Relation Algebra

§1. Notation

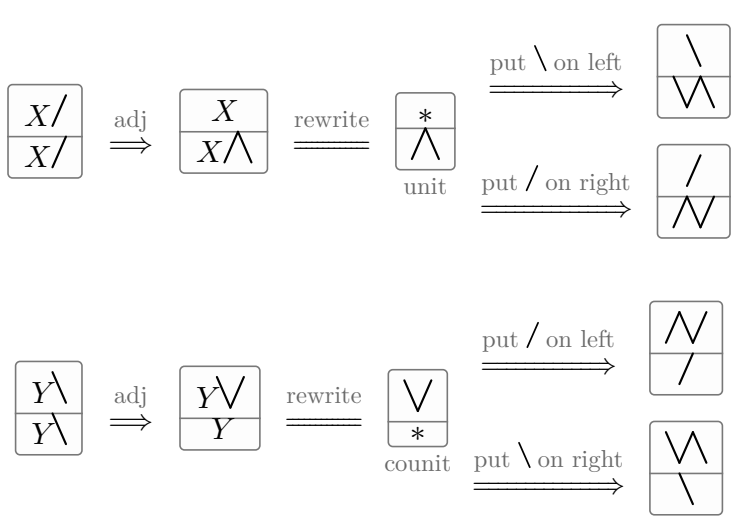
Traditionaly Adjunction F -| G is defined as $F(x) \leq y$ iff $x \leq G(y)$, and you derive laws like this:

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

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 $\bigwedge\!\!\!\bigvee = /$ and $\bigvee\!\!\!\bigwedge = \backslash$, and you just discovered string diagrams.

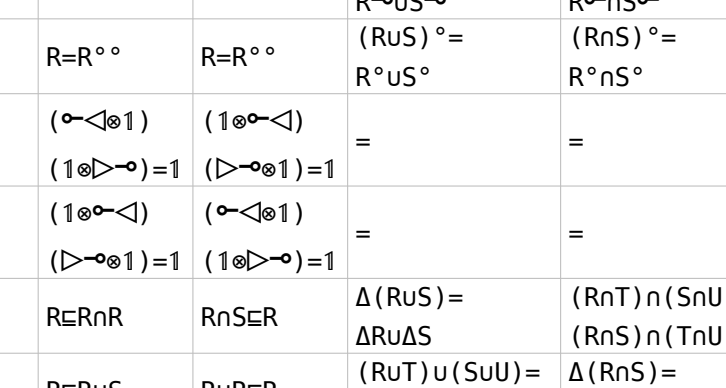
§1.1. Adjunctions

(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	⊸⊸=⊤	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
·∋⊣ $\frac{}{∋}$	Map(A,EB)→ (A→B)	𝟙	$\frac{f∋}{∋}$=f	$\frac{R}{∋}$∋=R	—	—	$\frac{\frac{f∋}{∋}}{∋}$∋ =f∋	$\frac{\frac{R}{∋}∋}{∋}$ =$\frac{R}{∋}$
$\frac{}{∋}$ ⊣·∋	(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.

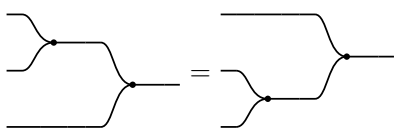
§2. Relations

(2a)

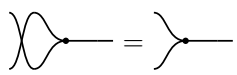
Rel is a poset-enriched category with $(\otimes, {}^\circ)$ where $\otimes$ is commutative cartisian product, and converse $\circ\dashv\circ$, and

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

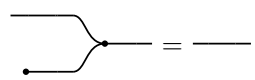
(2b)



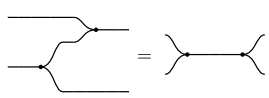
$\triangleright$ associative



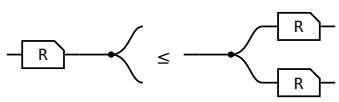
$\triangleright$ commutative



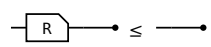
$\multimapinv$ is its unit



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



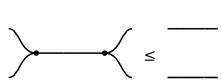
$R\triangleleft\leq\triangleleft(R\otimes R)$



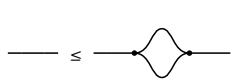
$R\multimapinv\leq\multimapinv$

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

(2.1a)



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



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

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

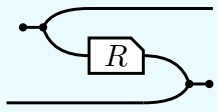
$\mathbb{1}\leq\multimap\multimapinv$

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:



$$R^\circ = (\multimap\!\lhd \otimes \mathbb{1})(\mathbb{1} \otimes R \otimes \mathbb{1})(\mathbb{1} \otimes \rhd\!\multimap)$$

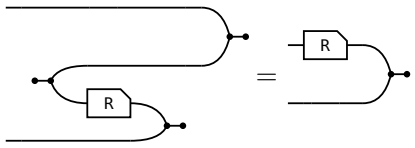
where $\multimap\!\lhd : \mathbb{I} \to a \otimes a$ opens a pair of wires out of nothing and $\rhd\!\multimap : b \otimes b \to \mathbb{I}$ closes one, so the input of R° is where the output of R was. Taking that formula as the definition would now be circular: $\lhd$ and $\multimap$ are $\rhd^\circ$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

(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)

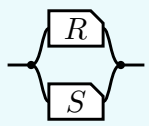


R° is DEFINED as the bending of $(R \otimes \mathbb{1}) \rhd\!\multimap$, so the slide claims only that unbending it gives that back — and unbending undoes bending for every arrow. That is the snake above with a passenger: the $\multimap\!\lhd$ bends the a strand down and the $\rhd\!\multimap$ brings 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)

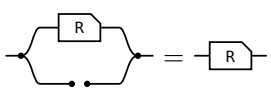
The **meet** of `R,S : a→b`, the paper's **convolution**, is `R∩S:=◁(R⊗S)▷` — copy the input, run `R` and `S` on the two copies, merge the results — so what comes out is what both of them do.



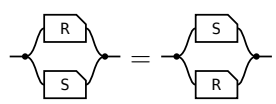
transcribed: a definition has no statement to export

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

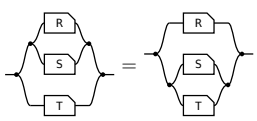
(4b)



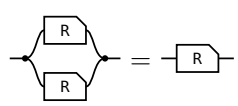
unit: one half of `T` per end — the merge's unit law absorbs the ⊶, the copy's counit law the ⊸



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

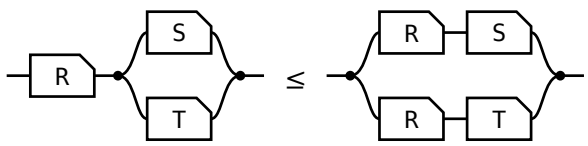


idempotent: the one that is not bookkeeping — the lax copy law is the whole of it, worked in allegory2

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)

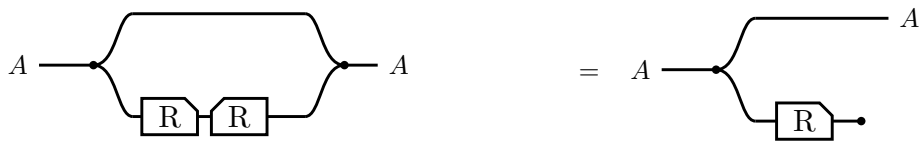
semi-distributivity, and what supplies it	**picture**
`R (S∩T)⊑RS∩RT` — the lax copy law. Equality exactly when `R` is single valued: the Maps section's `F(R∩S)=FR∩FS`.	

§5. Domain and range

(5a)

The **domain** $\mathsf{Dom}(R) \triangleq \mathbb{1} \cap RR^\circ$ and the **range** $\mathsf{Ran}\ R \triangleq \mathsf{Dom}(R^\circ)$.

(5b)



$\triangleright$ lands the return leg back on the value $\triangleleft$ handed out,
so $R^\circ \triangleright$ cuts to $\multimap$ — Frobenius

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

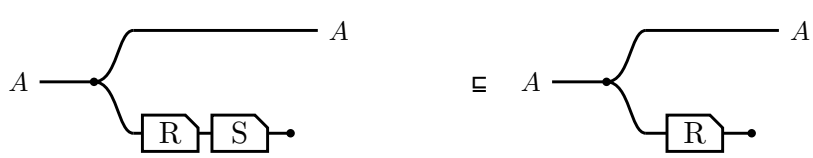
(5c)

$\mathsf{Dom}(R) \triangleq \mathbb{1} \cap RR^\circ$
$\mathsf{Dom}(R) \sqsubseteq A \Longleftrightarrow R \sqsubseteq AR$, for A coreflexive
$\mathsf{Dom}(RS) \sqsubseteq \mathsf{Dom}(R)$
$\mathsf{Dom}(R \cap S) = \mathbb{1} \cap SR^\circ$
R entire $\Longleftrightarrow \mathsf{Dom}(R) = \mathbb{1} \Longleftrightarrow \mathbb{1} \sqsubseteq RR^\circ$
R simple $\Longleftrightarrow R^\circ R \sqsubseteq \mathbb{1}$
R a map $\Longleftrightarrow R$ entire and simple
R, S entire $\Longrightarrow RS$ entire — likewise simple, likewise maps
RS entire $\Longrightarrow R$ entire

§5.1. Sliding the discard

$\mathsf{Dom}(RS) \sqsubseteq \mathsf{Dom}(R)$, 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)



$S \multimap \sqsubseteq \multimap$, the lax axiom for $\multimap$ in the first section
— the discard slides back past S

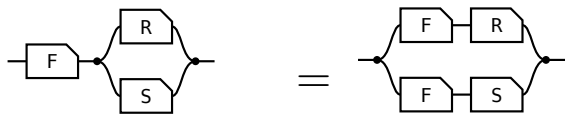
Equality is S entire, which is the same picture read as $\mathsf{Dom}(R) = \mathbb{1} \Longleftrightarrow R$ entire.

§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)

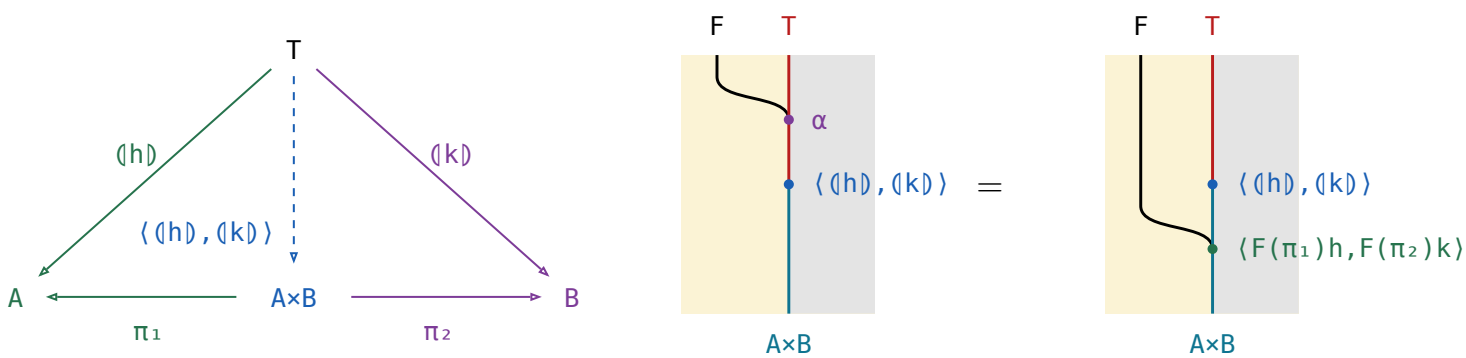
law	picture
$F(R\cap S)=FR\cap FS$ F single valued	 one person who admires both — a at A, b at B

§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 (B&dM p. 55).
`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 (11.6.1a) 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]`	B&dM Ex 3.6 (p. 57): apply Ex 3.4 to write `preds` as `⦇k⦈π₁`.

(7b)

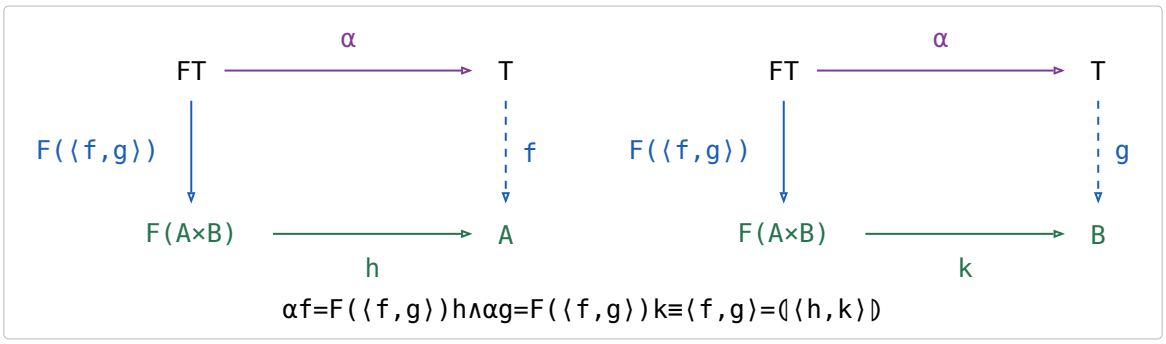


`⟨⦇h⦈,⦇k⦈⟩=⦇⟨F(π₁)h,F(π₂)k⟩⦈` banana split

§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 (7.1a) 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`; B&dM's Ex 3.4 is the other special case (p. 58). What it says: **any algebra on a product carrier is folded by a fork**.

(7.1a)



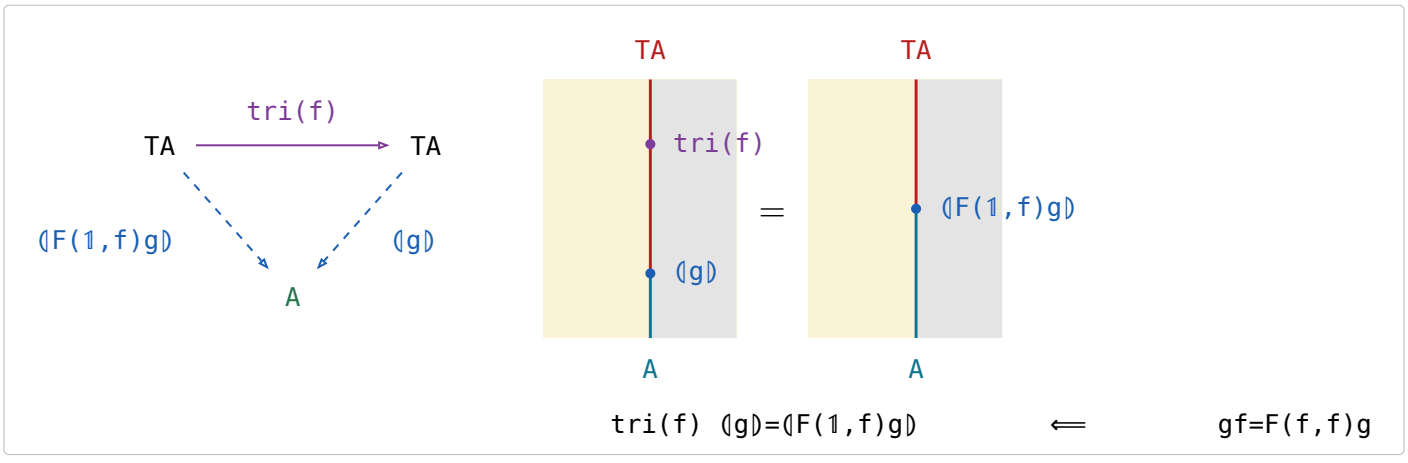
§7.2. Ruby triangles

(7.2a)

stage	definition
informally (p. 58)	`tri(f) [a₀,a₁,…,aᵢ,…,aₙ]=[a₀,f a₁,…,fⁱ aᵢ,…,fⁿ aₙ]`
for cons-lists (p. 59)	`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 (p. 59)	`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)



(7.2b) is Horner's rule, generalised from cons-lists to any initial type; B&dM call it that because for cons-lists it is the schoolbook method for evaluating a polynomial (p. 58). Fusion reduces it to `F(𝟙,T(f))α⦇g⦈=F(𝟙,⦇g⦈)F(𝟙,f)g` (p. 59).

§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`

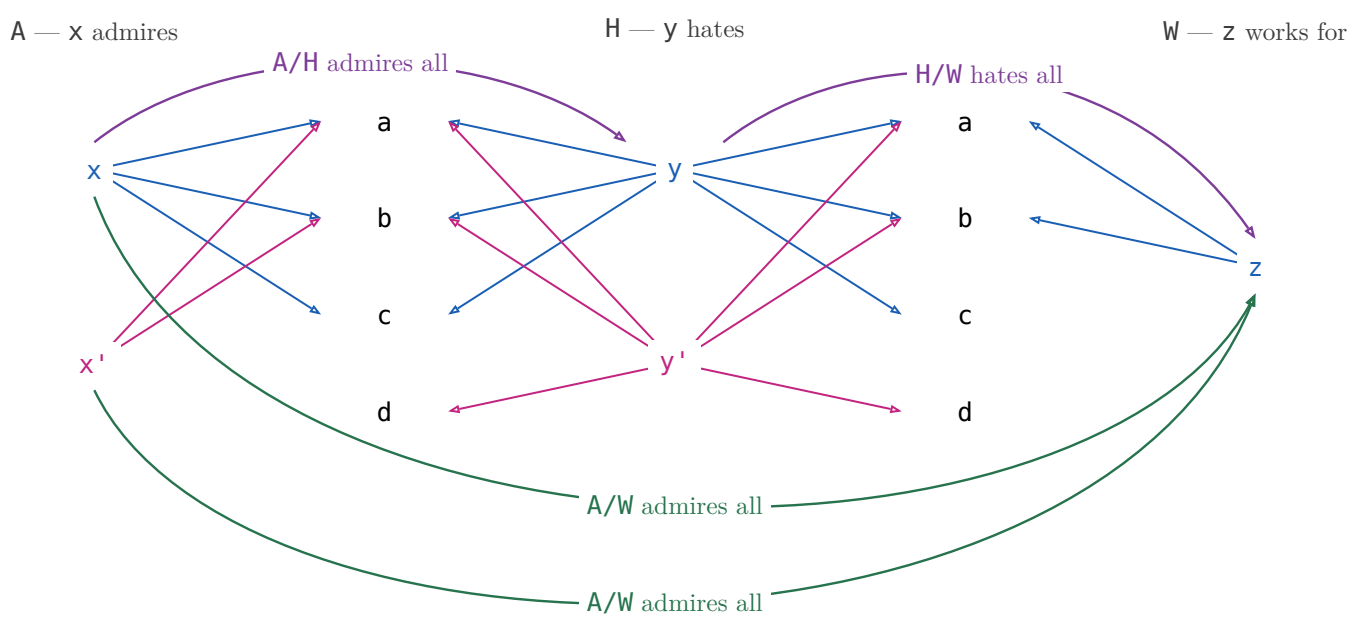
§8. / is all of

(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)

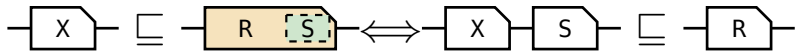
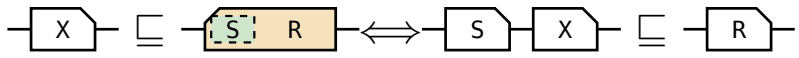
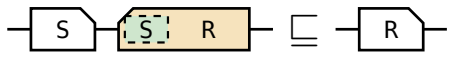
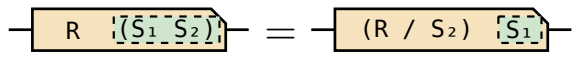
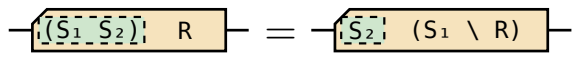
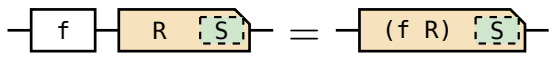
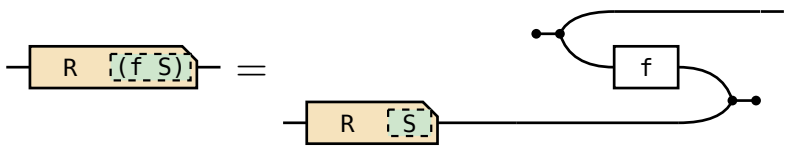
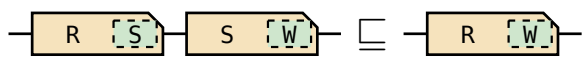
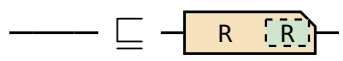
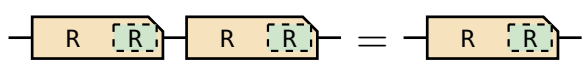
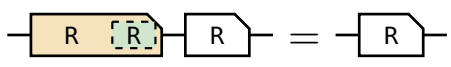
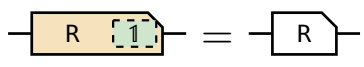
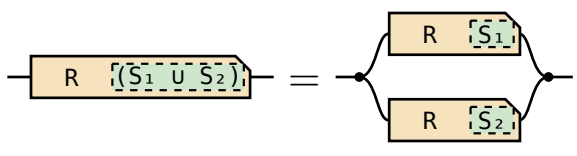
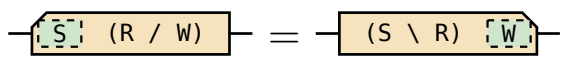


(8.1b)

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: ∩, ∪, ∘ and composition are what the Frobenius generators build, and / is none of those — it is posited, with nothing to unfold.

§9. $\frac{R}{S}$

(9a)

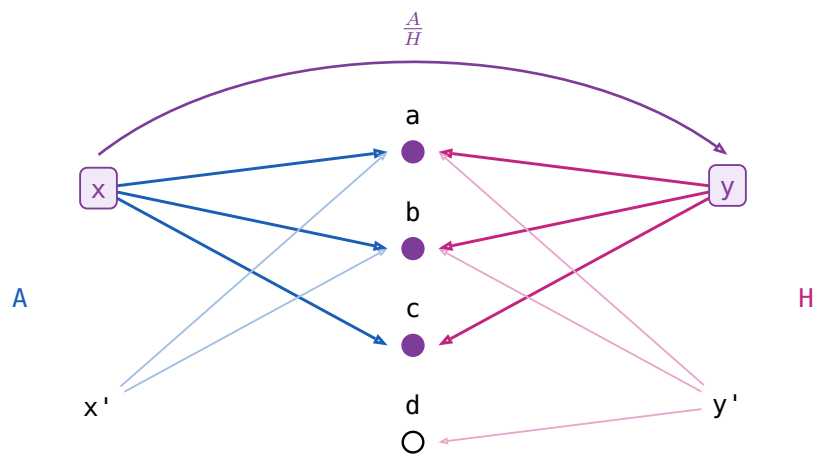
> $\frac{R}{S}$ ≜`(R/S)∩(S/R)°`. In `Rel` `x` and `y` has the same image: `∀p. (x R p⟺y S p)`

(9b)

`x (A/H) y` — `x` admires everyone `y` hates
`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

$$\begin{array}{ll} \ni_{\mathrm{R}}\square = R\square, \quad \ni_{\mathrm{R}} = \ni_{\mathrm{R}\,\square} & \\ \mathbb{1} \sqsubseteq (R/\ni_{\mathrm{R}})(\ni_{\mathrm{R}}/R) & \ni_{\mathrm{R}} \text{ is } \textbf{thick} \\ \frac{\ni_{\mathrm{R}}}{\ni_{\mathrm{R}}} = \mathbb{1} & \ni_{\mathrm{R}} \text{ is } \textbf{straight} \end{array}$$

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

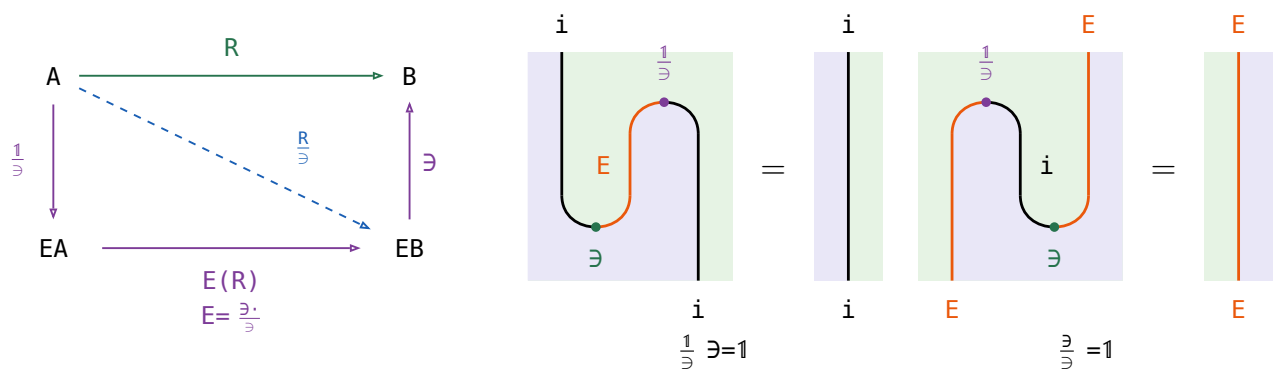
`∈≜∋° : A→EA`

(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)



$\frac{\mathsf{R}}{\ni}$ ∋=R EA is the powerset of A. B&dM write PA — standard mathematics, but here P is already the relator P(R).

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

§10.2. $\mathcal{A}$≅Kleisli(E)

(10.2a)

$$E(R)\triangleq \tfrac{\ni R}{\ni},\quad \tfrac{1}{\ni}\ :\ A\to EA,\quad \mathrm{union}=E(\ni)=\tfrac{\ni\ni}{\ni}\ :\ E(EA)\to EA$$

$$f\diamond g\triangleq f\ E(g)\ \mathrm{union}\ :\ A\to EC,\ \text{ for } f\ :\ A\to EB \text{ and } g\ :\ B\to EC$$

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

(10.2b)

$$\boxed{\tfrac{S}{\ni}\diamond\tfrac{R}{\ni}} \underset{\text{definition of }\diamond,\ \mathrm{union}=E(\ni)}{=\!=\!=\!=} \boxed{\tfrac{S}{\ni}\,E(\tfrac{R}{\ni})E(\ni)} \underset{E\text{ a functor, }E(X)E(Y)=E(XY)}{=\!=\!=\!=} \boxed{\tfrac{S}{\ni}\,E(\tfrac{R}{\ni}\ni)}$$

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

the first three steps only unfold $\diamond$ — E is a functor, so $E(\frac{R}{\ni})\mathrm{union}=E(\frac{R}{\ni}\ni)=E(R)$ 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 $\frac{1}{\ni}$, 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

(11a)

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 : $\mathcal{C}\to\mathcal{D}$ is a 2-functor between allegories:

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

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

(11b)

the statement	B&dM
For f a map, F(f) is a map and F(f°)=F(f)°.	Lemma 5.1
Over a **tabular** allegory a functor is a relator ⟺ it preserves °.	Theorem 5.1
F(R°)=F(R)° for every R, so F(R)° needs no bracket.	after Theorem 5.1, p. 113
Two relators agreeing on maps are equal.	Corollary 5.1
F(X∩Y)=F(X)∩F(Y) for X,Y coreflexive.	Ex 5.2
F(R∩S)⊑F(R)∩F(S), and strictly.	Ex 5.2, the restriction
F(Dom(R))=Dom(F(R)) for F preserving °.	—

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⟩

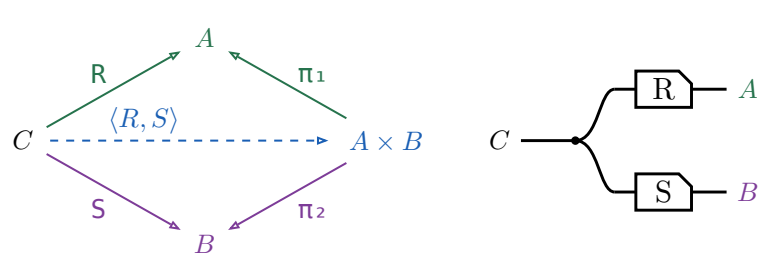
(11.1a)

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 ⊤.

