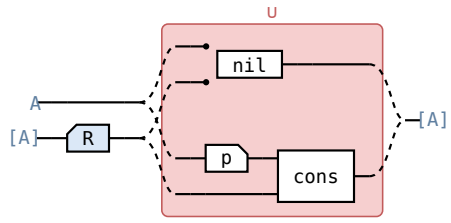
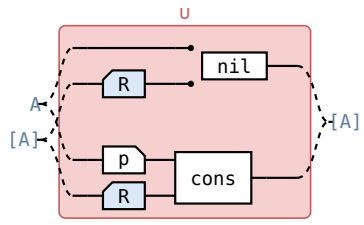
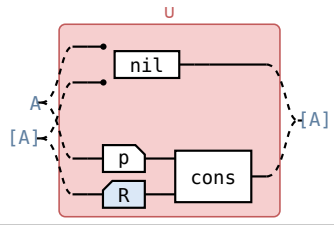
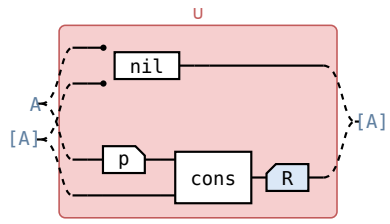
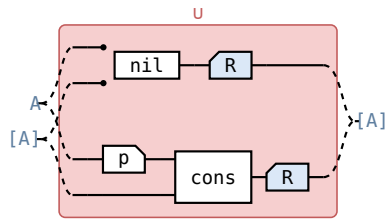
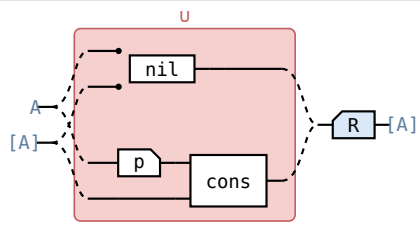
this same diagram is **subseq**'s: algebra **[nil,π₂ ∪ cons]**, type **[A]→[A]**

circuit — the fork is F([A])=1+A×[A]: nil above, the pair below	**Hinze–Marsden**
 α prefix list(p)	 the cons branch alone, without 1+ or ⊸ nil
=  F(prefix) [nil,⊸ nil ∪ cons] list(p) defining equation	 the cons operand of ⊸ nil ∪ cons
=  F(prefix) [nil,⊸ nil ∪ (p×list(p)) cons] list(p) through cons	 the (p×list(p)) cons operand of ⊸ nil ∪ (p×list(p)) cons
=  [nil,⊸ nil ∪ (p×(prefix list(p))) cons] relator, prefix entire	
=  F(prefix list(p))S prefix list(p) entire	

≙ reduce reads that off as prefix list(p)=⦇S⦈. fusion cannot: list(p) is not entire, (𝟙×list(p))⊸ nil⊏⊸ nil, and no algebra meets the side condition. ,

(𝟙×R°)(⊸ nil ∪ (p×𝟙) cons)⊑(⊸ nil ∪ (p×𝟙) cons)R° (3.3.3c)

shortening the tail and then taking the step lands inside taking the step and then shortening the result

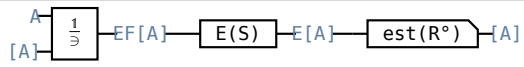
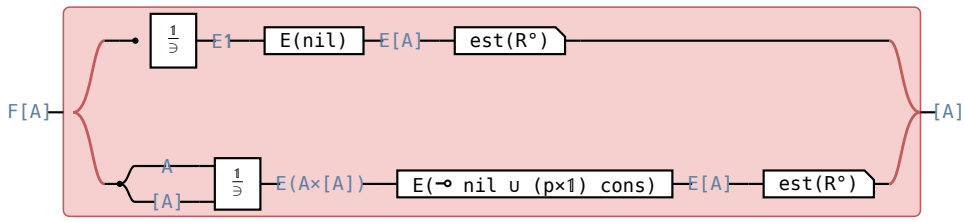
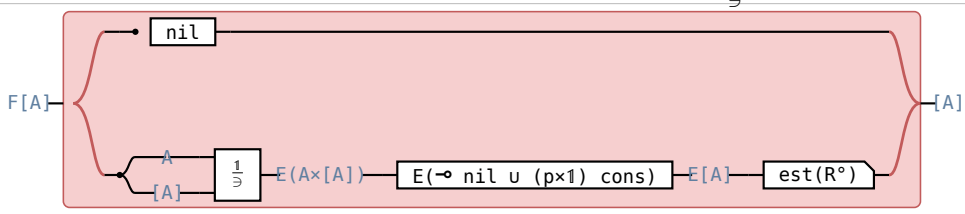
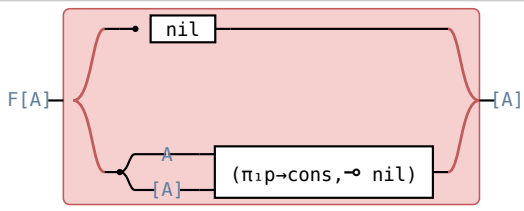
circuit — the cons branch of F(R°)S⊑SR°	**reason**
 (𝟙×R°)(⊸ nil ∪ (p×𝟙) cons)	
=  (𝟙×R°)⊸ nil ∪ (p×R°) cons	each operand is reached on its own (1.1a) — and (𝟙×R°)(p×𝟙) is p and R° on the pair's two strands at once
⊑  ⊸ nil ∪ (p×R°) cons	⊸ is the greatest arrow into 𝟙, so (𝟙×R°)⊸⊑⊸
⊑  ⊸ nil ∪ (p×𝟙) cons R°	cons length=(𝟙×length)π₂ succ with succ monotone — a shorter tail makes a shorter list
=  ⊸ nil R° ∪ (p×𝟙) cons R°	nil R°=nil ≜ takewhile — so the constant branch may carry the R° the other one already has
=  (⊸ nil ∪ (p×𝟙) cons)R°	one R° past the join is the two inside it (1.1a)

the nil branch, which no row above draws, is nil⊑nil R°.

(13.3.3d)

$\frac{S}{\ni}$ `est(R°)=[nil,(π₁p→cons,-∘ nil)]`

the longest of the lists the algebra allows is the `cons` where the head passes `p`, and `nil` where it does not

formula	reason
	
=  $[\frac{nil}{\ni}\ est(R°),\ \frac{-\circ\ nil\ \cup\ (p\times\mathbb{1})\ cons}{\ni}\ est(R°)]$	coproduct of maps
=  $[nil,\ \frac{-\circ\ nil\ \cup\ (p\times\mathbb{1})\ cons}{\ni}\ est(R°)]$	singleton, `R°` reflexive
= 	`nil R=T`

the set is `{nil}` where `p` fails on the head and `{nil,cons(a,xs)}` where it holds, and `nil` loses the second — ≜ `est` at a two-element set.

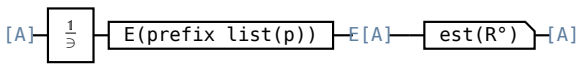
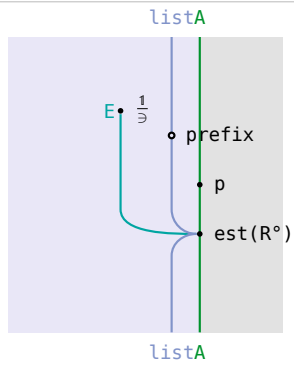
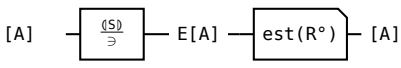
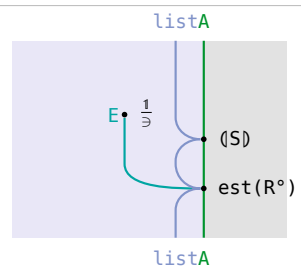
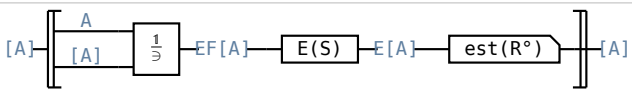
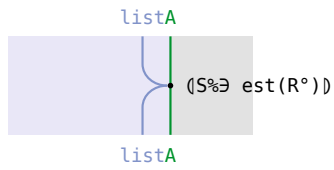
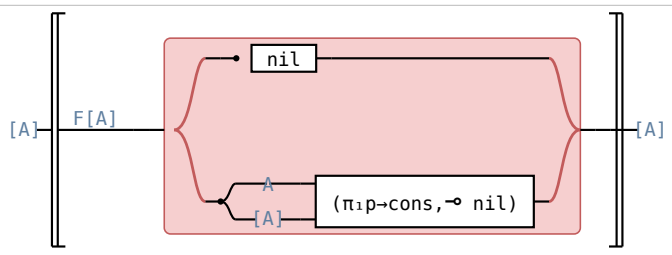
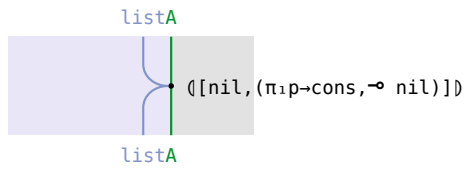
(13.3.3e)

name	definition	type	example	in words
`S`	`[nil,-∘ nil ∪ (p×𝟙) cons]` ≜ `takewhile`	`F([A])→[A]`	`(4,[2]) S [4,2]`, and `(3,[2]) S nil` only	`prefix`'s algebra with one extra `p` — stop, or keep a head that passes `p`

(13.3.3f)

takewhile(p)≜ $\frac{\text{prefix list(p)}}{\ni}$ est(R°)=⦇[nil,(π₁p→cons,⊸ nil)]⦈

takewhile: **takewhile(p)(x)** returns the longest prefix of **x** with the property that all its elements satisfy **p**; the catamorphism is the standard implementation.

circuit	Hinze–Marsden
 the specification — ≜ est's est(R°).	
=  (13.3.3b)	
⊒  greedy theorem at R°, with F(R°)S⊑SR° — (13.3.3c) — for its hypothesis: one longest p-prefix kept at each cons, instead of every p-prefix collected and one chosen at the end.	
=  (13.3.3d)	
takewhile(p)° takewhile(p)⊑prefix° prefix∩R∩R°⊑𝟙 takewhile(p)⊑prefix list(p) and (prefix list(p))° takewhile(p)⊑R — (13.1.5a) at est(R°) — and two prefixes of one list of equal length are equal, so takewhile(p) is simple: **the** longest, not **a** longest.	
$\frac{\text{prefix list(p)}}{\ni}$ est(R°) entire nil is always a p-prefix and R is connected on the prefixes of one list, so the longest exists.	
X⊑Y, X entire, Y simple ⟹X=Y ⦇[nil,(π₁p→cons,⊸ nil)]⦈ is a reduce of maps, hence entire — what turns the ⊒ above into the heading's =.	

§13.3.4. mss=⦇[zero⟨𝟙,$\frac{1}{\ni}$⟩,⟨(𝟙×π₁)⊕,⟨(𝟙×π₁)⊕ $\frac{1}{\ni}$,π₂π₂⟩ cup⟩]⦈ π₂ est(≥)

(13.3.4a)

FX=𝟙+A×X, A:=Int, α≜[nil,cons], sum=⦇[zero,plus]⦈ and segment=suffix prefix from (7a) and ≜ subseq.

head≜cons° π₁, wrap≜⟨𝟙,⊸ nil⟩ cons the head of a list and the one-element list, beside ≜ subseq's tail≜cons° π₂

⊕≜ $\frac{\text{⊸ zero ∪ plus}}{\ni}$ est(≥) the set at (a,b) is {0,a+b}, so ⊕ is the larger of the two,

(3.3.4b)

$\frac{\text{segment sum}}{\ni}$ est(≥)=$\frac{\text{suffix}}{\ni}$ E($\frac{\text{prefix sum}}{\ni}$ est(≥)) est(≥)

maximum segment sum problem: using **segment=suffix prefix**, the specification is expressed in this form

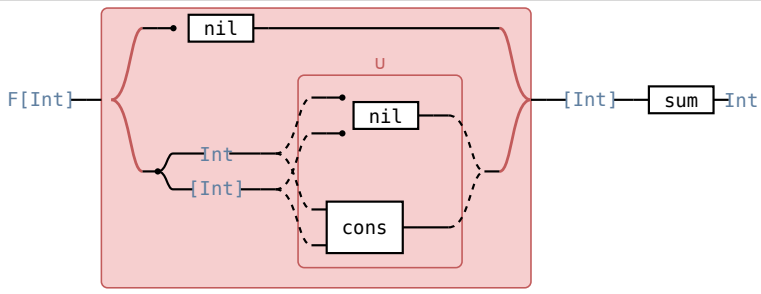
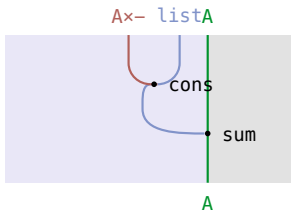
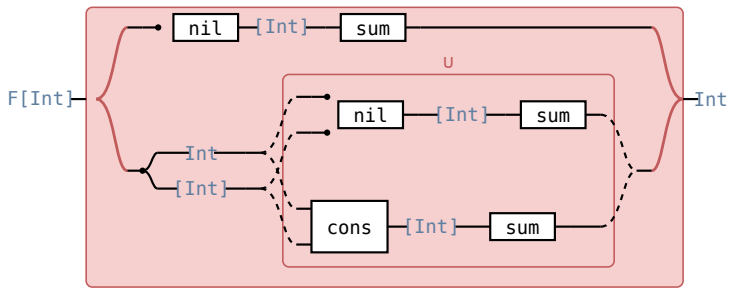
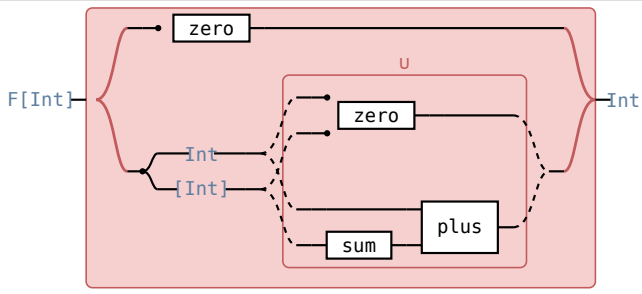
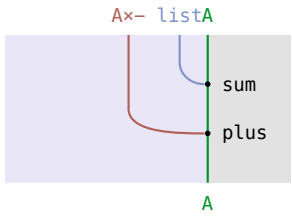
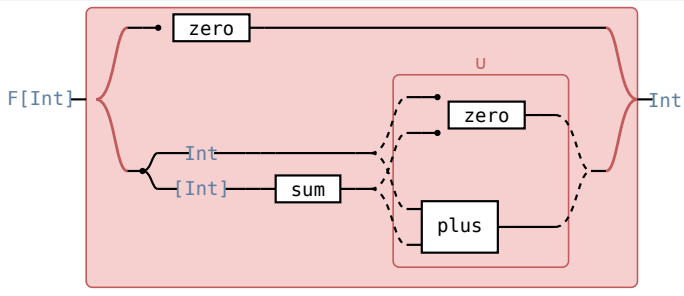
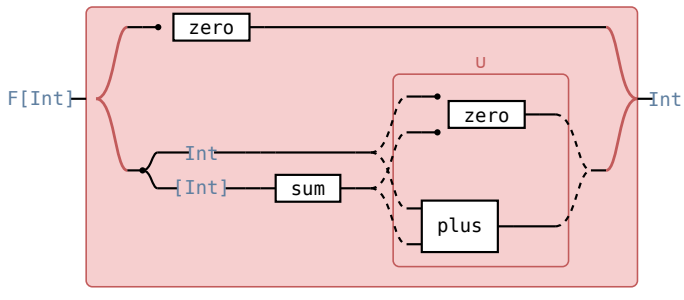
formula — one wire from [A] to A, its type written along it	**reason**
· [A] · $\boxed{\frac{\text{segment sum}}{\ni}}$ · EA · $\boxed{\text{est(R°)}}$ · A ·	
= · [A] · $\boxed{\frac{\text{suffix (prefix sum)}}{\ni}}$ · EA · $\boxed{\text{est(R°)}}$ · A ·	segment=suffix prefix ≜ subseq, ≜ mss
= · [A] · $\boxed{\frac{\text{suffix}}{\ni}}$ — E[A] — $\boxed{\text{E(prefix sum)}}$ · EA · $\boxed{\text{est(R°)}}$ · A ·	absorption (10b) — frac(S,∋) E(R)=frac(SR,∋) at S:=suffix, R:=prefix sum
= · [A] · $\boxed{\frac{\text{suffix}}{\ni}}$ — E[A] — $\boxed{\text{E}(\frac{\text{prefix sum}}{\ni})}$ · E(EA) · $\boxed{\text{union}}$ · EA · $\boxed{\text{est(R°)}}$ · A ·	$\frac{R}{\ni}$ ∋=R, union=E(∋) (10b)'s frac(R,∋)∋=R at R:=prefix sum and E(R)≜frac(∋R,∋); (13.1b)'s union≜frac(∋∋,∋); the middle equality is ≜ relator's F(RS)=F(R)F(S) at F:=E
= · [A] · $\boxed{\frac{\text{suffix}}{\ni}}$ — E[A] — $\boxed{\text{E}(\frac{\text{prefix sum}}{\ni})}$ · E(EA) · $\boxed{\text{E(est(≥))}}$ · EA · $\boxed{\text{est(R°)}}$ · A ·	(13.1b), the sets non-empty
= · [A] · $\boxed{\frac{\text{suffix}}{\ni}}$ — E[A] — $\boxed{\text{E}(\frac{\text{prefix sum}}{\ni}\ \text{est(≥)})}$ · EA · $\boxed{\text{est(R°)}}$ · A ·	relator ≜ relator — F(RS)=F(R)F(S) at F:=E

The union step is (13.1b)'s P(est(R)) est(R)=P(Dom(est(R))) union est(R) — every suffix has the empty prefix, so Dom is 𝟙 here — and P(f)=E(f) at the map it is applied to ((11.3d)).

(13.3.4c)

[nil,⊸ nil ∪ cons] sum=F(sum)[zero,⊸ zero ∪ plus]

fusion: **prefix** expressed as a catamorphism on cons-lists, ⦇[nil,⊸ nil ∪ cons]⦈, and this is the fusion condition that expresses **prefix sum** as a catamorphism

circuit — the fork is the bracket's case split F([A])=𝟙+A×[A]: nil above, the pair and its ∪ below	**Hinze–Marsden**
 [nil,⊸ nil ∪ cons] sum	 the cons operand of ⊸ nil ∪ cons
=  [nil sum,⊸ nil sum ∪ cons sum] coproduct of maps, composition over ∪	
=  [zero,⊸ zero ∪ (𝟙×sum) plus] sum's defining equation	 the (𝟙×sum) plus operand of ⊸ zero ∪ (𝟙×sum) plus
=  [zero,(𝟙×sum)(⊸ zero ∪ plus)] (𝟙×sum)⊸=⊸, sum entire	
=  F(sum) [zero,⊸ zero ∪ plus] relator	

fusion at α_B :=[nil,⊸ nil ∪ cons], S:=sum: the side condition, so prefix sum=⦇[zero,⊸ zero ∪ plus]⦈. prefix is the reduce, sum the map fused into it — the intermediate list is gone.

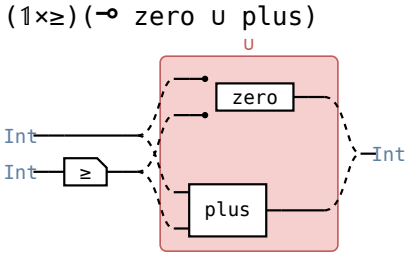
(13.3.4d)

(𝟙×≥)(⊸ zero ∪ plus)⊑(⊸ zero ∪ plus)≥

monotonic algebra: an F-algebra S is monotonic on a relation R if F(R)S⊑SR — the plus branch of F(≥)S⊑S≥; the zero branch is zero⊑zero≥

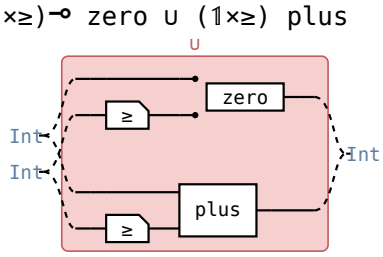
circuit — the head above, the running sum below; the tape is the ∪

(𝟙×≥)(⊸ zero ∪ plus)



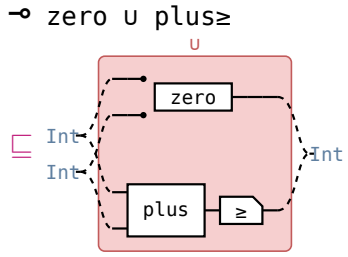
=

(𝟙×≥)⊸ zero ∪ (𝟙×≥) plus



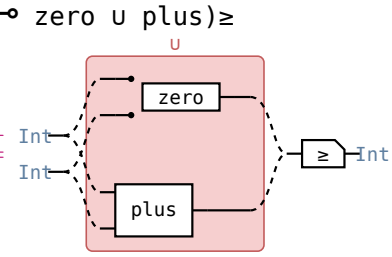
relator, composition over ∪

⊸ zero ∪ plus≥



(5.1a), (≥×≥) plus⊑plus≥; ≥ reflexive (≤×≤) plus⊑plus≤ is ≜ monotonic, written + there, and plus is a map, so it is monotonic on an order and on its opposite together, which carries it to ≥.

(⊸ zero ∪ plus)≥

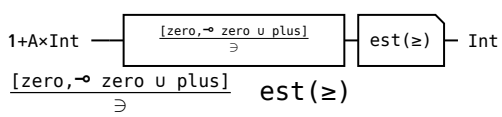


(13.3.4e)

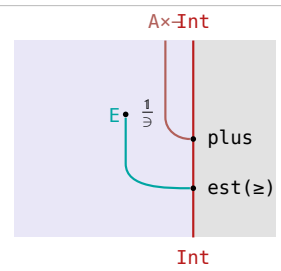
$\frac{\texttt{[zero,⊸ zero ∪ plus]}}{\ni}$ est(≥)=[zero,⊕]

the largest sum the algebra offers is zero from nothing and, from a head and a running sum, the larger of zero and the head added to it

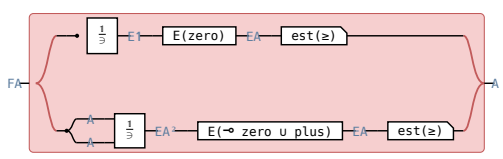
circuit — the tape is the coproduct: zero's branch above, plus's below



Hinze–Marsden

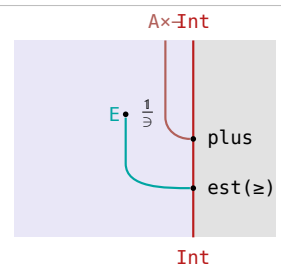


the zero arm of the bracket



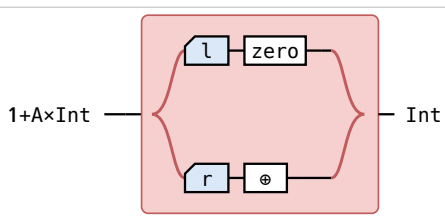
= $[\frac{\texttt{zero}}{\ni}\ \texttt{est(≥)},\ \frac{\texttt{⊸ zero ∪ plus}}{\ni}\ \texttt{est(≥)}]$

coproduct of maps — (11.2.1a) at T:=[zero,⊸ zero ∪ plus], then [U,V]Z=[UZ,VZ] — (11.2a), composition over ∪

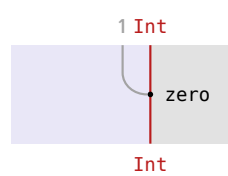


the plus operand of the lower arm's ⊸ zero ∪ plus, under its 𝟙%∋ E(…) and est(≥)

=



singleton, ≥ reflexive — (13.1b)'s $\frac{\mathbb{1}}{\ni}$ est(R)=𝟙∩R at R:=≥, zero a map; the lower branch is ⊕'s definition, ≜ mss, and no law



with (13.3.4d) the greedy theorem gives ⦇[zero,⊕]⦈⊑ $\frac{\texttt{prefix sum}}{\ni}$ est(≥) — and (13.3.4g) makes it an equality.