(11.1b)

$$\langle R,S\rangle\pi_1=\mathrm{Dom}(S)R\qquad \langle R,S\rangle\pi_2=\mathrm{Dom}(R)S$$

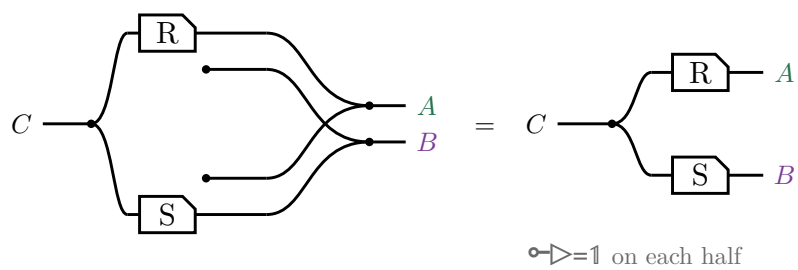
(11.1c)



A domain is coreflexive, so ⟨R,S⟩π₁⊑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. π₁=𝟙⊗⊸ discards the second component, so π₁°=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)

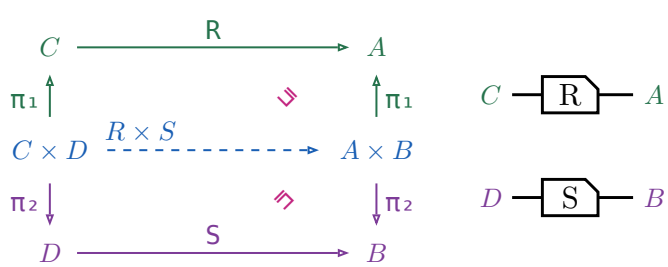


§11.1.1. Relational product R×S

(11.1.1a)

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

(11.1.1b)

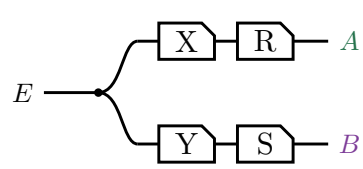


Right-then-up is (R×S)π₁, up-then-right is π₁R, and (R×S)π₁⊑π₁R, equality when S is entire. In Rel, (c,d) (R×S) (a,b) iff c R a and d S b — two strands side by side, no copy dot: R×S=R⊗S.

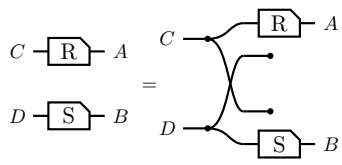
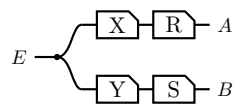
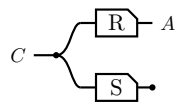
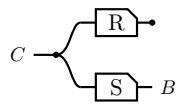
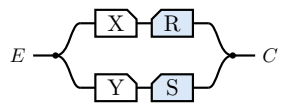
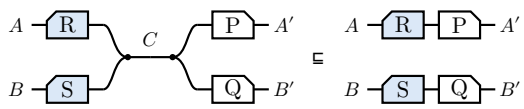
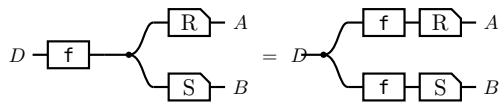
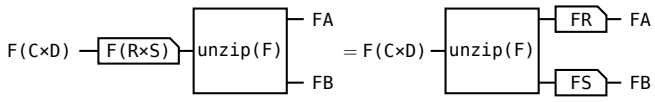
§11.1.2. Absorption

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

(11.1.2a)

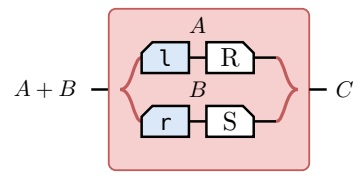
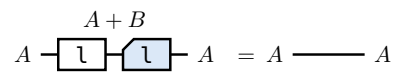
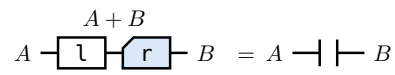
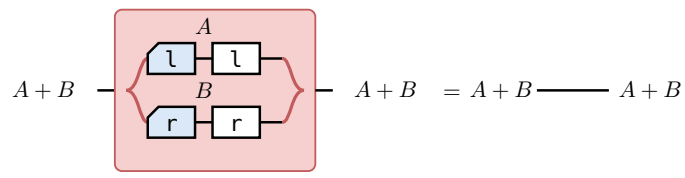


(11.1.2b)

	the law	picture
1	`R×S=⟨π₁R,π₂S⟩`	
2	`⟨X,Y⟩(R×S)=⟨XR,YS⟩` both sides are the same strokes — (11.1.2a). Its ⊑ half is Ex 5.8, that half at `S:=𝟙` and at `R:=𝟙` B&dM's (5.4), (5.5) — stages of their proof of this row; Ex 5.6 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`

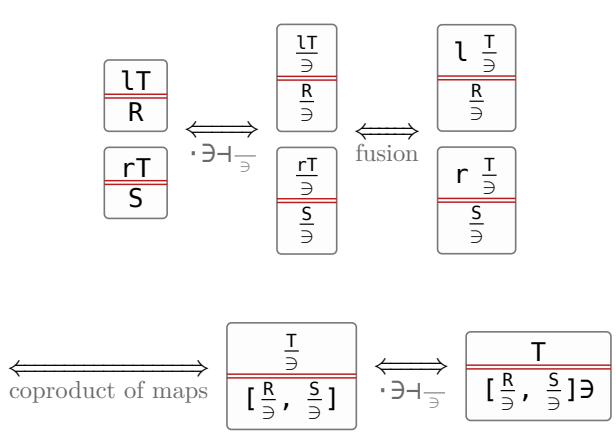
(11.2a)

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`	

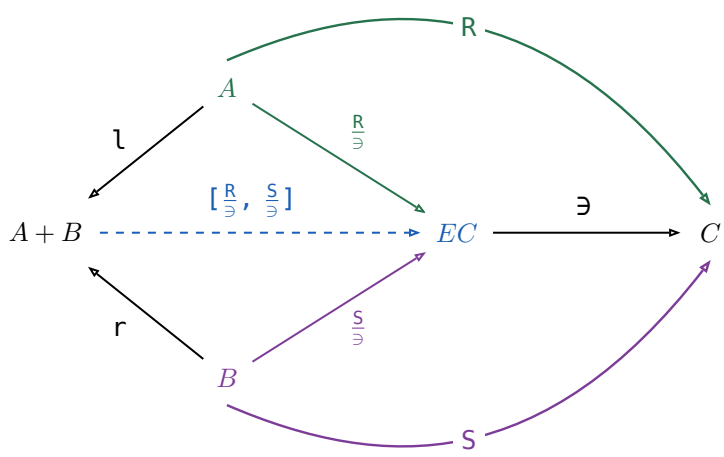
§11.2.1. `[R,S]≜[R⫤, S⫤]∋`

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 ⫤ 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)

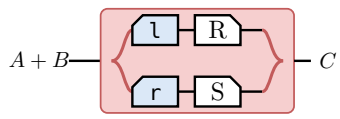
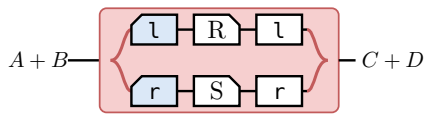
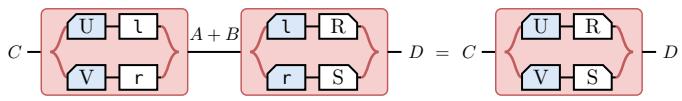
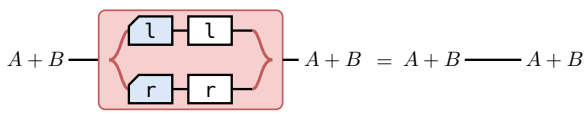


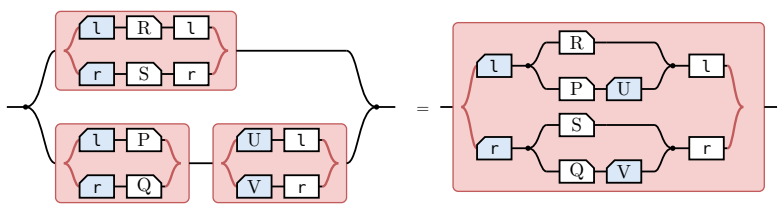
(11.2.1b)



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)

B&dM	the law	picture
(5.9)	`[R,S]=(l°R)∪(r°S)` (11.2a)'s first row	
(5.10)	`R+S=[Rl,Sr]`	
(5.11)	`[U,V]°[R,S]=(U°R)∪(V°S)` (11.2a)'s last row	
Ex 5.12	`X≜[𝟙,⊥]=l°` and `Y≜[⊥,𝟙]=r°`, so `(Xl)∪(Yr)=[l,r]=𝟙` which is (5.9)	
Ex 5.13	prove (5.11), and say why duality does not carry it over from the product law	

Ex 5.14	`(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,

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

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

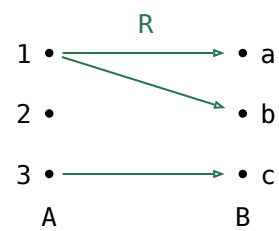
xs P(R) ys⟺(∀a∈xs. ∃b∈ys. a R b)∧(∀b∈ys. ∃a∈xs. a R b)

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

(11.4a)

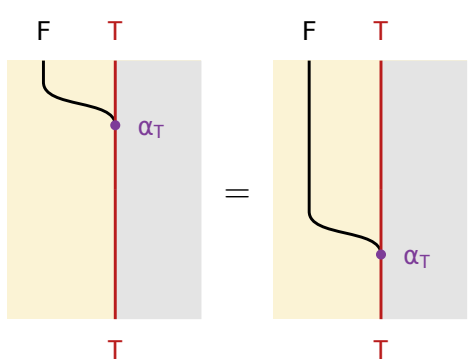
An **F-algebra** on A is a map $\alpha_A : FA \to A$. An **F-homomorphism** from α_A to α_B is a map $h : A \to B$ with $\alpha_A\ h = F(h)\alpha_B$. The **initial algebra** $\alpha_T : FT \to T$ is the F-algebra with exactly one F-homomorphism $(\!|\alpha_A|\!) : T \to A$ to every F-algebra α_A.

$$\begin{tikzcd} FA \arrow[r, "\alpha_A"] \arrow[d, "F(h)"'] & A \arrow[d, "h"] \\ FB \arrow[r, "\alpha_B"'] & B \end{tikzcd} \qquad \begin{tikzcd} FT \arrow[r, "\alpha_T"] \arrow[d, "F((\!|\alpha_A|\!))"'] & T \arrow[d, dashed, "(\!|\alpha_A|\!)"] \\ FA \arrow[r, "\alpha_A"'] & A \end{tikzcd}$$

§11.4.1. Reflection

(11.4.1a)

$$\begin{tikzcd} FT \arrow[r, "\alpha_T"] \arrow[d, "F(\mathbb{1})"'] & T \arrow[d, dashed, "\mathbb{1}"] \\ FT \arrow[r, "\alpha_T"'] & T \end{tikzcd}$$



$(\!|\alpha_T|\!) = \mathbb{1}$ (2.11)

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

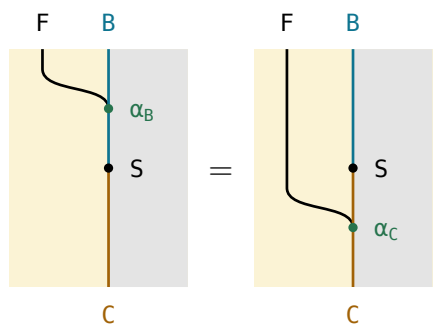
§11.4.2. Fusion

Fusion rewrites $(\!|\alpha_B|\!)S$ through a second algebra $\alpha_C : FC \to C$ along an arrow $S : B \to C$.

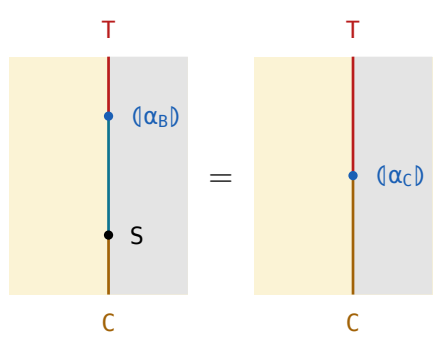
(11.4.2a)

$$\begin{tikzcd} FT \arrow[r, "\alpha_T"] \arrow[d, "F((\!|\alpha_B|\!))"'] & T \arrow[d, dashed, "(\!|\alpha_B|\!)"] \\ FB \arrow[r, "\alpha_B"] \arrow[d, "F(S)"'] & B \arrow[d, "S"] \\ FC \arrow[r, "\alpha_C"'] & C \end{tikzcd}$$

the side condition



the conclusion



$(\!|\alpha_B|\!)S = (\!|\alpha_C|\!) \Longleftarrow \alpha_B\ S = F(S)\alpha_C$ (2.12)

§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°)	A datatype acts on relations, not only on maps — the map of the converse is the converse of the map.

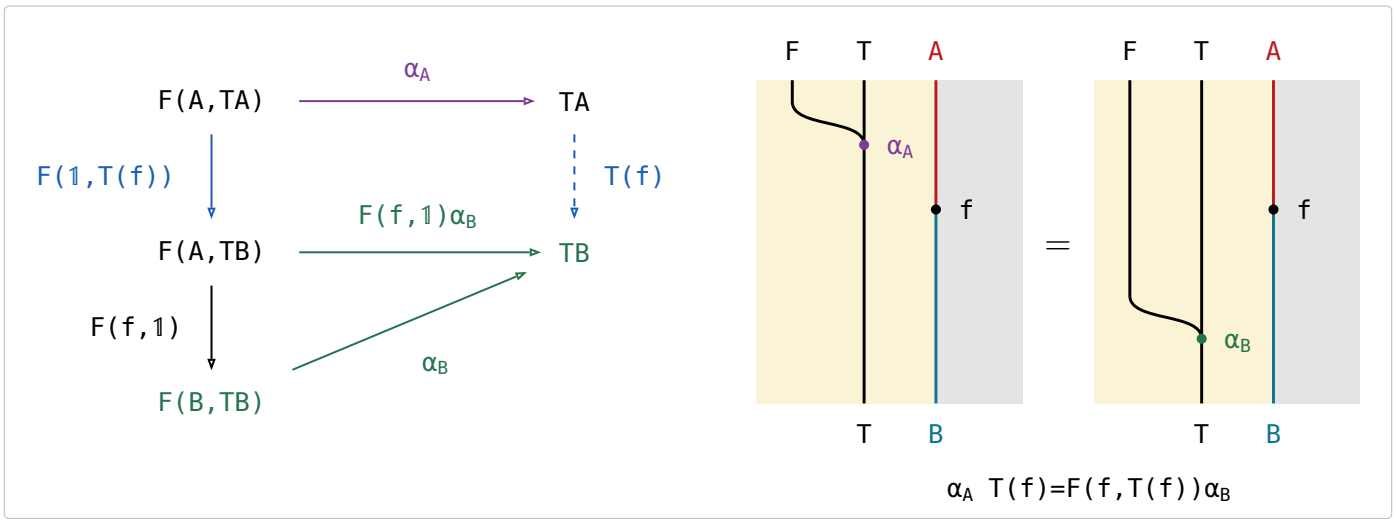
§11.5.1. Type functor

(11.5.1a)

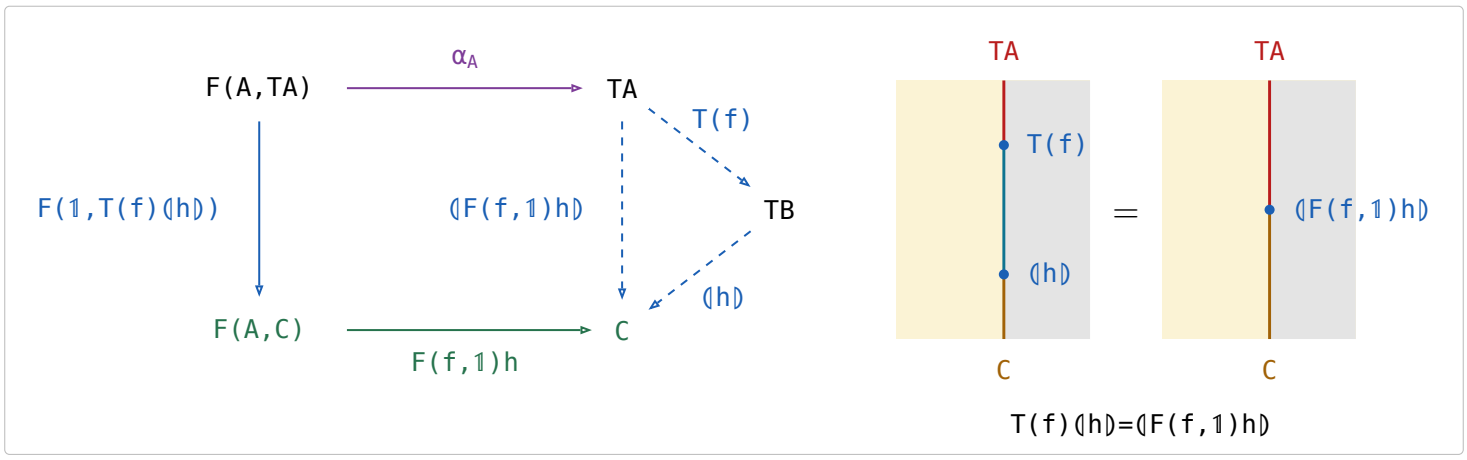
Let F be a bifunctor taking both the parameter A and the recursive position TA, with an initial algebra α_A : F(A,TA)→TA for every object A. Then T is a functor, acting on a map f : A→B by

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

(11.5.1b)



(11.5.1c)



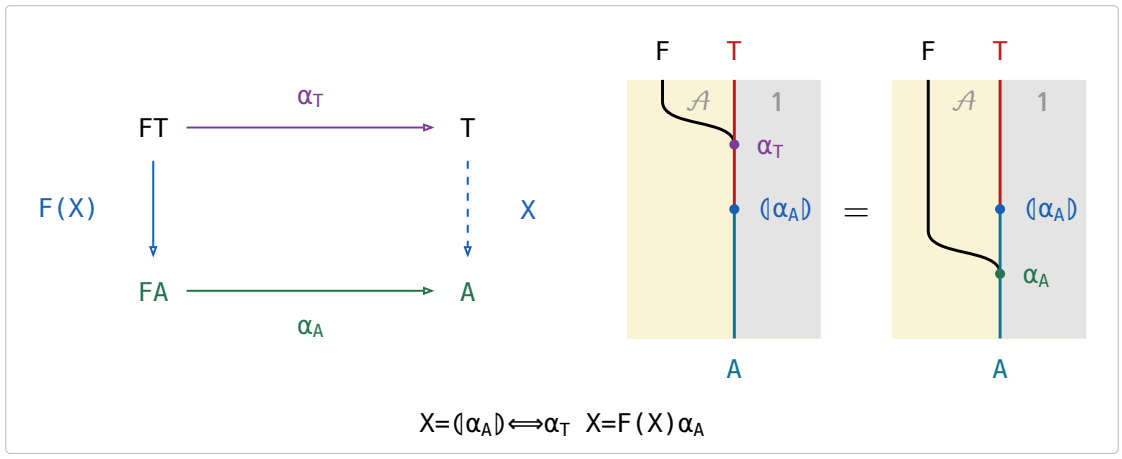
§11.6. Reduce

(11.6a)

> 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:

§11.6.1. The defining equation

(11.6.1a)



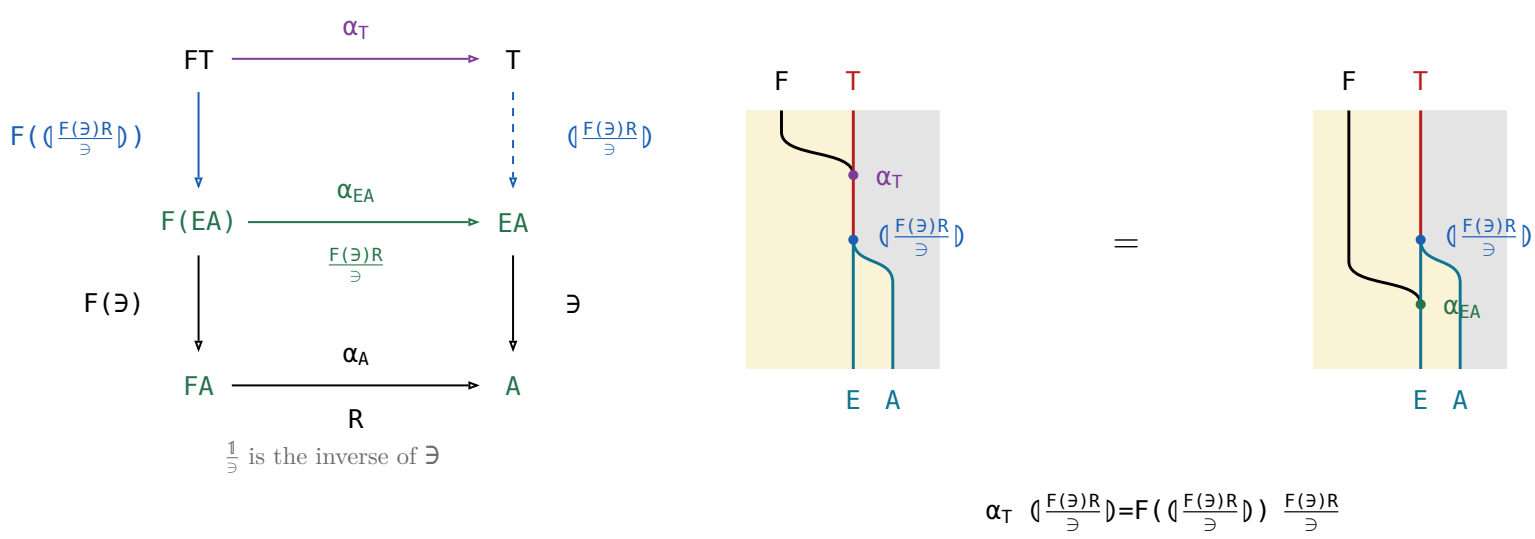
An F-algebra is a **weakened** natural transformation. A transformation F⇒𝟙 on $\mathcal{A}$ would need a component FX→X at every object and a commuting square at every arrow, but F-Algebra only need it works on T and A.

(11.6.1b)

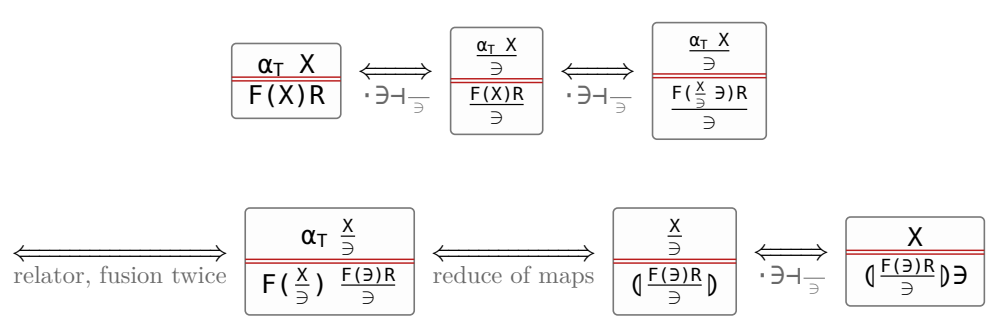
⦇[c,f]⦈=reduce(c,f)		
part	**Nat**	**[A]**
datatype	Nat::=zero|succ Nat	[A]::=nil|cons (A,[A])
base functor F	F(X)=1+X	F(X)=1+A×X
initial algebra α	α=[zero,succ] : 1+Nat→Nat	α=[nil,cons] : 1+A×[A]→[A]
the fold, pointwise	⦇[c,f]⦈ zero=c ⦇[c,f]⦈ (succ n)=f (⦇[c,f]⦈n)	⦇[c,f]⦈ nil=c ⦇[c,f]⦈ (cons (a,x))=f (a,⦇[c,f]⦈ x)

§11.6.2. ⦇R⦈=⦇$\frac{F(\ni)R}{\ni}$⦈∋

(11.6.2a)



(11.6.2b)



§11.6.3. φ(Y)⊑Y⟹(μX : φ(X))⊑Y

(11.6.3a)

φ a **monotonic** mapping of the hom-set A→B into itself: X⊑Y⟹φ(X)⊑φ(Y) lean:AOP.A6_2.Monotonic@66dddf1e.

(μX : φ(X)) the least X : A→B with φ(X)⊑X lean:AOP.A6_2.mu@4928a490.