(13.3.4f)

$[\mathsf{nil},\mathsf{cons}](\mathsf{g},\frac{\mathsf{suffix}}{\ni}\ \mathsf{E}(\mathsf{g}))=\mathsf{F}((\mathsf{g},\frac{\mathsf{suffix}}{\ni}\ \mathsf{E}(\mathsf{g})))\mathsf{k}$

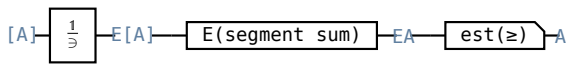
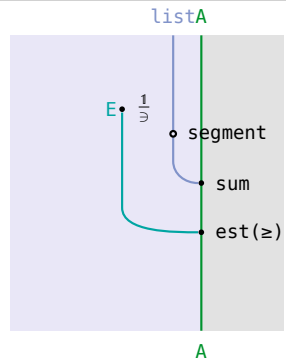
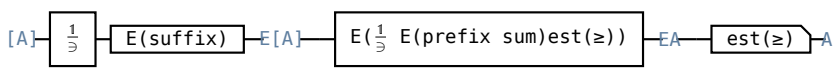
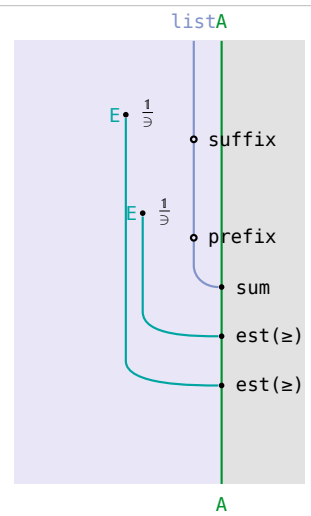
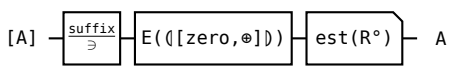
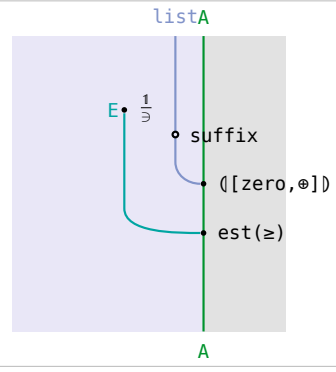
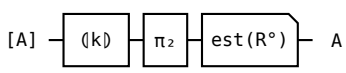
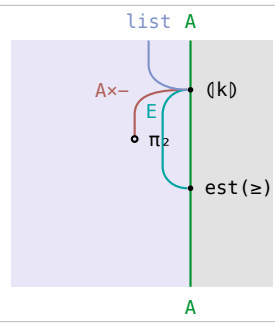
$\mathsf{k}\triangleq[\mathsf{zero}\langle\mathbb{1},\frac{\mathbb{1}}{\ni}\rangle,\langle\mathsf{w},\langle\mathsf{w}\ \frac{\mathbb{1}}{\ni},\pi_2\pi_2\rangle\ \mathsf{cup}\rangle]$, $\mathsf{w}\triangleq(\mathbb{1}\times\pi_1)\oplus$: the value at the whole list, paired with the set of the values at its suffixes, runs k's recursion.

the equation at that branch	why
$\mathsf{nil}\langle\mathsf{g},\frac{\mathsf{suffix}}{\ni}\ \mathsf{E}(\mathsf{g})\rangle=\mathsf{zero}\langle\mathbb{1},\frac{\mathbb{1}}{\ni}\rangle$	nil has one suffix, itself, and $\mathsf{g}(\mathsf{nil})=\mathsf{zero}$, so the set is the singleton $\{\mathsf{zero}\}$
$\mathsf{cons}\langle\mathsf{g},\frac{\mathsf{suffix}}{\ni}\ \mathsf{E}(\mathsf{g})\rangle=(\mathbb{1}\times\langle\mathsf{g},\frac{\mathsf{suffix}}{\ni}\ \mathsf{E}(\mathsf{g})))\langle\mathsf{w},\langle\mathsf{w}\ \frac{\mathbb{1}}{\ni},\pi_2\pi_2\rangle\ \mathsf{cup}\rangle$	$\mathsf{g}(\mathsf{cons}(\mathsf{a},\mathsf{x}))=\mathsf{a}\oplus(\mathsf{g}(\mathsf{x}))$, which is w reading $\mathsf{g}(\mathsf{x})$ off π_1; and the suffixes of $\mathsf{cons}(\mathsf{a},\mathsf{x})$ are $\mathsf{cons}(\mathsf{a},\mathsf{x})$ itself, whose value is that same w, together with those of x, which $\pi_2\pi_2$ carries — so the two sets meet at cup

(13.3.4g)

$\mathsf{mss}=(\!|\mathsf{k}|\!)\ \pi_2\ \mathsf{est}(\geq)$

maximum segment sum problem: $\frac{\mathsf{suffix}}{\ni}\ \mathsf{E}((\!|[\mathsf{zero},\oplus]|\!))$ expressed as the catamorphism $(\!|\mathsf{k}|\!)$, hence mss implemented by a linear-time algorithm, $\oplus\triangleq\frac{\multimap\ \mathsf{zero}\ \cup\ \mathsf{plus}}{\ni}\ \mathsf{est}(\geq)$ — $\triangleq\mathsf{mss}$; k and w — (13.3.4f).

circuit	Hinze–Marsden
 mss is the greatest of the segment sums — $\triangleq\mathsf{mss}$	
$=$  (13.3.4b)	
 $=\frac{\mathsf{suffix}}{\ni}\ \mathsf{E}((\!\|[\mathsf{zero},\oplus]\|\!))\ \mathsf{est}(\geq)$ the greedy theorem greedy theorem at $\mathsf{R}:=\geq$, $\mathsf{S}:=[\mathsf{zero},\multimap\ \mathsf{zero}\ \cup\ \mathsf{plus}]$ — (13.3.4d) is its condition and (13.3.4e) its $\frac{\mathsf{S}}{\ni}\ \mathsf{est}(\geq)$; its $\sqsubseteq$ is an $=$ because $(\!\|[\mathsf{zero},\oplus]\|\!)$ is entire and $\frac{\mathsf{prefix\ sum}}{\ni}\ \mathsf{est}(\geq)$ simple (13.3.3f)'s last row	
 $=(\!\|\mathsf{k}\|\!)\ \pi_2\ \mathsf{est}(\geq)$ $\triangleq\mathsf{reduce}$ at (13.3.4f)'s equation, so $(\!\|\mathsf{k}\|\!)=\langle(\!\|[\mathsf{zero},\oplus]\|\!),\frac{\mathsf{suffix}}{\ni}\ \mathsf{E}((\!\|[\mathsf{zero},\oplus]\|\!))\rangle$, of which π_2 is the row above	

one fold builds the $\mathsf{n+1}$ running maxima and the final $\mathsf{est}(\geq)$ reads them in one more pass, so mss is linear.

§13.3.5. filter(p)=⦇[nil,(π₁p→cons,π₂)]⦈

(13.3.5a)

name	definition	type	example	in words
F, α, p, R	as in ≜ takewhile			
π₂	π₂ where ≜ takewhile has ⊸ nil	A×[A]→[A]	π₂(3,[1,2])=[1,2]	drop the head, keep the tail
subseq	⦇[nil,π₂ ∪ cons]⦈ ≜ subseq	[A]→[A]	[3,1,2] subseq [3,2]	xs subseq ys⟺ys is xs with elements dropped — at each cons, drop the head or keep it
S	[nil,π₂ ∪ (p×𝟙) cons]	F([A])→[A]	(4,[2]) S [2] and (4,[2]) S [4,2], but (3,[2]) S [2] only	subseq's algebra with one extra p — drop the head, or keep a head that passes p
𝟙⊑π₂R cons°				the tail is one shorter than the cons, so π₂ loses the est(R°) at every step — where ≜ takewhile's loser is nil

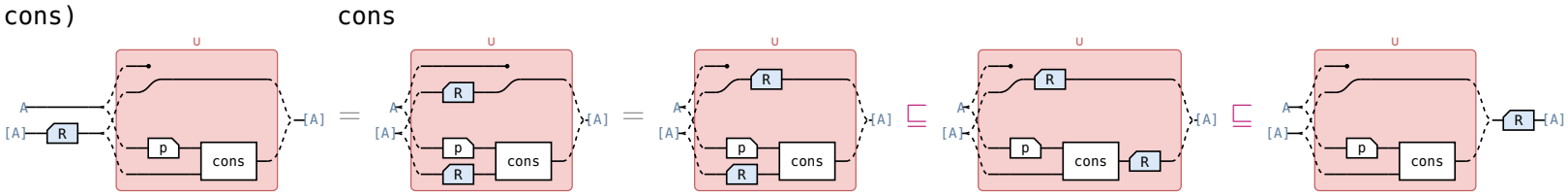
(3.3.5b)

(𝟙×R°)(π₂ ∪ (p×𝟙) cons)⊑(π₂ ∪ (p×𝟙) cons)R°

shortening the tail and then taking the step lands inside taking the step and then shortening the result

formula — the cons branch of F(R°)S⊑SR°; **reason** under each circuit

(𝟙×R°)(π₂ ∪ (p×𝟙) cons) (𝟙×R°)π₂ ∪ (p×R°) cons π₂R° ∪ (p×R°) cons π₂R° ∪ (p×𝟙) cons R° (π₂ ∪ (p×𝟙) cons)R°



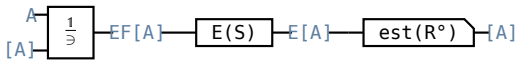
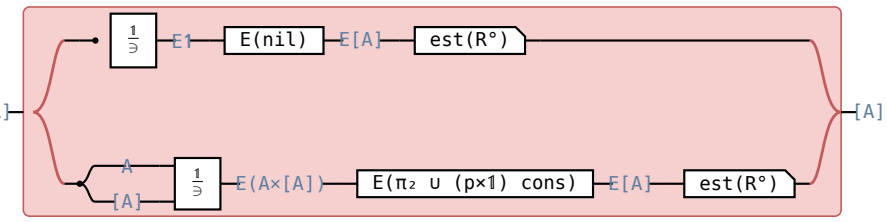
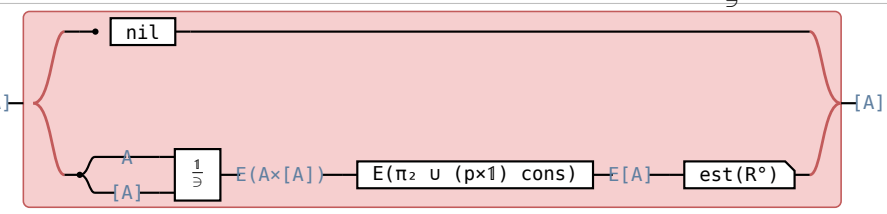
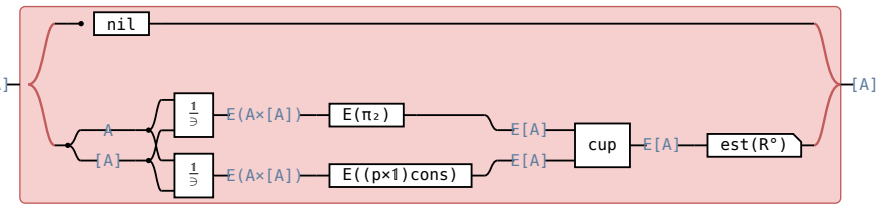
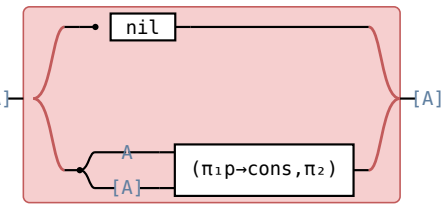
(𝟙×R°)(p×𝟙)=p×R° (1.1a) (𝟙×R°)π₂=π₂R° (12.1c) (p×R°) cons⊑(p×𝟙) cons R° (13.3.3c) (1.1a)

the nil branch: nil⊑nil R°.

(13.3.5c)

$\frac{S}{\ni}$ `est(R°)=[nil,(π₁p→cons,π₂)]`

the longest of the lists the algebra allows is the **cons** where the head passes **p**, and the tail where it does not

formula	reason
	
=  $[\frac{nil}{\ni}\ est(R°),\ \frac{\pi_2 \cup (p\times\mathbb{1})\ cons}{\ni}\ est(R°)]$	`S=[nil,π₂ ∪ (p×𝟙) cons]` ≜ `filter` — and the `%∋` of a coproduct of maps is the coproduct of their `%∋` (11.2.1a)
=  $[nil,\ \frac{\pi_2 \cup (p\times\mathbb{1})\ cons}{\ni}\ est(R°)]$	`nil%∋` is the singleton `{nil}`, and `est(R°)` of a singleton is its element because `R°` is reflexive ≜ `est`
= 	$\frac{\pi_2 \cup (p\times\mathbb{1})\ cons}{\ni} = \langle \frac{\pi_2}{\ni}, \frac{(p\times\mathbb{1})\ cons}{\ni} \rangle\ cup$ ≜ `cup`
= 	`𝟙⊑π₂R cons°` ≜ `filter` — `xs R cons(a,xs)`, so `est(R°)` returns the `cons` where `p a` puts it in the set and `xs` where the set is `{xs}` ≜ `est`

the head is dropped, not the whole tail: that is the one place `π₂` shows against (13.3.3d)'s `⊸ nil`.

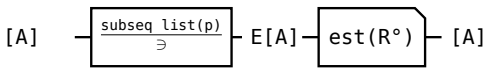
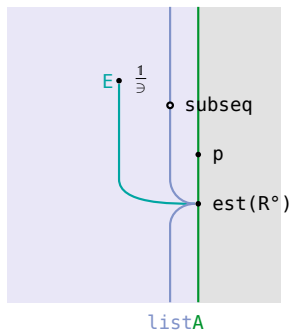
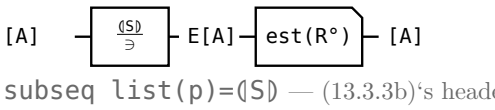
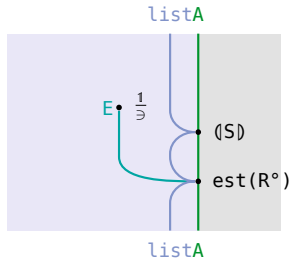
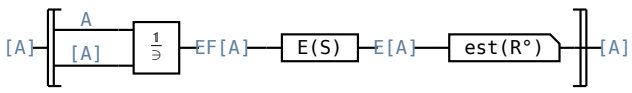
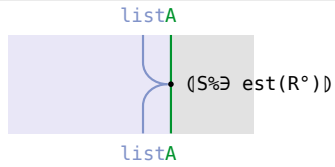
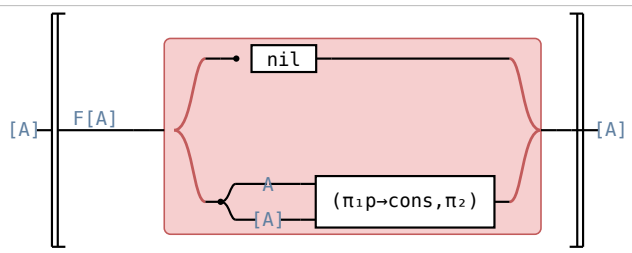
(13.3.5d)

name	definition	type	example	in words
`S`	`[nil,π₂ ∪ (p×𝟙) cons]` ≜ `filter`	`F([A])→[A]`	`(4,[2]) S [2]` and `(4,[2]) S [4,2]`, but `(3,[2]) S [2]` only	`subseq`'s algebra with one extra `p` — drop the head, or keep a head that passes `p`

(13.3.5e)

filter(p)=⦇[nil,(π₁p→cons,π₂)]⦈

filter: **filter(p) x** returns the longest subsequence of **x** with the property that all its elements satisfy **p**; the catamorphism is the standard program; **R** a preorder.

circuit — one wire, its type written along it	**Hinze–Marsden**
 ≜ subseq	
=  subseq list(p)=⦇S⦈ — (13.3.3b)'s header, subseq for prefix	
⊒  greedy theorem at R°, whose hypothesis F(R°)S⊑SR° is (13.3.5b)	
=  (13.3.5c)	
= the catamorphism is entire and filter(p) simple, so ⊒ is =	

(subseq list(p))°(subseq list(p))∩R∩R°⊑𝟙 fails — two p-subsequences of one list can be of equal length and different — so §13.3.3's uniqueness argument does not transfer. What survives est(R°) is the one keeping **exactly** the passing elements: a subsequence that drops a passing element is beaten by the one that keeps it.

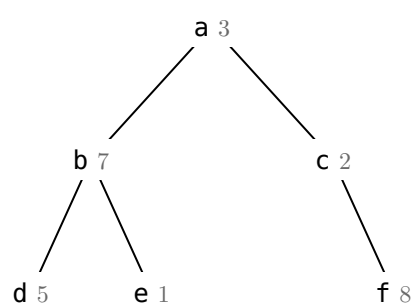
§13.4. Planning a company party

(13.4a)

definition	type	note
`tree A::=node (A,[tree A])`	$\mathcal{A}\to\mathcal{A}$	The company hierarchy: an employee, and the list of subtrees under them.
`F(A,B)=A×[B]`	$\mathcal{A}\times\mathcal{A}\to\mathcal{A}$	The base functor `tree` folds: an employee beside the recursive position, one layer deep.
`rating`	`A→Real`	What one employee is worth as a guest.
`cost≜list(rating) sum`	`[A]→Real`	What a guest list is worth.
`R≜cost≤cost°`	`[A]→[A]`	The preorder the guest list is maximised over.
`choose≜π₁ ∪ π₂`	`[A]×[A]→[A]`	Takes one of the two parties a subtree returns.
`include≜(𝟙×(list(π₂) concat)) cons`	`F(A,[A]×[A])→[A]`	The party that invites the root, which puts every immediate subtree's root out. A map.
`exclude≜(𝟙×(list(choose) concat))π₂`	`F(A,[A]×[A])→[A]`	The party that leaves the root out, so each subtree is free to choose. Not a map.
`S≜⟨include,exclude⟩`	`F(A,[A]×[A])→[A]×[A]`	The algebra: one step returns both parties of a subtree at once.
`party≜⦇S⦈ choose`	`tree A→[A]`	Every guest list the president's ruling allows.
the specification $\frac{\texttt{party}}{\ni}$ `est(R°)`	`tree A→[A]`	A guest list of greatest total conviviality.

§13.4.1. `include` and `exclude`

(13.4.1a)



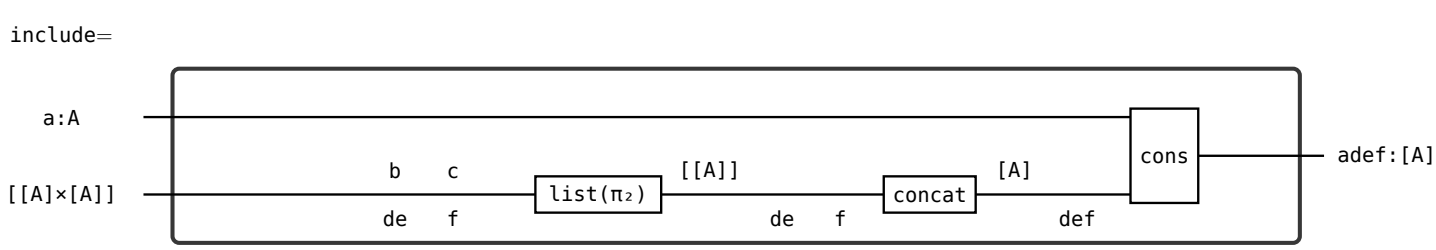
the small grey number is the employee's rating

(13.4.1b)

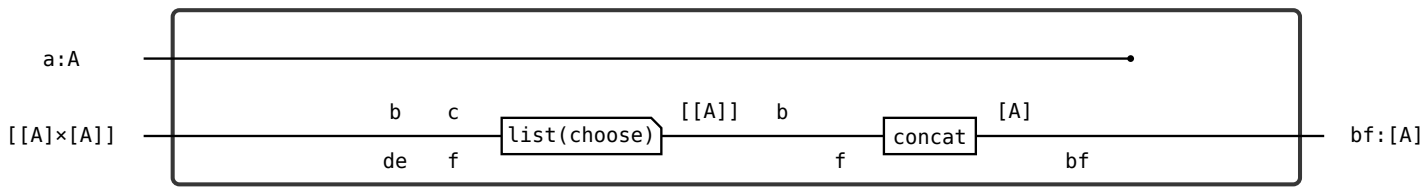
node	the algebra's input `A×[[A]×[A]]`	`include`	`exclude`
`d`, `e`, `f`	`(d,[])`	`[d]`	`[]`
`b`	`(b,[([d],[]),([e],[])])`	`[b]`	`[d,e]`, `[d]`, `[e]`, `[]`
`c`	`(c,[([f],[])])`	`[c]`	`[f]`, `[]`
`a`	`(a,[([b],…),([c],…)])`	`[a]+exclude(b)+exclude(c)`	`choose(b)+choose(c)`

with those ratings, `est(R°)` keeps `([b]=7,[d,e]=6)` at `b`, where `include` wins, and `([c]=2,[f]=8)` at `c`, where `exclude` wins, so at the root `include=[a,d,e,f]=17` beats `exclude=[b,f]=15`, and `choose` takes 17.

(13.4.1c)



exclude= (13.4.1d)




```
list(choose) [([d],[]),([e],[])]                                    (13.4.1e)
  1st item                ([d],[])  choose↦[d]  or  []
  2nd item                ([e],[])  choose↦[e]  or  []
  2×2=4 combinations:   [[d],[e]]   [[d],[]]   [[],[e]]   [[],[]]
  concat flattens:        [d,e]       [d]        [e]        []
```

§13.4.2. list((R×R)°)

```
[  ([b],[d,e])  ,  ([c],[f])  ]                                     (13.4.2a)
        |              |
        ▼              ▼        list((R×R)°), elementwise
[   ([b],[d])   ,  ([c],[])   ]

([b],[d,e])
  |   └── exclude: the best party in b's subtree when b does not come
  └────── include: the best party in b's subtree when b comes
```

(13.4.2b)

element	1st component include	2nd component exclude
1st, ([b],[d,e])	[b] R° [b] 7≥7, reflexivity	[d,e] R° [d] 6≥5, and ([b],[d,e]) is the pair est(R°) keeps at b
2nd, ([c],[f])	[c] R° [c] 2≥2, reflexivity	[f] R° [] 8≥0, and ([c],[f]) is the pair est(R°) keeps at c

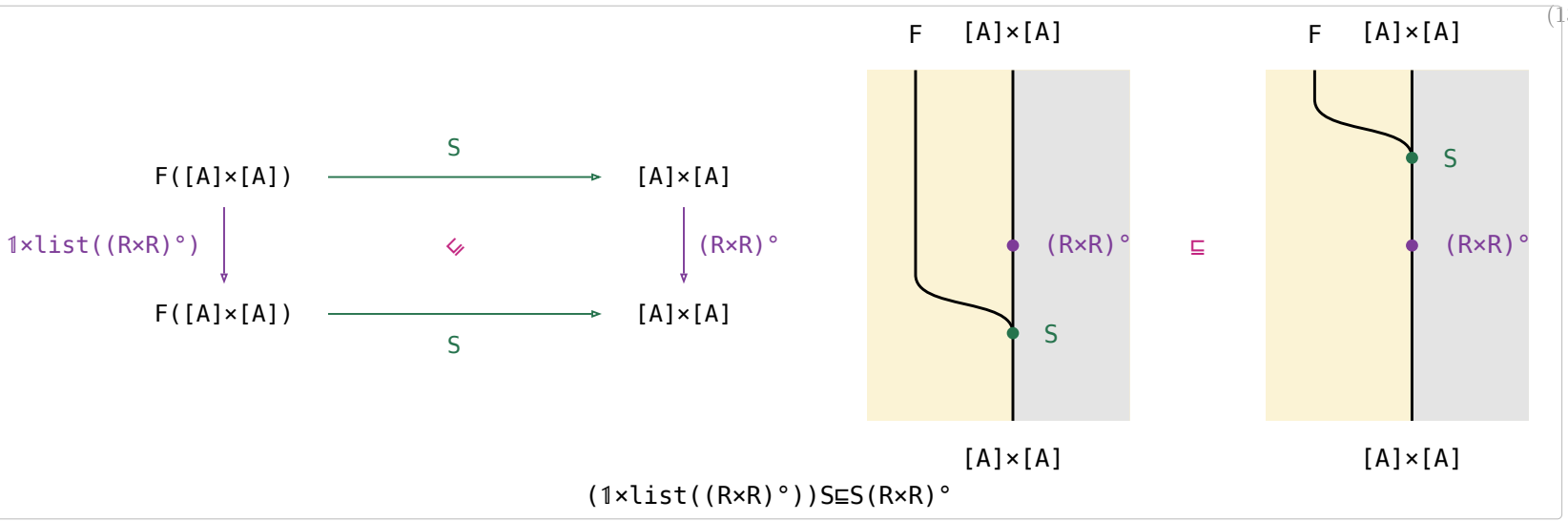
the first component of 𝟙×list((R×R)°) is the root employee.

R≜cost≤cost° is a preorder, so the components that do not move are related by reflexivity — (R×R)° demands a relation in **both** components, it cannot skip one.

R compares cost only, not contents: [f] R° [] is legal though [f] has an element [] does not. That is why the three leaves of §13.4.3's proof all reduce to "cost is a sum".

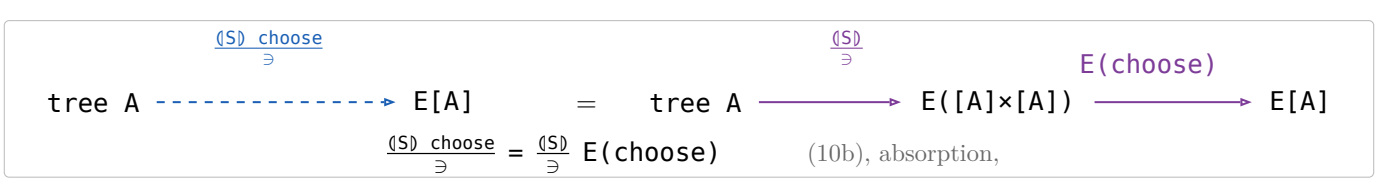
§13.4.3. (𝟙×list((R×R)°))S⊑S(R×R)° — S : F([A]×[A])→[A]×[A] monotonic on (R×R)°

(13.4.3a)



(𝟙×list((R×R)°))S⊑S(R×R)°

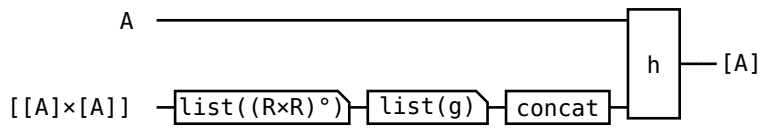
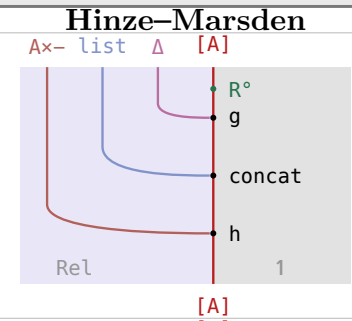
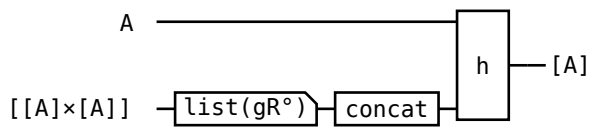
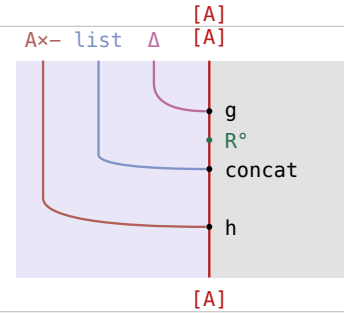
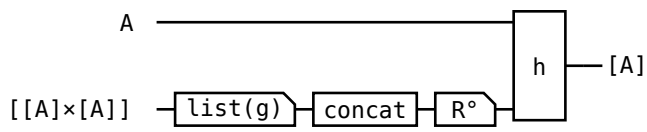
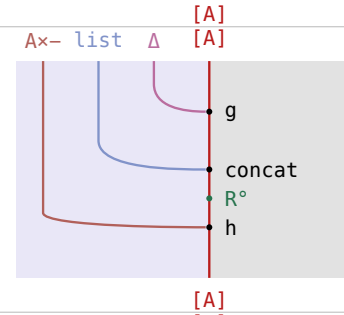
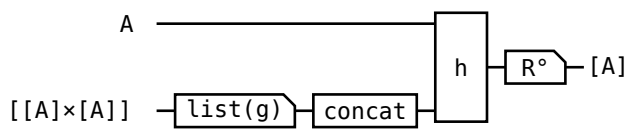
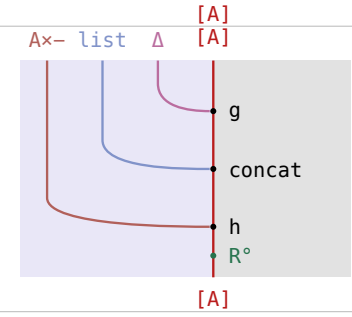
(13.4.3b)



(13.4.3c)

```
S≜(include,exclude)
(𝟙×list((R×R)°))S ⊑ S(R×R)°
(𝟙×list((R×R)°))include ⊑ include R°
(𝟙×list((R×R)°))exclude ⊑ exclude R°
```

bettering both parties of every subtree before the node's algebra runs gets no further than running it first and bettering the two parties it returns,

circuit	Hinze–Marsden
	
⊑ 	
⊑ 	
⊑ 	

g `(R×R)°g⊑gR°`
`g:=π₂` is `(R×R)°π₂=(Dom(π₁R°))π₂R°⊑π₂R°` , `g:=π₁` its mirror `(Dom(π₂R°))π₁R°⊑π₁R°` — 1 and 4 of (11.1.2b), then `Dom⊑𝟙`; `g:=choose≜π₁ ∪ π₂` is the union of the two (13.2c). `list` monotonic, ≜ relator

concat `list(R°)concat⊑concat R°`
a LEAF: no law above it. `cost` is a sum, so `cost(concat(xss))=sum(list(cost)(xss))` and a cheaper part makes a cheaper whole.

h `(𝟙×R°)h⊑hR°`
a LEAF: `cost(cons(a,xs)) =rating(a)+cost(xs)`, so a cheaper tail makes a cheaper list. For `h:=π₂` it is an EQUALITY `(𝟙×R°)π₂=(Dom(π₁))π₂R° =π₂R°` : `π₁` is a map, hence entire, so `Dom(π₁)=𝟙` — (5c)

(𝟙×(list(g) concat))h	g	h
include	π₂	cons
exclude	choose	π₂

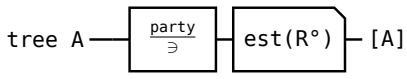
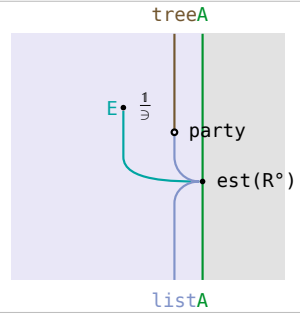
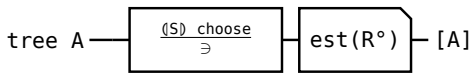
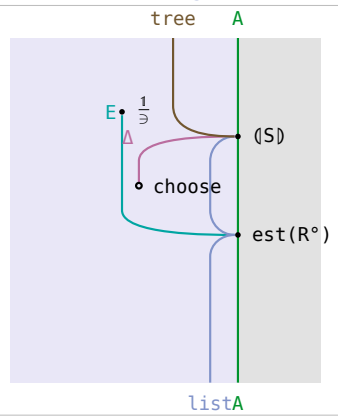
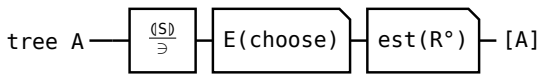
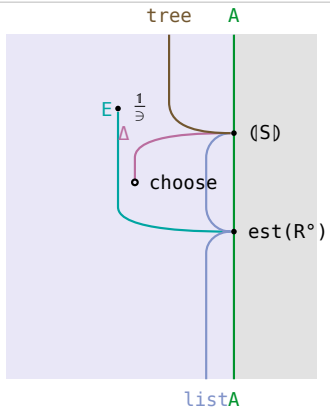
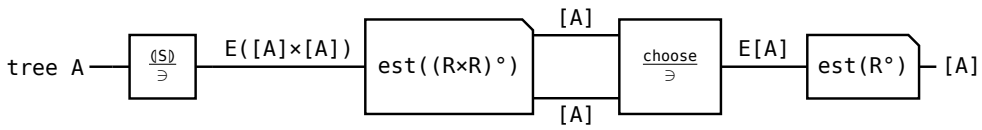
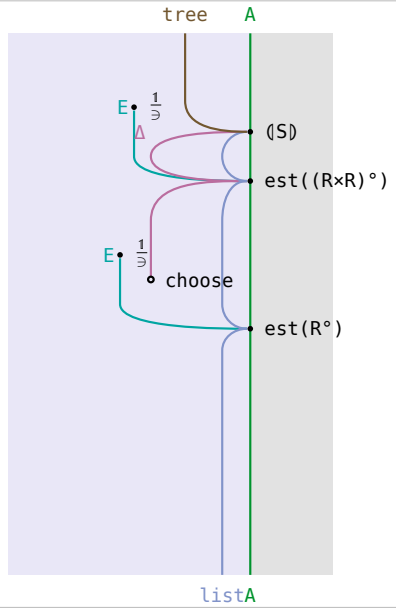
(13.4.3d)

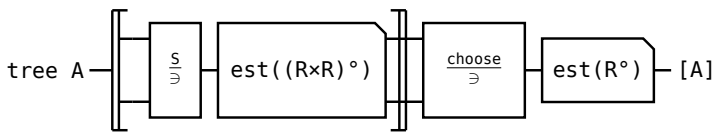
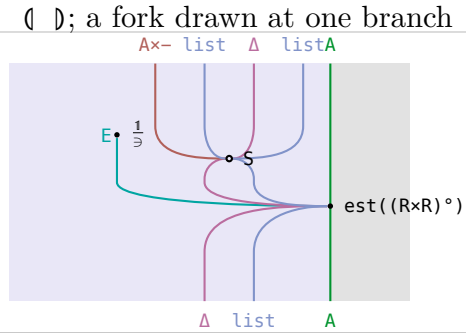
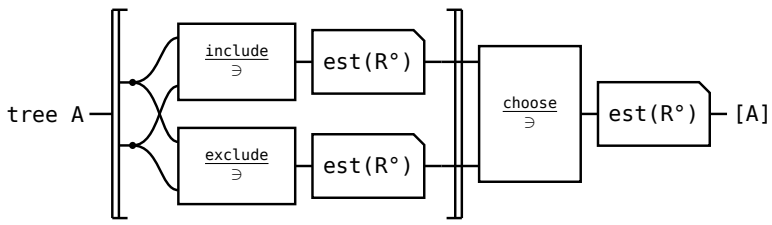
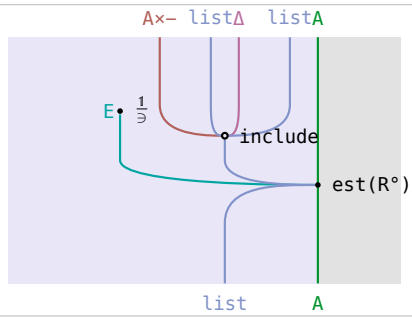
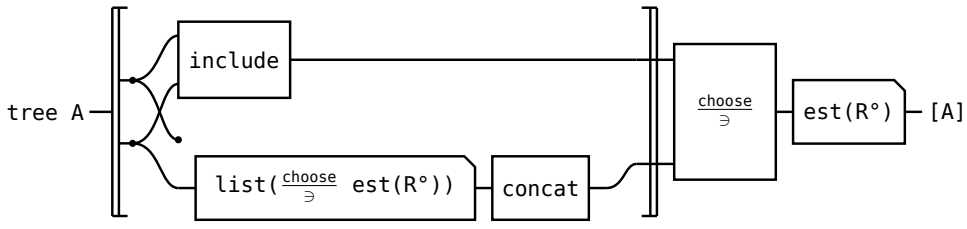
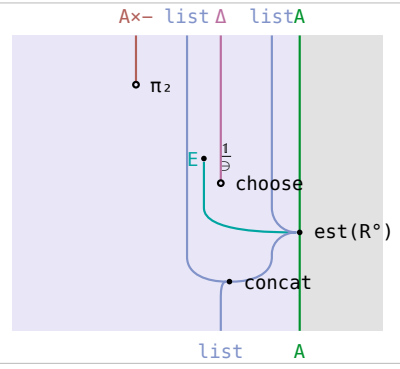
`R≜{(0,0)} : {0,1}→{0,1}`	not entire — undefined at `1`
`(R×R)choose` at `(0,1)`	nothing — `R×R` needs BOTH components related, and `R` is undefined at `1`
`choose R` at `(0,1)`	`0` — `choose` picks the first component `0`, and `R 0 0` holds
`((0,1),0)`	in `choose R`, not in `(R×R)choose`
`(R×R)choose=choose R`	iff `R` is entire — `Dom⊑𝟙` is the `g` row's only inequality step, and it is an equality iff `Dom(R)=𝟙`

§13.4.4. The derivation

$\frac{\text{party}}{\ni}$ `est(R°)∋⦇⟨include,π₂ list(`$\frac{\text{choose}}{\ni}$` est(R°)) concat⟩⦈` $\frac{\text{choose}}{\ni}$ `est(R°)` (13.4.4a)

the best of every guest list the president allows is one pass up the tree, each subtree handing up its best party with its boss in and its best with the boss out, and **choose** taking the better of the two at the root

circuit	**Hinze–Marsden** — outside the ⦇ ⦈
 the specification — ≜ `party`	
=  `party≜⦇S⦈ choose` — ≜ `party`	
=  $\frac{\text{⦇S⦈ choose}}{\ni}=\frac{\text{⦇S⦈}}{\ni}$ `E(choose)` — (13.4.3b)	
⊑  `(R×R)°choose⊑choose R°` — (13.4.3c)'s `g` row,	

circuit	**Hinze–Marsden** — inside the ⦇ ⦈; a fork drawn at one branch (13.4.4b)
 from here ⦇ ⦈ is drawn open — the two bars, with the algebra's own circuit between them; `(𝟙×list((R×R)°))S⊑S(R×R)°` at `(R×R)°`, (13.4.3a),	
 `⟨`include/∋ `est(R°),`exclude/∋ `est(R°)⟩⊑`S/∋ `est((R×R)°),`	
 `include` a map, `est(R°)` into each branch,	

§13.5. Shortest paths on a cylinder

(13.5a)

definition	type	note
F(A,X)=A+A×X	$\mathcal{A}\times\mathcal{A}\to\mathcal{A}$	The base functor: one square of the new column, alone or in front of the path so far.
L=list⁺	$\mathcal{A}\to\mathcal{A}$	A path is a non-empty list of squares, one per column crossed.
α the initial algebra	F(A,LA)→LA	α(5)=[5] and α(1,[5])=[1,5]: a path is started by one square, or extended by one.
N the n-tuple relator	$\mathcal{A}\to\mathcal{A}$	One component per row of a column, so the fold carries n answers at once.
R≜sum≤sum°	L Nat→L Nat	The cost of a path, which the cheapest minimises.
setify	NA→EA	setify(1,2,3,4)={1,2,3,4} — which row a component came from is forgotten.
moves	NA→E(NA)	moves(x)={up(x),x,down(x)} — rotated up, unrotated, rotated down.
trans	E(NA)→N(EA)	trans{(a,b,c),(x,y,z)}=({a,x},{b,y},{c,z}) — component k of the result is the set of the k-th components.
zip	F(NA,NB)→NF(A,B)	zip((1,2,3,4),({[5]},{[6]},{[7]},{[8]}))=((1,{[5]}),(2,{[6]}),(3,{[7]}),(4,{[8]})).
cp≜ $\frac{F(1,\ni)}{\ni}$	F(A,EB)→E(F(A,B))	cp(1,{[5],[6],[8]})={(1,[5]),(1,[6]),(1,[8])}.
gen≜F(1,moves trans N(union)) zip N(cp P(α))	F(NA,N(E(LA)))→N(E(LA))	gen((1,2,3,4),({[5]},{[6]},{[7]},{[8]})) is worked out in (13.5.1b).
paths≜⦇gen⦈ setify union	L N Nat→E(L Nat)	paths[(1,2,3,4),(5,6,7,8)] is the union of (13.5.1b)'s four sets: 12 paths, 3 from each entry row.
the specification paths est(R)	L N Nat→L Nat	A cheapest path from the entry side to the exit side.

§13.5.1. N(E(L A))

(13.5.1a)

```
1   5
2   6
3   7      [(1,2,3,4),(5,6,7,8)] : L N Nat
4   8
```

(13.5.1b)

```
⦇gen⦈[(5,6,7,8)] = ({[5]},{[6]},{[7]},{[8]})

gen((1,2,3,4),({[5]},{[6]},{[7]},{[8]})) =
    ( {[1,5],[1,6],[1,8]},
      {[2,5],[2,6],[2,7]},
      {[3,6],[3,7],[3,8]},
      {[4,5],[4,7],[4,8]} )
```

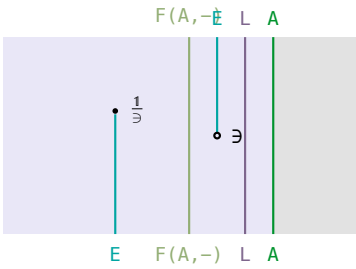
(13.5.1c)

N	the 4-tuple, one component per row of the entry column
E	the set of paths that can start in that row
L	a path is the non-empty list of the squares it crosses

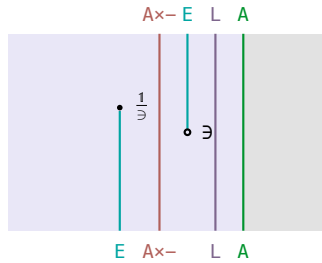
row 1's set has no `[1,7]` — from row 1 only rows 4, 1 and 2 are reachable — and it does have `[1,8]`, because the cylinder glues the bottom row to the top. That is `moves trans N(union)` of ≜ `paths`: component `k` collects the paths of rows `k-1`, `k`, `k+1`.