(11.6.3b)

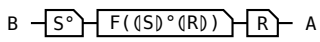
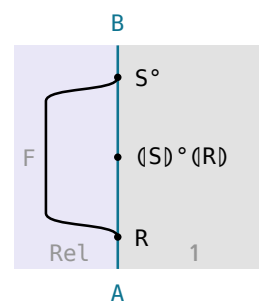
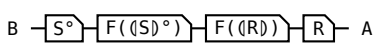
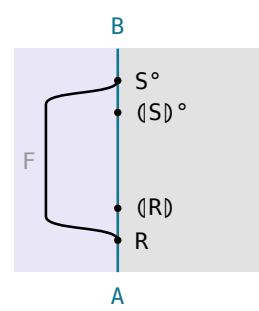
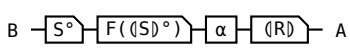
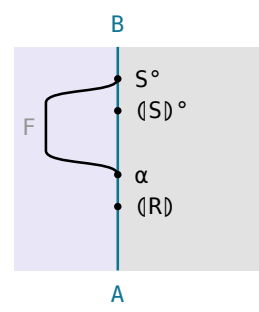
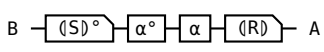
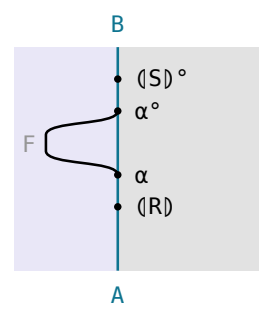
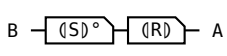
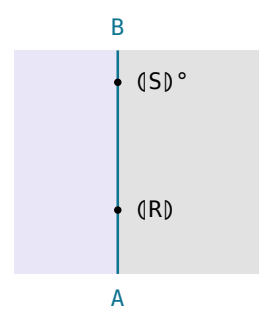
the law	what it says
φ(Y)⊑Y⟹(μX : φ(X))⊑Y Theorem 6.1, lean:AOP.A6_2.mu_le@9918bd39	to bound (μX : φ(X)) above, exhibit one Y the body does not grow past — the half §11.6.4 and every chapter after it uses
φ((μX : φ(X)))=(μX : φ(X)) Theorem 6.1, lean:AOP.A6_2.mu_fixed@2d3d1a8a	**Knaster–Tarski**: the 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⦈

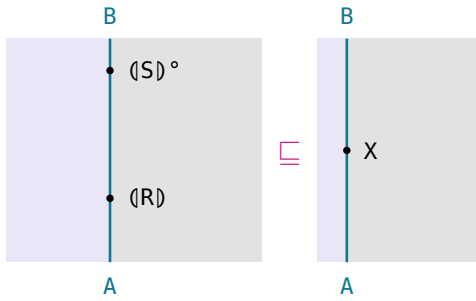
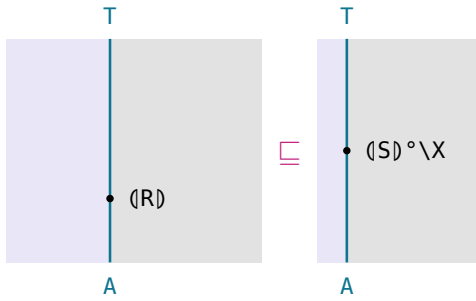
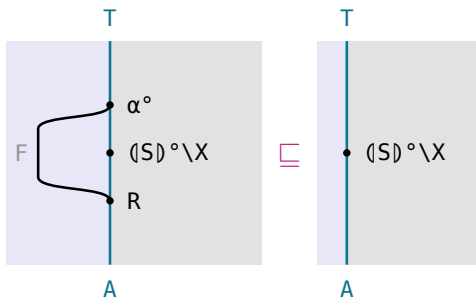
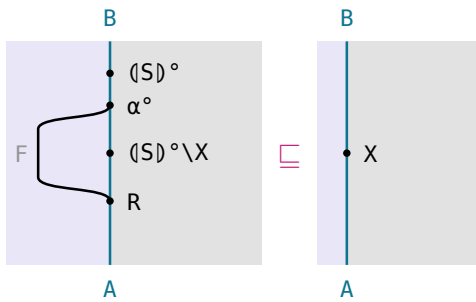
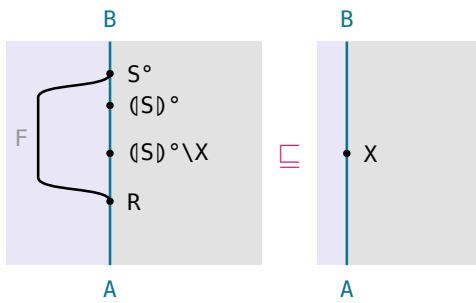
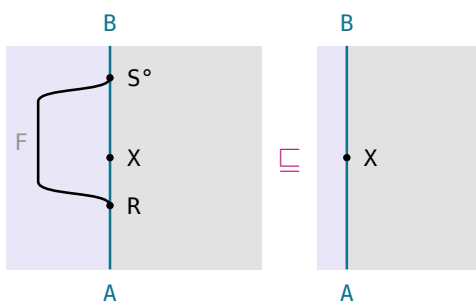
Theorem 6.2, lean:AOP.A6_3.hylo_fixed@42010f9f; R : FA→A, S : FB→B, α : FT→T initial

circuit	Hinze–Marsden
 S°F(⦇S⦈°⦇R⦈)R the body at ⦇S⦈°⦇R⦈	
=  S°F(⦇S⦈°)F(⦇R⦈)R F(RS)=F(R)F(S) — (11a)	
=  S°F(⦇S⦈°)α⦇R⦈ (11.6.1a) at R: F(⦇R⦈)R=α⦇R⦈	
=  ⦇S⦈°α°α⦇R⦈ (11.6.1a) at S conversed: ⦇S⦈°α°=S°F(⦇S⦈)°, and F(⦇S⦈)°=F(⦇S⦈°) — (11b)	
=  ⦇S⦈°⦇R⦈ α°α=𝟙, Lambek — lean:AOP.A6_2.InitialAlgebra.recip_alpha_alpha@5a99c7f6	

(11.6.4b)

S°F(X)R⊑X⟹⦇S⦈°⦇R⦈⊑X

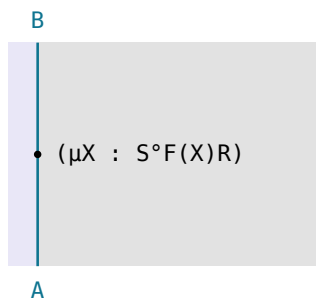
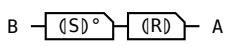
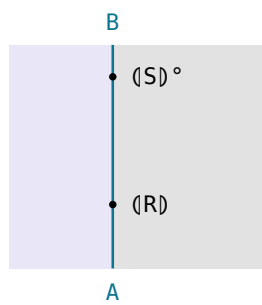
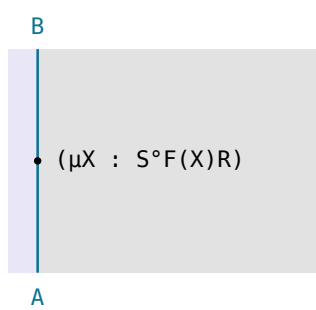
Theorem 6.2, `lean:AOP.A6_3.hylo_le_of_prefixed@b892517c`

circuit	Hinze–Marsden
B ⦇S⦈° ⦇R⦈ ⊑ X A ⦇S⦈°⦇R⦈⊑X the conclusion	
T ⦇R⦈ ⊑ ⦇S⦈°\X A ⇔ ⦇R⦈⊑⦇S⦈°\X (1.1a)'s S·⊣S\ at ⦇S⦈°	
T α° F(⦇S⦈°\X) R ⊑ ⦇S⦈°\X A ⇐ α°F(⦇S⦈°\X)R⊑⦇S⦈°\X (6.2) ⦇R⦈=(μX : α°F(X)R) — (11.6.1a) and (11.6.3b); `lean:AOP.A6_2.relCata_le_of_prefixed@9f98060a`	
B ⦇S⦈° α° F(⦇S⦈°\X) R ⊑ X A ⇔ ⦇S⦈°α°F(⦇S⦈°\X)R⊑X (1.1a)'s S·⊣S\ at ⦇S⦈°	
B S° F(⦇S⦈°) F(⦇S⦈°\X) R ⊑ X A ⇔ S°F(⦇S⦈°)F(⦇S⦈°\X)R⊑X ⦇S⦈°α°=S°F(⦇S⦈°) — (11.6.4a)	
B S° F(X) R ⊑ X A ⇐ S°F(X)R⊑X F(RS)=F(R)F(S) — (11a) — and ⦇S⦈°(⦇S⦈°\X)⊑X — (1.1a)	

(11.6.4c)

⦇S⦈°⦇R⦈=(μX : S°F(X)R)

Theorem 6.2, `lean:AOP.A6_3.hylo_eq_mu@c60df971`

circuit	Hinze–Marsden
 (μX : S°F(X)R) (11.6.3a) at φ(X):=S°F(X)R	
 ⦇S⦈°⦇R⦈ ⊑ (11.6.3b)'s φ(Y)⊑Y⟹(μX : φ(X))⊑Y at Y:=⦇S⦈°⦇R⦈, whose S°F(⦇S⦈°⦇R⦈)R=⦇S⦈°⦇R⦈ is (11.6.4a); `lean:AOP.A6_2.mu_le_of_fixed@8ea2332b`	
 (μX : S°F(X)R) ⊑ (11.6.4b) at X:=(μX : S°F(X)R), whose S°F((μX : S°F(X)R))R⊑(μX : S°F(X)R) is (11.6.3b)'s φ((μX : φ(X)))=(μX : φ(X)); `lean:AOP.A6_2.mu_prefixed@fc0a1dca`	

§12. Combinatorial functions

(12a)

definition	type	note
`[A]::=nil|cons (A,[A])`	$\mathcal{A}\to\mathcal{A}$	B&dM's `listr` under the short name it keeps from p. 125 on.
`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]]`	`cat` restricted to a non-empty first argument, 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`. Ex 7.41; `est(R°)` is (13.1a)
`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`. Ex 7.39
`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. Ex 7.40

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

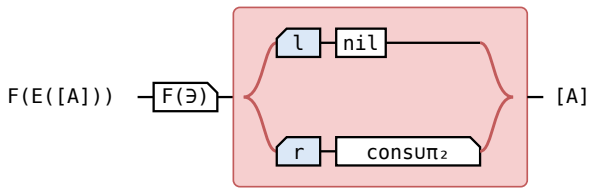
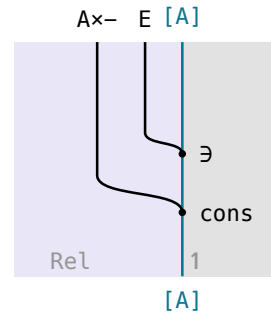
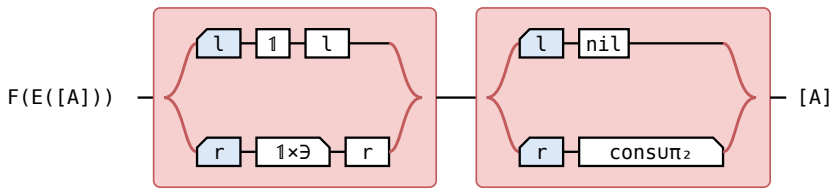
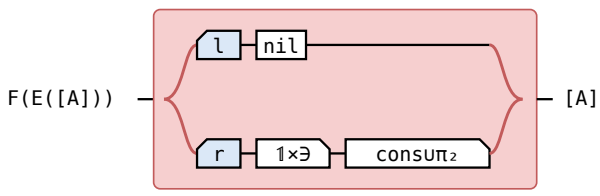
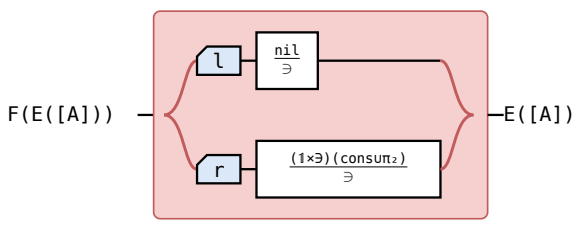
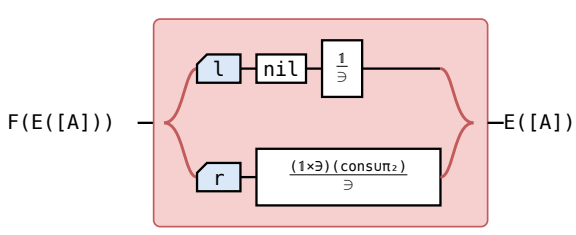
(12.1a)

`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)

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

(11.6.2b) at subseq=⦇[nil,consuπ₂]⦈ — (12a)

circuit — the fork is F([A])=1+A×[A]: nil above, the pair below	**Hinze–Marsden** — the cons operand
 $\frac{F(\ni)[nil,consu\pi_2]}{\ni}$	
=  $\frac{(\mathbb{1}+\mathbb{1}\times\ni)[nil,consu\pi_2]}{\ni}$ F(X)=1+A×X — (12a)	
=  $\frac{[nil,(\mathbb{1}\times\ni)(consu\pi_2)]}{\ni}$ R+S≜[Rl,Sr], l[R,S]=R, r[R,S]=S — (11.2a)	
=  $[\frac{nil}{\ni}, \frac{(\mathbb{1}\times\ni)(consu\pi_2)}{\ni}]$ (11.2.1a) at T:=[nil,(1×∋)(consuπ₂)]	
=  $[nil\ \frac{\mathbb{1}}{\ni}, \frac{(\mathbb{1}\times\ni)(consu\pi_2)}{\ni}]$ (10b), $\frac{f}{\ni} = f\ \frac{\mathbb{1}}{\ni}$ for f a map, at f:=nil	

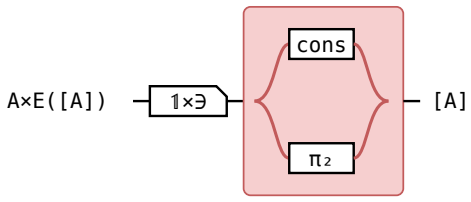
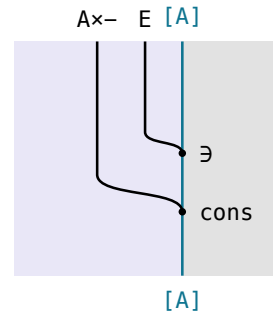
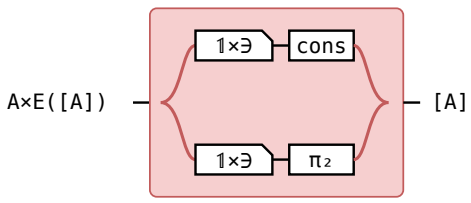
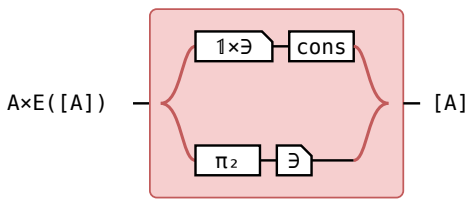
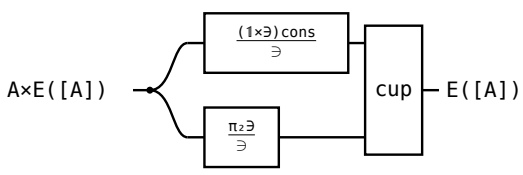
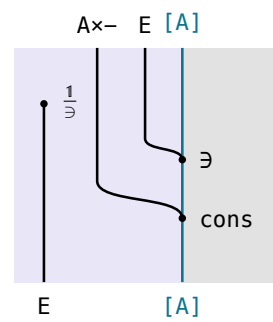
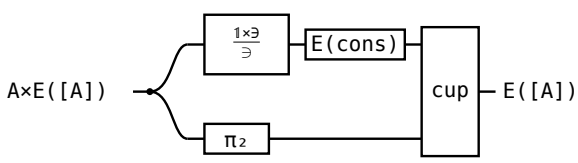
(12.1c)

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

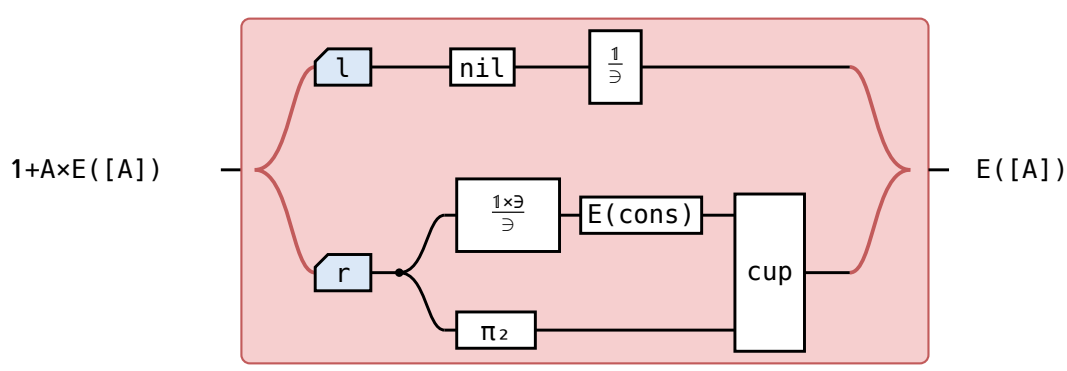
(12.1d)

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

B&dM §5.6, p. 124

circuit — the tape IS the ∪, and neither branch injects	**Hinze–Marsden** — the cons operand
 $\frac{(\mathbb{1}\times\ni)(\mathtt{cons}\cup\pi_2)}{\ni}$ (12.1b)'s second branch	
=  $\frac{(\mathbb{1}\times\ni)\mathtt{cons}\cup(\mathbb{1}\times\ni)\pi_2}{\ni}$ $T(X_1\cup X_2)=TX_1\cup TX_2$ — (1.2b)	
=  $\frac{(\mathbb{1}\times\ni)\mathtt{cons}\cup\pi_2\ni}{\ni}$ $(\mathbb{1}\times\ni)\pi_2=\pi_2\ni$ — (11.1.1b) at π_2, an equality because $\mathbb{1}$ is entire	
=  $\langle \frac{(\mathbb{1}\times\ni)\mathtt{cons}}{\ni}, \frac{\pi_2\ni}{\ni} \rangle\ \mathtt{cup}$ $\frac{R\cup S}{\ni} = \langle \frac{R}{\ni}, \frac{S}{\ni} \rangle\ \mathtt{cup}$ — (12.1a)	
=  $\langle \frac{\mathbb{1}\times\ni}{\ni}\ \mathtt{E(cons)}, \pi_2 \rangle\ \mathtt{cup}$ (10b), absorption $\frac{S}{\ni}\ \mathtt{E}(R)=\frac{SR}{\ni}$ at $S:=\mathbb{1}\times\ni$, $R:=\mathtt{cons}$; fusion and $\frac{\ni}{\ni}=\mathbb{1}$ on the π_2 operand	

(12.1e)



B&dM §5.6, p. 124, which writes Pcons; cons is a map, and there P(cons)=E(cons) — (11.3d).

§13. Optimisation Problems

§13.1. est(R)≜∋∩(∈\R°)

(13.1a)

For R : A→A, est(R)≜∋∩(∈\R°) : EA→A lean:AOP.A7_1.est@e39806f8.

xs (est(R)) x⟺x∈xs∧(∀y∈xs. x R y) B&dM's min R, the same predicate under the same letter; his R : A←A reads x R y as the arrow y→x, 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)

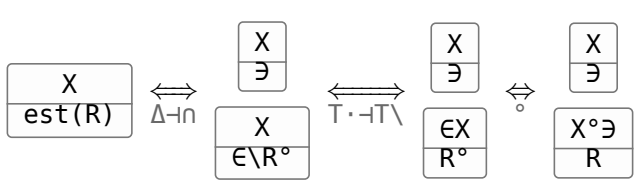
P(R) : EA→EB, xs P(R) ys⟺(∀x∈xs. ∃y∈ys. x R y)∧(∀y∈ys. ∃x∈xs. x R y) every x and every y has a partner, (11.3b)

(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 a preorder	a minimum in each set, then a minimum of those
P(est(R)) est(R)=P(Dom(est(R))) union est(R) R a preorder	the same as an equality, once empty sets are dropped

§13.1.1. X⊑est(R)⟺X⊑∋ and X°∋⊑R

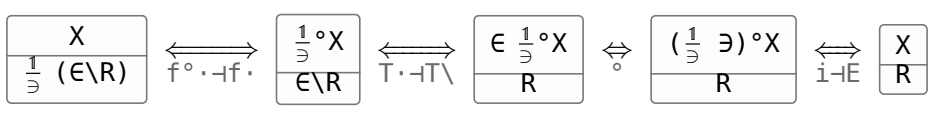
(13.1.1a)



lean:AOP.A7_1.le_est_iff@81855810

§13.1.2. $\frac{1}{\ni}$ (∈\R)=R

(13.1.2a)



lean:AOP.A7_1.singletonMap_comp_lb@9d4759cf

§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)

$$\begin{array}{c} X \\ \hline \overset{S}{\cup}(\in\backslash R) \end{array} \xleftrightarrow{f^\circ\cdot\dashv f\cdot} \begin{array}{c} \overset{S}{\cup}{}^\circ X \\ \hline \in\backslash R \end{array} \xleftrightarrow{T\cdot\dashv T\backslash} \begin{array}{c} \in \overset{S}{\cup}{}^\circ X \\ \hline R \end{array} \overset{\circ}{\Leftrightarrow} \begin{array}{c} (\overset{S}{\cup}\ni)^\circ X \\ \hline R \end{array} \xleftrightarrow{\cdot\ni\dashv\overset{\ }{\ni}} \begin{array}{c} S^\circ X \\ \hline R \end{array} \xleftrightarrow{T\cdot\dashv T\backslash} \begin{array}{c} X \\ \hline S^\circ\backslash R \end{array}$$

lean:AOP.A7_1.Λ_comp_lb@f14ab8b5

§13.1.4. union (∈\R)=∈\(∈\R)

(13.1.4a)

$$\begin{array}{c} X \\ \hline \text{union}\ (\in\backslash R) \end{array} \xleftrightarrow{f^\circ\cdot\dashv f\cdot} \begin{array}{c} \text{union}^\circ\ X \\ \hline \in\backslash R \end{array} \xleftrightarrow{T\cdot\dashv T\backslash} \begin{array}{c} \in\text{union}^\circ\ X \\ \hline R \end{array} \overset{\circ}{\Leftrightarrow} \begin{array}{c} (\text{union}\ni)^\circ X \\ \hline R \end{array} \xleftrightarrow{\cdot\ni\dashv\overset{\ }{\ni}} \begin{array}{c} (\ni\ni)^\circ X \\ \hline R \end{array} \xleftrightarrow{\circ,\ T\cdot\dashv T\backslash} \begin{array}{c} X \\ \hline \in\backslash(\in\backslash R) \end{array}$$

lean:AOP.A7_1.bigUnion_comp_lb@ed13dc7a

§13.1.5. $\overset{S}{\cup}$ est(R)=Sn(S°\R°)

(13.1.5a)

$$\begin{array}{c} X \\ \hline \overset{S}{\cup}\ \text{est}(R) \end{array} \xleftrightarrow{f^\circ\cdot\dashv f\cdot} \begin{array}{c} \overset{S}{\cup}{}^\circ X \\ \hline \text{est}(R) \end{array} \xleftrightarrow{\Delta\dashv\cap} \begin{array}{c} \overset{S}{\cup}{}^\circ X \\ \hline \ni \end{array} \ \begin{array}{c} \overset{S}{\cup}{}^\circ X \\ \hline \in\backslash R^\circ \end{array} \xleftrightarrow{f^\circ\cdot\dashv f\cdot} \begin{array}{c} X \\ \hline \overset{S}{\cup}\ \ni \end{array} \ \begin{array}{c} X \\ \hline \overset{S}{\cup}\ (\in\backslash R^\circ) \end{array} \xleftrightarrow{\cdot\ni\dashv\overset{\ }{\ni},\ (13.1.3a)} \begin{array}{c} X \\ \hline S \end{array} \ \begin{array}{c} X \\ \hline S^\circ\backslash R^\circ \end{array} \xleftrightarrow{\Delta\dashv\cap} \begin{array}{c} X \\ \hline S\cap(S^\circ\backslash R^\circ) \end{array}$$

lean:AOP.A7_1.Λ_comp_est@4c6d38d0

§13.1.6. $\overset{S}{\cup}$ est(R)= $\overset{S}{\cup}$ est(R∩S°S)

(13.1.6a)

$$\begin{array}{c} X \\ \hline \overset{S}{\cup}\ \text{est}(R\cap S^\circ S) \end{array} \xleftrightarrow{(13.1.5a),\ \circ} \begin{array}{c} X \\ \hline S\cap(S^\circ\backslash(R^\circ\cap S^\circ S)) \end{array} \xleftrightarrow{\Delta\dashv\cap,\ T\cdot\dashv T\backslash} \begin{array}{c} X \\ \hline S \end{array} \ \begin{array}{c} S^\circ X \\ \hline R^\circ\cap S^\circ S \end{array}$$

$$\xleftrightarrow{\Delta\dashv\cap,\ S^\circ\cdot\ \text{monotone}} \begin{array}{c} X \\ \hline S \end{array} \ \begin{array}{c} S^\circ X \\ \hline R^\circ \end{array} \xleftrightarrow{T\cdot\dashv T\backslash,\ \Delta\dashv\cap} \begin{array}{c} X \\ \hline S\cap(S^\circ\backslash R^\circ) \end{array} \xleftrightarrow{(13.1.5a)} \begin{array}{c} X \\ \hline \overset{S}{\cup}\ \text{est}(R) \end{array}$$

lean:AOP.A7_1.Λ_comp_est_context@e8052cf4

§13.1.7. P(f) est(R)=est(fRf°) f

(13.1.7a)

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

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

§13.1.8. P(S) est(R)⊑(∋S)∩(∈\(SR°))

(13.1.8a)

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

lean:AOP.A7_1.powerRel_comp_est_le@d8b5692c

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

(13.1.9a)

$$\begin{array}{c} \texttt{P(est(R)) est(R)} \\ \hline \texttt{union est(R)} \end{array} \underset{\text{(13.1.5a)}}{\Longleftrightarrow} \begin{array}{c} \texttt{P(est(R)) est(R)} \\ \hline \texttt{(∋∋)∩((∋∋)°\R°)} \end{array} \xleftarrow{\quad \texttt{°}, \texttt{Δ⊣∩}, \texttt{T·⊣T\quad}} \begin{array}{c} \texttt{P(est(R)) est(R)} \\ \hline \texttt{∋∋} \end{array} \begin{array}{c} \texttt{∈∈P(est(R))} \\ \texttt{est(R)} \\ \hline \texttt{R°} \end{array} \xleftarrow{\quad \text{UP of } \texttt{est}, \texttt{R} \text{ transitive} \quad} \begin{array}{c} \texttt{P(est(R))∋} \\ \hline \texttt{∋est(R)} \end{array} \begin{array}{c} \texttt{∈P(est(R))} \\ \hline \texttt{est(R)∈} \end{array}$$

R a preorder

lean:AOP.A7_1.powerRel_est_le_bigUnion@d0c726a1, R transitive

§13.2. Lax natural transformations (LaT)

(13.2a)

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.