§13.5.2. The cross product cp≜$\frac{\texttt{F(1,∋)}}{\texttt{∋}}$

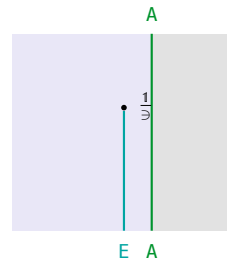
(13.5.2a)



cp on all of `F(A,E(L A))`



the `A×−` summand: `∋` picks one path, `𝟙%∋` collects the results



the `A` summand, no `E` to distribute: `𝟙%∋` alone, `a↦{a}`

(13.5.2b)

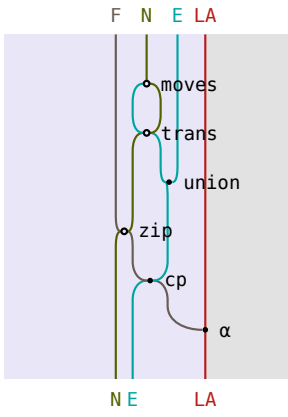
type	note
`F(A,−)=A+A×−`	one wire carries the whole functor, both summands with it; the `∋` under it is `F(𝟙,∋)`
`∋:E(L A)→L A`	the `∋` inside `F(𝟙,∋)`: a set of paths, one of them — `L A` is one path, `A` one square
`F(𝟙,∋)=𝟙+𝟙×∋`	`:A+A×E(L A)→A+A×L A`, the relator acting on each summand
`∋:E(A+A×L A)→A+A×L A`	the `∋` under the bar: a set of cells, one of them — a different `∋`
`cp:A+A×E(L A)→E(A+A×L A)`	$\frac{\texttt{R}}{\texttt{∋}}$ turns `R:X→Y` into `X→E Y`
`cp=`$\frac{\texttt{1}}{\texttt{∋}}$ `E(F(𝟙,∋))`	what the picture draws: the unit makes the outer `E`, leaving one `∋` — (10b)

(13.5.2c)

```
w      = (1,{[5],[6],[8]})            : A×E(L A)     the right summand
𝟙×∋    : A×E(L A) → A×L A                            𝟙 keeps the square 1, ∋ picks one path
         w ↦ (1,[5]), (1,[6]), (1,[8]) : A×L A        one output per path in w
cp(w)  = {(1,[5]),(1,[6]),(1,[8])}    : E(A+A×L A)   the three of them, collected
```

§13.5.3. gen=F(𝟙,moves trans N(union)) zip N(cp P(α))

(13.5.3a)



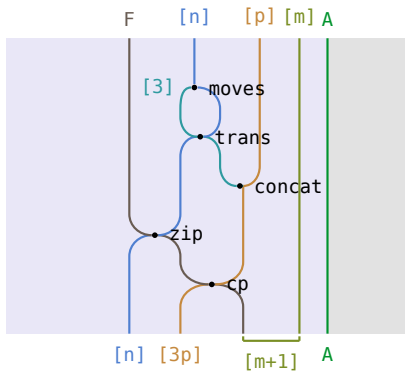
```
u = ((1,2,3,4),({[5]},{[6]},{[7]},{[8]}))      : F(N A,N(E(L A)))                                  (13.5.3b)
F(𝟙,moves trans N(union))                        𝟙 keeps the column, the path SETS move
  moves({[5]},{[6]},{[7]},{[8]})
   = {({[6]},{[7]},{[8]},{[5]}),
      ({[5]},{[6]},{[7]},{[8]}),
      ({[8]},{[5]},{[6]},{[7]})}                 : E(N(E(L A)))  down, unmoved, up
  trans(that)
   = ({{[6]},{[5]},{[8]}},{{[7]},{[6]},{[5]}},
      {{[8]},{[7]},{[6]}},{{[5]},{[8]},{[7]}}) : N(E(E(L A)))  row k gets rows k-1, k, k+1
  N(union)(that)
   = ({[5],[6],[8]},{[5],[6],[7]},
      {[6],[7],[8]},{[5],[7],[8]})               : N(E(L A))     every path into row k, none dropped
zip(that)
   = ((1,{[5],[6],[8]}),(2,{[5],[6],[7]}),
      (3,{[6],[7],[8]}),(4,{[5],[7],[8]}))       : N(F(A,E(L A)))  each row: its square, and the
                                                                    paths it may be put in front of
N(cp P(α))(that)
   = ({[1,5],[1,6],[1,8]},{[2,5],[2,6],[2,7]},
      {[3,6],[3,7],[3,8]},{[4,5],[4,7],[4,8]}) : N(E(L A))     cp pairs the square with each path,
                                                               α prefixes it: α(1,[5])=[1,5]
```

§13.5.4. gen on Vec: F(𝟙,moves trans Vec(n)(concat)) zip Vec(n)(cp)

(13.5.4a)

	on L=list⁺, 13.5.3	on Vec	note
the input	L(N A)	A[m][n]≜(A[n])[m]	m columns of n squares, outermost index first — the matrix itself.
a path	L A, a non-empty list	A[1]≜A, A[m+1]≜A×A[m]	Through m columns, m squares: the count is in the type. A one-square path is the square; Vec(m) is the functor X↦X[m].
the paths into a row	E(L A)	A[p][m], p=3^(m-1)	Three moves per column crossed, so this count is in the type too; a set forgets it.
α	[wrap,cons] F(A,LA)→LA	wrap=𝟙:A→A[1] cons=𝟙:A×A[m]→A[m+1]	Both are identities by the definition of A[m], so α drops out of gen and Q.
moves	NA→E(NA)	X[n]→X[3][n]	moves(x)=(up(x),x,down(x)), a 3×n matrix.
trans	E(NX)→N(EX)	X[3][n]→X[n][3]	The transpose.
union	E(EX)→EX	concat:X[3][p]→X[3p]	The three candidate lists, end to end.
zip	F(NA,NB)→NF(A,B)	F(A[n],B[n])→F(A,B)[n]	Unchanged.
cp	F(A,EB)→E(F(A,B))	A×B[p]→(A×B)[p]	The square paired with each of the p candidates.
gen	F(𝟙,moves trans N(union)) zip N(cp P(α)) F(NA,N(E(LA))) →N(E(LA))	F(𝟙,moves trans Vec(n)(concat)) zip Vec(n)(cp) F(A[n],A[n][p][m])→A[n][3p][m+1]	The same composite without α; the type shown is the cons side, the wrap side is 𝟙:A[n]→A[n].
⦇gen⦈	L(N A)→N(E(L A))	A[m][n]→A[n][p][m]	Columns in, rows out.
Q, 13.5.6	F(𝟙,moves trans N(est(R))) zip N(α)	F(𝟙,moves trans Vec(n)(est(R))) zip, est(R):X[3]→X	The cheapest of three; no p anywhere. ⦇Q⦈:A[m][n]→A[n][m] has the type of a transpose.

(13.5.4b)

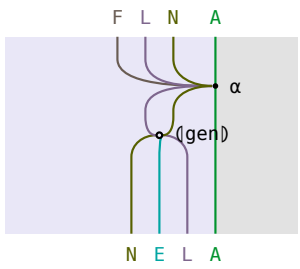


(13.5.4c)

```
a row is the outer index, a cell one path
u = ((1,2,3,4),C)                                  : F(A[n],A[n])
    C = 5                                            one candidate per row, one square long:
        6                                            a column of squares
        7
        8
F(𝟙,moves trans Vec(n)(concat))                      𝟙 keeps the column, the candidates move
  moves(C)                                           down, unmoved, up: a 3×n matrix
   = 6 7 8 5
     5 6 7 8
     8 5 6 7                                       : A[3][n]
  trans(that)                                        the transpose: row k gets rows k-1, k, k+1
   = 6 5 8
     7 6 5
     8 7 6
     5 8 7                                         : A[n][3]
  Vec(n)(concat)(that) = that                        concat:X[3][1]→X[3] is 𝟙
zip(that)                                            each row: its square, and its candidates
   = (1, 6 5 8)
     (2, 7 6 5)
     (3, 8 7 6)
     (4, 5 8 7)                                    : F(A,A[3])[n]
Vec(n)(cp)(that)                                     cp pairs the square with each candidate,
   = [1,6] [1,5] [1,8]                               and a pair is a two-square path
     [2,7] [2,6] [2,5]
     [3,8] [3,7] [3,6]
     [4,5] [4,8] [4,7]                             : A[n][3][2]
```

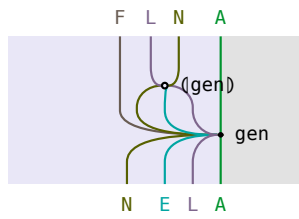
§13.5.5. gen is an F-algebra; ⦇gen⦈: α⦇gen⦈=F(𝟙,⦇gen⦈)gen

(13.5.5a)



α puts the column back on the list, then the fold reads all of it

=



the fold reads the rest under F, then one gen puts the column in front

(13.5.5b)

```
xs = [(1,2,3,4),(5,6,7,8)] = α((1,2,3,4),[(5,6,7,8)])    : L(N A)

⦇gen⦈(xs) = gen((1,2,3,4), ⦇gen⦈[(5,6,7,8)])
```

both halves are in (13.5.1b): the tail folds to `({[5]},{[6]},{[7]},{[8]})`, one path per row and each of them one square long, and gen on it is (13.5.3b)'s walk.

§13.5.6. Q : F(N A,N(L A))→N(L A), an F(N A,–)-algebra: Q=F(𝟙,moves trans N(est(R))) zip N(α)

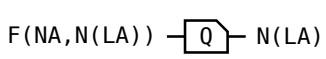
```
Q : F(N A,N(L A)) → N(L A)                    the fold's algebra: a new column, one path per row   (13.5.6a)

u = ((1,2,3,4),([5],[6],[7],[8]))             : F(N A,N(L A))
F(𝟙,moves trans N(est(R)))                       𝟙 keeps the column, the paths move
  moves([5],[6],[7],[8])
   = {([6],[7],[8],[5]),
      ([5],[6],[7],[8]),
      ([8],[5],[6],[7])}                      : E(N(L A))      down, unmoved, up
  trans(that)
   = ({[6],[5],[8]},{[7],[6],[5]},
      {[8],[7],[6]},{[5],[8],[7]})            : N(E(L A))      row k gets rows k-1, k, k+1
  N(est(R))(that)
   = ([5],[5],[6],[5])                        : N(L A)         the cheapest into each row —
                                                               chosen BEFORE the new square
zip(that)
   = ((1,[5]),(2,[5]),(3,[6]),(4,[5]))        : N(F(A,L A))    each row: its square, and the one
                                                               predecessor that survived
N(α)(that)
   = ([1,5],[2,5],[3,6],[4,5])                : N(L A)         α(1,[5])=[1,5]
```

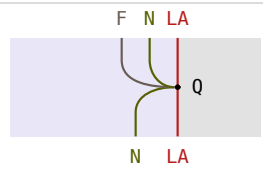
Q=[N(wrap),(𝟙×moves trans N(est(R))) zip' N(cons)] (13.5.6b)

at the last column **Q** starts one path per row, and at each earlier one it puts each square in front of the cheapest of the three kept paths it can step to — the algebra (13.5.6c)'s last step folds

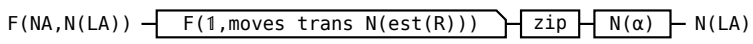
circuit — the fork is F(NA,N(LA))=NA+NA×N(LA)

Hinze–Marsden

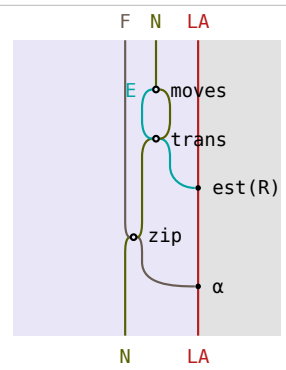




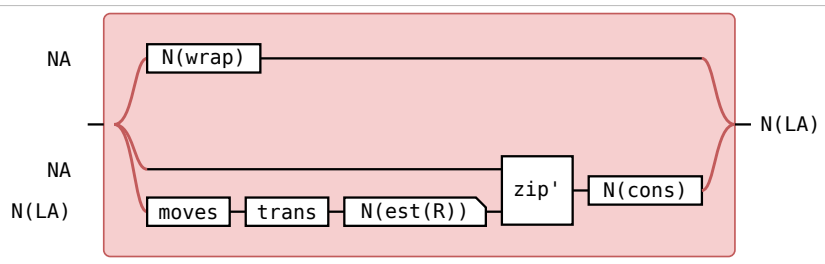
=



the fusion condition (13.5.6d) read as a definition,



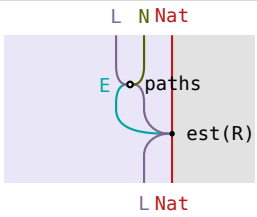
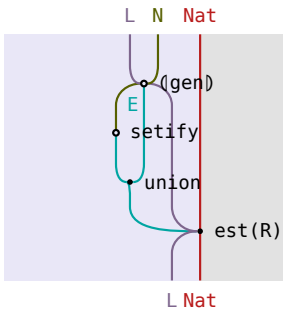
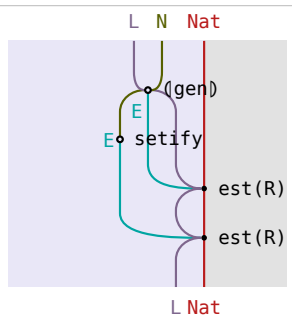
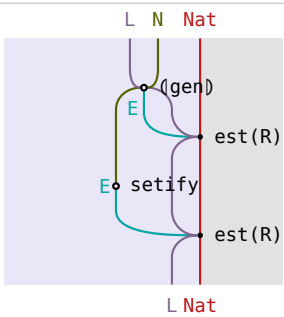
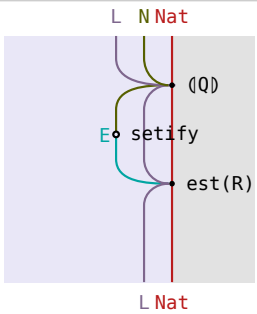
=



zip=𝟙+zip', α=[wrap,cons]

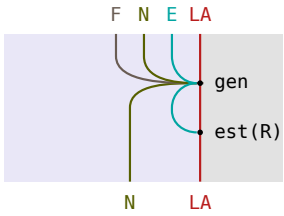
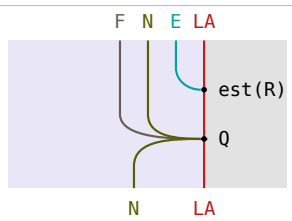
paths est(R)⊒⦇Q⦈ setify est(R) (13.5.6c)

shortest paths on a cylinder: the dynamic programming approach to exhaustive search allows a path of least cost to be found in O(n×m) time; Q is (13.5.6b)'s algebra.

circuit	Hinze–Marsden
L N Nat ─ paths ─ est(R) ─ L Nat	
= L N Nat ─ ⦇gen⦈ ─ setify ─ union ─ est(R) ─ L Nat ≜ paths at paths	
⊑ L N Nat ─ ⦇gen⦈ ─ setify ─ P(est(R)) ─ est(R) ─ L Nat P(est(R)) est(R)⊑union est(R) — a minimum in each set, then a minimum of those; R transitive	
⊑ L N Nat ─ ⦇gen⦈ ─ N(est(R)) ─ setify ─ est(R) ─ L Nat setify lax natural	
⊑ L N Nat ─ ⦇Q⦈ ─ setify ─ est(R) ─ L Nat fusion at (13.5.6d)	

gen N(est(R))⊒F(𝟙,N(est(R)))Q (13.5.6d)

fusion: the condition for fusion in (13.5.6c)'s last step, used to derive a definition of Q.

circuit	Hinze–Marsden
F(NA,N(E(LA))) ─ gen ─ N(est(R)) ─ N(LA)	
⊑ F(NA,N(E(LA))) ─ F(𝟙,N(est(R))) ─ Q ─ N(LA) (7.13), then zip, trans, moves lax natural	

(7.13) is F(𝟙,est(R))α⊑cp P(α) est(R), (13.3.1c) at the map α with $\frac{\text{F(𝟙,∋)α}}{\ni}$ =cp P(α): extending every path in a set and then taking a minimum is beaten by extending one minimum. It is the crux here, not the greedy theorem.

§13.6. The security van problem

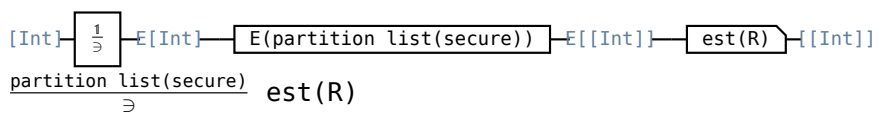
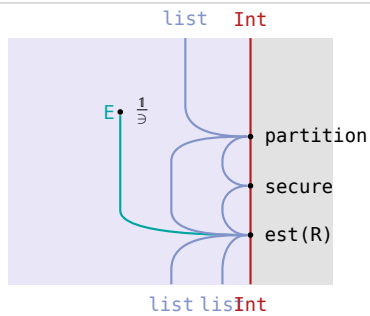
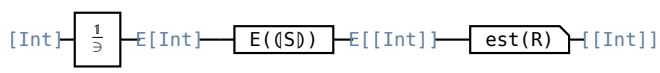
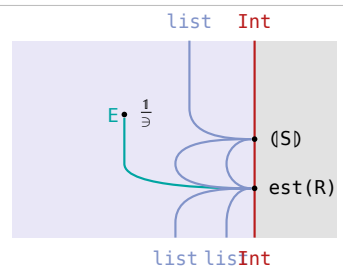
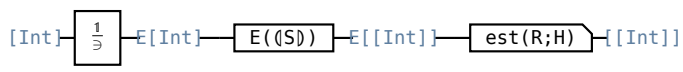
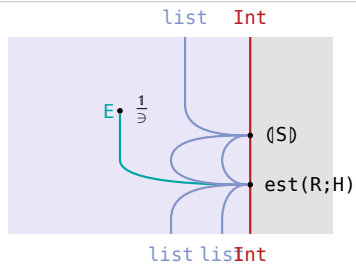
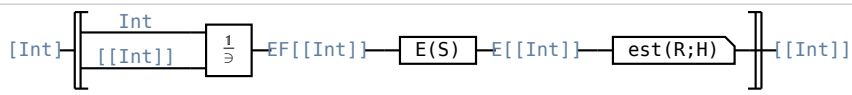
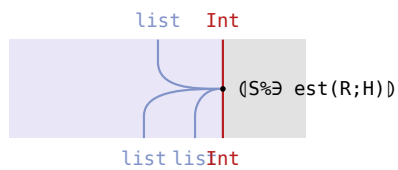
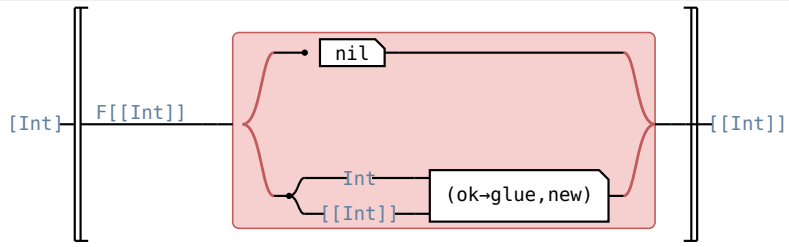
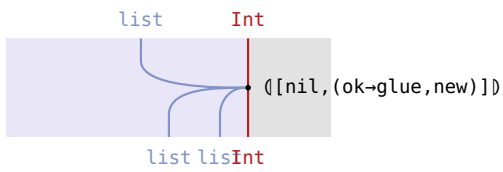
(13.6a)

definition	type	note
R≜length≤length°	[[Int]]→[[Int]]	The order the schedule is minimised over: fewer van visits is better.
ceiling≜ $\frac{\text{prefix sum}}{\ni}$ est(≥)	[Int]→Int	The highest the bank's balance reaches over a stretch of transactions.
floor≜ $\frac{\text{prefix sum}}{\ni}$ est(≤)	[Int]→Int	The lowest it reaches, so ceiling−floor is the cash the stretch has to carry.
secure the coreflexive on x with bmax(ceiling x,ceiling x−floor x)≤N	[Int]→[Int]	The stretches one van visit can serve: some starting reserve keeps the cash between 0 and N.
ok the coreflexive on (a,xs) with xs non-empty and [a]+head(xs) secure	Int×[[Int]] →Int×[[Int]]	The test the final program runs, in place of old's secure.
new≜(wrap×𝟙) cons	Int×[[Int]]→[[Int]]	Calls the van: the transaction opens a segment of its own.
glue≜(𝟙×cons°) assocl (cons×𝟙) cons	Int×[[Int]]→[[Int]]	Puts the transaction on the front of the first segment.
old≜(𝟙×cons°) assocl ((cons secure)×𝟙) cons	Int×[[Int]]→[[Int]]	glue restricted to a first segment that stays secure.
partition=⦇[nil,new ∪ glue]⦈	[Int]→[[Int]]	Every splitting of the transactions into consecutive segments.
S≜[nil,new ∪ old]	1+Int×[[Int]]→[[Int]]	The algebra whose fold is every splitting of the transactions into secure segments.
partition list(secure)=⦇S⦈	[Int]→[[Int]]	Fusion: keeping only secure segments is what turns glue into old.
H≜(head prefix° head°) ∪ (nil° nil)	[[Int]]→[[Int]]	One schedule's first segment is a prefix of the other's, or both are empty.
R;H≜R∩(R°⇒H)	[[Int]]→[[Int]]	The refinement of R that makes both halves of S monotonic.
|R|≜R∩¬R°	[[Int]]→[[Int]]	The strict part R splits into: R;H=|R| ∪ (R∩H).
the specification $\frac{\text{partition list(secure)}}{\ni}$ est(R)	[Int]→[[Int]]	A schedule with the fewest secure segments.

(13.6b)

$\frac{\texttt{partition list(secure)}}{\ni}$ `est(R)⊒⦇[nil,(ok→glue,new)]⦈`

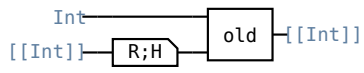
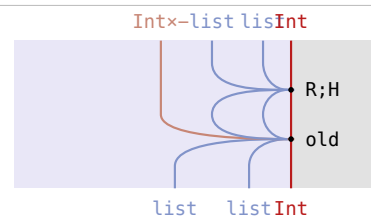
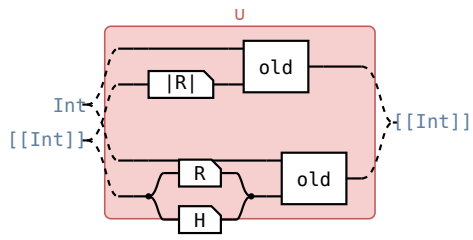
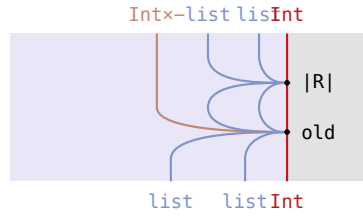
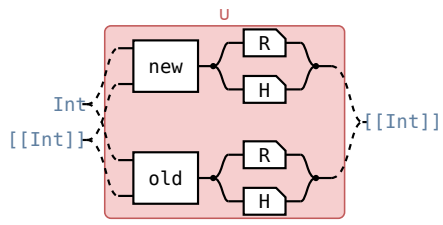
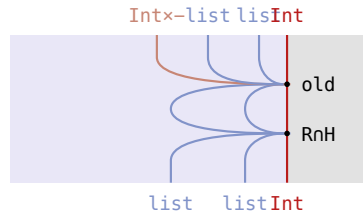
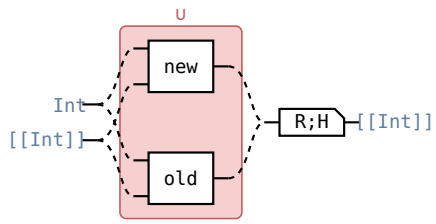
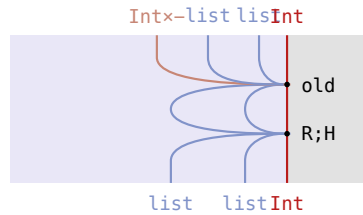
the fewest secure segments the transactions can be cut into are one pass along them, the next transaction glued onto the open segment wherever that segment stays secure and the van called where it does not

circuit	Hinze–Marsden
 $\frac{\texttt{partition list(secure)}}{\ni}$ `est(R)` the specification — ≜ `secure`	
 = $\frac{\texttt{⦇S⦈}}{\ni}$ `est(R)` `partition list(secure)=⦇S⦈` — ≜ `secure`, fusion at `secure prefix⊑prefix secure`	
 ⊑ $\frac{\texttt{⦇S⦈}}{\ni}$ `est(R;H)` `R;H⊑R` — ≜ `secure`; (7.15) `(𝟙×R)old⊑(new ∪ old)R` is FALSE, the shorter partition need not stay secure, where (7.14) `(𝟙×R)new⊑(new ∪ old)R` holds,	
 ⊑ $\frac{\texttt{S}}{\ni}$ `est(R;H)⦈` greedy theorem at `R;H`, its hypothesis `F(R;H)S⊑S(R;H)` the `old` half (7.17) — (13.6c) — and the `new` half (7.16), which rests on (7.18) `(𝟙×⊤)new⊑new H`	
 ⊑ `⦇[nil,(ok→glue,new)]⦈` `old⊑new (R;H)°`: `old` returns the shorter result wherever it returns one, and `ok` is where it does	

(13.6c)

(𝟙×(R;H))old⊑(new ∪ old)(R;H)

gluing the transaction onto a better schedule for the rest gets no further than gluing it on, or calling the van, and bettering the whole schedule after,

circuit — the `old` branch of each union	**Hinze–Marsden**
	 the `old` operand of `new ∪ old`, in every row
=  `(𝟙×|R|)old ∪ (𝟙×(R∩H))old` `R;H=|R| ∪ (R∩H)` — ≜ `secure`, ∪ distributes,	
⊑  `new (R∩H) ∪ old (R∩H)` (7.19) and (7.20) on `|R|`, (7.21) on `R∩H`	
⊑  `(new ∪ old)(R;H)` `X∩Y⊑X;Y`, converses	

§14. Thinning Algorithms

§14.1. Thinning

For $Q : A \to A$, $\text{thin } Q \triangleq (\ni/\ni) \cap (\in \backslash (Q^\circ \in)) : EA \to EA$ (8.1). (14.1a)

$ys\ (\text{thin } Q)\ xs \iff xs \subseteq ys \wedge (\forall a \in ys.\ \exists b \in xs.\ b\ Q\ a)$

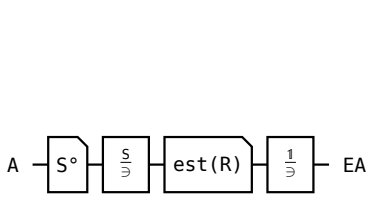
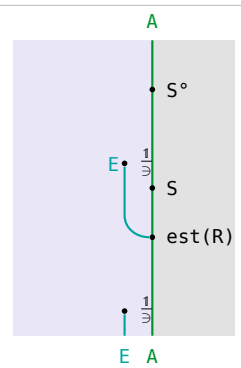
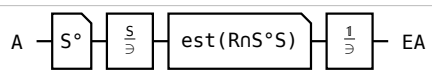
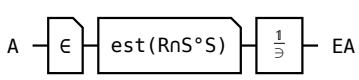
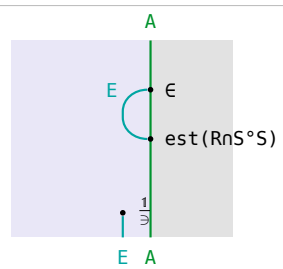
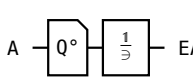
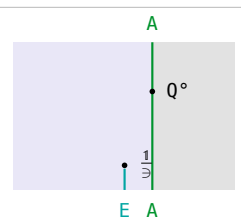
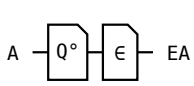
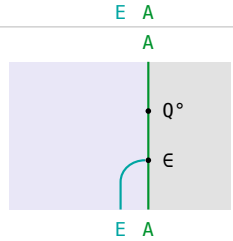
(14.1b)

the law	what it says
$X \sqsubseteq \frac{S}{\ni}\ \text{thin } Q \iff X \ni \sqsubseteq S$ and $S^\circ X \sqsubseteq Q^\circ \in$	everything kept is an S-value, and every S-value has a Q-lower bound among the kept ones
$Q \sqsubseteq R \implies \text{thin } Q \sqsubseteq \text{thin } R$	the fewer pairs Q relates, the fewer subsets count as thinnings
$\mathbb{1} \sqsubseteq \text{thin } Q$, and $\text{thin } Q$ is a preorder if Q is	keeping everything is always a legal thinning
$\text{est}(R) = \text{thin } Q\ \text{est}(R)$ $Q \sqsubseteq R$, both preorders	**thin-introduction**: thinning first cannot lose an R-minimum
$\text{thin } Q \sqsupseteq \text{est}(Q)\ \frac{\mathbb{1}}{\ni}$ (8.2)	**thin-elimination**: keeping one element is a thinning, but its domain is the sets $\text{est}(Q)$ is defined on
$\frac{S}{\ni}\ \text{thin } Q \sqsupseteq \frac{S}{\ni}\ \text{est}(R)\ \frac{\mathbb{1}}{\ni}$ (8.3), $R \cap (S^\circ S) \sqsubseteq Q$ — (14.1c)	the usable variant: R need only refine Q between values S gives one argument
$\text{union thin } Q \sqsupseteq P(\text{thin } Q)\ \text{union}$ (8.4)	thinning each member set is a thinning of the union

(14.1c)

$\frac{S}{\ni}\ \text{thin } Q \sqsupseteq \frac{S}{\ni}\ \text{est}(R)\ \frac{\mathbb{1}}{\ni}$

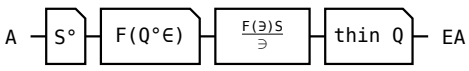
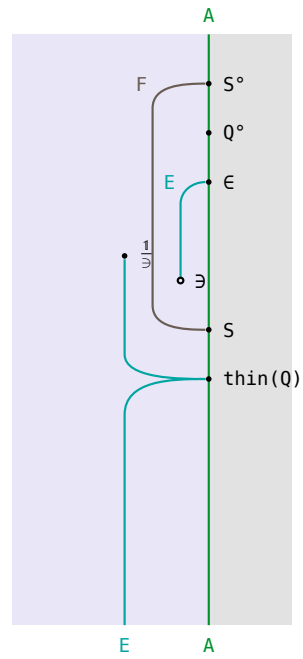
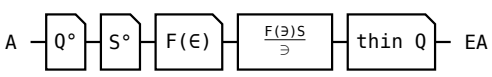
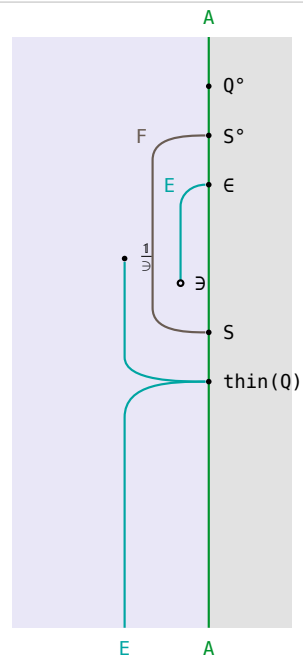
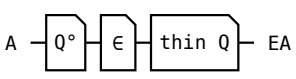
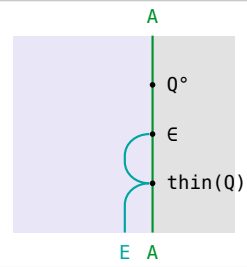
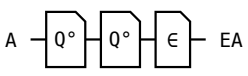
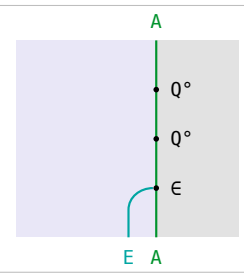
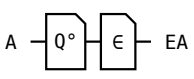
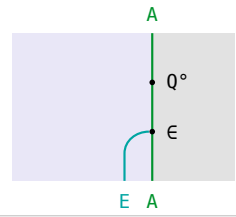
keeping one R-least of the S-values is a thinning, once R refines Q between the values S gives one argument — $R \cap (S^\circ S) \sqsubseteq Q$, Q a preorder

circuit — one wire, A to EA	**Hinze–Marsden**
$\frac{S}{\ni}\ \text{est}(R)\ \frac{\mathbb{1}}{\ni} \ni \sqsubseteq S$ and $S^\circ \frac{S}{\ni}\ \text{est}(R)\ \frac{\mathbb{1}}{\ni} \sqsubseteq Q^\circ \in$ (14.1b) at $X \triangleq \frac{S}{\ni}\ \text{est}(R)\ \frac{\mathbb{1}}{\ni}$	
$\Leftarrow$  the first is $\frac{\mathbb{1}}{\ni} \ni = \mathbb{1}$ — (10b) — then $\frac{S}{\ni}\ \text{est}(R) = S \cap (S^\circ \backslash R^\circ) \sqsubseteq S$ — (13.1b); the second is this chain, $- \sqsubseteq Q^\circ \in$	
$=$  $\frac{S}{\ni}\ \text{est}(R) = \frac{S}{\ni}\ \text{est}(R \cap S^\circ S)$ — (13.1b)	
$\sqsubseteq$  $S^\circ \frac{S}{\ni} \sqsubseteq \in$, since $\frac{S}{\ni} \ni = S$ with $\frac{S}{\ni}$ a map — (10b)	
$\sqsubseteq$  $\in\ \text{est}(R \cap S^\circ S) \sqsubseteq (R \cap S^\circ S)^\circ$ — (13.1.1a) at $X \triangleq \text{est}(R \cap S^\circ S)$, conversed — then $R \cap (S^\circ S) \sqsubseteq Q$	
$\sqsubseteq$  $\frac{\mathbb{1}}{\ni} \sqsubseteq \in$, since $\frac{\mathbb{1}}{\ni} \ni = \mathbb{1}$ with $\frac{\mathbb{1}}{\ni}$ a map — (10b)	

(14.1d)

$(\!|\frac{F(\ni)S}{\ni}$ thin $Q|\!)\sqsubseteq(\!|\frac{(\!|S|\!)}{\ni}$ thin Q

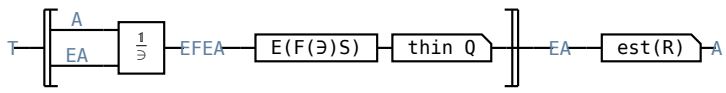
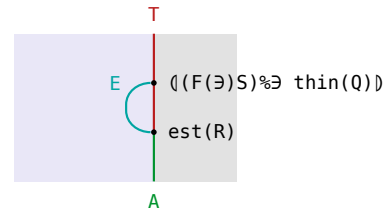
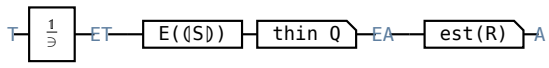
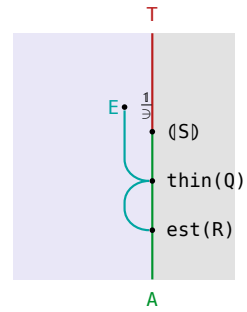
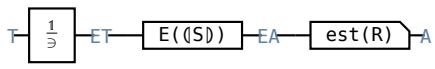
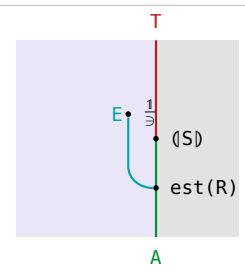
thinning at every step of the reduce is a thinning of the whole candidate set — S monotonic on Q, Q a preorder

circuit — one wire, A to EA	**Hinze–Marsden**
$(\!\|\frac{F(\ni)S}{\ni}$ thin $Q\|\!)\ni\sqsubseteq(\!\|S\|\!)$ and $(\!\|S\|\!)^\circ(\!\|\frac{F(\ni)S}{\ni}$ thin $Q\|\!)\sqsubseteq Q^\circ\in$ (14.1b) at $X\triangleq(\!\|\frac{F(\ni)S}{\ni}$ thin $Q\|\!)$, $(\!\|S\|\!)$ for its S	
$\Leftarrow$  the first by fusion; (11.6.4b) at the bound $Q^\circ\in$ reduces the second to $-\sqsubseteq Q^\circ\in$	
$\sqsubseteq$  $S^\circ F(Q^\circ)\sqsubseteq Q^\circ S^\circ$ — (13.3b) at S, conversed; $F(R)^\circ=F(R^\circ)$ — (11b)	
$\sqsubseteq$  $S^\circ F(\in)\frac{F(\ni)S}{\ni}\sqsubseteq\in$, since $\frac{F(\ni)S}{\ni}\ni=F(\ni)S$ with $\frac{F(\ni)S}{\ni}$ a map — (10b)	
$\sqsubseteq$  $\in$ thin $Q\sqsubseteq Q^\circ\in$, the $\in\backslash(Q^\circ\in)$ half of $\triangleq$ thin — (1.1a)	
$=$  $Q^\circ Q^\circ=Q^\circ$, Q a preorder	

(14.1e)

$$(\!|\tfrac{F(\ni)S}{\ni}\ \text{thin}\ Q|\!)\ \text{est}(R) \sqsubseteq (\!|\tfrac{S}{\ni}|\!)\ \text{est}(R)$$

the thinning fold refines the optimisation problem itself — S monotonic on Q, Q⊑R, both preorders

circuit — one wire, T to A	**Hinze–Marsden**
	
⊑  thinning theorem	
=  est(R)=thin Q est(R) — (14.1b), Q⊑R	

§14.2. Paths in a layered network

(14.2a)

F(A,X)=A+A×X, L=list⁺ with initial algebra α≜[wrap,cons] : F(A,LA)→LA.

wrapz≜⟨wrap,zero⟩, consw(a,(xs,n))=(cons(a,xs),wt(a,head(xs))+n).

cost≜(\|[wrapz,consw]\|)π₂, (\|[wrapz,consw]\|)=⟨𝟙,cost⟩, R≜cost≤cost°.

Q≜R∩(head head°), S≜F(𝟙,∋)α, $\frac{F(\ni,\mathbb{1})}{\ni}$ =𝟙+cpl, $\frac{F(\mathbb{1},\ni)}{\ni}$ =𝟙+cpr, step≜cpr P(cons) est(R).

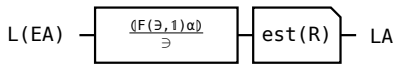
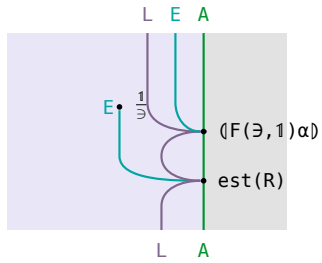
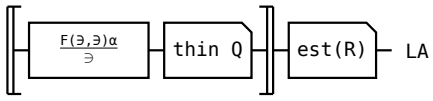
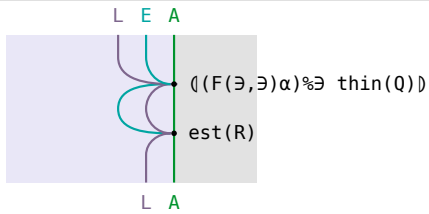
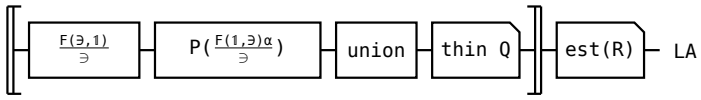
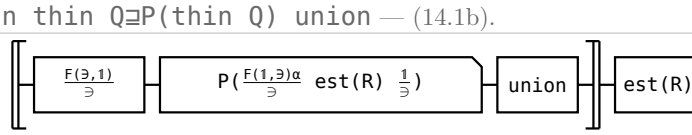
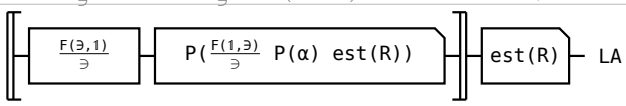
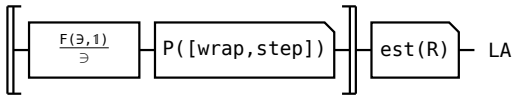
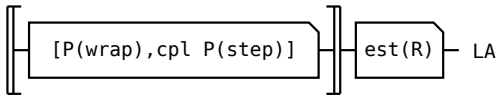
(14.2b)

the law	**what it says**
F(∋,Q)α⊑F(∋,𝟙)αQ	F(∋,𝟙)α is monotonic on Q; on R it is not, since the next edge can cost arbitrarily much
S head⊑[𝟙,π₁] S≜F(𝟙,∋)α	S head is simple, which gives R∩(S°S)⊑Q: between two paths S builds from one argument, equal cost and equal head already means Q

(14.2c)

$\frac{(\!|F(\ni,\mathbb{1})\alpha|\!)}{\ni}$ `est(R)`$\sqsupseteq$`(|[P(wrap),cpl P(step)]|) est(R)`

a least-cost path in a layered network, as a fold over the layers

circuit — one wire, `L(EA)` to `LA`; the algebra inside the functorial box	**Hinze–Marsden**
 $= \frac{L(\ni)}{\ni}$ `est(R)`	
$\sqsupseteq$  (14.1e), at `F(∋,𝟙)α` monotonic on `Q` — (14.2b).	
$=$  `F(∋,∋)=F(∋,𝟙)F(𝟙,∋)`; $\frac{F(\ni,\mathbb{1})F(\mathbb{1},\ni)\alpha}{\ni}=\frac{F(\ni,\mathbb{1})}{\ni}$ `P(`$\frac{F(\mathbb{1},\ni)\alpha}{\ni}$`) union`.	
$\sqsupseteq$  `union thin Q`$\sqsupseteq$`P(thin Q) union` — (14.1b).	
$\sqsupseteq$ $\frac{S}{\ni}$ `thin Q`$\sqsupseteq\frac{S}{\ni}$ `est(R)` $\frac{\mathbb{1}}{\ni}$ — (14.1b) at `S`$\triangleq$`F(𝟙,∋)α`, `R∩(S°S)`$\sqsubseteq$`Q` — (14.2b).	
$=$  `P(`$\frac{\mathbb{1}}{\ni}$`) union=`$\mathbb{1}$; `α` a map, so $\frac{F(\mathbb{1},\ni)\alpha}{\ni}=\frac{F(\mathbb{1},\ni)}{\ni}$ `P(α)`.	
$=$  $\frac{F(\mathbb{1},\ni)}{\ni}$ `P(α) est(R)=[wrap,step]` — $\triangleq$ `cost`	
$=$  $\frac{F(\ni,\mathbb{1})}{\ni}$ `=`$\mathbb{1}$`+cpl` — $\triangleq$ `cost`	

§14.3. Implementing thin

(14.3a)

`setify : [A]→EA`, `cup : EA×EA→EA`, `cp(F)≜ F(∋)/∋`, `listcp(F) : F([A])→[FA]`, `sort P≜setify° ordered P` for `P` a connected preorder.

`thinlist Q` is any `thinlist Q⊑subseq` with `thinlist Q setify⊑setify thin Q`; one is `⦇[nil,bump Q]⦈`, `bump Q (a,[])=[a]`, `bump Q (a,[b]#xs)=(b Q a→[a]#xs,a Q b→[b]#xs,[a]#[b]#xs)`.

Binary thinning data: `S=(f₁p₁) ∪ (f₂p₂)` with `p₁`, `p₂` coreflexive; `Q` a preorder with `Q⊑R` and both `f₁p₁`, `f₂p₂` monotonic on `Q`; `P` a connected preorder with both `f₁`, `f₂` monotonic on `P`; `gᵢ≜list(fᵢ) filter(pᵢ)`.

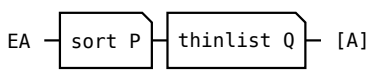
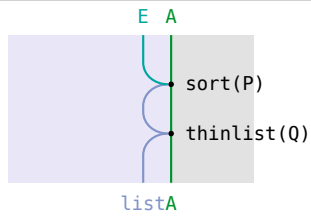
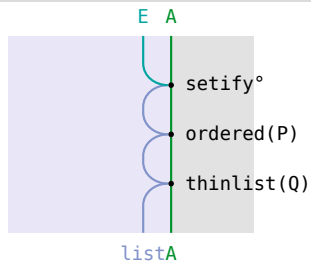
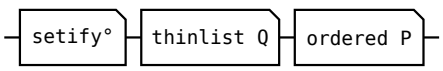
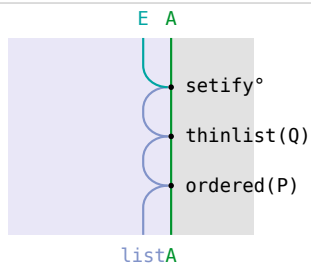
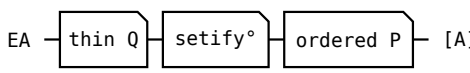
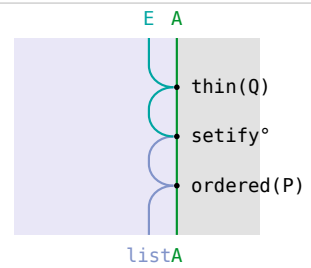
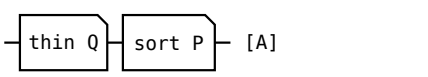
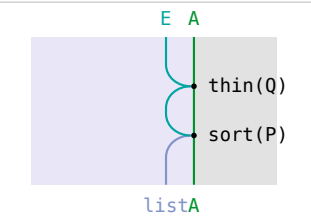
(14.3b)

the law	what it says
`thinlist Q xs=[minlist Q xs]` (8.5), `Q` connected, `xs` non-empty	what thinning should come to when it can: one element
`sort P thinlist Q⊑thin Q sort P` (8.6) — (14.3c)	thinning a sorted list is a thinning of the set — this is what `thinlist Q⊑subseq` buys
`sort P minlist Q⊑est(Q)` (8.7)	a minimum of the sorted list is a minimum of the set
`sort(fPf°) list(f)⊑P(f) sort P` (8.8)	shunt a function through a sort
`sort P filter(p)⊑E(p) sort P` (8.9), `p` coreflexive	filtering a sorted list sorts the restricted set
`(sort P×sort P) merge P⊑cup sort P` (8.10)	merging two sorted lists sorts their union
`F(sort P) listcp(F)⊑cp(F) sort(FP)` (8.11), `F` linear	`listcp(F)` is the list implementation of the cartesian product `cp(F)`

(14.3c)

`sort P thinlist Q⊑thin Q sort P`

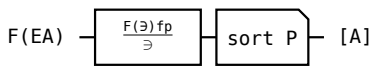
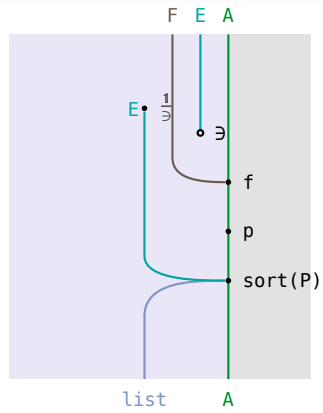
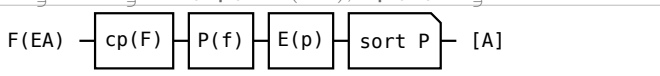
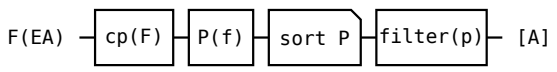
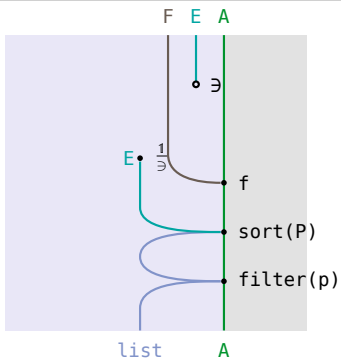
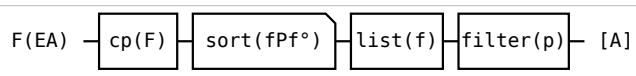
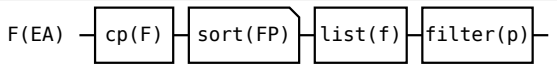
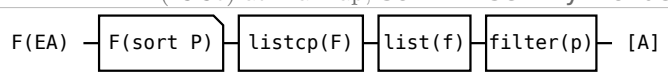
a thinning of the sorted list lists a thinning of the set — `P` a connected preorder, `thinlist Q⊑subseq`.