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

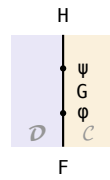
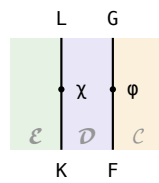
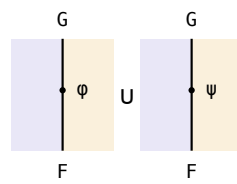
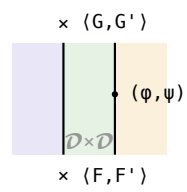
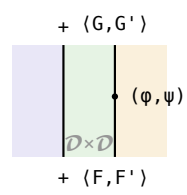
Lax at every **map** already gives LaT, and at a map the inequation is an equality $G(f)\varphi=\varphi F(f)$: laxness is about relations only. Theorem 5.2

(13.2b)

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

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

(13.2c)

closed under	commutative diagram	string diagram
composition $\psi\varphi$	$\begin{tikzcd} HA \arrow[r, "\psi_A"] \arrow[d, dotted, "H(R)"'] \arrow[dr, phantom, "\Leftarrow"] & GA \arrow[r, "\varphi_A"] \arrow[d, dotted, "G(R)"'] \arrow[dr, phantom, "\Leftarrow"] & FA \arrow[d, dotted, "F(R)"'] \\ HB \arrow[r, "\psi_B"'] & GB \arrow[r, "\varphi_B"'] & FB \end{tikzcd}$ $H(R)\psi_B \sqsubseteq \psi_A G(R)$ 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 $\chi\circ\varphi$	$\begin{tikzcd} L(GA) \arrow[r, "\chi_{GA}K(\varphi_A)"] \arrow[d, dotted, "L(G(R))"'] \arrow[dr, phantom, "\Leftarrow"] & K(FA) \arrow[d, dotted, "K(F(R))"'] \\ L(GB) \arrow[r, "\chi_{GB}K(\varphi_B)"'] & K(FB) \end{tikzcd}$ $\varphi : G \Rightarrow F$ 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 $\varphi\cup\psi$	$\begin{tikzcd} GA \arrow[r, "\varphi_A"] \arrow[d, dotted, "G(R)"'] \arrow[dr, phantom, "\Leftarrow"] & FA \arrow[d, dotted, "F(R)"'] \\ GB \arrow[r, "\varphi_B"'] & FB \end{tikzcd} \cup \begin{tikzcd} GA \arrow[r, "\psi_A"] \arrow[d, dotted, "G(R)"'] \arrow[dr, phantom, "\Leftarrow"] & FA \arrow[d, dotted, "F(R)"'] \\ GB \arrow[r, "\psi_B"'] & FB \end{tikzcd}$ $G(R)\varphi_B \sqsubseteq \varphi_A F(R)$ 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 $K(\varphi)$	$\begin{tikzcd} K(GA) \arrow[r, "K(\varphi_A)"] \arrow[d, dotted, "K(G(R))"'] \arrow[dr, phantom, "\Leftarrow"] & K(FA) \arrow[d, dotted, "K(F(R))"'] \\ K(GB) \arrow[r, "K(\varphi_B)"'] & K(FB) \end{tikzcd}$ $G(R)\varphi_B \sqsubseteq \varphi_A F(R)$ gives $K(G(R))K(\varphi_B) \sqsubseteq K(\varphi_A)K(F(R))$	
product $\varphi\times\psi$	$\begin{tikzcd} GA\times G'A \arrow[r, "\varphi_A\times\psi_A"] \arrow[d, dotted, "G(R)\times G'(R)"'] \arrow[dr, phantom, "\Leftarrow"] & FA\times F'A \arrow[d, dotted, "F(R)\times F'(R)"'] \\ GB\times G'B \arrow[r, "\varphi_B\times\psi_B"'] & FB\times F'B \end{tikzcd}$ $\varphi : G \Rightarrow F$ 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 $\varphi+\psi$	$\begin{tikzcd} GA+G'A \arrow[r, "\varphi_A+\psi_A"] \arrow[d, dotted, "G(R)+G'(R)"'] \arrow[dr, phantom, "\Leftarrow"] & FA+F'A \arrow[d, dotted, "F(R)+F'(R)"'] \\ GB+G'B \arrow[r, "\varphi_B+\psi_B"'] & FB+F'B \end{tikzcd}$ $\varphi : G \Rightarrow F$ 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 $[\varphi,\psi] = (\varphi+\psi)\rhd$, and $\rhd$ 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 — φ,ψ : G⇒F lax natural does NOT give φ∩ψ 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≜{0,1}, R≜{(0,0),(1,0)}, φ≜π₁∩π₂ : Δ⇒Id

π₁,π₂ : Δ⇒Id are both LaTs (13.4.3c)'s g row and π₁∩π₂={((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→B	a relator F : $\mathcal{C}\to\mathcal{D}$
2-cell	R⊑S	a LaT φ : G⇒F (13.2a)
order	R⊑S	φ⊑ψ componentwise
union	R∪S	φ∪ψ, **survives** (13.2c)
product	R×S (11.1.1a)	φ×ψ, **survives**; a TENSOR, not a categorical product; SYMMETRIC (11.1b) (13.2c)
coproduct	R+S	φ+ψ, **survives**; a BIPRODUCT (13.2c)
meet	R∩S	φ∩ψ 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 𝟙 z=𝟘	the constant relator at such a z, initial and terminal

§13.2.3. R⊑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≜const A, F≜const B — the relators X↦A and X↦B, each sending every arrow to 𝟙 (13.2a)

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

not every LaT is constant: ∋ : P⇒Id (11.3d) π₂ : (A×−)⇒Id (12.1c) ◁ : Id⇒Δ (2b)

§13.3. Monotonic algebras

(13.3a)

An F-algebra φ : FA→A is **monotonic on** R : A→A when it is lax at R at G:=F, F:=Id: F(R)φ⊑φR. 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.

For a map f : FA→A that is f°F(R)f⊑R (1.1a)'s f°·⊣f· at X:=F(R)f, Y:=R, equivalently F(R)⊑fRf° ·f⊣·f° then f°·⊣f·.

(≤×≤)+⊑+≤ — addition on Nat is monotonic on ≤, which at the point level reads c=a+b∧a≤a'∧b≤b'⟹c≤a'+b'.

(13.3b)

$$\begin{tikzcd} FA \arrow[r, "\varphi"] \arrow[d, "F(R)"'] \arrow[dr, phantom, "\Leftarrow"] & A \arrow[d, "R"] \\ FA \arrow[r, "\varphi"'] & A \end{tikzcd}$$

F(R)φ⊑φR

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

(13.3.1a)

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

+ distributes over ≤, at the point level min xs+min ys=min{x+y|x∈xs∧y∈ys} for xs, ys non-empty and min≜est(≤).

(13.3.1b)

$$\begin{tikzcd} F(EA) \arrow[r, "\frac{F(\ni)f}{\ni}"] \arrow[d, "F(est(R))"'] \arrow[dr, phantom, "\Leftarrow"] & EA \arrow[d, "est(R)"] \\ FA \arrow[r, "f"'] & A \end{tikzcd}$$

$F(est(R))f \sqsubseteq \frac{F(\ni)f}{\ni} est(R)$

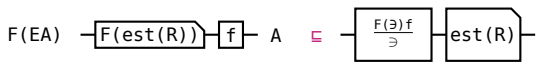
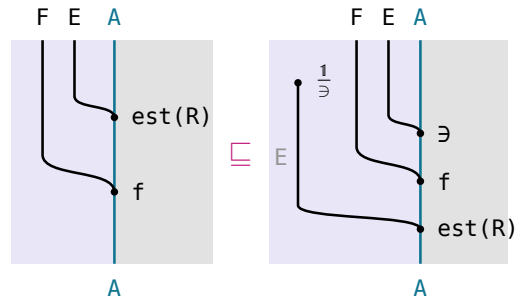
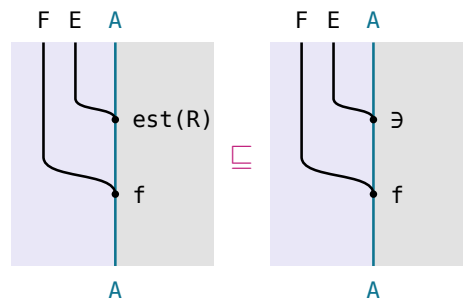
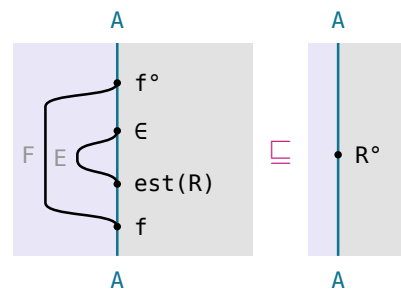
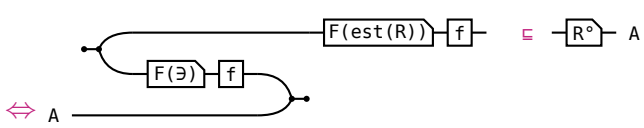
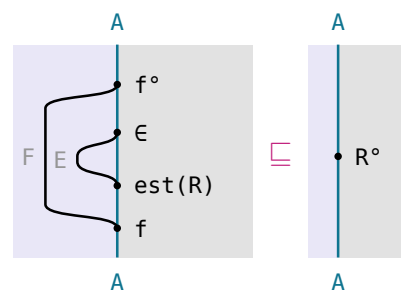
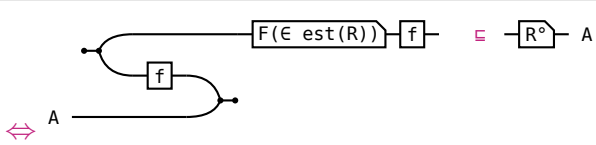
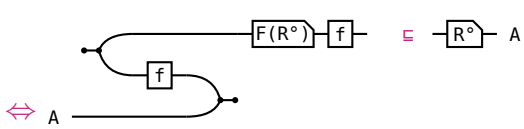
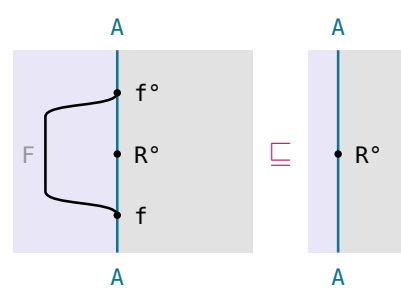
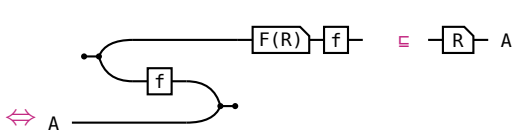
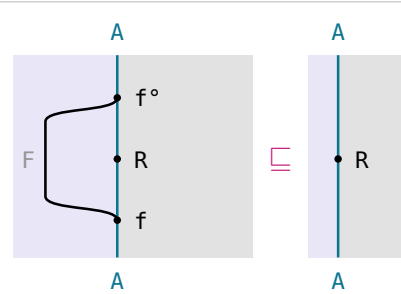
$$\begin{tikzcd} \begin{matrix} E\ Nat\times E\ Nat \\ (xs,ys) \end{matrix} \arrow[r, "\frac{(\ni\times\ni)+}{\ni}"] \arrow[d, "est(\le)\times est(\le)"'] \arrow[dr, phantom, "\Leftarrow"] & \begin{matrix} E\ Nat \\ \{x+y|x\in xs\wedge y\in ys\} \end{matrix} \arrow[d, "est(\le)"] \\ \begin{matrix} Nat\times Nat \\ (min\ xs,min\ ys) \end{matrix} \arrow[r, "+"'] & \begin{matrix} Nat \\ min\ xs+min\ ys \end{matrix} \end{tikzcd}$$

$(est(\le)\times est(\le))+ \sqsubseteq \frac{(\ni\times\ni)+}{\ni} est(\le)$

(13.3.1c)

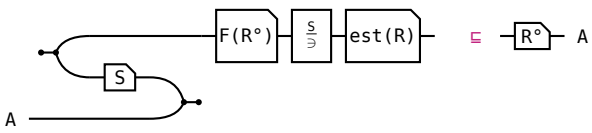
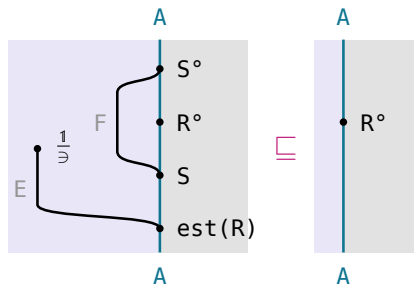
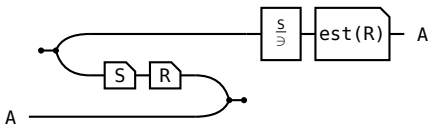
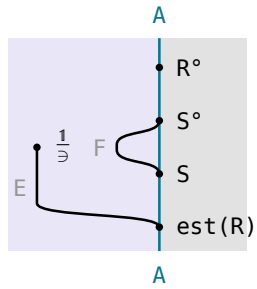
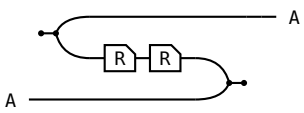
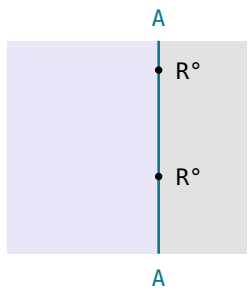
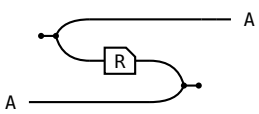
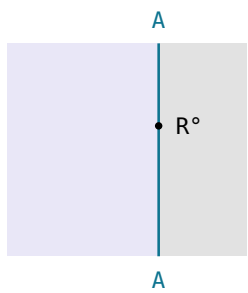
$f°F(R)f⊑R \Longleftrightarrow F(est(R))f⊑ \frac{F(∋)f}{∋}\ est(R)$

Theorem 7.1, f a map, R reflexive

circuit	Hinze–Marsden
 $F(est(R))f⊑ \frac{F(∋)f}{∋}\ est(R)$ $\frac{S}{∋} = \frac{1}{∋}\ E(S),\ S≜F(∋)f$ def. f distributes over R — (13.3.1a), (10.1a)	
$F(est(R))f ⊑ F(∋)f$ and ⇔ $(F(∋)f)°F(est(R))f ⊑ R°$ (13.1.5a) at $F(∋)f$; $F(est(R))f⊑(F(∋)f)°$ $\backslash R°⟺(F(∋)f)°F(est(R))f⊑R°$ — (8.1c) $est(R)≜∋∩(∈\backslash R°)$ — (13.1a)	 and 
⇔  $(F(∋)f)°F(est(R))f⊑R°$ $est(R)⊑∋$ — (13.1a); the first conjunct is free	
⇔  $f°F(∈ est(R))f⊑R°$ converse, relators: $(F(∋)f)°=f°F(∈)$ and $F(∈)F(est(R))=F(∈ est(R))$	
⇔  $f°F(R°)f⊑R°$ $∈ est(R)=R°$, R reflexive — (13.1a)	
⇔  $f°F(R)f⊑R$ f a map, so monotonic on R and $R°$ together	

§13.3.2. Greedy Theorem: $(\!|\frac{S}{\ni}\ \mathrm{est}(R)|\!) \sqsubseteq \frac{(\!|S|\!)}{\ni}\ \mathrm{est}(R)$, given S monotoic on R and R a preorder

(13.3.2a)

circuit	Hinze–Marsden
$(\!\|\frac{S}{\ni}\ \mathrm{est}(R)\|\!) \sqsubseteq \frac{(\!\|S\|\!)}{\ni}\ \mathrm{est}(R)$ the conclusion: one minimum kept at each step is below every result collected and one minimum taken at the end	
$(\!\|\frac{S}{\ni}\ \mathrm{est}(R)\|\!) \sqsubseteq (\!\|S\|\!)$ and $\Leftrightarrow (\!\|S\|\!)^\circ(\!\|\frac{S}{\ni}\ \mathrm{est}(R)\|\!) \sqsubseteq R^\circ$ (13.1.5a) at $(\!\|S\|\!)$; $(\!\|\frac{S}{\ni}\ \mathrm{est}(R)\|\!) \sqsubseteq (\!\|S\|\!)^\circ \backslash R^\circ \Longleftrightarrow (\!\|S\|\!)^\circ(\!\|\frac{S}{\ni}\ \mathrm{est}(R)\|\!) \sqsubseteq R^\circ$ — (8.1c)	
$\Leftarrow \frac{S}{\ni}\ \mathrm{est}(R) \sqsubseteq S$ the left conjunct; (13.1.5a), and $(\!\|-\|\!)$ is monotone	
 $\Leftarrow S^\circ F(R^\circ)(\!\|\frac{S}{\ni}\ \mathrm{est}(R)\|\!) \sqsubseteq R^\circ$ the right conjunct; $(\!\|S\|\!)^\circ(\!\|\frac{S}{\ni}\ \mathrm{est}(R)\|\!)$ is the least X with $X = S^\circ F(X)(\!\|\frac{S}{\ni}\ \mathrm{est}(R)\|\!)$ (11.6.4c) — so Knaster–Tarski leaves this one inequation $\frac{S}{\ni} = \frac{1}{\ni} E(S)$ — (10.1a)	
 $\sqsubseteq R^\circ S^\circ(\!\|\frac{S}{\ni}\ \mathrm{est}(R)\|\!)$ $S^\circ F(R^\circ) \sqsubseteq R^\circ S^\circ$, the converse of the hypothesis $F(R)S \sqsubseteq SR$	
 $\sqsubseteq R^\circ R^\circ$ $S^\circ(\!\|\frac{S}{\ni}\ \mathrm{est}(R)\|\!) \sqsubseteq S^\circ(S^\circ \backslash R^\circ) \sqsubseteq R^\circ$ — (13.1.5a), (8.1c)	
 $\sqsubseteq R^\circ$ 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]) (12a)	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]⦈ (12a)	[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

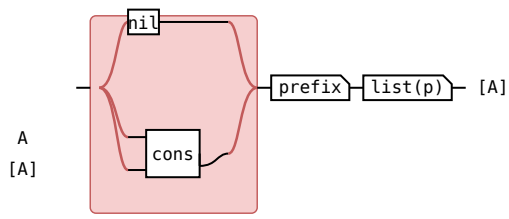
nil R=⊤, nil R°=nil nil is the shortest list — below every list, and above only itself, so it loses every est(R°)

(3.3.3b)

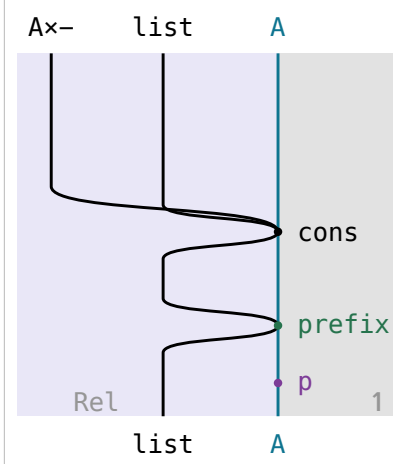
α prefix list(p)=F(prefix list(p))S

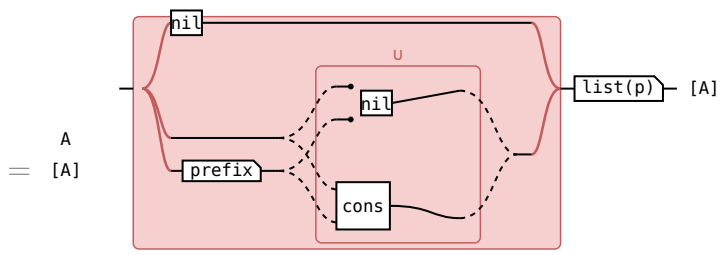
building the list and then keeping a **p**-passing prefix of it is keeping one of the tail first, and then building with **S**

circuit — the fork is `F([A])=1+A×[A]`: `nil` above, the pair below | **Hinze–Marsden**



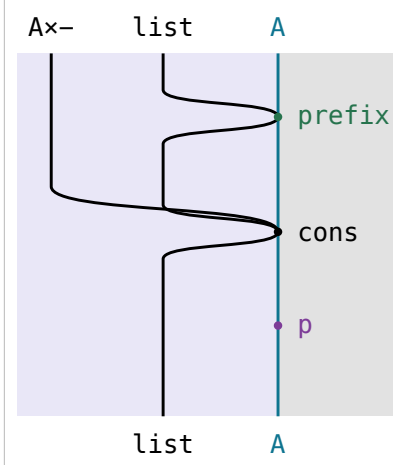
α prefix list(p)

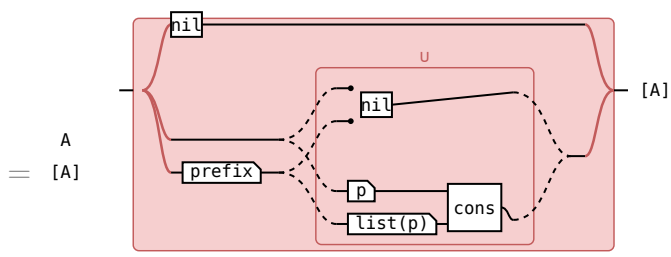




F(prefix) [nil,⊸ nil ∪ cons] list(p)

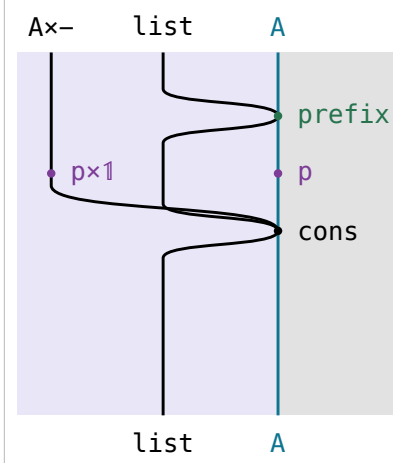
defining equation

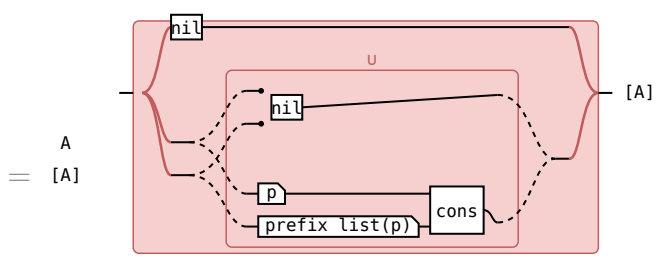




F(prefix) [nil,⊸ nil ∪ (p×list(p)) cons]

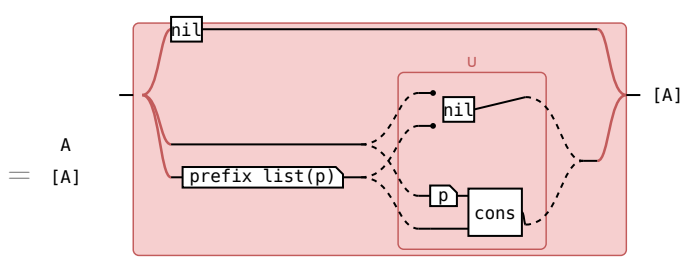
list(p) through cons





[nil,⊸ nil ∪ (p×(prefix list(p))) cons]

relator, `prefix` entire



F(prefix list(p))S

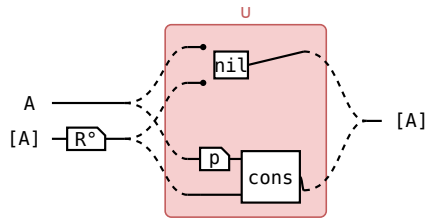
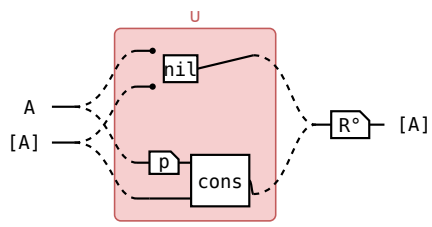
`prefix list(p)` entire

(11.6.1a) reads that off as `prefix list(p)=⦇S⦈`. (11.4.2a) cannot: `list(p)` is not entire, `(𝟙×list(p))⊸ nil⊏⊸ nil`, and no algebra meets the side condition.

(13.3.3c)

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

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

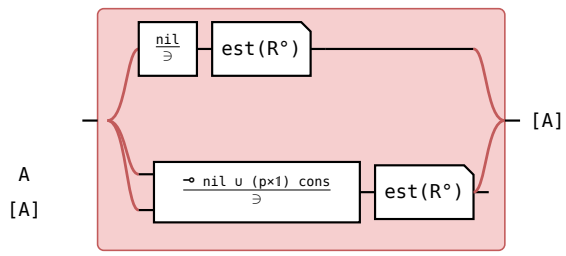
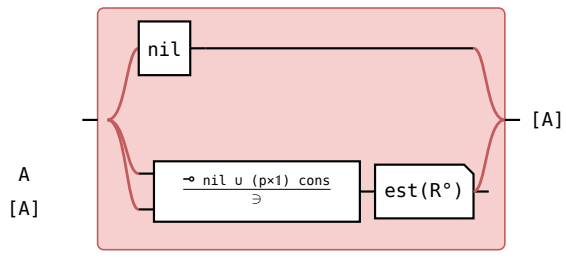
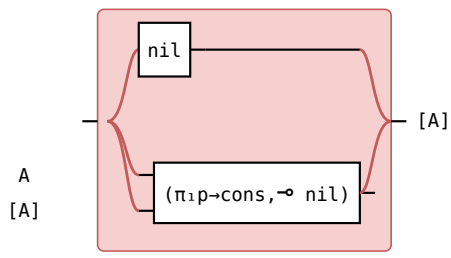
formula — the cons branch of F(R°)S⊑SR°	**reason**
(𝟙×R°)⊸ nil⊑⊸ nil R°	nil R°=nil (13.3.3a) — the right side is ⊸ nil, and (𝟙×R°)⊸⊑⊸
(𝟙×R°)(p×𝟙) cons⊑(p×𝟙) cons R°	composition (13.2c) at (𝟙×R°) (p×𝟙)=(p×R°)=(p×𝟙)(𝟙×R°) and (𝟙×R°) cons⊑cons R°
 (𝟙×R°)(⊸ nil ∪ (p×𝟙) cons)	
⊑  (⊸ nil ∪ (p×𝟙) cons)R°	union (13.2c) at φ:=⊸ nil, ψ:=(p×𝟙) cons

the nil branch is nil⊑nil R°. (𝟙×R°) cons⊑cons R° is cons length=(𝟙×length)π₂ succ with succ monotone — a shorter tail makes a shorter list.

(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**
F([A]) — $\frac{S}{\ni}$ E([A]) — est(R°) — [A] $\qquad \frac{S}{\ni}$ est(R°)	
=  $[\frac{nil}{\ni}$ est(R°), $\frac{⊸ nil ∪ (p×𝟙) cons}{\ni}$ est(R°)]	coproduct of maps
=  [nil, $\frac{⊸ nil ∪ (p×𝟙) cons}{\ni}$ est(R°)]	singleton, R° reflexive
=  [nil,(π₁p→cons,⊸ nil)]	nil R=⊤

the set is {nil} where p fails on the head and {nil,cons(a,xs)} where it holds, and nil loses the second — (13.1a) at a two-element set.