circuit — one wire, `EA` to `[A]`	**Hinze–Marsden**
	
= `sort P≜setify° ordered P` — ≜ `thinlist`	
⊑  `ordered P thinlist Q⊑thinlist Q ordered P`, since `thinlist Q⊑subseq` — ≜ `thinlist` — and a subsequence of a `P`-ordered list is `P`-ordered	
⊑  ≜ `thinlist`'s `thinlist Q setify⊑setify thin Q` after `setify°`, at `setify°setify⊑𝟙` for `setify` simple — (5c) — then `·setify⊣·setify°` — (1.3a)	
=  `sort P≜setify° ordered P` — ≜ `thinlist`	

(14.3d)

$\frac{\mathsf{F(\ni)fp}}{\ni}$ **sort P⊒F(sort P) listcp(F) list(f) filter(p)**

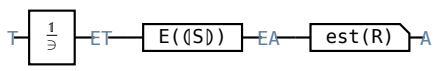
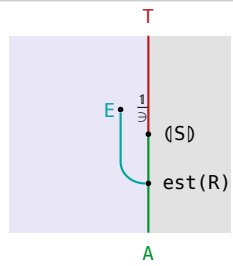
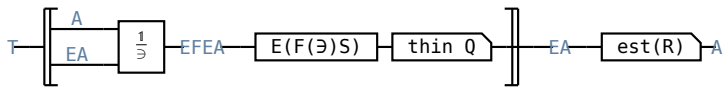
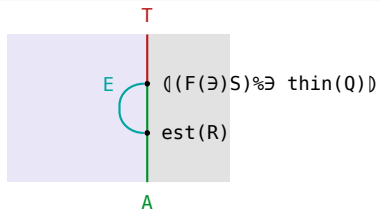
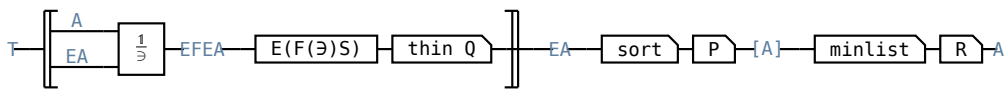
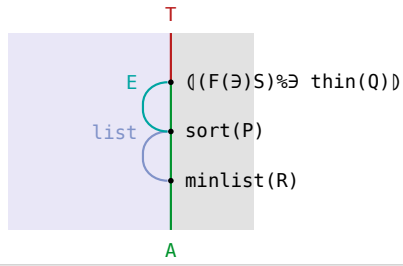
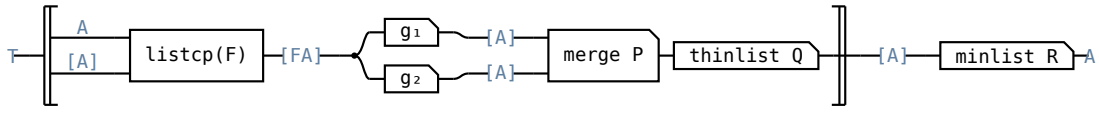
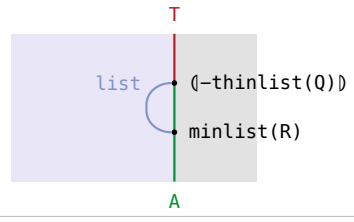
one sorted list built from sorted arguments, instead of a set built and then sorted — **f : FA→A** monotonic on **P**, **p** coreflexive, **F** linear.

circuit — one wire, F(EA) to [A]	**Hinze–Marsden**
	
= $\frac{\mathsf{F(\ni)fp}}{\ni}$ $=\frac{\mathsf{F(\ni)}}{\ni}$ E(fp) — (10b); cp(F)$\triangleq\frac{\mathsf{F(\ni)}}{\ni}$ — ≜ thinlist	
=  E(fp)=E(f)E(p); E(f)=P(f) for f a map — (11.3d)	
⊑  sort P filter(p)⊑E(p) sort P — (14.3b)	
⊑  sort(fPf°) list(f)⊑P(f) sort P — (14.3b)	
⊑  FP⊑fPf° — (13.3b) at f a map; sort P≜setify° ordered P grows with P — ≜ thinlist	
⊑  F(sort P) listcp(F)⊑cp(F) sort(FP) — (14.3b), F linear	

(14.3e)

$\frac{(\!|S|\!)}{\ni}$ est(R)⊒(⦇listcp(F) ⟨g₁,g₂⟩ merge P thinlist Q⦈ minlist R

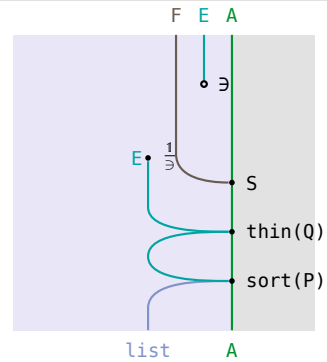
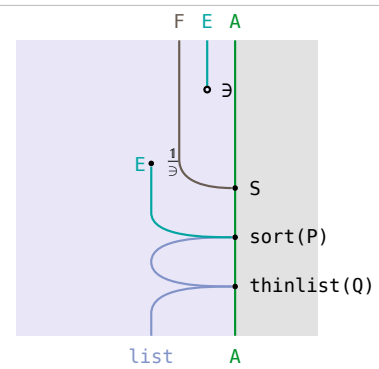
a fold on sorted lists of partial solutions, thinned at every step — at ≜ **thinlist**'s binary thinning data.

circuit — one wire, T to A	**Hinze–Marsden**
	
⊑  (14.1e) at f₁p₁ and f₂p₂ monotonic on Q — ≜ thinlist	
⊑  sort P minlist R⊑est(R) — (14.3b) at its Q≜R	
⊑  fusion at (14.3f)	

(14.3f)

$\frac{F(\ni)S}{\ni}$ thin Q sort P⊒F(sort P) listcp(F) ⟨g₁,g₂⟩ merge P thinlist Q

sorting the candidate set is what turns the thinning algebra into an algebra on lists — the side condition of binary thinning theorem's last step.

circuit — one wire, F(EA) to [A]	**Hinze–Marsden**
F(EA) — $\frac{F(\ni)S}{\ni}$ — thin Q — sort P — [A]	
⊑ F(EA) — $\frac{F(\ni)S}{\ni}$ — sort P — thinlist Q — [A] sort P thinlist Q⊑thin Q sort P — (14.3b)	
F(EA) — $\langle\frac{F(\ni)f_1p_1}{\ni},\frac{F(\ni)f_2p_2}{\ni}\rangle$ — cup — sort P — thinlist Q — [A] = $\langle\frac{F(\ni)f_1p_1}{\ni},\frac{F(\ni)f_2p_2}{\ni}\rangle$ cup sort P thinlist Q S=(f₁p₁) ∪ (f₂p₂) — ≜ thinlist, then ≜ cup	
F(EA) — $\langle\frac{F(\ni)f_1p_1}{\ni},\frac{F(\ni)f_2p_2}{\ni}\rangle$ — sort P×sort P — merge P — thinlist Q — [A] ⊑ $\langle\frac{F(\ni)f_1p_1}{\ni},\frac{F(\ni)f_2p_2}{\ni}\rangle$(sort P×sort P) merge P thinlist Q (sort P×sort P) merge P⊑cup sort P — (14.3b)	
F(EA) — $\langle\frac{F(\ni)f_1p_1}{\ni}$ sort P,$\frac{F(\ni)f_2p_2}{\ni}$ sort P$\rangle$ — merge P — thinlist Q — [A] = $\langle\frac{F(\ni)f_1p_1}{\ni}$ sort P,$\frac{F(\ni)f_2p_2}{\ni}$ sort P$\rangle$ merge P thinlist Q ⟨X,Y⟩(sort P×sort P)=⟨X sort P,Y sort P⟩ — (11.1.2b)	
F(EA) — F(sort P) — listcp(F) — ⟨g₁,g₂⟩ — merge P — thinlist Q — [A] ⊑ F(sort P) listcp(F) ⟨g₁,g₂⟩ merge P thinlist Q (14.3d) at f₁, p₁ and at f₂, p₂, then X⟨g₁,g₂⟩⊑⟨Xg₁,Xg₂⟩ — (11.1.2b); gᵢ≜list(fᵢ) filter(pᵢ) — ≜ thinlist	

§14.4. The knapsack problem

(14.4a)

vol,wt : Item→Real, value≜list(vol) sum, weight≜list(wt) sum .

subseq=⦇[nil,cons] ∪ [nil,π₂]⦈ , , within w the coreflexive on xs with weight xs≤w, 0≤w.

R≜value≥value° , Q≜R∩(weight≤weight°) , P≜R , .

FA=1+Item×A, listcp(F)=wrap+cpr, g₁≜list([nil,cons]) filter(within w) =[list(nil),h₁], g₂≜list([nil,π₂])=[list(nil),h₂].

h₁≜list(cons) filter(within w), h₂≜list(π₂).

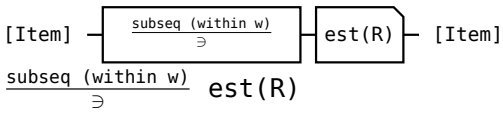
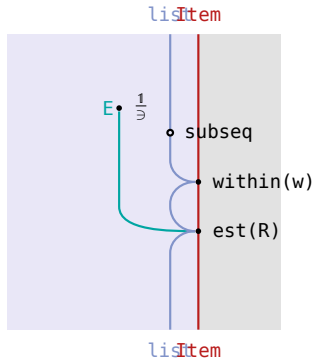
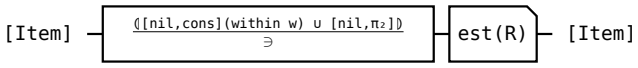
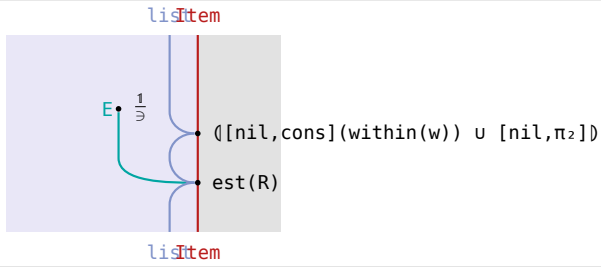
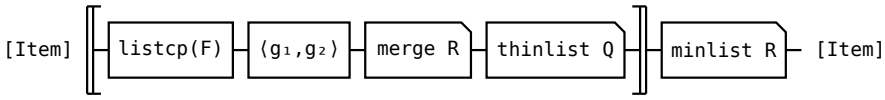
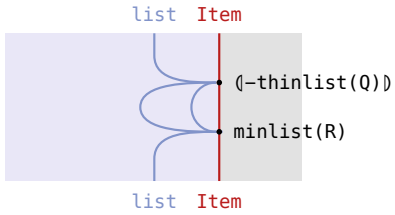
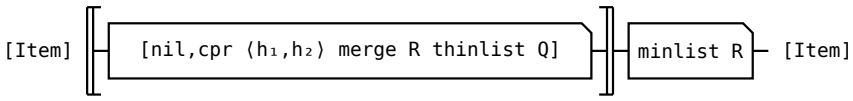
(14.4b)

the law	what it says
(𝟙×R) (cons (within w))⊑cons (within w)R FALSE	not monotonic on R: a selection of greater value need not still fit once one more item goes in
(𝟙×Q) (cons (within w))⊑cons (within w)Q (𝟙×Q)π₂⊑π₂Q ,	both halves are monotonic on Q once ties in value are broken by weight

(14.4c)

$\frac{\texttt{subseq (within w)}}{\ni}$ `est(R)`$\sqsupseteq$`⦇[nil,cpr ⟨h₁,h₂⟩ merge R thinlist Q]⦈ minlist R`

the knapsack problem, as a fold that thins the packings kept at each item

circuit — one wire, `[Item]` to `[Item]`; the algebra inside the functorial box	**Hinze–Marsden**
 $\frac{\texttt{subseq (within w)}}{\ni}$ `est(R)`	
$=$  $\frac{\texttt{⦇[nil,cons](within w) ∪ [nil,π₂]⦈}}{\ni}$ `est(R)` fusion, weights non-negative.	
$\sqsupseteq$  `⦇listcp(F) ⟨g₁,g₂⟩ merge R thinlist Q⦈ minlist R` binary thinning theorem, at P≜R, F linear, Q from (14.4b)	
$=$  `⦇[nil,cpr ⟨h₁,h₂⟩ merge R thinlist Q]⦈ minlist R` `listcp(F)=wrap+cpr`, `gᵢ=[list(nil),hᵢ]` — ≜ `within`; `minlist R` is `head`, packings coming out in descending value	

§14.5. The paragraph problem

(14.5a)

Line=list⁺ Word, Para=list⁺ Line, FA=Word+Word×A, listcp(F)=wrap+cpr.

new(a,xs)=[[a]]⧺xs, glue(a,xs)=[[a]⧺head(xs)]⧺tail(xs), partition≜⦇[wrap wrap,new ∪ glue]⦈ : list⁺ Word→Para.

width≜⦇[length,(length×𝟙) plus succ]⦈, 0≤length a, fits w the coreflexive on a line x with width x≤w, ok w the coreflexive on [x]⧺xs with width x≤w.

white w x=w−width x, collect≜list(sqr) sum, waste w≜init list(white w) collect.

R≜(waste w)≤(waste w)° , Q≜R∩(head head°) , P≜T , .

g₁≜list([wrap wrap,new]), g₂≜list([wrap wrap,glue]) filter(ok w), start≜wrap wrap wrap, h₁≜list(new), h₂≜list(glue) filter(ok w).

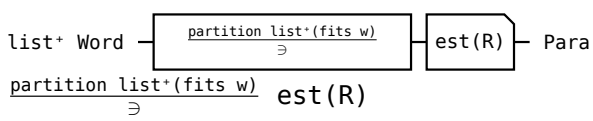
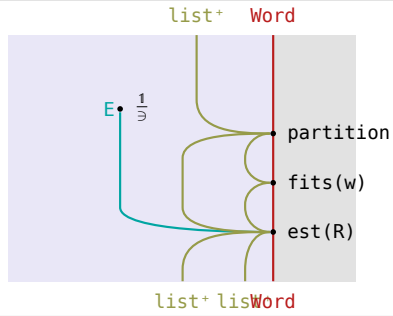
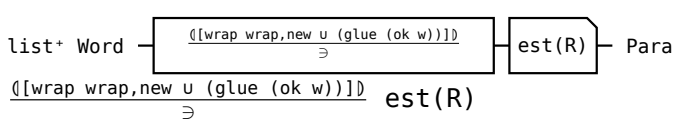
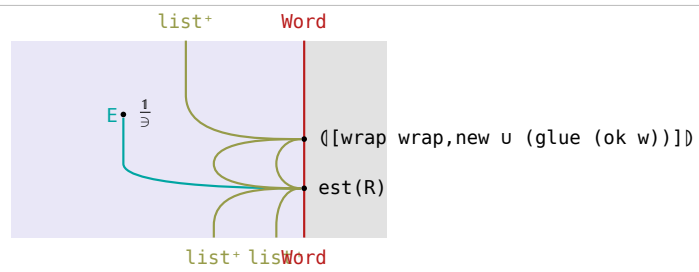
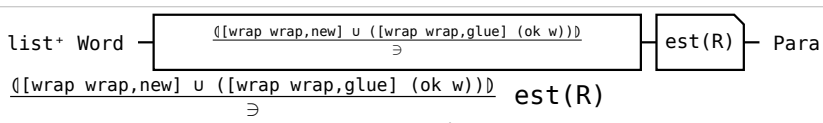
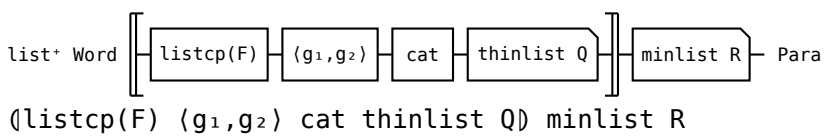
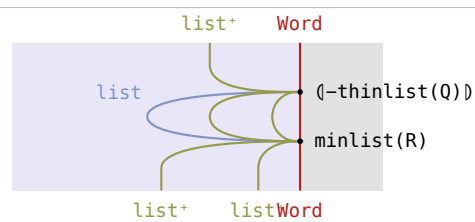
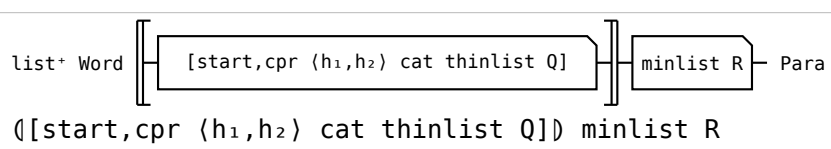
(14.5b)

the law	what it says
(𝟙×R) glue⊑glue R FALSE	glue is not monotonic on R: the waste of a paragraph depends on its whole first line, so no greedy algorithm solves this
(𝟙×Q) new⊑new Q (𝟙×Q) (glue (ok w))⊑glue (ok w)Q cons monotonic on collect≤collect°. ,	both halves are monotonic on Q once ties in waste are broken by the first line
merge T=cat; P≜head prefix head° also serves	T needs no sorting at all, and prefix is a linear order on first lines of paragraphs of one input

(14.5c)

$\frac{\text{partition list}^+\text{(fits w)}}{\ni}$ est(R)⊒⦇[start,cpr ⟨h₁,h₂⟩ cat thinlist Q]⦈ minlist R

a paragraph laid out as a fold that thins the layouts kept at each word

circuit — one wire, list⁺ Word to Para; the algebra inside the functorial box	Hinze–Marsden
	
=  fusion, every word fits on a line by itself.	
=  the algebra as (f₁p₁) ∪ (f₂p₂), p₁≜𝟙 — ≜ thinlist.	
⊑  binary thinning theorem, at P≜T with merge T=cat, Q from (14.5b)	
=  listcp(F)=wrap+cpr, gᵢ along the coproduct — ≜ partition	

§14.6. Bitonic tours

(14.6a)

FA=(City×City)+(City×A), the base functor of cons-lists of length at least two; listcp(F)=wrap+cpr; tc : City×City→Real, neither positive nor symmetric.

start (a,b)=([a,b],[a,b]), dropl (a,([b]+xs,ys))=([a]+xs,[a]+ys), dropr (a,(xs,[b]+ys))=([a]+xs,[a]+ys), tour≜⦇[start,dropl ∪ dropr]⦈.

cost (xs,ys)=outcost xs+incost ys, outcost $[a_0,\dots,a_n]$=tc $(a_0,a_1)+\dots+$tc (a_{n-1},a_n), incost $[a_0,\dots,a_n]$=tc $(a_1,a_0)+\dots+$tc (a_n,a_{n-1}).

next≜tail head, next2≜next×next, head2≜head×head, R≜cost≤cost°, Q≜R∩(next2 next2°)∩(head2 head2°), P≜T, g_1≜list([start,dropl]), g_2≜list([start,dropr]).

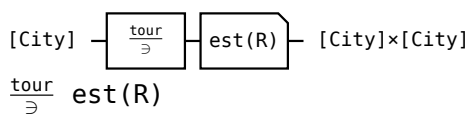
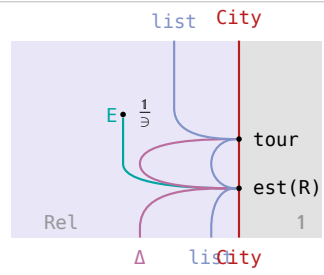
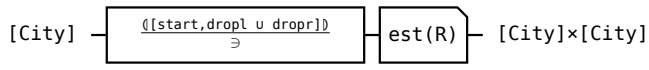
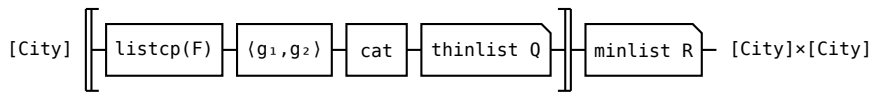
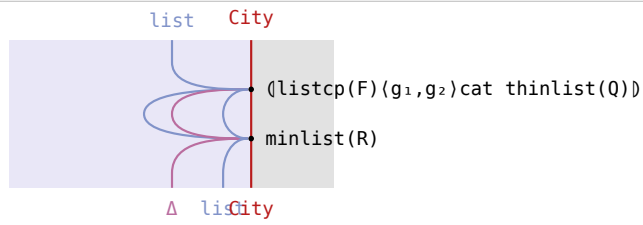
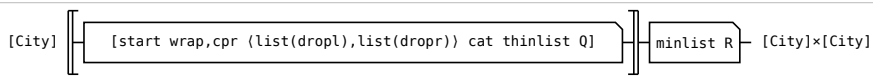
(14.6b)

the law	what it says
(𝟙×R) dropl⊑dropl R FALSE (𝟙×R) dropr⊑dropr R FALSE	neither drop is monotonic on R: the two edges it adds and removes depend on head and next of both lists
(𝟙×Q) dropl⊑dropl Q (𝟙×Q) dropr⊑dropr Q	both are, once ties in cost are broken by the two second cities — the heads already agree among tours of one input

(14.6c)

$\overset{\text{tour}}{\ni}$ est(R)∋⦇[start wrap,cpr (list(dropl),list(dropr)) cat thinlist Q]⦈ minlist R

a least-cost bitonic tour, as a fold that thins the tours kept at each city

circuit — one wire, [City] to [City]×[City]; the algebra inside the functorial box	**Hinze–Marsden**
 $\overset{\text{tour}}{\ni}$ est(R)	
=  $\overset{\text{⦇[start,dropl ∪ dropr]⦈}}{\ni}$ est(R) tour≜⦇[start,dropl ∪ dropr]⦈ — ≜ tour	
⊑  ⦇listcp(F) ⟨g_1,g_2⟩ cat thinlist Q⦈ minlist R binary thinning theorem, at P≜T with merge T=cat, Q from (14.6b)	
=  ⦇[start wrap,cpr ⟨list(dropl),list(dropr)⟩ cat thinlist Q]⦈ minlist R listcp(F)=wrap+cpr, g_i=[list(start),list(drop$_i$)] — ≜ tour; quadratic, two tours added per step	

§15. Dynamic Programming

§15.1. Theory

(15.1a)

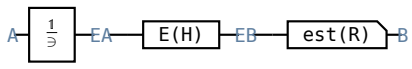
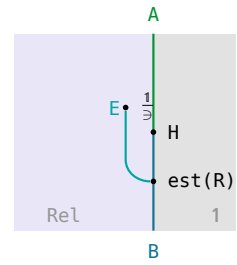
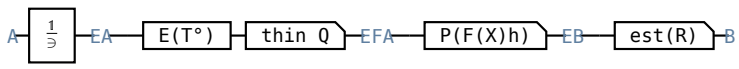
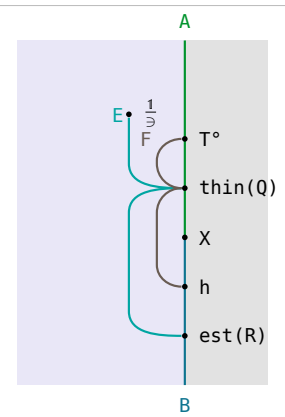
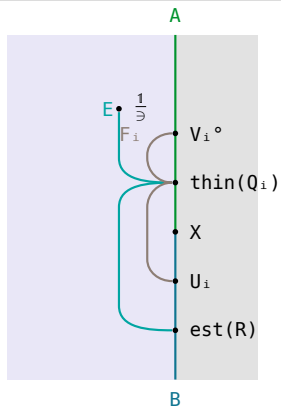
h : FB→B a map, T : FA→A an F-algebra, R : B→B.