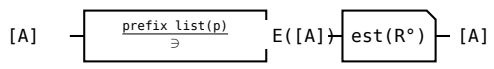
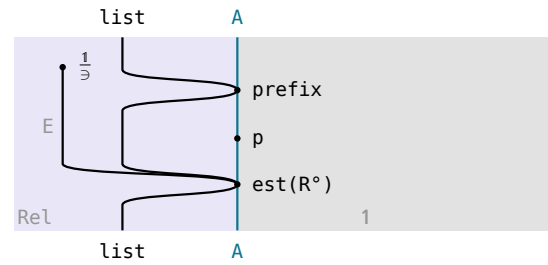
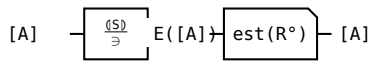
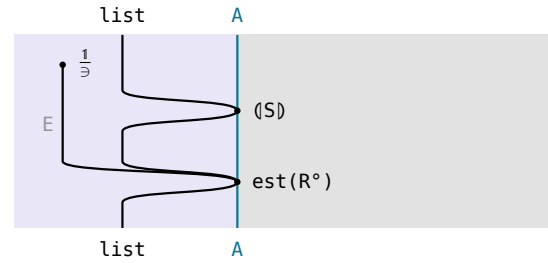
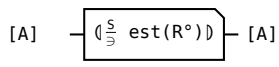
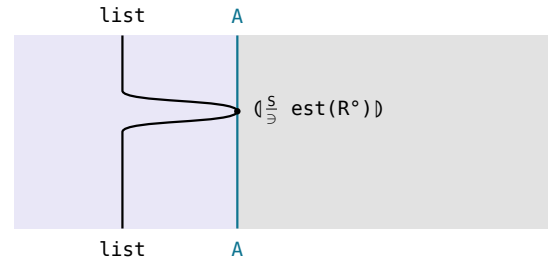
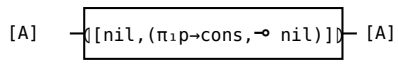
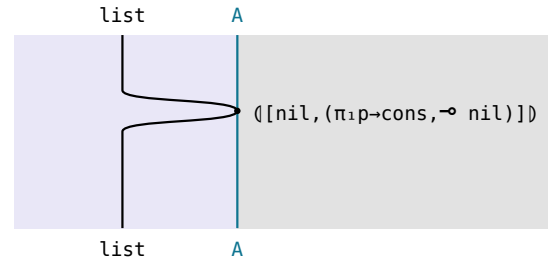
(13.3.3e)

name	**definition**	**type**	**example**	**in words**
S	[nil,⊸ nil ∪ (p×𝟙) cons] (13.3.3a)	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)]⦈

Ex 7.39: the longest prefix all of whose elements pass **p**

circuit	Hinze–Marsden
 $\frac{\text{prefix list(p)}}{\ni}$ est(R°) the specification — (13.1a)'s est(R°)	
=  $\frac{⦇S⦈}{\ni}$ est(R°) (13.3.3b)	
 ⊒ ⦇$\frac{S}{\ni}$ est(R°)⦈ (13.3.2a) 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	
 = ⦇[nil,(π₁p→cons,⊸ nil)]⦈ (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 wrap,((𝟙×head)⊕,π₂) cons]⦈ est(≥)

(13.3.4a)

FX=1+A×X, A:=Int, α≜[nil,cons], sum=⦇[zero,plus]⦈ and segment=suffix prefix from (7a) and (12a).

head≜cons° π₁, wrap≜⟨𝟙,⊸ nil⟩ cons the head of a list and the one-element list, beside (12a)'s tail≜cons° π₂

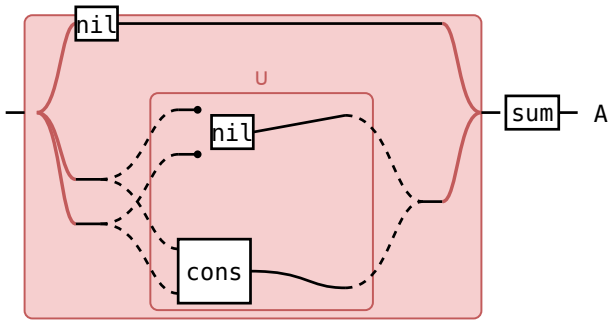
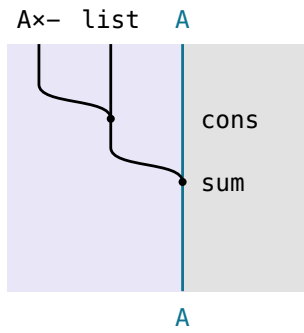
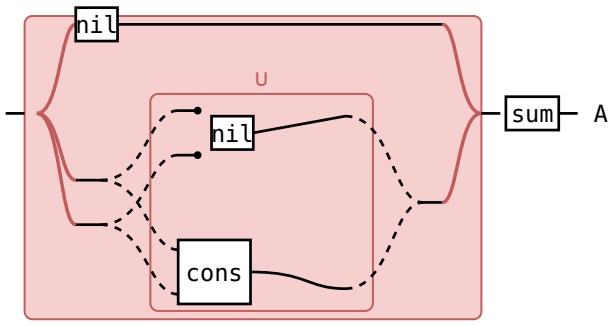
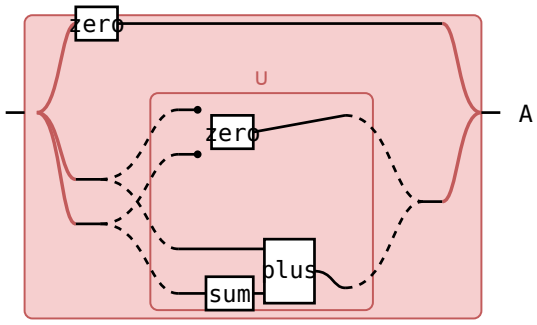
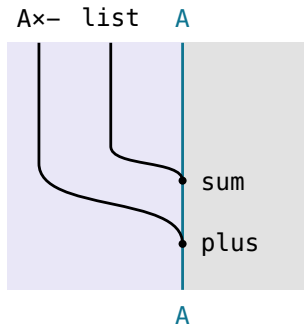
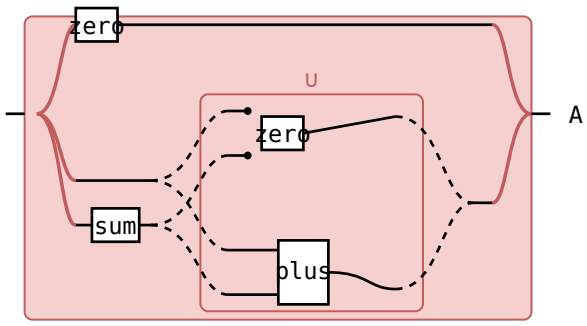
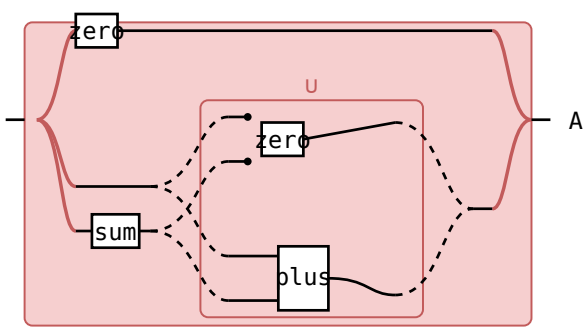
⊕≜ $\frac{\text{⊸ zero ∪ plus}}{\ni}$ est(≥) B&dM's oplus=max(Λ(zerouplus)); the set at (a,b) is {0,a+b}, so ⊕ is the larger of the two

$\frac{\texttt{segment sum}}{\ni}$ est(≥)=$\frac{\texttt{suffix}}{\ni}$ E($\frac{\texttt{prefix sum}}{\ni}$ est(≥)) est(≥)	(3.3.4b)
formula — one wire from [A] to A, its type written along it	**reason**
· [A] · $\boxed{\frac{\texttt{segment sum}}{\ni}}$ · EA · $\boxed{\texttt{est(≥)}}$ · A · $\qquad \frac{\texttt{segment sum}}{\ni}$ est(≥)	
= · [A] · $\boxed{\frac{\texttt{suffix (prefix sum)}}{\ni}}$ · EA · $\boxed{\texttt{est(≥)}}$ · A · $\qquad \frac{\texttt{suffix (prefix sum)}}{\ni}$ est(≥)	segment=suffix prefix (12a), (13.3.4a)
= · [A] · $\boxed{\frac{\texttt{suffix}}{\ni}}$ · E([A]) · $\boxed{\texttt{E(prefix sum)}}$ · EA · $\boxed{\texttt{est(≥)}}$ · A · $\qquad \frac{\texttt{suffix}}{\ni}$ E(prefix sum) est(≥)	absorption (10b) — frac(S,∋) E(R)=frac(SR,∋) at S:=suffix, R:=prefix sum
= · [A] · $\boxed{\frac{\texttt{suffix}}{\ni}}$ · E([A]) · $\boxed{\texttt{E}(\frac{\texttt{prefix sum}}{\ni})}$ · E(EA) · $\boxed{\texttt{union}}$ · EA · $\boxed{\texttt{est(≥)}}$ · A · $\qquad \frac{\texttt{suffix}}{\ni}$ E($\frac{\texttt{prefix sum}}{\ni}$) union est(≥)	$\frac{\texttt{R}}{\ni}$ ∋=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 (11a)'s F(RS)=F(R)F(S) at F:=E
= · [A] · $\boxed{\frac{\texttt{suffix}}{\ni}}$ · E([A]) · $\boxed{\texttt{E}(\frac{\texttt{prefix sum}}{\ni})}$ · E(EA) · $\boxed{\texttt{E(est(≥))}}$ · EA · $\boxed{\texttt{est(≥)}}$ · A · $\qquad \frac{\texttt{suffix}}{\ni}$ E($\frac{\texttt{prefix sum}}{\ni}$) E(est(≥)) est(≥)	(13.1b), the sets non-empty
= · [A] · $\boxed{\frac{\texttt{suffix}}{\ni}}$ · E([A]) · $\boxed{\texttt{E}(\frac{\texttt{prefix sum}}{\ni}\ \texttt{est(≥)})}$ · EA · $\boxed{\texttt{est(≥)}}$ · A · $\qquad \frac{\texttt{suffix}}{\ni}$ E($\frac{\texttt{prefix sum}}{\ni}$ est(≥)) est(≥)	relator (11a) — F(RS)=F(R)F(S) at F:=E

B&dM's max(P(max(Λ(sum prefix))))Λsuffix, mirrored. 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]

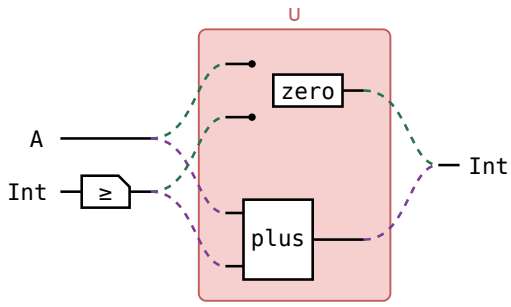
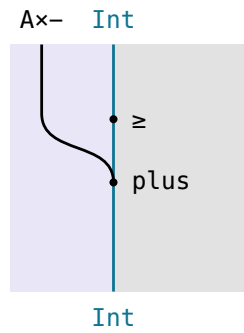
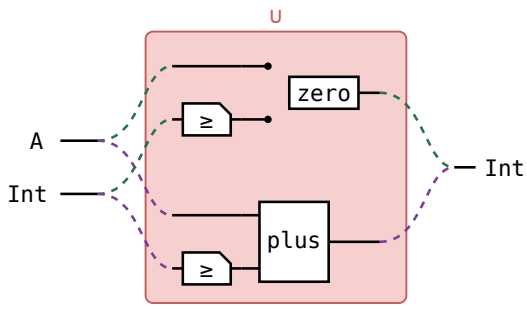
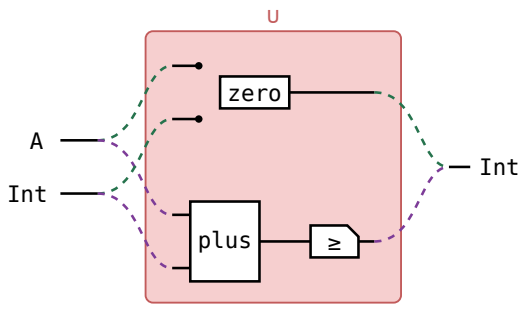
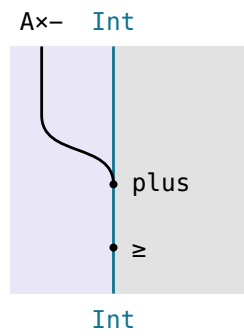
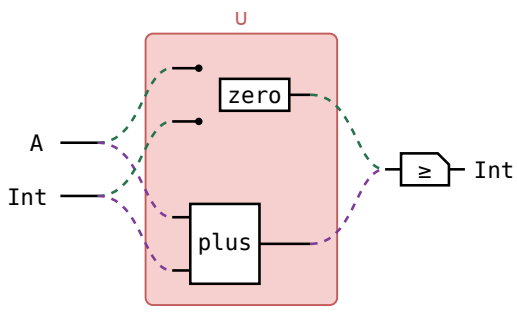
circuit — the fork is the bracket's case split F([A])=𝟙+A×[A]: nil above, the pair and its ∪ below	**Hinze–Marsden**
 [nil,⊸ nil ∪ cons] sum	
=  [nil sum,⊸ nil sum∪cons sum] coproduct of maps, composition over ∪	
=  [zero,⊸ zero ∪ (𝟙×sum) plus] sum's defining equation	
=  [zero,(𝟙×sum)(⊸ zero ∪ plus)]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

(11.4.2a) 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)≥

the **plus** branch of **F(≥)S⊑S≥**; the **zero** branch is **zero⊑zero≥**

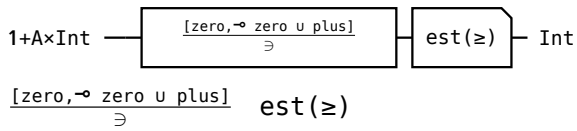
circuit — the head above, the running sum below; the tape is the ∪	**Hinze–Marsden**
 (𝟙×≥)(⊸ zero ∪ plus)	
=  (𝟙×≥)⊸ zero ∪ (𝟙×≥) plus relator, composition over ∪	
⊑  ⊸ zero ∪ plus≥ (5.1a), (≥×≥) plus⊑plus≥; (≤×≤) plus⊑plus≤ is (13.3a), written + there, and plus is a map, so it is monotonic on an order and on its opposite together, which carries it to ≥.	
⊑  (⊸ zero ∪ plus)≥ ≥ reflexive	

(13.3.4e)

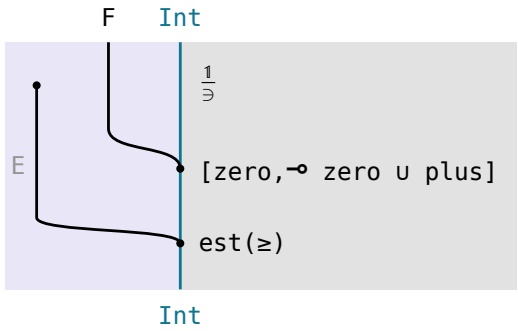
$\frac{\texttt{[zero,-∘ zero ∪ plus]}}{\ni}$ `est(≥)=[zero,⊕]`

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

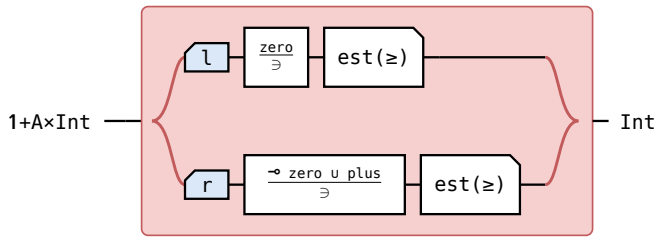
Hinze–Marsden



$\frac{\texttt{[zero,-∘ zero ∪ plus]}}{\ni}$ `est(≥)`



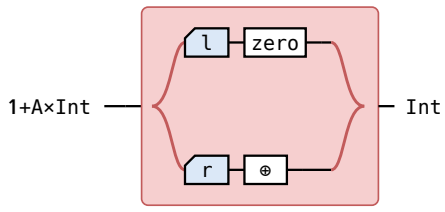
=



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

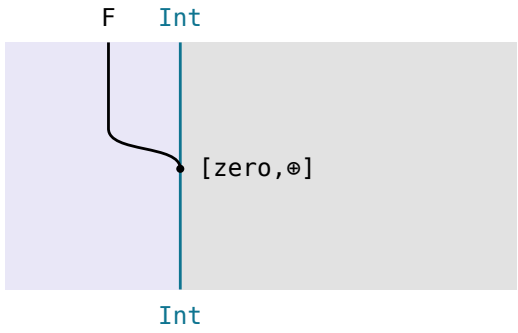
coproduct of maps — (11.2.1a) at `T:=[zero,-∘ zero ∪ plus]`, then `[U,V]Z=[UZ,VZ]` — (11.2a), composition over `∪`

=



`[zero,⊕]`

singleton, ≥ reflexive — (13.1b)'s $\frac{1}{\ni}$ `est(R)=𝟙∩R` at `R:=≥`, `zero` a map; the lower branch is ⊕'s definition, (13.3.4a), and no law

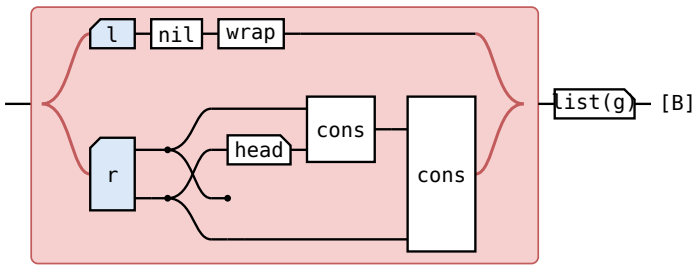


with (13.3.4d) the greedy theorem gives `⦇[zero,⊕]⦈⊑` $\frac{\texttt{prefix sum}}{\ni}$ `est(≥)` — B&dM's own containment — and (13.3.4g) makes it an equality.

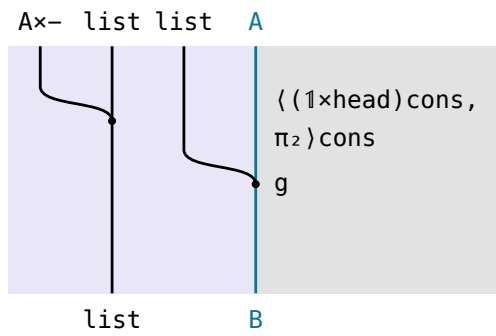
[nil wrap,⟨(𝟙×head) cons,π₂⟩ cons] list(g)=F(list(g))[c wrap,⟨(𝟙×head)f,π₂⟩ cons] (13.3.4f)

circuit — the **1** branch of **1+A×[[A]]** above, the root and its list of tails below

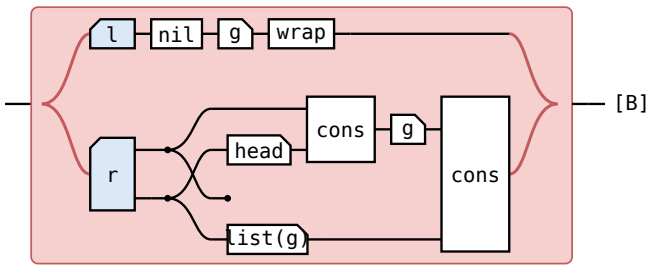
Hinze–Marsden



[nil wrap,⟨(𝟙×head) cons,π₂⟩ cons] list(g)
g≜⦇[c,f]⦈ and tails=⦇[nil wrap,⟨(𝟙×head) cons,π₂⟩ cons]⦈, since tails(cons(a,xs)) is cons(cons(a,head(tails xs)),tails xs)

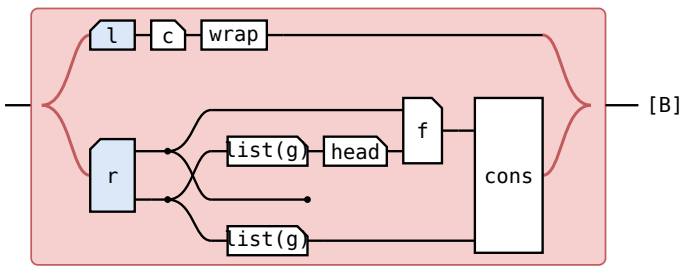


=



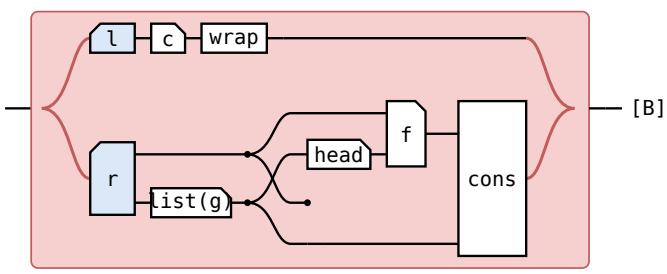
[nil g wrap,⟨(𝟙×head) cons g,π₂ list(g)⟩ cons]
coproduct of maps, wrap and cons natural

=

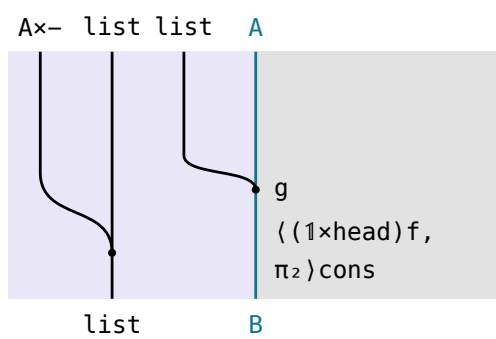


[c wrap,⟨(𝟙×(list(g) head))f,(𝟙×list(g))π₂⟩ cons]
g's defining equation, list(g) head=head g; the π₂ slide π₂list(g)=(𝟙×list(g))π₂ is 1 and 4 of (11.1.2b), Dom(π₁)=𝟙 ((5c)); list(g) head=head g is the one step the note has no law for — head is undefined at nil, and the equality is a fact about tails, which never returns an empty list, not a naturality square

=



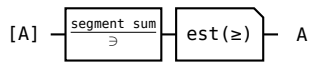
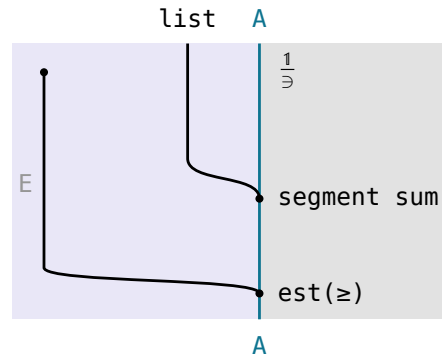
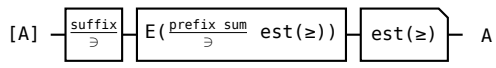
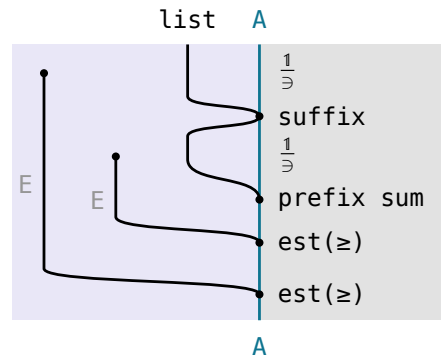
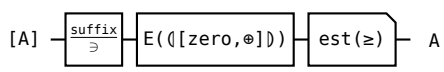
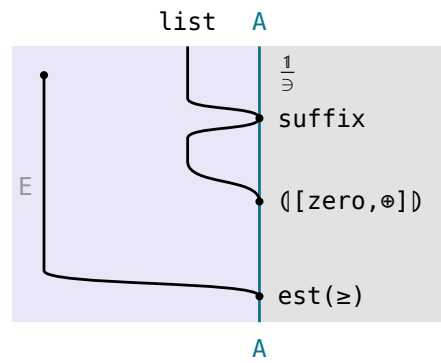
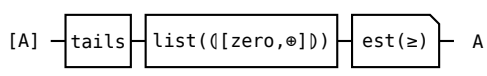
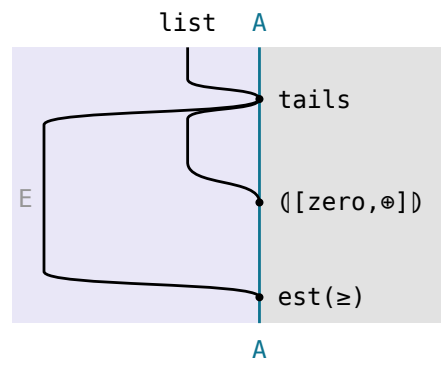
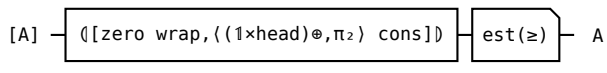
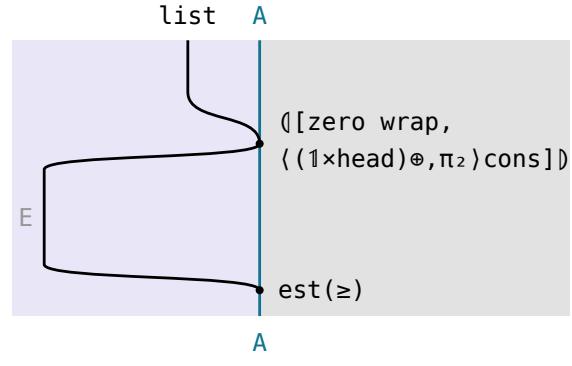
F(list(g))[c wrap,⟨(𝟙×head)f,π₂⟩ cons]
fork, relator — the fork slide is 7 of (11.1.2b), 𝟙×list(g) a map there — and it is at c,f:=zero,⊕. (11.4.2a) then reads off tails list(g)=⦇[c wrap,⟨(𝟙×head)f,π₂⟩ cons]⦈, the heading's fold.



(13.3.4g)

mss=⦇[zero wrap,⟨(𝟙×head)⊕,π₂⟩ cons]⦈ est(≥)

Ex 7.40, $\oplus \overset{A}{=} \frac{\multimap \text{zero} \cup \text{plus}}{\ni}$ est(≥) — (13.3.4a)

circuit	Hinze–Marsden
 segment sum / ∋ est(≥) mss is the greatest of the segment sums — (13.3.4a)	
=  suffix / ∋ E(prefix sum / ∋ est(≥)) est(≥) (13.3.4b)	
=  suffix / ∋ E(⦇[zero,⊕]⦈) est(≥) the greedy theorem (13.3.2a) at R:=≥, S:=[zero,⊸ zero ∪ plus] — (13.3.4d) is its condition and (13.3.4e) its $\frac{S}{\ni}$ est(≥); its ⊑ is an = because ⦇[zero,⊕]⦈ is entire and $\frac{\text{prefix sum}}{\ni}$ est(≥) simple (13.3.3f)'s last row	
=  tails list(⦇[zero,⊕]⦈) est(≥) tails implements $\frac{\text{suffix}}{\ni}$ and list(f) implements E(f) — (12a)	
=  ⦇[zero wrap,⟨(𝟙×head)⊕,π₂⟩ cons]⦈ est(≥) (11.4.2a) at (13.3.4f)'s side condition, c,f:=zero,⊕	

one fold builds the n+1 running maxima and the final est(≥) reads them in one more pass, so mss is linear.