H≜⦇T⦈°⦇h⦈ : A→B, M≜ $\frac{H}{\ni}$ est(R) the problem to be solved, (μX : G(X)) as in ≜ μ.

(15.1b)

$\frac{H}{\ni}$ est(R)⊒(μX : $\frac{T°}{\ni}$ thin Q P(F(X)h) est(R))

an optimum over everything H returns is reached by taking the input apart every way T allows, dropping the parts that can never win, solving each of the rest and keeping one optimum

circuit — one wire, A to B	**Hinze–Marsden**
 the problem to be solved, H≜⦇T⦈°⦇h⦈ — ≜ H	
 h monotonic on R and Q a preorder with QF(H)h⊑F(H)hR; thin Q as in (14.1b). This is the same with the thinning step dropped — 𝟙⊑thin Q, so the body and with it the fixed point only shrink. Knaster–Tarski leaves (9.1) $\frac{T°}{\ni}$ P(F(M)h) est(R)⊑M; M=H∩(H°\R°) splits that into (9.2) $\frac{T°}{\ni}$ P(F(M)h) est(R)⊑H and (9.3) H°$\frac{T°}{\ni}$ P(F(M)h) est(R)⊑R°, and both use only (9.4) = (7.10) P(X)est(R)⊑(∃X)∩(∈\(XR°)) — (13.1.8a)	
= Proposition 9.1 at T=[V₁,V₂], h=[U₁,U₂], Q=Q₁+Q₂, V₂V₁°=⊥: FA is usually a coproduct, and disjoint ranges split the fixed point into one branch per summand. The fixed point is unique and entire — T° followed by F's membership relation inductive, $\frac{T°}{\ni}$ finite and non-empty, R connected	

(15.1c)

the condition	**how it is discharged**
F(R)h⊑hR Proposition 9.2, R≜cost≤cost°, h cost=F(cost)k, F(≤)k⊑k≤	monotonicity when the cost is itself a fold with a step k monotonic on ≤
F(R∩(H°H))h⊑hR Proposition 9.3, R≜cost≤cost°, h cost=F(⦇cost,H°⦈)k, F(≤×𝟙)k⊑k≤, H° simple;	monotonicity **in context**: k may also read the input the part was built from
QF(H)h⊑F(H)hR at Q≜F(U,V) Proposition 9.4, U, V preorders, F(U,R)h⊑hR, VH⊑HR	both conditions at once, split along the two arguments of a bifunctor

§15.2. The string edit problem

(15.2a)

Op::=cpy Char|del Char|ins Char, F(A,B)=1+(A×B), α≜[nil,cons].

edit≜⦇[base,step]⦈ : [Op]→[Char]×[Char] , base returning ([],[]), step (cpy a,(xs,ys))=([a]⧺xs,[a]⧺ys), step (del a,(xs,ys))=([a]⧺xs,ys), step (ins a,(xs,ys))=(xs,[a]⧺ys).

length≜⦇[zero,π₂ succ]⦈ , R≜length≤length° , U≜⊤, V≜suffix°×suffix° , Q≜𝟙+(U×V) , empty the coreflexive on (xs,ys) with both lists empty.

unstep implements $\frac{\text{step}°}{\ni}$ thin(U×V) : unstep ([a]⧺xs,[])=[(del a,(xs,[]))], unstep ([],[b]⧺ys)=[(ins b,([],ys))], unstep ([a]⧺xs,[b]⧺ys)=(a=b→[(cpy a,(xs,ys))],[(del a,(xs,[b]⧺ys)),(ins b,([a]⧺xs,ys))]).

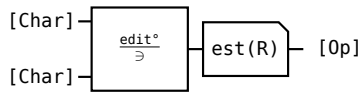
(15.2b)

$\frac{\texttt{edit°}}{\ni}$ `est(R)∋mle, mle=(empty→nil,unstep list((𝟙×mle)cons) minlist R)`

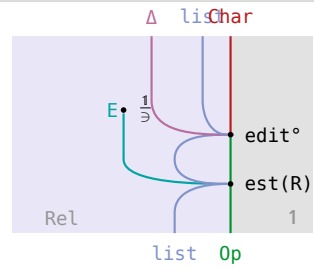
a shortest edit sequence from which both strings can be reconstituted is one pass over the two of them, each step copying, deleting or inserting one character and the best sequence for what is left taken from the entries already computed

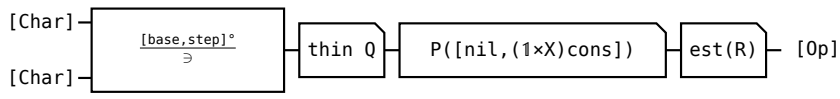
circuit — two wires, one per string

Hinze–Marsden

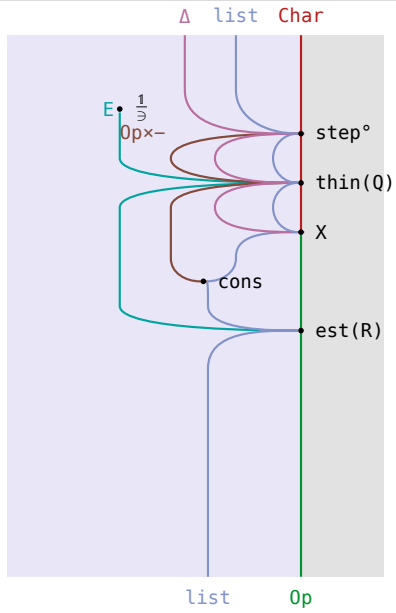


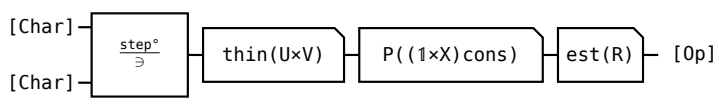
the specification — ≜ `edit`



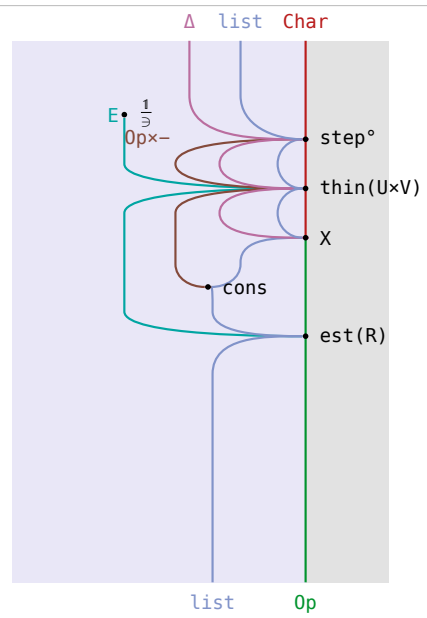


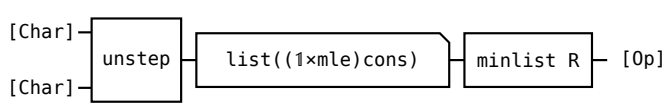
at `Q≜𝟙+(U×V)`. Monotonicity `F(R)α⊑αR` is Proposition 9.2 at `length≜⦇[zero,π₂ succ]⦈` with `succ` monotonic on ≤, so `cons` is monotonic on `R`. The thinning condition `QF(𝟙,edit°)α⊑F(𝟙,edit°)αR` over `F(Op,[Char]×[Char])` is Proposition 9.4 at `U≜⊤` — `F(⊤,R)α⊑αR`, left as an exercise, so any two operations may be compared — and `V≜suffix°×suffix°` , — `edit (𝟙×suffix)⊑R° edit` , `edit (suffix×𝟙)⊑R° edit` . `suffix=tail*` and `BA⊑CB⟹BA*⊑C*B` cut those to one step each: drop the operation that produced the head, or weaken its `cpy` to a `del`, never lengthening the sequence



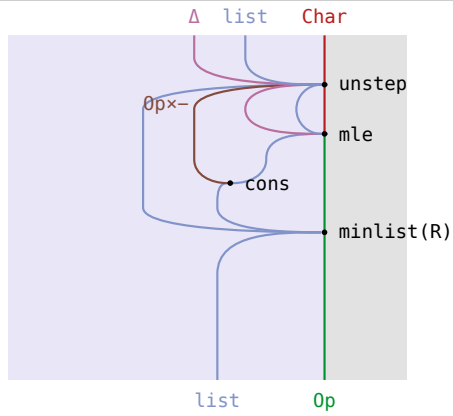


Proposition 9.1: `base` and `step` have disjoint ranges, so the fixed point splits into one branch per summand — `empty`, the coreflexive on `(xs,ys)` with both lists empty, is where `base` returns





`unstep` implements $\frac{\texttt{step°}}{\ni}$ `thin(U×V)` — at most two decompositions survive, a copy beating a delete or an insert wherever it is available — and `minlist R` implements `est(R)`. The same subproblem is solved many times over, so the running time is exponential in the two lengths




```
mle(xs,ys)=head(column(xs,ys)),
column(xs,ys)=[mle(u,ys)|u←tails(xs)]
column(xs)=⦇[fstcol(xs),nextcol(xs)]⦈, fstcol=list(del) tails
```

the tabulation: `mle(xs,ys)` needs `mle(u,v)` for every tail `u` of `xs` and `v` of `ys`, so the columns are built right to left

```
column(xs)([b]#ys)=nextcol(xs)(b,column(xs)(ys))
nextcol(xs)(b,us)=⦇[base(b,last(us)),step(b)]⦈(xus),
xus=zip(xs,zip(init(us),tail(us)))
base(b,u)=[[ins(b)]#u]
step(b)((a,(u,v)),ws)=(a=b→[[cpy(a)]#v]#ws, [bmin(R)([del(a)]#w,
[ins(b)]#u)]#ws)
w=head(ws)
```

; these `base`, `step` are not `edit`'s. An entry depends on the one below it (a delete), the one to its right (an insert), and the one below that (a copy) — quadratic in the two lengths

§15.3. Optimal bracketing

(15.3a)

tree A::=tip A|bin (tree A,tree A), FX=A+X², so F(R)=𝟙+R²; h≜[tip,bin], flatten≜⦇[wrap,cat]⦈ : tree A→list⁺ A , H=flatten°.

⟨cost,size⟩≜⦇[opt,opb]⦈, opt≜⟨zero,st⟩, opb ((cx,sx),(cy,sy))=(cb (sx,sy)+cx+cy,sb (sx,sy)).

sb associative, so size=flatten sz for a map sz R≜cost≤cost° , g≜[zero,(𝟙×sz)² opb π₁], single the coreflexive on singleton lists.

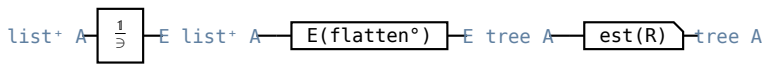
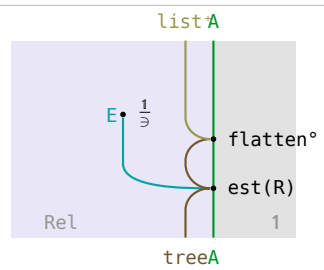
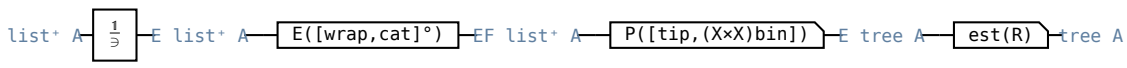
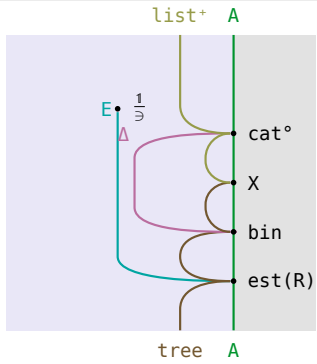
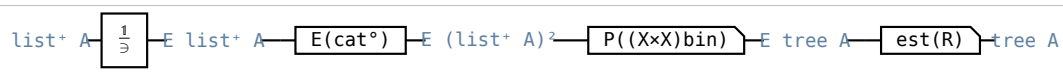
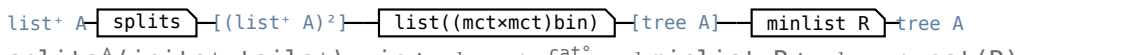
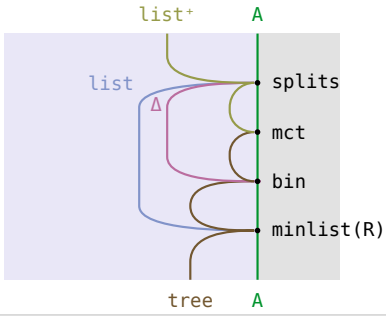
splits≜⟨inits⁺,tails⁺⟩ zip, an implementation of $\frac{\text{cat}^\circ}{\ni}$; array≜inits list(row), row≜tails list(mct), col≜inits list(mct).

mix≜zip list(bin) minlist R, next≜⟨π₁,mix⟩ snoc, process≜((tip wrap)×𝟙) loop(next).

(15.3b)

$\frac{\text{flatten}^\circ}{\ni}$ **est(R)⊇mct, mct=(single→head tip,(init col,tail row) mix)**

a least-cost bracketing of $a_1 \oplus \cdots \oplus a_n$ is read off an array holding one best tree per non-empty segment, each entry built from the column to its left and the row below it

circuit — one wire, list⁺ A to tree A	**Hinze–Marsden**
 the specification — ≜ flatten	
 split the list in every way, bracket both halves, join. The condition is monotonicity **in context**, ⊒ F(R∩(flatten flatten°))h⊑hR — only trees with the same flattening are compared — which is Proposition 9.3 at H°=flatten, a map, with (9.5) [tip,bin] cost=(𝟙+⟨cost,flatten⟩²)g (the cost of a node reads only the cost and the flattening of its two subtrees) and (9.6) (𝟙+(≤×𝟙)²)g⊑g≤ (g monotonic on ≤ in its two cost arguments)	
=  Proposition 9.1: wrap and cat have disjoint ranges, and single is the coreflexive on singleton lists, where wrap returns	
⊒  splits≜⟨inits⁺,tails⁺⟩ zip implements $\frac{\text{cat}^\circ}{\ni}$ and minlist R implements est(R). Exponential, since the segments of one list overlap	
= mct=(single→head tip,⟨init col,tail row⟩ mix) (9.7) the tabulation: mct xs is needed for every non-empty segment xs, so the values are held as an array of rows, array≜inits list(row), row≜tails list(mct), col≜inits list(mct), mix≜zip list(bin) minlist R	
= col=(single→head tip wrap,⟨init col,tail row⟩ next) (9.8) cons col=(𝟙×array) process (9.9) row=(single→head tip wrap,⟨mct,tail row⟩ cons) (9.10) a column extends the column to its left, a row the row below it; (9.9) is (9.8) rewritten as a loop, next≜⟨π₁,mix⟩ snoc, process≜((tip wrap)×𝟙) loop(next)	
= array=⦇[fstcol,addcol]⦈, fstcol≜tip wrap wrap addcol≜⟨π₁ tip wrap,step⟩ cons, step≜⟨process tail,π₂⟩ zip list(cons) the program: one fold building the array column by column, cubic in the length of the input	

§15.4. Data compression

(15.4a)

`[A]::=nil|snoc ([A],A)`, `list⁺ A::=wrap A|snoc (list⁺ A,A)`, `String=[Char]`, `Code::=sym Char|ptr (String,String⁺)`, `F(Code,String)=1+(String×Code)`, `α≜[nil,snoc]`.

`decode≜⦇[nil,extend]⦈ : [Code]→String`, `extend (xs,sym a)=xs#[a]`, `extend (xs,ptr (ys,zs))=xs#zs` when `ys#zs` is a proper prefix of `xs#zs`; `H=decode°`.

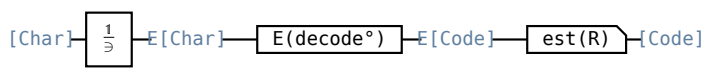
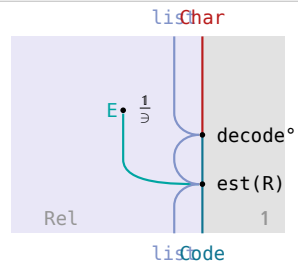
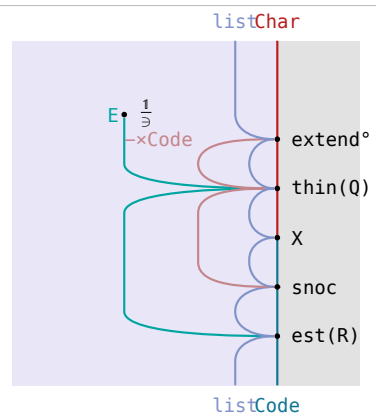
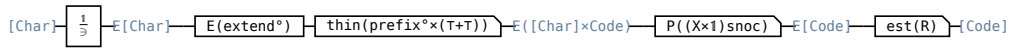
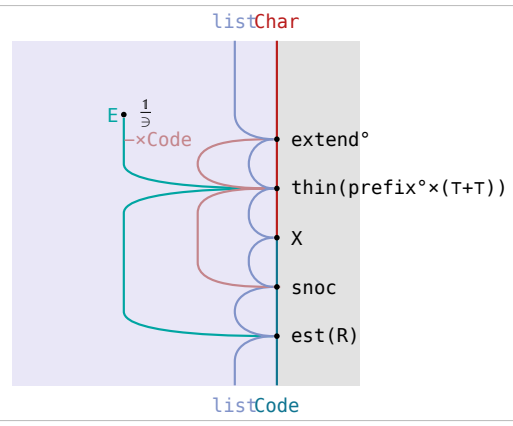
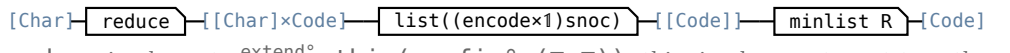
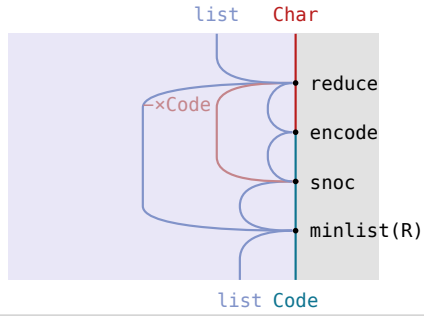
`size≜⦇[zero,distr [𝟙×c,𝟙×p] plus]⦈` with `c`, `p` the constant costs of a symbol and a pointer; `R≜size≤size°`, `Q≜F(T+T,prefix°)=𝟙+(prefix°×(T+T))`, the two `T` on symbols and on pointers.

`lrt ws=est(prefix°×(T+T)) {(xs,(ys,zs))|xs#zs=ws, ys#zs` a proper prefix of `ws}`, the longest repeated tail; `reduce (ws#[a])=(zs≠[]→[(ws,sym a),(xs,ptr (ys,zs))],[(ws,sym a)])` with `(xs,(ys,zs))=lrt (ws#[a])`.

(15.4b)

$\frac{\texttt{decode°}}{\ni}$ `est(R)⊒encode`, `encode=(null→nil,reduce list((encode×𝟙)snoc) minlist R)`

a smallest code sequence decoding to the given string is built from the right, each step emitting the last character as a symbol or ending with a pointer back into what has already been decoded

circuit — one wire, `String` to `[Code]`	**Hinze–Marsden**
 the specification — ≜ `decode`	
at `Q≜F(T+T,prefix°)=𝟙+(prefix°×(T+T))`, the two `T` on symbols and on pointers. Monotonicity `F(R)α⊑αR` is routine, the two costs being constants: `(T+T)[c,p]=[c,p]`. ⊒ Proposition 9.4 splits the thinning condition in two — `F(T+T,R)α⊑αR`, and `decode prefix⊑R° decode`, for which `decode init⊑R° decode` is enough: dropping the last character of the output shortens or removes the last code element and never raises the cost. So between a symbol and a pointer nothing can be decided in advance, and between two pointers the longer match wins	
 = Proposition 9.1: `nil` and `extend` have disjoint ranges. The decompositions of one string are $\frac{\texttt{extend°}}{\ni}$ `(ws#[a])={(ws,sym a)} ∪ {(xs,ptr (ys,zs))|xs#zs=ws#[a], ys#zs` a proper prefix of `ws}` — take the last character as a symbol, or end with a pointer	
 ⊒ `reduce` implements $\frac{\texttt{extend°}}{\ni}$ `thin(prefix°×(T+T))`: thinning leaves at most two, the symbol and the pointer of the longest repeated tail `lrt`. Again exponential; the book gives no tabulation for it	

§16. Greedy Algorithms

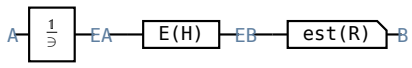
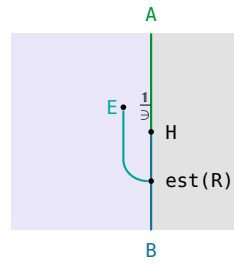
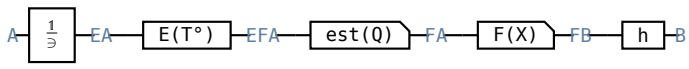
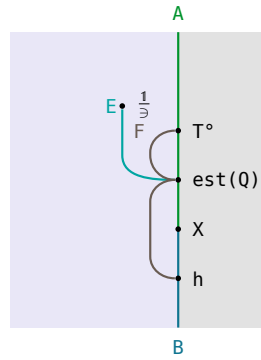
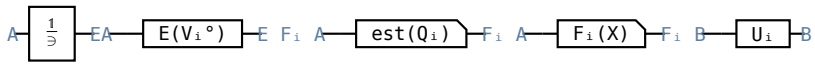
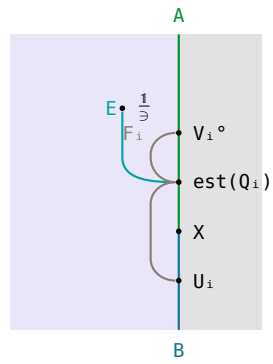
§16.1. Theory

h, T, R, H, M as in ≜ H; additionally Q a **connected** preorder on the sets $\frac{T°}{∋}$ returns, so that $\frac{T°}{∋}$ est(Q) is entire. (16.1a)

(16.1b)

$\frac{H}{∋}$ est(R)∋(μX : $\frac{T°}{∋}$ est(Q) F(X)h)

the same optimum reached by keeping ONE decomposition at each step, so that no set is ever carried and the recursion runs on values alone

circuit — one wire, A to B	**Hinze–Marsden**
 the problem to be solved, H≜⦇T⦈°⦇h⦈ — ≜ H, Q	
⊒ 	
=  Proposition 10.1 at T=[V₁,V₂], h=[U₁,U₂], Q=Q₁+Q₂, V₂V₁°=⊥	

§16.2. The detab-entab problem

detab≜⦇[nil,expand]⦈ : String→String over snoc-lists, α≜[nil,snoc], H=detab°; expand (xs,a)=(a=TB→fill xs,xs∗[a]), fill xs=xs∗blanks (n−(col xs) mod n). (16.2a)

col≜⦇[zero,count]⦈, count (c,a)=(a=NL→0,c+1); TB the tab, BL the blank, NL the newline, tab stops every n columns.

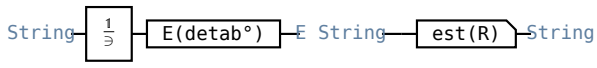
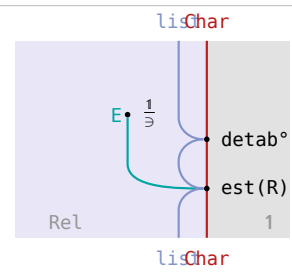
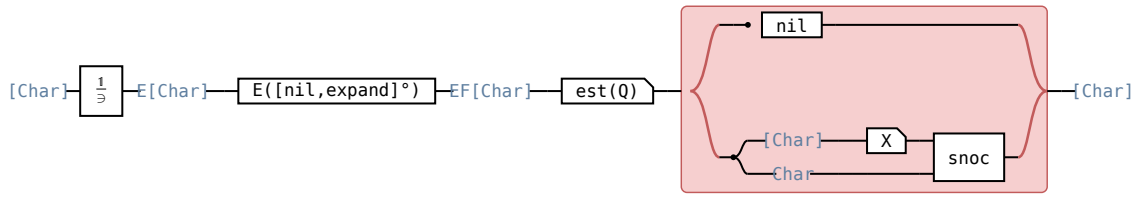
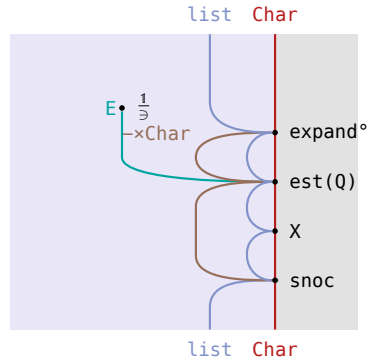
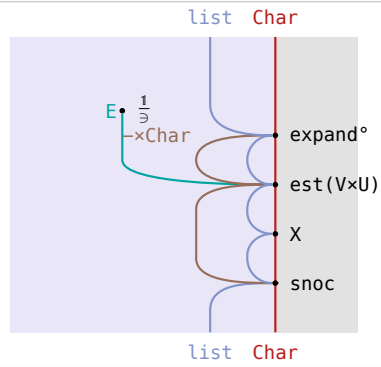
R≜length≤length° , U the preorder with a U b⟺a=TB∨a=b, V≜prefix°n(fill fill°) , Q≜𝟙+(V×U) .

unfill xs the shortest prefix of xs with fill (unfill xs)=fill xs; tbc the trailing blank count, triple≜⟨unfill entab,⟨tbc,col⟩⟩.

(16.2b)

$\frac{\texttt{detab°}}{\ni}$ `est(R)∋entab, entab=triple assocl π₁ (𝟙×blanks) cat`

the shortest input **detab** expands to the given output is one pass along that output, holding each blank back and cashing the held blanks in for a tab wherever the column reaches a tab stop

circuit — one wire, `String` to `String`	**Hinze–Marsden**
 the specification — ≜ `detab`; `detab entab=𝟙` and nothing shorter does	
 at `Q≜𝟙+(V×U)` : one character of input is decided at each step. `F(⊤,R)α⊑αR`. `detab prefix⊑R° detab` is FALSE , — at `n=8`, `detab [a,b,c,d,e,TB]=[a,b,c,d,e,BL,BL,BL]`, whose prefix `[a,b,c,d,e,BL,BL]` is longer than any input giving it, and `detab V°⊑R° detab` holds. `expand V°⊑expand ∪ (π₁V°)` — shortening the output either leaves the last step alone or discards it	
Proposition 10.1: `nil` and `expand` have disjoint ranges. The greedy step is to emit a tab whenever a tab is legal, consuming all the blanks back to the previous tab stop	
`entab xs=entab (unfill xs)⧺blanks (tbc xs)` (10.1) what makes `triple≜⟨unfill entab,⟨tbc,col⟩⟩` a snoc-list reduce: the output splits at the last tab stop	

`triple=⦇[base,op]⦈, entab=triple assocl π₁ (𝟙×blanks) cat`
`base` returns `([],(0,0))`, `op ((xs,(t,c)),a)=`
`(a=BL∧(c+1) mod n≠0→(xs,(t+1,c+1)), a=BL→(xs⧺[TB],(0,c+1)),`
`a=NL→(xs⧺blanks t⧺[NL],(0,0)), (xs⧺blanks t⧺[a],(0,c+1)))`
the program: one pass carrying the column and the count of pending blanks	

§16.3. The minimum tardiness problem

(16.3a)

`FX=1+(X×Job)`, `α≜[nil,snoc]` on schedules, `β≜[nil,snag]` on bags, `snag` putting a job into a bag; `bagify≜⦇β⦈ : [Job]→Bag Job`, `H=bagify°`.

`ct, dt, wt : Job→Real` the completion, due and weighting quantities of a job; `penalty(xs,j)=(sum(list(ct)(xs))+ct(j)−dt(j))×wt(j)`.

`cost≜` $\frac{\texttt{prefix}}{\ni}$ `P(α° [zero,penalty]) est(≥)`, `cost []=0`, `cost (xs⧺[j])=bmax (cost xs,penalty (xs,j))`, `R≜cost≤cost°`.

`perm≜bagify bagify°=⦇[nil,add]⦈`, `add (xs,j)=ys⧺[j]⧺zs` for some `xs=ys⧺zs`.

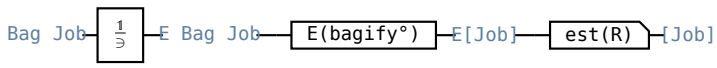
`k≜[zero,assocr (𝟙×((bagify°×𝟙) penalty)) bmax]`, `f≜[zero,(bagify°×𝟙) penalty]`, `Q≜f≤f°`, `Q'≜(bagify°×𝟙) penalty≤penalty°(bagify×𝟙)`.

(16.3b)

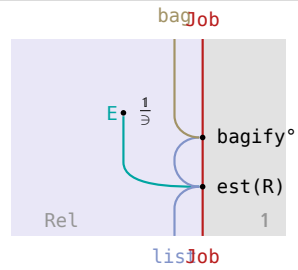
$\frac{\text{bagify}^\circ}{\ni}$ `est(R)∋schedule, schedule=(null→nil,pick (schedule×𝟙) snoc)`

an ordering of the given bag with least maximum penalty is got by taking a job of least penalty out of the bag, putting it last, and scheduling what is left the same way

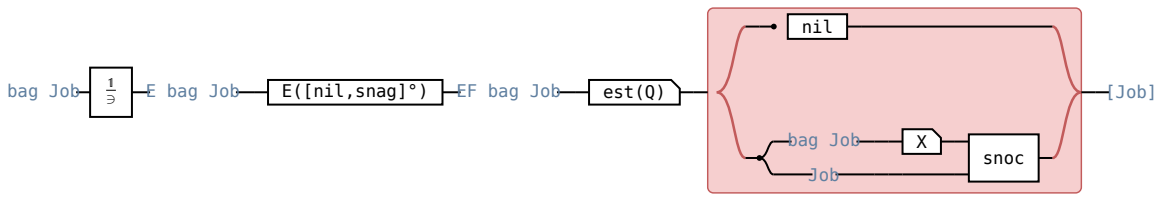
circuit — one wire, `Bag Job` to `[Job]` | **Hinze–Marsden**



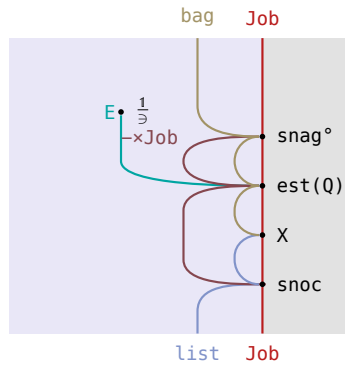
the specification — ≜ `bagify`



⊒

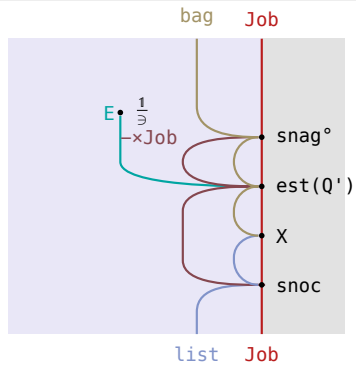


No greedy **reduce** exists — one would also solve every prefix of the input, and the best schedule of a prefix need not extend to a best schedule of the whole

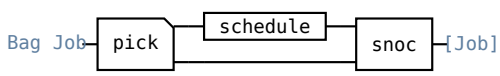


=

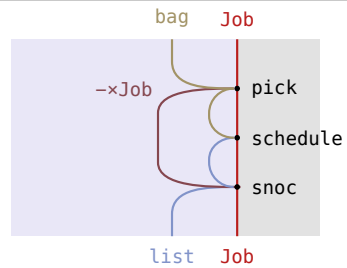
Proposition 10.1: `nil` and `snag` have disjoint ranges



⊒



`pick`$\sqsubseteq\frac{\text{snag}^\circ}{\ni}$ `est(Q')`, a partial function, quadratic in the number of jobs



§16.4. The TeX problem

(16.4a)

`intern≜val round : Decimal→[0,2¹⁶)`, `val≜⦇[zero,shift]⦈`, `shift (d,r)=(d+r)/10`, `round r` rounds `2¹⁶r` to the nearest integer: `round r=n⟺2n−1<2¹⁷r<2n+1`.

`interval n=((2n−1)/2¹⁷,(2n+1)/2¹⁷)`, `r inrange (a,b)⟺a<r<b`, `round°=interval inrange`, `R≜length≤length°`.

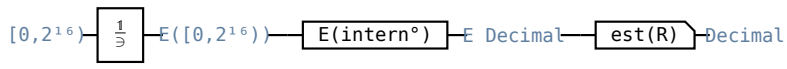
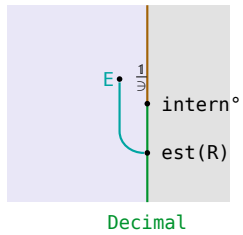
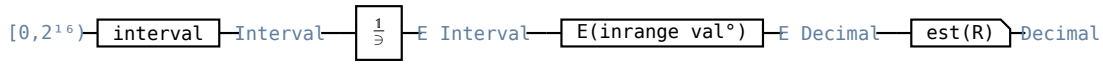
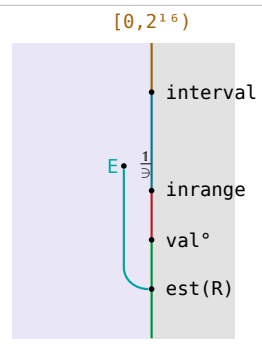
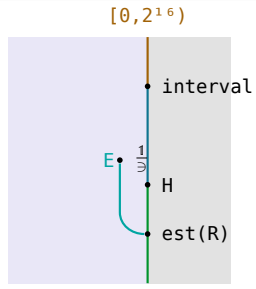
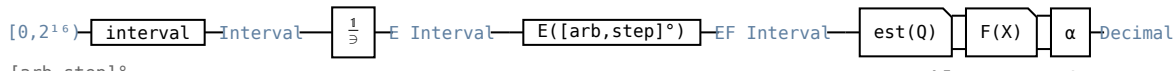
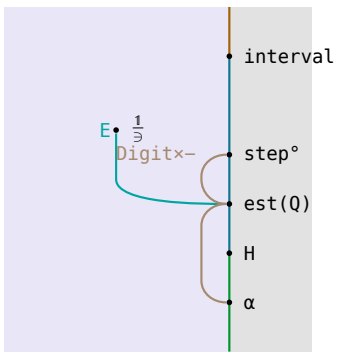
`Interval` the pairs `(a,b)` with `0<b<1` and `a<b` (10.9) `[arb,step] : 1+(Digit×Interval)→Interval`, `step (d,(a,b))=((d+a)/10,(d+b)/10)`.

`FX=1+(Digit×X)`, `α≜[nil,cons]`, `H≜⦇[arb,step]⦈°`, `! : Digit×Interval→1`, `Q≜(l°!°r) ∪ 𝟙` `l`, `r` are (11.2a)'s injections into `FX=1+(Digit×X)`, so `l : 1→F(Interval)` and `r : Digit×Interval→F(Interval)`, `w≜2¹⁷`.

(16.4b)

$\frac{\texttt{intern°}}{\ni}$ `est(R)⊒extern, extern(n)=f(2n−1,2n+1)`

a shortest decimal whose internal representation is the given multiple of 2^{-16} is got by emitting the one digit the interval of admissible reals allows, until that interval contains zero and the empty decimal will do

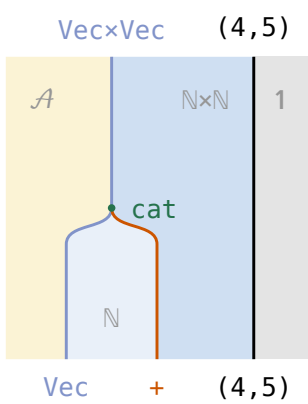
circuit — one wire, `[0,2¹⁶)` to `Decimal`	**Hinze–Marsden**
 the specification — ≜ `intern`	
=  `round°` is not a map, but `interval` is, so it comes out of the transpose	
= fusion: `val inrange°=⦇[arb,step]⦈` — the converse of `val`, cut down to intervals, is a reduce on cons-lists	
⊑  $\frac{\texttt{[arb,step]°}}{\ni}$ returns at most two elements — stop, or take one more digit — and `! nil⊑cons R°` makes it stop whenever stopping is legal	
= `extern=interval f, f(a,b)=(a<0→[],[d]+f(10a−d,10b−d))` the program, with `d` the digit above	
= `extern(n)=f(2n−1,2n+1), f(p,q)=(p≤0→[],[d]+f(10p−w·d,10q−w·d))` `d=(10q) div w`: the same in integer arithmetic only, as chapter 3 required of `intern` — every interval reached is `(p/w,q/w)`	

§17. Indexed types

(17a)

definition	type	note
`Vec`	$\mathbb{N}\to\mathcal{A}$	`n↦Vec(n,Int)`, the lists of length `n`. $\mathbb{N}$ is the discrete category — identities only — and `N` of $\triangleq$ `paths` is `Vec(4)`.
`+`	$\mathbb{N}\times\mathbb{N}\to\mathbb{N}$	Any function between discrete categories is a functor, so the index arithmetic is a wire.
`Vec×Vec`	$\mathbb{N}\times\mathbb{N}\to\mathcal{A}$	`(m,n)↦Vec(m)×Vec(n)`, pairing and product as one functor.
`cat`	`Vec×Vec⇒Vec∘+`	`cat : Vec(m)×Vec(n)→Vec(m+n)`, natural in `(m,n)`; over a discrete category naturality is empty.

(17b)



the bottom cut reads `Vec(+(4,5))`: `9` is written nowhere — the `+` wire beside `(4,5)` is it

§18. Appendix

§18.1. P(S) est(R)=(∋S)∩(∈\(SR°))

(p,q) tabulates W≜(∋S)∩(∈\(SR°)), and y≜ $\frac{(p\ni S)\cap(qR)}{\ni}$, a map. (18.1a)

(18.1b)

$$\begin{array}{c}\mathtt{p°y}\\ \hline \mathtt{P(S)}\end{array} \quad \begin{array}{c}\mathtt{y°q}\\ \hline \mathtt{est(R)}\end{array} \xLeftarrow{\mathtt{p°q=W},\ \mathbb{1}\sqsubseteq\mathtt{yy°}} \begin{array}{c}\mathtt{W}\\ \hline \mathtt{P(S)\ est(R)}\end{array}$$

(18.1c)

$$\begin{array}{c}\mathtt{p°q}\\ \hline \ni\mathtt{S}\end{array} \quad \begin{array}{c}\mathbb{1}\\ \hline \mathtt{R}\end{array}\ \text{R reflexive} \xLeftarrow{\Delta\dashv\cap,\ \mathtt{f°}\cdot\dashv\mathtt{f}\cdot} \begin{array}{c}\mathtt{q}\\ \hline \mathtt{(p\ni S)\cap(qR)}\end{array} \xLeftarrow{\mathtt{f°}\cdot\dashv\mathtt{f}\cdot,\ \cdot\ni\dashv\frac{}{\ni}} \begin{array}{c}\mathtt{y°q}\\ \hline \ni\end{array}$$

$$\begin{array}{c}\mathtt{R°q°q}\\ \hline \mathtt{R°}\end{array}\ \text{q simple} \xLeftarrow{\cdot\ni\dashv\frac{}{\ni},\ °,\ \text{meet}} \begin{array}{c}\mathtt{\in y°q}\\ \hline \mathtt{R°}\end{array}$$

(18.1d)

$$\begin{array}{c}\mathtt{p°y\ni}\\ \hline \ni\mathtt{S}\end{array} \quad \begin{array}{c}\mathtt{p\ni}\\ \hline \mathtt{y\ni S°}\end{array} \xLeftarrow{\Delta\dashv\cap,\ \cdot\mathtt{T}\dashv\mathtt{T}/,\ °,\ \mathtt{f°}\cdot\dashv\mathtt{f}\cdot} \begin{array}{c}\mathtt{p°y}\\ \hline \mathtt{P(S)}\end{array}$$

$$\begin{array}{c}\mathtt{p°p\ni S}\\ \hline \ni\mathtt{S}\end{array}\ \text{p simple} \xLeftarrow{\cdot\ni\dashv\frac{}{\ni},\ \text{meet}} \begin{array}{c}\mathtt{p°y\ni}\\ \hline \ni\mathtt{S}\end{array}$$

$$\begin{array}{c}\mathtt{W}\\ \hline \mathtt{\in\backslash(SR°)}\end{array}\ \text{W's right half} \xLeftarrow{°,\ \mathtt{T}\cdot\dashv\mathtt{T}\backslash} \begin{array}{c}\mathtt{q°p\ni}\\ \hline \mathtt{RS°}\end{array} \xLeftarrow{\mathbb{1}\sqsubseteq\mathtt{qq°}} \begin{array}{c}\mathtt{p\ni}\\ \hline \mathtt{(p\ni)\cap(qRS°)}\end{array} \xLeftarrow{\text{modular law}} \begin{array}{c}\mathtt{p\ni}\\ \hline \mathtt{y\ni S°}\end{array}$$

§18.2. P(est(R)) est(R)=P(Dom(est(R))) union est(R)

(18.2a)

$$\begin{array}{c}\mathtt{P(Dom(est(R))\ est(R))\ est(R)}\\ \hline \mathtt{Dom(est(R))\ est(R)=est(R)}\end{array} \quad \mathtt{P(est(R))\ est(R)}$$

$$\mathtt{P(Dom(est(R)))\ est(R)} \xrightarrow[\text{(13.1.9a)}]{\sqsubseteq} \mathtt{P(Dom(est(R)))\ union\ est(R)} \quad \overline{\mathtt{P}\ \text{a relator}}$$

(18.2b)

$$\begin{array}{c}\mathtt{P(Dom(est(R)))\ union\ est(R)}\\ \hline \mathtt{P(est(R))\ est(R)}\end{array} \xLeftarrow{\text{(18.1b) at } \mathtt{S:=est(R)},\ \Delta\dashv\cap,\ \mathtt{T}\cdot\dashv\mathtt{T}\backslash} \begin{array}{c}\mathtt{P(Dom(est(R)))\ union\ est(R)}\\ \hline \ni\mathtt{est(R)}\end{array}$$

$$\begin{array}{c}\mathtt{\in P(Dom(est(R)))\ union\ est(R)}\\ \hline \mathtt{est(R)R°}\end{array}$$

$$\begin{array}{c}\mathtt{P(Dom(est(R)))\ union\ est(R)}\\ \hline \ni\mathtt{est(R)}\end{array} \xLeftarrow{\mathtt{P(Dom(est(R)))}\sqsubseteq\mathbb{1}} \begin{array}{c}\mathtt{union\ est(R)}\\ \hline \ni\mathtt{est(R)}\end{array}$$

$$\begin{array}{c}\mathtt{(\ni\ni)\cap(\in\backslash(\in\backslash R°))}\\ \hline \ni\mathtt{est(R)}\end{array} \xLeftarrow{\mathtt{T(U\cap V)}\sqsubseteq\mathtt{TU\cap TV},\ \cdot\ni\dashv\frac{}{\ni},\ \text{(13.1.4a)}} \begin{array}{c}\mathtt{\ni(\ni\cap(\in(\in\backslash(\in\backslash R°))))}\\ \hline \mathtt{\ni(\ni\cap(\in\backslash R°))}\end{array}\ \text{counit of } \mathrm{T}\cdot\dashv\mathrm{T}\backslash \xLeftarrow{\text{modular law}} \begin{array}{c}\mathtt{(\ni\ni)\cap(\in\backslash(\in\backslash R°))}\\ \hline \ni\mathtt{est(R)}\end{array}$$

$$\begin{array}{c}\mathtt{Dom(est(R))\in union\ est(R)}\\ \hline \mathtt{est(R)R°}\end{array} \xLeftarrow{\in\ \text{lax natural}} \begin{array}{c}\mathtt{\in P(Dom(est(R)))\ union\ est(R)}\\ \hline \mathtt{est(R)R°}\end{array}$$

$$\begin{array}{c}\mathtt{est(R)\in est(R)}\\ \hline \mathtt{est(R)R°}\end{array}\ \text{UP of est} \xLeftarrow{\mathtt{est(R)°}\sqsubseteq\in,\ \in\in\mathtt{union}\sqsubseteq\in} \begin{array}{c}\mathtt{est(R)\ est(R)°\in union\ est(R)}\\ \hline \mathtt{est(R)R°}\end{array} \xLeftarrow{\mathtt{Dom(est(R))}\sqsubseteq\mathtt{est(R)\ est(R)°}} \begin{array}{c}\mathtt{Dom(est(R))\in union\ est(R)}\\ \hline \mathtt{est(R)R°}\end{array}$$