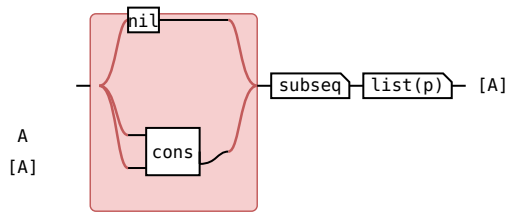
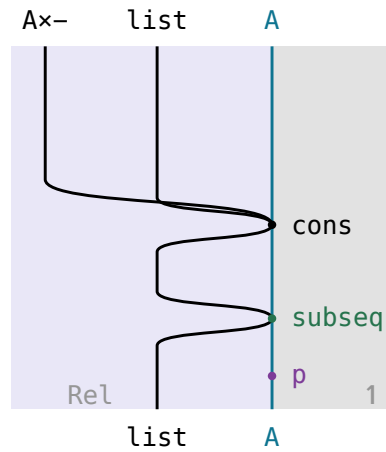
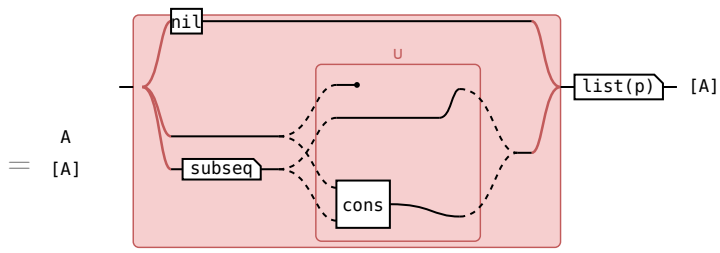
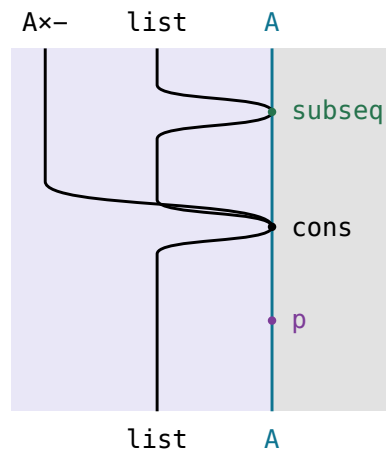
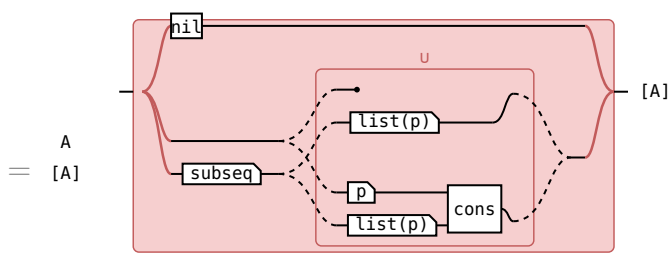
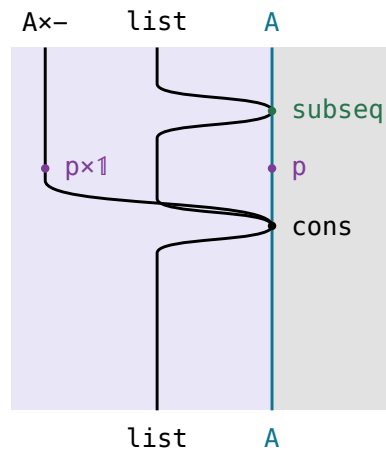
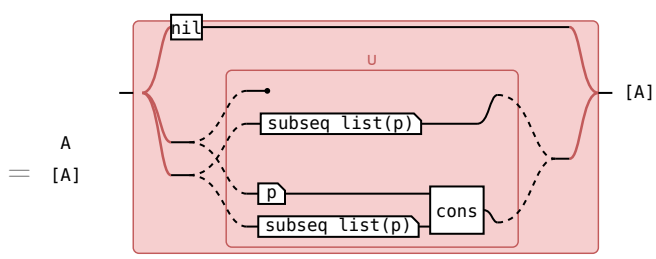
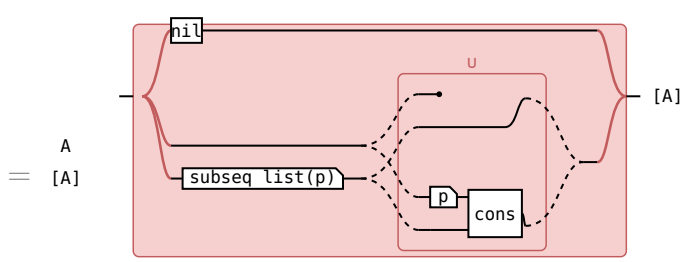
§13.3.5. filter(p)=⦇[nil,(π₁p→cons,π₂)]⦈

(13.3.5a)

name	definition	type	example	in words
F, α, p, R	as in (13.3.3a)			
π₂	π₂ where (13.3.3a) has ⊸ nil	A×[A]→[A]	π₂(3,[1,2])=[1,2]	drop the head, keep the tail
subseq	⦇[nil,π₂ ∪ cons]⦈ (12a)	[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 (13.3.3a)'s loser is nil

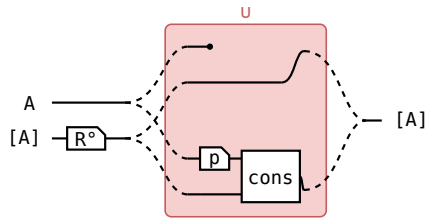
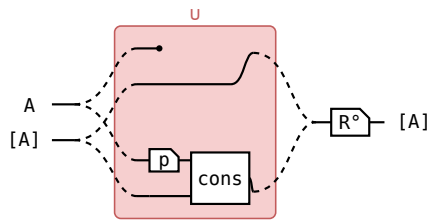
α subseq list(p)=F(subseq list(p))S (13.3.5b)

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

circuit — the fork is `F([A])=1+A×[A]`: `nil` above, the pair below	**Hinze–Marsden**
 α subseq list(p)	
=  F(subseq) [nil,π₂ ∪ cons] list(p) defining equation	
=  F(subseq) [nil,(𝟙×list(p))π₂ ∪ (p×list(p)) cons] list(p) through cons, π₂ natural	
=  [nil,(𝟙×(subseq list(p)))π₂ ∪ (p×(subseq list(p))) cons] relator, subseq entire	
=  F(subseq list(p))S subseq list(p) entire	

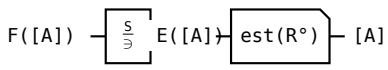
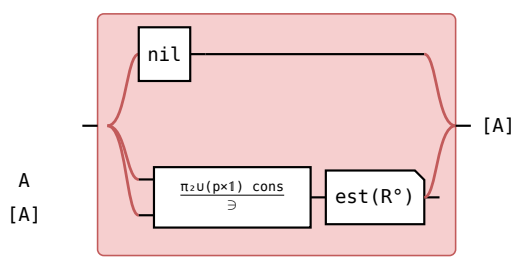
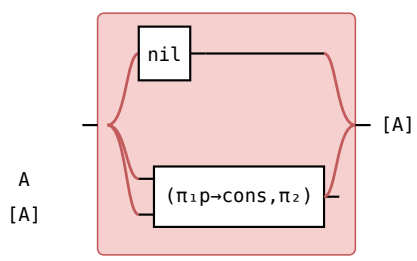
(11.6.1a) reads that off as `subseq list(p)=(|S|)` — (13.3.3b)'s chain with π₂ for ⊸ `nil`. Here (11.4.2a) is blocked for the same reason, and π₂ is the branch that carries `list(p)` out by naturality ((12.1c) at `∋:=list(p)`).

(13.3.5c)

(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**
(𝟙×R°)π₂=π₂R°	π₂ natural (12.1c) — the right side is an EQUALITY here
(𝟙×R°)(p×𝟙) cons⊑(p×𝟙) cons R°	(13.3.3c)'s cons branch, unchanged
 (𝟙×R°)(π₂∪(p×𝟙) cons)	
⊑  (π₂∪(p×𝟙) cons)R°	union (13.2c) at X:=π₂, Y:=(p×𝟙) cons

F(R°)S⊑SR°, the nil branch again nil⊑nil R°. The first step is an equality: (13.3.3c) buys its ⊸ nil branch with nil R°=nil, and π₂ needs only its naturality square.

(13.3.5d)

$\frac{S}{\ni}$ 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**
 $\frac{S}{\ni}$ est(R°)	
=  [nil, $\frac{\pi_2\cup(p\times\mathbb{1})\ \text{cons}}{\ni}$ est(R°)]	(13.3.3d)'s first two steps
=  [nil,(π₁p→cons,π₂)]	𝟙⊑π₂R cons°

at (a,xs) the set is {xs} where p fails on a and {xs,cons(a,xs)} where it holds, and xs loses the second — (13.1a) at a two-element set. The head is dropped, not the whole tail: that is the one place π₂ shows against (13.3.3d)'s ⊸ nil.

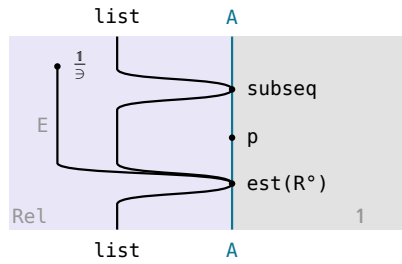
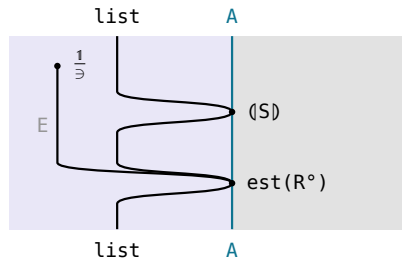
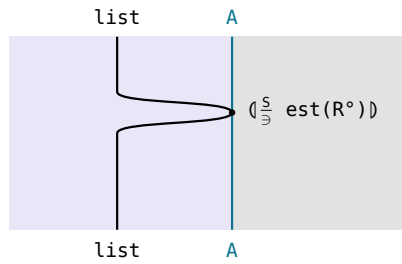
(13.3.5e)

name	**definition**	**type**	**example**	**in words**
S	[nil,π₂∪(p×𝟙) cons] (13.3.5a)	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

(13.3.5f)

filter(p)=⦇[nil,(π₁p→cons,π₂)]⦈

Ex 7.41, **R** a preorder: the longest subsequence all of whose elements pass **p**

circuit — one wire, its type written along it	**Hinze–Marsden**
[A] — $\frac{\text{subseq list(p)}}{\ni}$ E([A]) — est(R°) — [A] filter(p)$\triangleq\frac{\text{subseq list(p)}}{\ni}$ est(R°) (12a)	
= [A] — $\frac{⦇S⦈}{\ni}$ E([A]) — est(R°) — [A] $\frac{⦇S⦈}{\ni}$ est(R°) subseq list(p)=⦇S⦈ — (13.3.5b)	
⊑ [A] — ⦇$\frac{S}{\ni}$ est(R°)⦈ — [A] ⦇$\frac{S}{\ni}$ est(R°)⦈ (13.3.2a) at R°, whose hypothesis F(R°)S⊑SR° is (13.3.5c)	
= [A] — ⦇[nil,(π₁p→cons,π₂)]⦈ — [A] ⦇[nil,(π₁p→cons,π₂)]⦈ (13.3.5d)	
= ⦇[nil,(π₁p→cons,π₂)]⦈ entire and filter(p) simple an entire relation below a simple one equals it; the simplicity is (12a)'s Ex 7.41, not a law of this note	

(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.

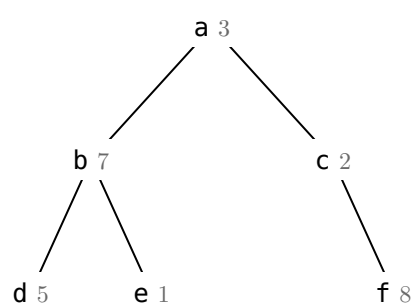
§13.4. Planning a company party

(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)



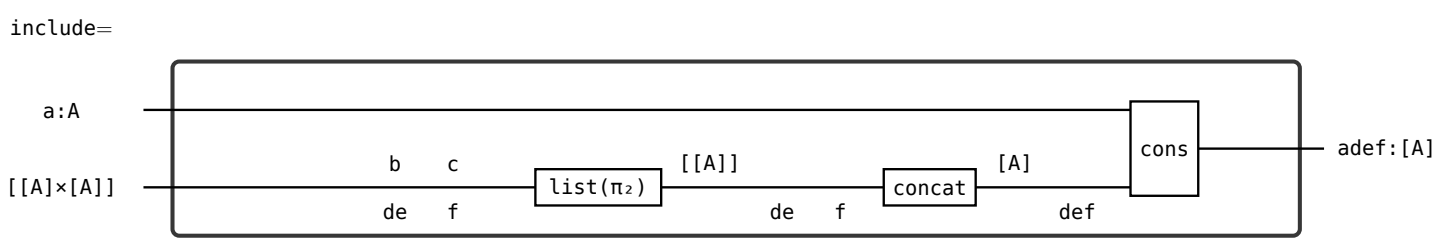
the small grey number is the employee's rating

(13.4.1b)

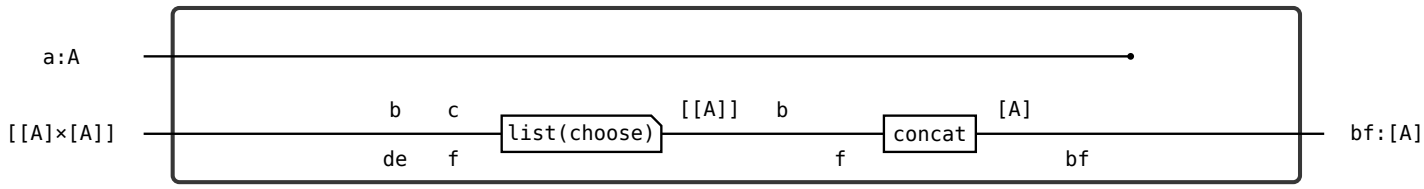
node	the algebra's input `A×[[A]×[A]]`	**`include`**	**`exclude`**
`d`, `e`, `f`	`(d,[])`	`[d]`	`[]`
`b`	`(b,[([d],[]),([e],[])])`	`[b]`	`[d,e]`, `[d]`, `[e]`, `[]`
`c`	`(c,[([f],[])])`	`[c]`	`[f]`, `[]`
`a`	`(a,[([b],…),([c],…)])`	`[a]⧺exclude(b)⧺exclude(c)`	`choose(b)⧺choose(c)`

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)



(13.4.1d)



(13.4.1e)

```
list(choose) [([d],[]),([e],[])]
  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)°)`

(13.4.2a)

```
[  ([b],[d,e])  ,  ([c],[f])  ]
        |              |
        ▼              ▼        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`

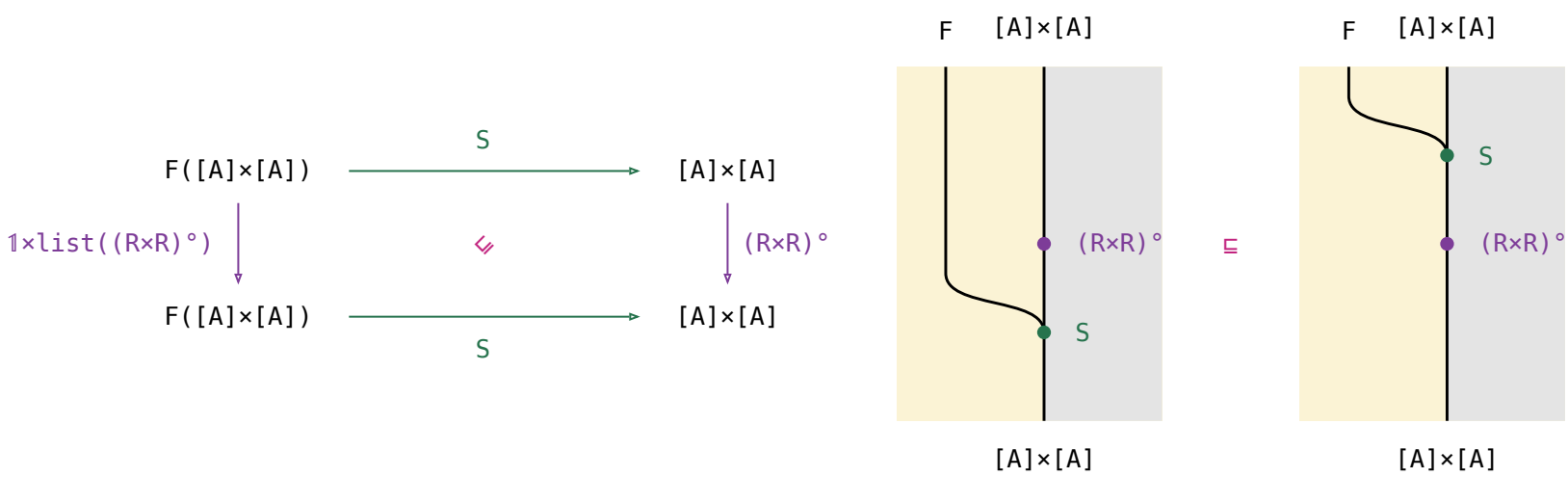
the first component of `𝟙×list((R×R)°)` is the root employee.

`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)

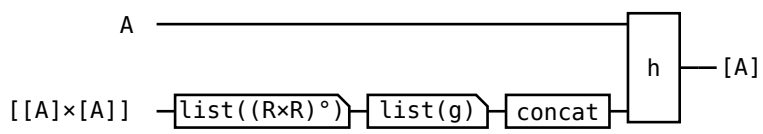
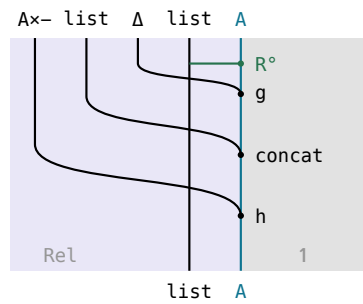
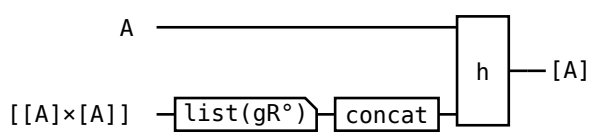
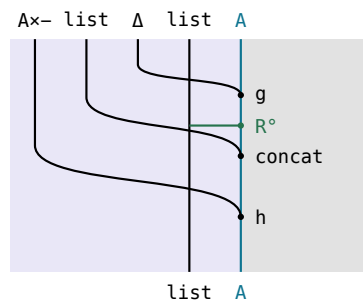
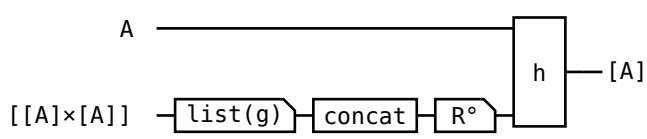
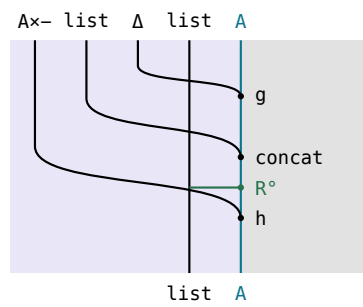
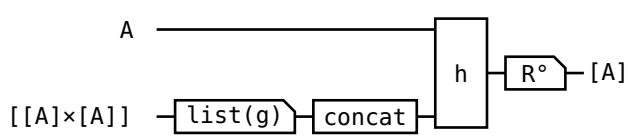
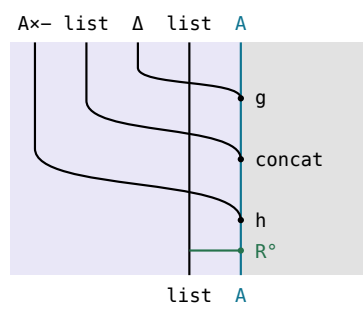
$$\text{tree A} \xrightarrow[\ni]{(\!|S|\!)\ \text{choose}} \text{E[A]} \quad = \quad \text{tree A} \xrightarrow[\ni]{(\!|S|\!)} \text{E([A}\times\text{[A])} \xrightarrow{\text{E(choose)}} \text{E[A]}$$

$$\frac{(\!|S|\!)\ \text{choose}}{\ni} = \frac{(\!|S|\!)}{\ni}\ \text{E(choose)} \qquad \text{(10b), absorption}$$

(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, (11a)

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. B&dM's exercise.

h `(𝟙×R°)h⊑hR°`
a LEAF: `cost(cons(a,xs)) =rating(a)+cost(xs)`, so a cheaper tail makes a cheaper list. B&dM's exercise. 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

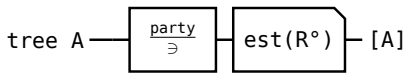
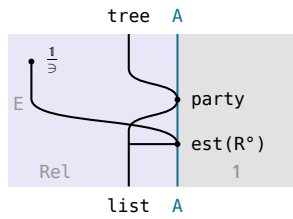
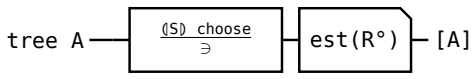
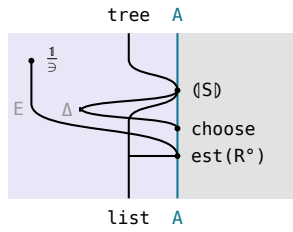
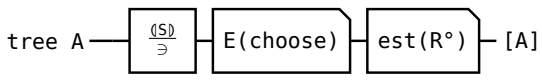
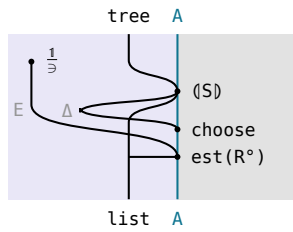
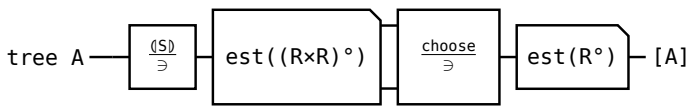
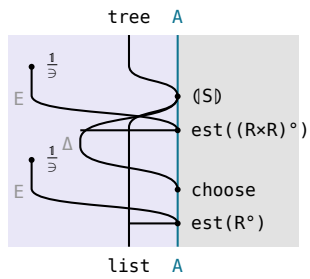
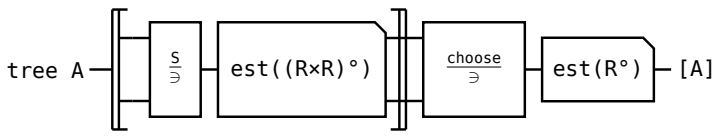
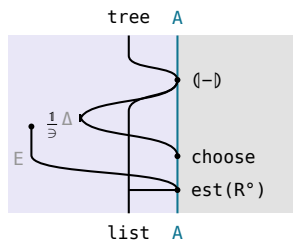
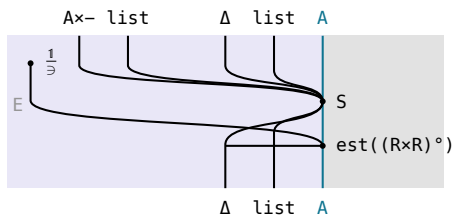
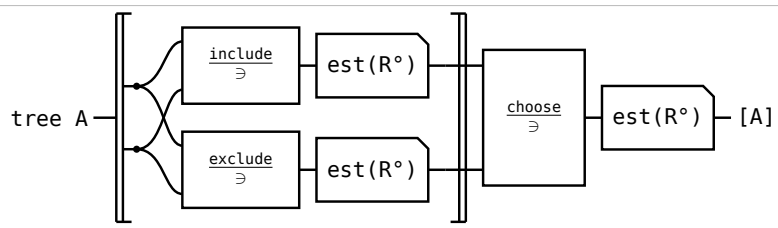
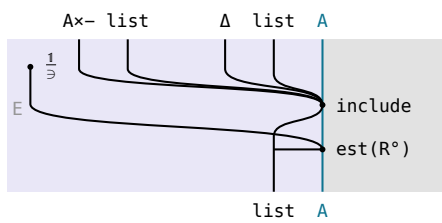
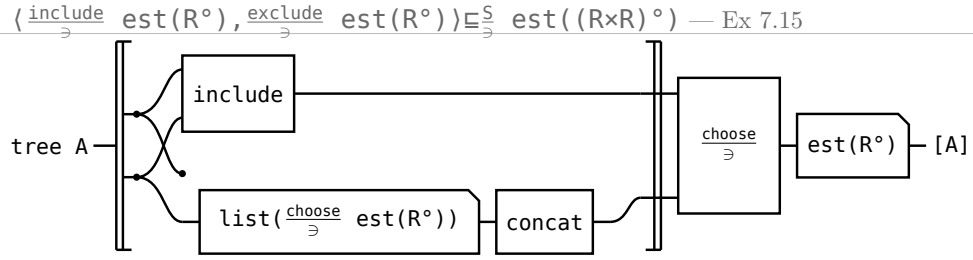
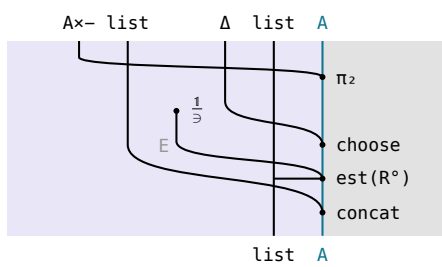
(13.4.3d)

`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`

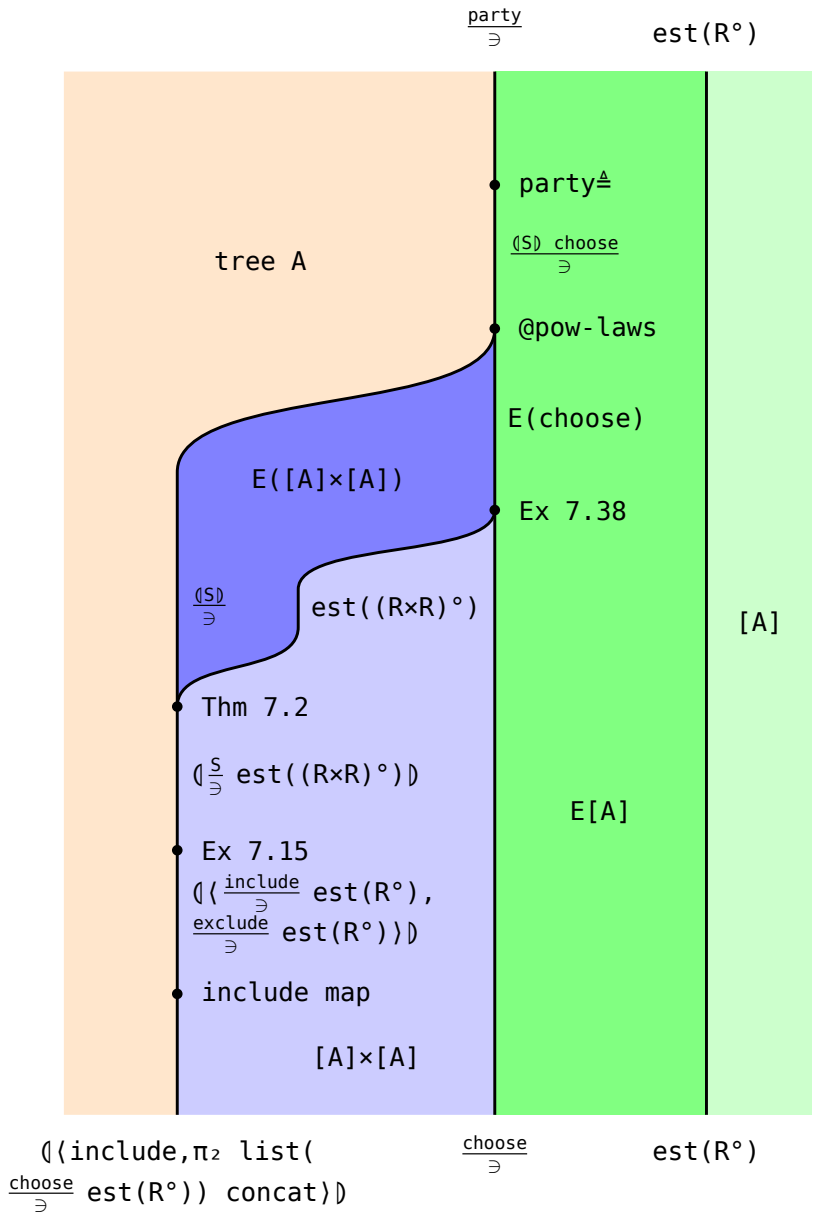
§13.4.4. The derivation

$\frac{\text{party}}{\ni}$ 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 ⦇ ⦈ above, inside it below; a fork drawn at one branch
 $\frac{\text{party}}{\ni}$ est(R°) the specification — (13.4a)	
=  $\frac{\text{⦇S⦈ choose}}{\ni}$ est(R°) party≜⦇S⦈ choose — (13.4a)	
=  $\frac{\text{⦇S⦈}}{\ni}$ E(choose) est(R°) $\frac{\text{⦇S⦈ choose}}{\ni}$=$\frac{\text{⦇S⦈}}{\ni}$ E(choose) — (13.4.3b)	
⊑  $\frac{\text{⦇S⦈}}{\ni}$ est((R×R)°) $\frac{\text{choose}}{\ni}$ est(R°) (R×R)°choose⊑choose R° — Ex 7.38, (13.4.3c)'s g row	
⊑  ⦇$\frac{\text{S}}{\ni}$ est((R×R)°)⦈ $\frac{\text{choose}}{\ni}$ est(R°) (𝟙×list((R×R)°))S⊑S(R×R)° — Theorem 7.2 at (R×R)°, (13.4.3a)	 
⊑  ⦇⟨$\frac{\text{include}}{\ni}$ est(R°), $\frac{\text{exclude}}{\ni}$ est(R°)⟩⦈ $\frac{\text{choose}}{\ni}$ est(R°) ⟨$\frac{\text{include}}{\ni}$ est(R°), $\frac{\text{exclude}}{\ni}$ est(R°)⟩⊑$\frac{\text{S}}{\ni}$ est((R×R)°) — Ex 7.15	
⊑  ⦇⟨include,π₂ list($\frac{\text{choose}}{\ni}$ est(R°)) concat⟩⦈ $\frac{\text{choose}}{\ni}$ est(R°) include a map, est(R°) into each branch	

(13.4.4b)



include map	⊑	include a map, est(R°) into each branch
Ex 7.15	⊑	the fork splits
Thm 7.2	⊑	(𝟙×list((R×R)°))S⊑S(R×R)°
Ex 7.38	⊑	(R×R)°choose⊑choose R°
@pow-laws	=	absorption
party≜	=	definition of party

§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=[wrap,cons]`: 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`	Forgets which row a component came from, so the `n` answers become one set.
`moves`	`NA→E(NA)`	The three columns a path may step to: rotated up, unrotated, rotated down.
`trans`	`E(NA)→N(EA)`	Transposes a set of tuples, so every row collects its own paths.
`zip`	`F(NA,NB)→NF(A,B)`	Commutes `N` with the base functor, pairing each new square with its own row's paths.
`cp≜` $\frac{\texttt{F(1,∋)}}{\texttt{∋}}$	`F(A,EB)→E(F(A,B))`	Turns a square paired with a set of paths into the set of extensions of those paths.
`generate≜F(𝟙,moves trans N(union)) zip N(cp P(α))`	`F(NA,N(E(LA)))→N(E(LA))`	One fold step: extends every path of every row by the new column.
`paths≜⦇generate⦈ setify union`	`L N Nat→E(L Nat)`	Every path across the cylinder.
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      [(1,2,3,4),(5,6,7,8)] : L N Nat
3  7
4  8
```

(13.5.1b)

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

generate((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]} )
```

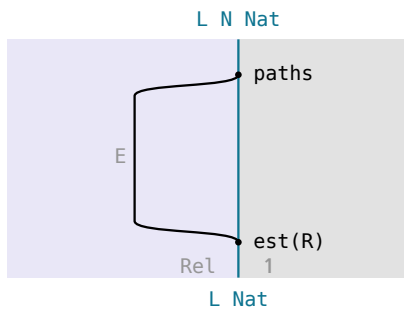
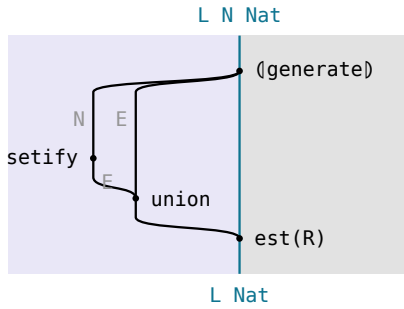
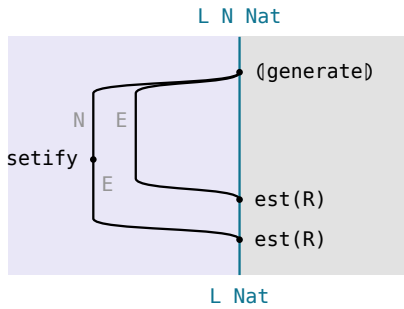
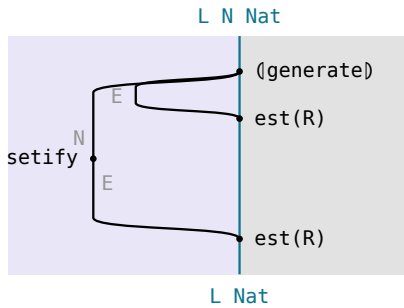
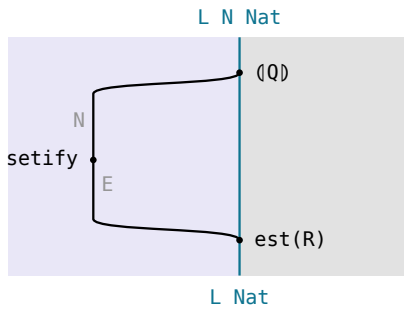
(13.5.1c)

`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 (13.5a): component `k` collects the paths of rows `k-1`, `k`, `k+1`.

§13.5.2. The derivation

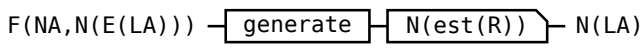
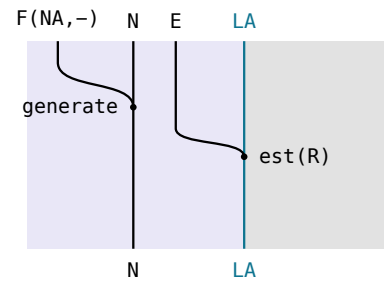
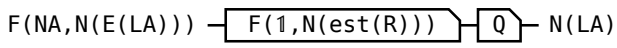
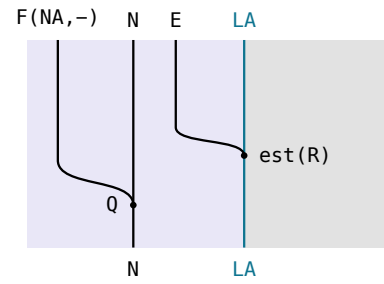
(13.5.2a)

paths est(R)≡⦇Q⦈ setify est(R) B&dM §7.4, p. 182; Q is (13.5.2c)'s algebra	
circuit	**Hinze–Marsden**
L N Nat ─ paths ─ est(R) ─ L Nat paths est(R)	
L N Nat ─ ⦇generate⦈ ─ setify ─ union ─ est(R) ─ L Nat = ⦇generate⦈ setify union est(R) (13.5a) at paths	
L N Nat ─ ⦇generate⦈ ─ setify ─ P(est(R)) ─ est(R) ─ L Nat ⊑ ⦇generate⦈ setify P(est(R)) est(R) (13.1b) at union, R a preorder	
L N Nat ─ ⦇generate⦈ ─ N(est(R)) ─ setify ─ est(R) ─ L Nat ⊑ ⦇generate⦈N(est(R)) setify est(R) setify lax natural	
L N Nat ─ ⦇Q⦈ ─ setify ─ est(R) ─ L Nat ⊑ ⦇Q⦈ setify est(R) (11.4.2a) at (13.5.2b)	

(13.5.2b)

generate N(est(R))∋F(𝟙,N(est(R)))Q

the fusion condition of (13.5.2a)'s last step

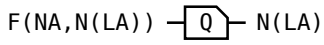
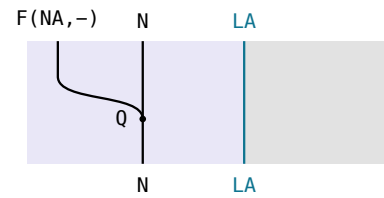
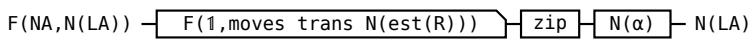
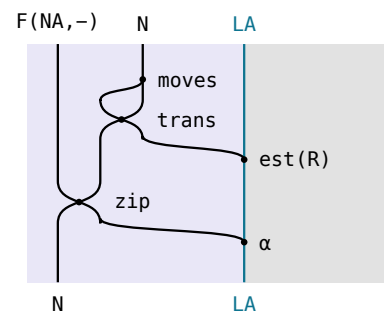
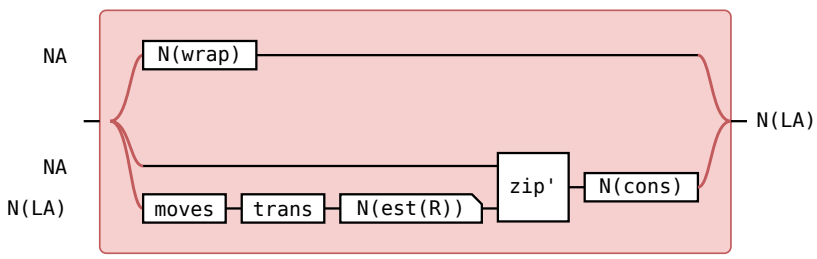
circuit	Hinze–Marsden
 generate N(est(R))	
⊒  F(𝟙,N(est(R)))Q (7.13), then zip, trans, moves lax natural	

(7.13) is F(𝟙,est(R))α⊑cp P(α) est(R), (13.3.1c) at the map α with $\frac{\mathtt{F(\mathbb{1},\ni)\alpha}}{\ni}$ =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.

(13.5.2c)

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

the algebra (13.5.2a)'s last step folds

circuit — the fork is F(NA,N(LA))=NA+NA×N(LA)	Hinze–Marsden
 Q	
=  F(𝟙,moves trans N(est(R))) zip N(α) the fusion condition read as a definition	
=  [N(wrap),(𝟙×moves trans N(est(R))) zip' N(cons)] zip=𝟙+zip', α=[wrap,cons]	

§13.6. The security van problem

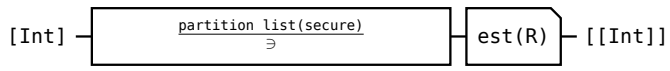
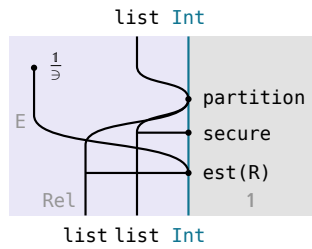
(13.6a)

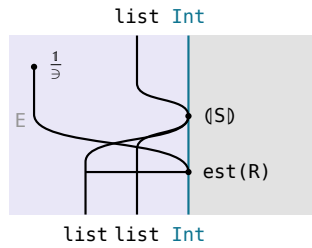
definition	type	note
R≜length≤length°	[[Int]]→[[Int]]	The order the schedule is minimised over: fewer van visits is better.
ceiling≜ $\frac{\text{prefix sum}}{\ni}$ est(≥)	[Int]→Int	The highest the bank's balance reaches over a stretch of transactions.
floor≜ $\frac{\text{prefix sum}}{\ni}$ est(≤)	[Int]→Int	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.

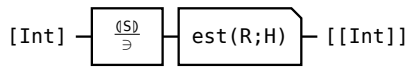
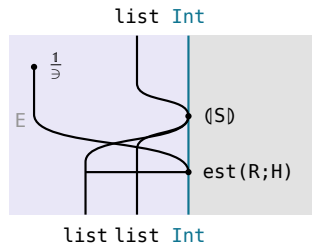
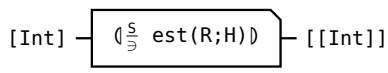
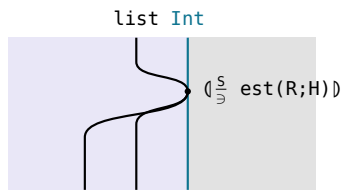
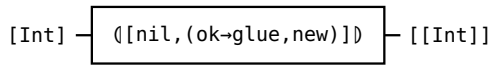
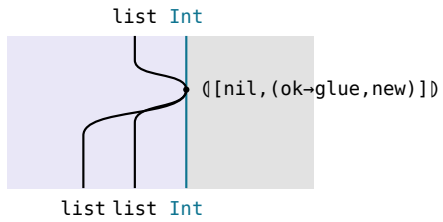
(13.6b)

$\frac{\texttt{partition list(secure)}}{\ni}$ `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 — (13.6a)	

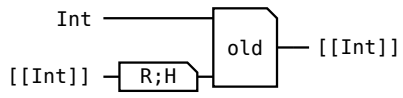
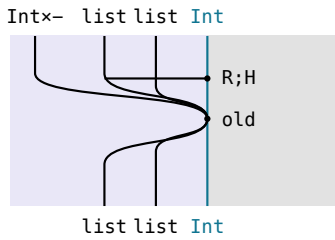
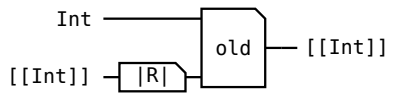
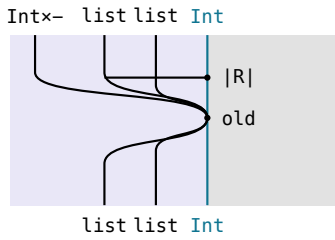
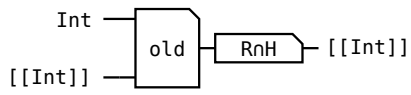
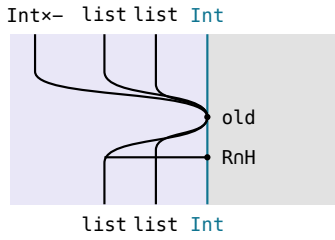
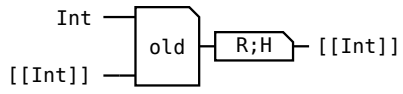
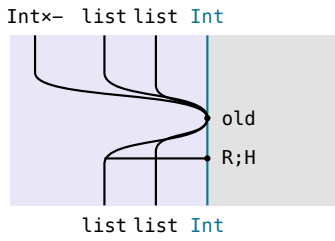
= $\frac{\texttt{⦇S⦈}}{\ni}$ `est(R)`
`partition list(secure)=⦇S⦈` — (13.6a), (11.4.2a) at `secure`
`prefix⊑prefix secure`
	

 ⊒ $\frac{\texttt{⦇S⦈}}{\ni}$ `est(R;H)` `R;H⊑R` — (13.6a); (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	
 ⊒ `⦇S/∋ est(R;H)⦈` (13.3.2a) 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⊑(newᴗold)(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**
 `(𝟙×(R;H))old`	
=  `(𝟙×|R|)oldᴗ(𝟙×(R∩H))old` `R;H=|R|ᴗ(R∩H)` — (13.6a), `ᴗ` distributes	
⊑  `new (R∩H)ᴗold (R∩H)` (7.19) and (7.20) on `|R|`, (7.21) on `R∩H`	
⊑  `(newᴗold)(R;H)` `X∩Y⊑X;Y`, converses	

§14. Thinning Algorithms

§14.1. Thinning

(14.1a)

For Q : A→A, thin Q≙(∋/∋)∩(∈\(Q°∈)) : EA→EA (8.1).

ys (thin Q) xs⟺xs⊆ys∧(∀a∈ys. ∃b∈xs. b Q a)

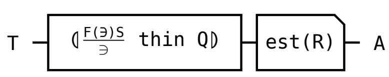
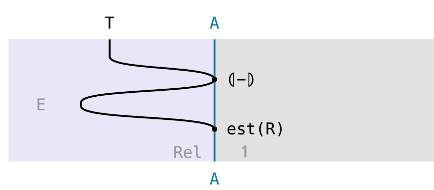
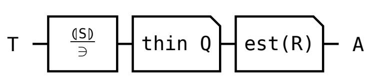
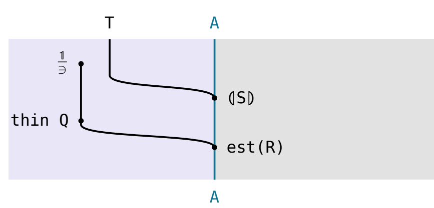
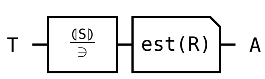
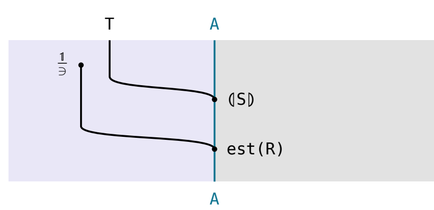
(14.1b)

the law	what it says
X⊑ $\frac{S}{\ni}$ thin Q⟺X∋⊑S and S°X⊑Q°∈	everything kept is an S-value, and every S-value has a Q-lower bound among the kept ones
Q⊑R⟹thin Q⊑thin R	the fewer pairs Q relates, the fewer subsets count as thinnings
𝟙⊑thin Q, and thin Q is a preorder if Q is	keeping everything is always a legal thinning
est(R)=thin Q est(R) Q⊑R, both preorders	**thin-introduction**: thinning first cannot lose an R-minimum
thin Q⊒est(Q) $\frac{𝟙}{\ni}$ (8.2)	**thin-elimination**: keeping one element is a thinning, but its domain is the sets est(Q) is defined on
$\frac{S}{\ni}$ thin Q⊒ $\frac{S}{\ni}$ est(R) $\frac{𝟙}{\ni}$ (8.3), R∩(S°S)⊑Q	the usable variant: R need only refine Q between values S gives one argument
union thin Q⊒P(thin Q) union (8.4)	thinning each member set is a thinning of the union
⦇$\frac{F(\ni)S}{\ni}$ thin Q⦈⊑ $\frac{⦇S⦈}{\ni}$ thin Q Theorem 8.1, S monotonic on Q	the **thinning theorem**: thinning at every step beats thinning only at the end

(14.1c)

⦇$\frac{F(\ni)S}{\ni}$ thin Q⦈ est(R)⊑$\frac{⦇S⦈}{\ni}$ est(R)

Corollary 8.1, S monotonic on Q, Q⊑R, both preorders

	circuit — one wire, T to A	**Hinze–Marsden**
	 ⦇$\frac{F(\ni)S}{\ni}$ thin Q⦈ est(R)	
⊑	 $\frac{⦇S⦈}{\ni}$ thin Q est(R) Theorem 8.1 — (14.1b)	
=	 $\frac{⦇S⦈}{\ni}$ est(R) 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]⦈π_2, ⦇[wrapz,consw]⦈=⟨𝟙,cost⟩, R≙cost≤cost°.

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

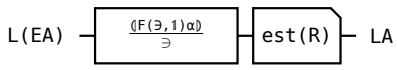
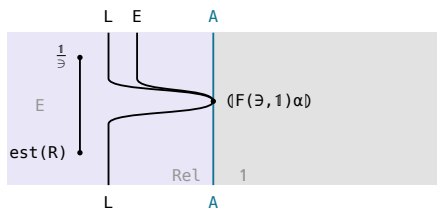
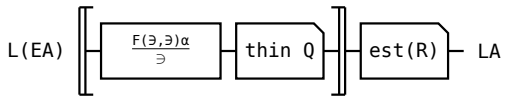
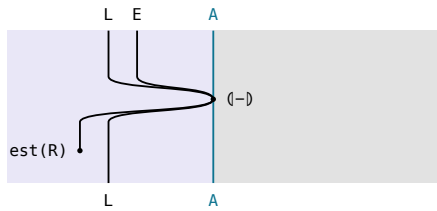
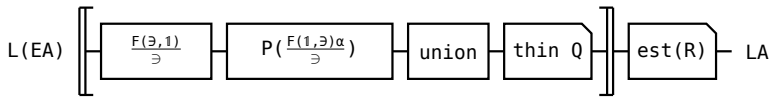
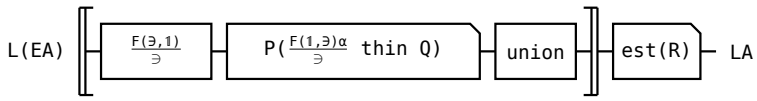
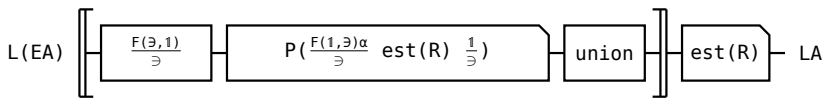
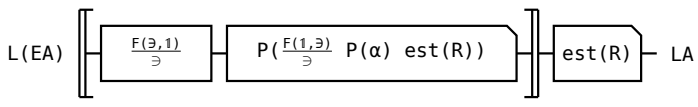
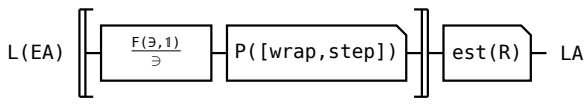
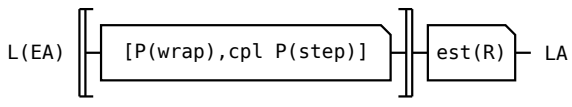
(14.2b)

the law	what it says
$F(\ni,Q)\alpha \sqsubseteq F(\ni,\mathbb{1})\alpha Q$	$F(\ni,\mathbb{1})\alpha$ is monotonic on Q; on R it is not, since the next edge can cost arbitrarily much
$S\ \mathrm{head} \sqsubseteq [\mathbb{1},\pi_1]$ $\quad S \triangleq F(\mathbb{1},\ni)\alpha$	$S\ \mathrm{head}$ is simple, which gives $R \sqcap (S \circ S) \sqsubseteq Q$: between two paths S builds from one argument, equal cost and equal head already means Q

(14.2c)

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

B&dM §8.2, p. 198

circuit — one wire, $L(EA)$ to LA; the algebra inside the functorial box	**Hinze–Marsden**
 $\frac{(\!\|F(\ni,\mathbb{1})\alpha\|\!)}{\ni}\ \mathrm{est}(R) = \frac{L(\ni)}{\ni}\ \mathrm{est}(R)$	
$\sqsupseteq$  $(\!\|\frac{F(\ni,\ni)\alpha}{\ni}\ \mathrm{thin}\ Q\|\!)\ \mathrm{est}(R)$ Corollary 8.1 — (14.1c), at $F(\ni,\mathbb{1})\alpha$ monotonic on Q — (14.2b)	
$=$  $(\!\|\frac{F(\ni,1)}{\ni}\ P(\frac{F(1,\ni)\alpha}{\ni})\ \mathrm{union}\ \mathrm{thin}\ Q\|\!)\ \mathrm{est}(R)$ $F(\ni,\ni)=F(\ni,\mathbb{1})F(\mathbb{1},\ni)$; $\frac{F(\ni,1)F(1,\ni)\alpha}{\ni}=\frac{F(\ni,1)}{\ni}$ $P(\frac{F(1,\ni)\alpha}{\ni})$ union	
$\sqsupseteq$  $(\!\|\frac{F(\ni,1)}{\ni}\ P(\frac{F(1,\ni)\alpha}{\ni}\ \mathrm{thin}\ Q)\ \mathrm{union}\|\!)\ \mathrm{est}(R)$ $\mathrm{union}\ \mathrm{thin}\ Q \sqsupseteq P(\mathrm{thin}\ Q)\ \mathrm{union}$ — (14.1b)	
$\sqsupseteq$  $(\!\|\frac{F(\ni,1)}{\ni}\ P(\frac{F(1,\ni)\alpha}{\ni}\ \mathrm{est}(R)\ \frac{1}{\ni})\ \mathrm{union}\|\!)\ \mathrm{est}(R)$ $\frac{S}{\ni}\ \mathrm{thin}\ Q \sqsupseteq \frac{S}{\ni}\ \mathrm{est}(R)\ \frac{1}{\ni}$ — (14.1b) at $S \triangleq F(\mathbb{1},\ni)\alpha$, $R \sqcap (S \circ S) \sqsubseteq Q$ — (14.2b)	
$=$  $(\!\|\frac{F(\ni,1)}{\ni}\ P(\frac{F(1,\ni)}{\ni}\ P(\alpha)\ \mathrm{est}(R))\|\!)\ \mathrm{est}(R)$ $P(\frac{1}{\ni})\ \mathrm{union}=\mathbb{1}$; α a map, so $\frac{F(1,\ni)\alpha}{\ni}=\frac{F(1,\ni)}{\ni}\ P(\alpha)$	
$=$  $(\!\|\frac{F(\ni,1)}{\ni}\ P([\mathrm{wrap},\mathrm{step}])\|\!)\ \mathrm{est}(R)$ $\frac{F(1,\ni)}{\ni}\ P(\alpha)\ \mathrm{est}(R)=[\mathrm{wrap},\mathrm{step}]$ — (14.2a)	
$=$  $(\!\|[P(\mathrm{wrap}),\mathrm{cpl}\ P(\mathrm{step})]\|\!)\ \mathrm{est}(R)$ $\frac{F(\ni,1)}{\ni}=\mathbb{1}+\mathrm{cpl}$ — (14.2a)	

§14.3. Implementing thin

(14.3a)

$\mathtt{setify} : [A]\to EA$, $\mathtt{cup} : EA\times EA\to EA$, $\mathtt{cp}(F)\triangleq \frac{F(\ni)}{\ni}$, $\mathtt{listcp}(F) : F([A])\to[FA]$, $\mathtt{sort}\ P\triangleq \mathtt{setify}^\circ\ \mathtt{ordered}\ P$ for P a connected preorder.

$\mathtt{thinlist}\ Q$ is any $\mathtt{thinlist}\ Q\sqsubseteq\mathtt{subseq}$ with $\mathtt{thinlist}\ Q\ \mathtt{setify}\sqsubseteq\mathtt{setify}\ \mathtt{thin}\ Q$; one is $(\!|[\mathtt{nil},\mathtt{bump}\ Q]|\!)$, $\mathtt{bump}\ Q\ (a,[])=[a]$, $\mathtt{bump}\ Q\ (a,[b]+\!\!+xs)=(b\ Q\ a\to[a]+\!\!+xs, a\ Q\ b\to[b]+\!\!+xs,[a]+\!\!+[b]+\!\!+xs)$.

Binary thinning data: $S=(f_1p_1)\cup(f_2p_2)$ with p_1, p_2 coreflexive; Q a preorder with $Q\sqsubseteq R$ and both f_1p_1, f_2p_2 monotonic on Q; P a connected preorder with both f_1, f_2 monotonic on P; $g_i\triangleq\mathtt{list}(f_i)\ \mathtt{filter}(p_i)$.

(14.3b)

the law	what it says
$\mathtt{thinlist}\ Q\ xs=[\mathtt{minlist}\ Q\ xs]$ (8.5), Q connected, xs non-empty	what thinning should come to when it can: one element
$\mathtt{sort}\ P\ \mathtt{thinlist}\ Q\sqsubseteq\mathtt{thin}\ Q\ \mathtt{sort}\ P$ (8.6)	thinning a sorted list is a thinning of the set — this is what $\mathtt{thinlist}\ Q\sqsubseteq\mathtt{subseq}$ buys
$\mathtt{sort}\ P\ \mathtt{minlist}\ Q\sqsubseteq\mathtt{est}(Q)$ (8.7)	a minimum of the sorted list is a minimum of the set
$\mathtt{sort}(fPf^\circ)\ \mathtt{list}(f)\sqsubseteq P(f)\ \mathtt{sort}\ P$ (8.8)	shunt a function through a sort
$\mathtt{sort}\ P\ \mathtt{filter}(p)\sqsubseteq E(p)\ \mathtt{sort}\ P$ (8.9), p coreflexive	filtering a sorted list sorts the restricted set
$(\mathtt{sort}\ P\times\mathtt{sort}\ P)\ \mathtt{merge}\ P\sqsubseteq\mathtt{cup}\ \mathtt{sort}\ P$ (8.10)	merging two sorted lists sorts their union
$F(\mathtt{sort}\ P)\ \mathtt{listcp}(F)\sqsubseteq\mathtt{cp}(F)\ \mathtt{sort}(FP)$ (8.11), F linear	$\mathtt{listcp}(F)$ is the list implementation of the cartesian product $\mathtt{cp}(F)$
$F(\mathtt{sort}\ P)\ \mathtt{listcp}(F)\ \mathtt{list}(f)\ \mathtt{filter}(p)\sqsubseteq \frac{F(\ni)fp}{\ni}\ \mathtt{sort}\ P$ Lemma 8.1, f monotonic on P, p coreflexive	one sorted list built from sorted arguments, instead of a set built and then sorted
$(\!|\mathtt{listcp}(F)\ \langle g_1,g_2\rangle\ \mathtt{merge}\ P\ \mathtt{thinlist}\ Q|\!)\ \mathtt{minlist}\ R \sqsubseteq \frac{(\!|S|\!)}{\ni}\ \mathtt{est}(R)$ Theorem 8.2	the **binary thinning theorem**: a fold on sorted lists of partial solutions, thinned at every step

§14.4. The knapsack problem

(14.4a)

$\mathtt{vol},\mathtt{wt} : \mathtt{Item}\to\mathtt{Real}$, $\mathtt{value}\triangleq\mathtt{list}(\mathtt{vol})\ \mathtt{sum}$, $\mathtt{weight}\triangleq\mathtt{list}(\mathtt{wt})\ \mathtt{sum}$.

$\mathtt{subseq}=(\!|[\mathtt{nil},\mathtt{cons}]\cup[\mathtt{nil},\pi_2]|\!)$, $\mathtt{within}\ w$ the coreflexive on xs with $\mathtt{weight}\ xs\le w$.

$R\triangleq\mathtt{value}\ge\mathtt{value}^\circ$, $Q\triangleq R\cap(\mathtt{weight}\le\mathtt{weight}^\circ)$, $P\triangleq R$.

$FA=1+\mathtt{Item}\times A$, $\mathtt{listcp}(F)=\mathtt{wrap}+\mathtt{cpr}$, $g_1\triangleq\mathtt{list}([\mathtt{nil},\mathtt{cons}])\ \mathtt{filter}(\mathtt{within}\ w)=[\mathtt{list}(\mathtt{nil}),h_1]$, $g_2\triangleq\mathtt{list}([\mathtt{nil},\pi_2])=[\mathtt{list}(\mathtt{nil}),h_2]$.

$h_1\triangleq\mathtt{list}(\mathtt{cons})\ \mathtt{filter}(\mathtt{within}\ w)$, $h_2\triangleq\mathtt{list}(\pi_2)$.

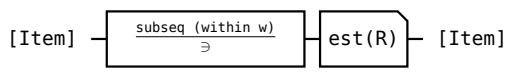
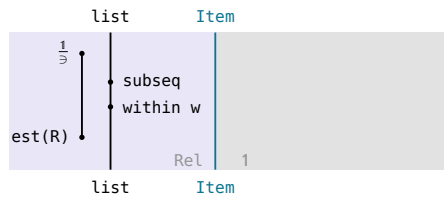
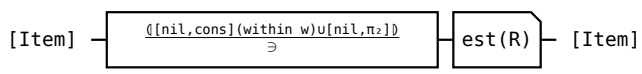
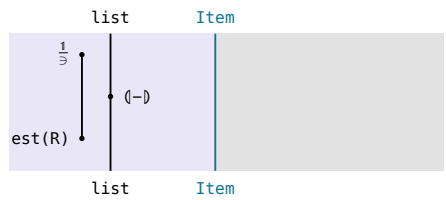
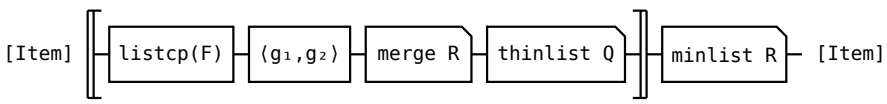
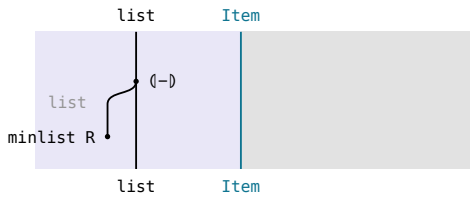
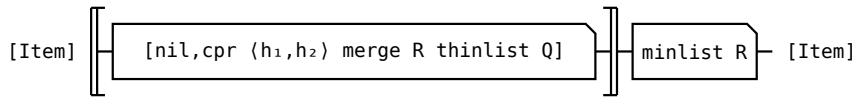
(14.4b)

the law	what it says
$(\mathbb{1}\times R)\ (\mathtt{cons}\ (\mathtt{within}\ w))\sqsubseteq\mathtt{cons}\ (\mathtt{within}\ w)R$ FALSE	not monotonic on R: a selection of greater value need not still fit once one more item goes in
$(\mathbb{1}\times Q)\ (\mathtt{cons}\ (\mathtt{within}\ w))\sqsubseteq\mathtt{cons}\ (\mathtt{within}\ w)Q$ $(\mathbb{1}\times Q)\pi_2\sqsubseteq\pi_2Q$	both halves are monotonic on Q once ties in value are broken by weight

(14.4c)

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

B&dM §8.4, p. 206

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)` (11.4.2a), weights non-negative	
⊑  `⦇listcp(F) ⟨g₁,g₂⟩ merge R thinlist Q⦈ minlist R` Theorem 8.2 — (14.3b), 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]` — (14.4a); `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,newuglue]⦈ : list⁺ Word→Para.
  width≜⦇[length,(length×𝟙) plus succ]⦈, 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≜Rn(head head°), P≜⊤.
  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

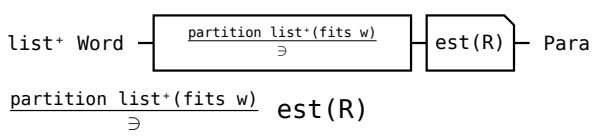
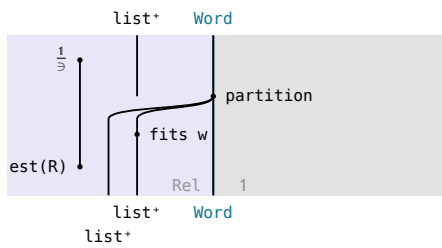
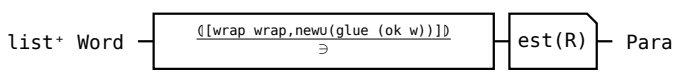
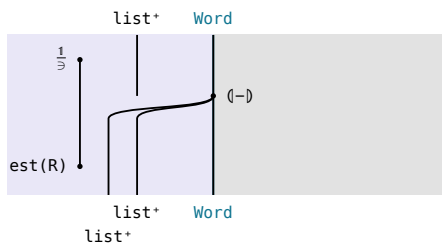
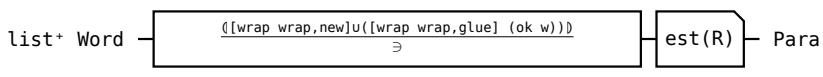
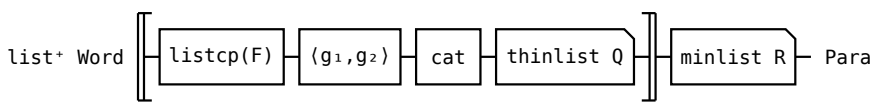
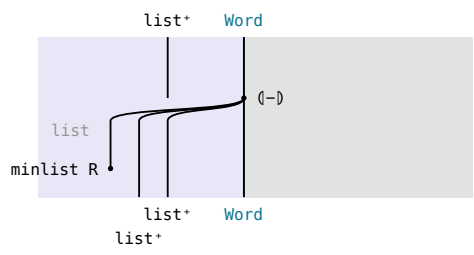
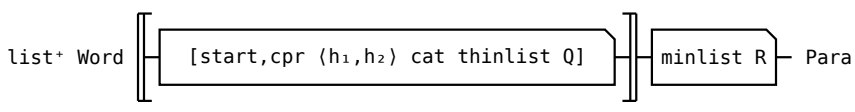
`(𝟙×Q) new⊑new Q`
`(𝟙×Q) (glue (ok w))⊑glue (ok w)Q`
`cons` monotonic on `collect≤collect°`
	both halves are monotonic on `Q` once ties in waste are broken by the first line

`merge ⊤=cat`; `P≜head prefix head°` also serves	`⊤` needs no sorting at all, and `prefix` is a linear order on first lines of paragraphs of one input

(14.5c)

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

B&dM §8.5, p. 210

circuit — one wire, `list⁺ Word` to `Para`; the algebra inside the functorial box	**Hinze–Marsden**
 $\frac{\texttt{partition list}^+\texttt{(fits w)}}{\ni}$ `est(R)`	
=  $\frac{\texttt{⦇[wrap wrap,newu(glue (ok w))]⦈}}{\ni}$ `est(R)` (11.4.2a), every word fits on a line by itself	
=  $\frac{\texttt{⦇[wrap wrap,new]u([wrap wrap,glue] (ok w))⦈}}{\ni}$ `est(R)` the algebra as `(f₁p₁)∪(f₂p₂)`, `p₁≜𝟙` — (14.3a)	
⊑  `⦇listcp(F) ⟨g₁,g₂⟩ cat thinlist Q⦈ minlist R` Theorem 8.2 — (14.3b), at `P≜⊤` with `merge ⊤=cat`, `Q` from (14.5b)	
=  `⦇[start,cpr ⟨h₁,h₂⟩ cat thinlist Q]⦈ minlist R` `listcp(F)=wrap+cpr`, `gᵢ` along the coproduct — (14.5a)	

§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,dropludropr]⦈.

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

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

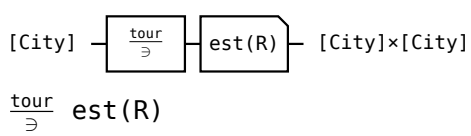
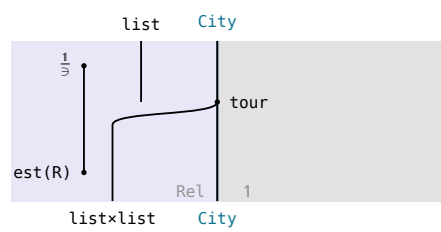
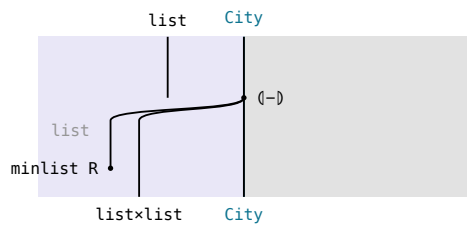
(14.6b)

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

B&dM §8.6, p. 215

circuit — one wire, [City] to [City]×[City]; the algebra inside the functorial box	**Hinze–Marsden**
 $\overset{\text{tour}}{\ni}$ est(R)	
[City] ⦇[start,dropludropr]⦈ ∋ est(R) [City]×[City] = ⦇[start,dropludropr]⦈ ∋ est(R) tour≜⦇[start,dropludropr]⦈ — (14.6a)	
[City] [listcp(F) ⟨g_1,g_2⟩ cat thinlist Q] minlist R [City]×[City] ⊒ ⦇listcp(F) ⟨g_1,g_2⟩ cat thinlist Q⦈ minlist R Theorem 8.2 — (14.3b), at P≜⊤ with merge ⊤=cat, Q from (14.6b)	
[City] [[start wrap,cpr ⟨list(dropl),list(dropr)⟩ cat thinlist Q]] minlist R [City]×[City] = ⦇[start wrap,cpr ⟨list(dropl),list(dropr)⟩ cat thinlist Q]⦈ minlist R listcp(F)=wrap+cpr, g_i=[list(start),list(drop$_i$)] — (14.6a); 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 (11.6.3a).

(15.1b)

the law	what it says
(μX : $\frac{T°}{\ni}$ P(F(X)h) est(R))⊑M Theorem 9.1, h monotonic on R	**dynamic programming**: decompose in every way, solve each part, keep one optimum per part
$\frac{T°}{\ni}$ P(F(M)h) est(R)⊑M (9.1)	all Knaster–Tarski leaves to prove
$\frac{T°}{\ni}$ P(F(M)h) est(R)⊑H (9.2) H° $\frac{T°}{\ni}$ P(F(M)h) est(R)⊑R° (9.3)	(9.1) split by M=H∩(H°\R°)
P(X) est(R)⊑(∃X)∩(∈\(XR°)) (9.4) = (7.10), (13.1.8a)	the only fact about est either proof uses
(μX : $\frac{T°}{\ni}$ thin Q P(F(X)h) est(R))⊑M Theorem 9.2, Q a preorder with QF(H)h⊑F(H)hR; thin Q as in (14.1b)	the same with a thinning step: decompositions that can never win are dropped
the fixed point is unique, and entire Theorem 6.3, T° followed by F's membership relation inductive; $\frac{T°}{\ni}$ finite and non-empty, R connected	when the recursion can be refined to a recursive function
$\frac{[V_1,V_2]°}{\ni}$ thin(Q_1+Q_2) P([U_1,U_2]) est(R) =(Ran V_1→W_1,W_2) W_i≜ $\frac{V_i°}{\ni}$ thin(Q_i) P(U_i) est(R) Proposition 9.1, V_2V_1°=⊥	FA is usually a coproduct, and disjoint ranges split the fixed point into one branch per summand
F(R)h⊑hR QF(H)h⊑F(H)hR	the two conditions to be checked, in the order the three propositions below discharge them
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
Q≜F(U,V) Proposition 9.4, U, V preorders, F(U,R)h⊑hR, VH⊑HR	both conditions of Theorem 9.2 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,π_2 succ]⦈, R≜length≤length°, U≜⊤, V≜suffix°×suffix°, Q≜𝟙+(U×V), empty the coreflexive on (xs,ys) with both lists empty.

unstep implements $\frac{step°}{\ni}$ 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)

the law	what it says
$\frac{\texttt{edit}^\circ}{\ni}$ est(R)	the specification: a shortest edit sequence from which both strings can be reconstituted
F(R)α⊑αR Proposition 9.2 at length, succ monotonic on ≤	cons is monotonic on R, so Theorem 9.1 already applies
QF(𝟙,edit°)α⊑F(𝟙,edit°)αR	the thinning condition sought, over F(Op,[Char]×[Char])
F(⊤,R)α⊑αR left as an exercise	U≜⊤ costs nothing: any two operations may be compared
edit (𝟙×suffix)⊑R° edit edit (suffix×𝟙)⊑R° edit	the other half of Proposition 9.4 for V≜suffix°×suffix°
edit (𝟙×tail)⊑R° edit edit (tail×𝟙)⊑R° edit suffix=tail*, and BA⊑CB⟹BA*⊑C*B	one step is enough: drop the operation that produced the head, or weaken its cpy to a del — never lengthening the sequence
(μX : $\frac{\texttt{[base,step]}^\circ}{\ni}$ thin Q P([nil,(𝟙×X) cons]) est(R))⊑ $\frac{\texttt{edit}^\circ}{\ni}$ est(R) Theorem 9.2	a copy, when available, beats a delete or an insert
X=(empty→nil, $\frac{\texttt{step}^\circ}{\ni}$ thin(U×V) P((𝟙×X) cons) est(R)) Proposition 9.1	base and step have disjoint ranges
mle=(empty→nil,unstep list((𝟙×mle) cons) minlist R)	the program: at most two decompositions survive, but the same subproblem is solved many times, so the running time is exponential
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))
	each column is a fold built bottom to top, over xs zipped with the adjacent pairs of the column to its right

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 $\frac{\texttt{cat}^\circ}{\ni}$; 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)

the law	what it says
$\frac{\texttt{flatten}^\circ}{\ni}$ est(R)	the specification: a least-cost bracketing of $a_1 \oplus \cdots \oplus a_n$, the tree whose flattening is the given list
[tip,bin] cost=(𝟙+⟨cost,flatten⟩²) g (9.5)	the cost of a node reads only the cost and the flattening of its two subtrees
(𝟙+(≤×𝟙)²) g⊑g≤ (9.6)	g is monotonic on ≤ in its two cost arguments
F(R∩(flatten flatten°))h⊑hR Proposition 9.3, H°=flatten a map	monotonicity in context: only trees with the same flattening are compared
(μX : $\frac{\texttt{[wrap,cat]}^\circ}{\ni}$ P([tip,(X×X) bin]) est(R))⊑ $\frac{\texttt{flatten}^\circ}{\ni}$ est(R) Theorem 9.1	split the list in every way, bracket both halves, join
X=(single→wrap° tip, $\frac{\texttt{cat}^\circ}{\ni}$ P((X×X) bin) est(R)) Proposition 9.1	wrap and cat have disjoint ranges
mct=(single→head tip,splits list((mct×mct) bin) minlist R)	the program; 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
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
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)

the law	what it says
$\frac{\texttt{decode°}}{\ni}$ `est(R)`	the specification: a smallest code sequence decoding to the given string
`F(R)α⊑αR` `(T+T)[c,p]=[c,p]`	monotonicity is routine, the two costs being constants

`F(T+T,R)α⊑αR`
`decode prefix⊑R° decode`
Proposition 9.4 at `Q≜F(T+T,prefix°)`
	the thinning condition split in two: pointers are ordered only by how much input they consume, symbols and pointers not at all

`decode init⊑R° decode`	enough for the second: drop the last character of the output by shortening or removing the last code element, never raising the cost
`(μX :` $\frac{\texttt{[nil,extend]°}}{\ni}$ `thin Q P([nil,(X×𝟙) snoc]) est(R))⊑` $\frac{\texttt{decode°}}{\ni}$ `est(R)` Theorem 9.2	between a symbol and a pointer nothing can be decided in advance; between two pointers the longer match wins
`X=(null→nil,` $\frac{\texttt{extend°}}{\ni}$ `thin(prefix°×(T+T)) P((X×𝟙) snoc) est(R))` Proposition 9.1	`nil` and `extend` have disjoint ranges
$\frac{\texttt{extend°}}{\ni}$ `(ws♯[a])={(ws,sym a)}∪` `{(xs,ptr (ys,zs))|xs♯zs=ws♯[a]`, `ys♯zs` a proper prefix of `ws}`	the decompositions of one string: 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 `lrt`
`encode=(null→nil,reduce list((encode×𝟙) snoc) minlist R)`	the program, again exponential; the book gives no tabulation for it

§16. Greedy Algorithms

§16.1. Theory

(16.1a)

h, T, R, H, M as in (15.1a); additionally Q a **connected** preorder on the sets $\frac{T^\circ}{\ni}$ returns, so that $\frac{T^\circ}{\ni}$ est(Q) is entire.

(16.1b)

the law	what it says
$(\mu X : \frac{T^\circ}{\ni}\ \mathrm{est}(Q)\ F(X)h) \sqsubseteq M$ Theorem 10.1, the hypotheses of Theorem 9.2	**greedy**: one decomposition is kept at every step, so P and $\frac{}{\ni}$ disappear from the recursion
$\frac{[V_1,V_2]^\circ}{\ni}\ \mathrm{est}(Q_1+Q_2)\ [U_1,U_2]$ $=(\mathrm{Ran}\ V_1 \to W_1, W_2)$ $W_i \triangleq \frac{V_i^\circ}{\ni}\ \mathrm{est}(Q_i)\ U_i$ Proposition 10.1, $V_2 V_1{}^\circ = \bot$	Proposition 9.1 with est for thin
$Q \triangleq F(U,V)$ Proposition 9.4	still the way to get both conditions, but $\frac{T^\circ}{\ni}$ est(Q) must now be entire as well
Theorem 7.2	that greedy theorem chooses among the **results** of one relational step of a reduce; Theorem 10.1 chooses among the **decompositions** of the input, for an arbitrary T rather than an initial algebra, at the cost of h being a map

§16.2. The detab-entab problem

(16.2a)

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

col�triangleq⦇[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�triangleq length≤length°, U the preorder with a U b⟺a=TB∨a=b, V�triangleq prefix°∩(fill fill°), Q�triangleq𝟙+(V×U).

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

(16.2b)

the law	what it says
$\frac{\texttt{detab}^\circ}{\ni}$ est(R)	the specification: a shortest input detab expands to the given output — detab entab=𝟙 and nothing shorter does
F(⊤,R)α⊑αR left as an exercise	U may be any preorder on characters; a U b⟺a=TBva=b puts TB below every character, so est prefers a tab to a blank
detab prefix⊑R° detab 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
detab V°⊑R° detab V≜prefix°∩(fill fill°)	true once only the prefixes that do not cross a tab stop are allowed
nil V°=nil fill V°=fill snoc V°⊑snoc∪(π₁V°) left as exercises	the three properties of V the derivation rests on; the second is what fill fill° was added for
expand V°⊑expand∪(π₁V°)	the claim they add up to: shortening the output either leaves the last step alone or discards it
detab V°⊑detab∪(init detab V°) init inductive, so the greatest solution is the unique one	hence detab V°⊑prefix detab, and prefix⊑R° finishes Proposition 9.4
(μX : $\frac{\texttt{[nil,expand]}^\circ}{\ni}$ est(Q) (𝟙+(X×𝟙)) [nil,snoc])⊑ $\frac{\texttt{detab}^\circ}{\ni}$ est(R) Theorem 10.1	greedy: one character of input is decided at each step
X=(null→nil, $\frac{\texttt{expand}^\circ}{\ni}$ est(V×U) (X×𝟙) snoc) Proposition 10.1	nil and expand have disjoint ranges
$\frac{\texttt{expand}^\circ}{\ni}$ (xs♯[a])={(ys,TB)|fill ys=xs♯[a]}∪{(xs,a)} the first set is non-empty iff a=BL and col (xs♯[a]) mod n=0	the last output character came from a tab only if it is a blank landing exactly on a tab stop
$\frac{\texttt{expand}^\circ}{\ni}$ est(V×U) (xs♯[a])= (a=BL∧col (xs♯[a]) mod n=0→(unfill xs,TB),(xs,a))	the greedy step: 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 a snoc-list reduce: the output splits at the last tab stop
triple=⦇[base,op]⦈ and entab=triple assocl π₁ (𝟙×blanks) cat	the program: one pass carrying the column and the count of pending blanks
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)))	hold a blank back, cash the held blanks in for a tab at a tab stop, flush them at a newline, flush them before anything else

§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°.

(16.3b)

the law	what it says
$\frac{\texttt{bagify°}}{\ni}$ est(R)	the specification: an ordering of the given bag of jobs with least maximum penalty
F(R∩(bagify bagify°))α⊑αR (10.2)	monotonicity in context — only schedules of one bag are compared
(Q∩(ββ°))F(bagify°)α⊑F(bagify°)αR (10.3)	the greedy condition, also in context
cost (xs#[j])=bmax (cost xs,penalty (perm xs,j)) α cost=F(⟨cost,bagify⟩)k, F(≤×𝟙)k⊑k≤ Proposition 9.3	(10.2): penalty sums the completion times of xs, and a sum does not see the order
α cost=⟨g,h⟩ bmax (10.4) g≜[zero,penalty] (10.5) h≜[zero,π_1 cost] (10.6)	the cost of a schedule split into the penalty of its last job and the cost of the rest
add⊑π_1R (10.7) β bagify°=F(bagify°) [nil,add] (10.8)	adding a job never lowers the cost; and bagify° is itself a reduce on bags. Together: β bagify°⊑F(bagify°)h≤cost°
F(bagify)Q°F(bagify°)=g≥g°, met by Q≜f≤f°, f≜[zero,(bagify°×𝟙) penalty]	the specification Q has to satisfy, and the choice that meets it: a Q-minimum is a job of least penalty
no greedy **reduce** exists	a greedy snoc-list reduce 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
X=(null→nil, $\frac{\texttt{snag°}}{\ni}$ est(Q) (X×𝟙) snoc) Theorem 10.1, Proposition 10.1	nil and snag have disjoint ranges
schedule=(null→nil,pick (schedule×𝟙) snoc) pick⊑ $\frac{\texttt{snag°}}{\ni}$ est(Q), a partial function	the program: repeatedly remove a job of least penalty and put it last; quadratic in the number of jobs

§16.4. The TeX problem

(16.4a)

intern≜val round : Decimal→[0,2^{16}), val≜⦇[zero,shift]⦈, shift (d,r)=(d+r)/10, round r rounds 2^{16}r to the nearest integer: round r=n⟺2n−1<2^{17}r<2n+1.

interval n=((2n−1)/2^{17},(2n+1)/2^{17}), 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]⦈=H°, ! : Digit×Interval→1, Q≜(l°!°r)∪𝟙, w≜2^{17}.

(16.4b)

the law	what it says
$\frac{\texttt{intern°}}{\ni}$ `est(R)=interval` $\frac{\texttt{inrange val°}}{\ni}$ `est(R)`	the specification: a shortest decimal whose internal representation is the given multiple of 2^{-16}. `round°` is not a map, but `interval` is, so it comes out of the $\frac{}{\ni}$
`val inrange°=⦇[arb,step]⦈` fusion; `zero inrange°=arb`, `shift inrange°=(𝟙×inrange°) step`	the converse of `val`, cut down to intervals, is a reduce on cons-lists
`F(R)α⊑αR`	`cons` is monotonic on `R` — routine
`α°F(h)Q°⊑R°α°F(h)`	the greedy condition of Theorem 10.1, converted
$\frac{\texttt{arb°}}{\ni}$ `(a,b)=(a<0→{*},{})` $\frac{\texttt{step°}}{\ni}$ `(a,b)={(d,(10a−d,10b−d))|0<10b−d<1}`	the decompositions of an interval: stop, or take one more digit
`0<10b−d₁<1` and `0<10b−d₂<1` imply `d₁=d₂`	`step°` is in fact a map, `d` the digit with `0<10b−d<1`, so $\frac{\texttt{[arb,step]°}}{\ni}$ returns at most two elements and `Q` need only choose between them
`Q≜(l°!°r)∪𝟙`, and `! nil⊑cons R°` from `! nil length⊑cons length≥`	stop whenever stopping is legal: the empty decimal is shorter than any other
`(μX :` $\frac{\texttt{[arb,step]°}}{\ni}$ `est(Q) F(X)α)⊑` $\frac{\texttt{intern°}}{\ni}$ `est(R)` Theorem 10.1	greedy: emit the one digit the interval allows, until the interval contains zero
`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`, `w=2¹⁷`
	the same in integer arithmetic only, as chapter 3 required of `intern`: every interval reached is `(p/w,q/w)`

§17. Appendix

§17.1. P(S) est(R)=(∋S)∩(∈\(SR°))

(17.1a)

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

(17.1b)

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

(17.1c)

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

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

(17.1d)

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

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

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

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

(17.2a)

$$\mathtt{P(est(R))\ est(R)} \xLeftrightarrow[\mathtt{Dom(est(R))\ est(R)=est(R)}]{} \mathtt{P(Dom(est(R))\ est(R))\ est(R)}$$

$$\xLeftrightarrow[\mathtt{P}\ \text{a relator}]{} \mathtt{P(Dom(est(R)))P(est(R))\ est(R)} \xLeftarrow[(13.1.9a)]{\sqsubseteq} \mathtt{P(Dom(est(R)))\ union\ est(R)}$$

(17.2b)

$$\begin{array}{c}\mathtt{P(Dom(est(R)))\ union\ est(R)}\\ \hline \mathtt{\ni est(R)}\end{array} \quad \begin{array}{c}\mathtt{\in P(Dom(est(R)))\ union\ est(R)}\\ \hline \mathtt{est(R)R°}\end{array} \xLeftarrow[\text{(17.1b) at } \mathtt{S:=est(R)},\ \mathtt{\Delta\dashv\cap},\ \mathtt{T\cdot\dashv T\backslash}]{} \begin{array}{c}\mathtt{P(Dom(est(R)))\ union\ est(R)}\\ \hline \mathtt{P(est(R))\ est(R)}\end{array}$$

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

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

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

$\Longleftarrow$ Dom(est(R))⊑est(R) est(R)°

est(R) est(R)°∈union est(R)
est(R)R°

$\Longleftarrow$ est(R)°⊑∈, ∈∈union⊑∈

est(R)∈est(R)
est(R)R°

UP of est