Mathematical Foundations of Bidirectional Transformations

Michael Johnson

CoACT (Center of Australian Category Theory)
Macquarie University Sydney

BXSS, Oxford, July 25-29, 2016

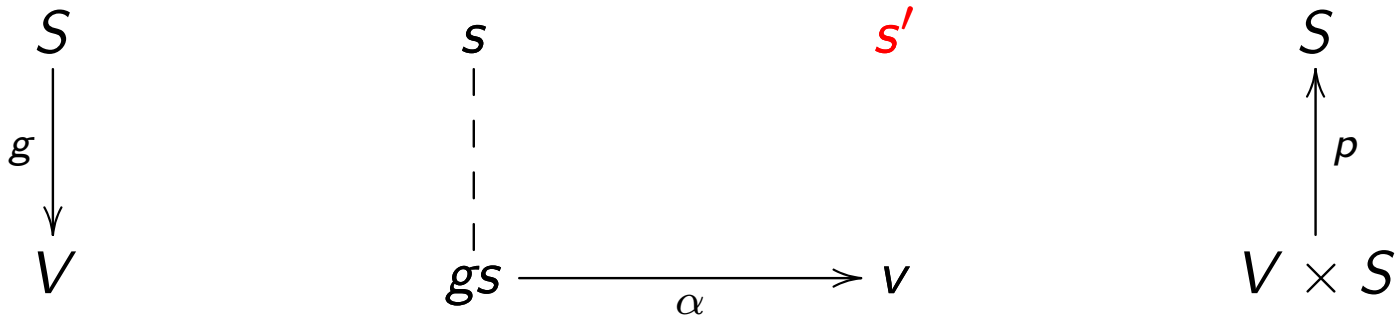
Did I say I would "spiral"?

If you lost it on Wednesday it's ok — I'm going to start again, again

Are state spaces really just sets?

Recall

Lens:



Satisfying axioms PutGet, GetPut and PutPut

Such things are exactly the algebras for the monad $\Delta\Sigma$ on Set/V

(Notice that the arrow $\alpha : gs \longrightarrow v$ is not used.)

What should be the domain of p?

A put depends on an old system state s and a new view state v.
So, pairs (s, v)? No — eg v unreachable from gs .

What should be the domain of p?

A put depends on an old system state s and a new view state v.
So, pairs (s, v)? No — eg v unreachable from gs .

A new view state v comes from (but does not in general determine) the orginal view state $g(s)$ and the state transition chosen.

What should be the domain of p?

A put depends on an old system state s and a new view state v.
So, pairs (s, v)? No — eg v unreachable from gs .

A new view state v comes from (but does not in general determine) the orginal view state $g(s)$ and the state transition chosen.

So we need $(s, v, \alpha : g(s) \longrightarrow v)$.

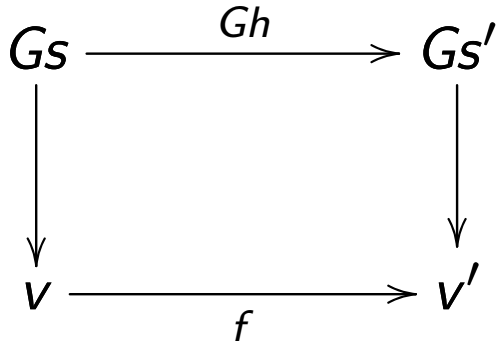
We need a domain for p which has as objects, triples of the form

$$(s, v, \alpha : g(s) \longrightarrow v).$$

Comma categories and generalised slice categories

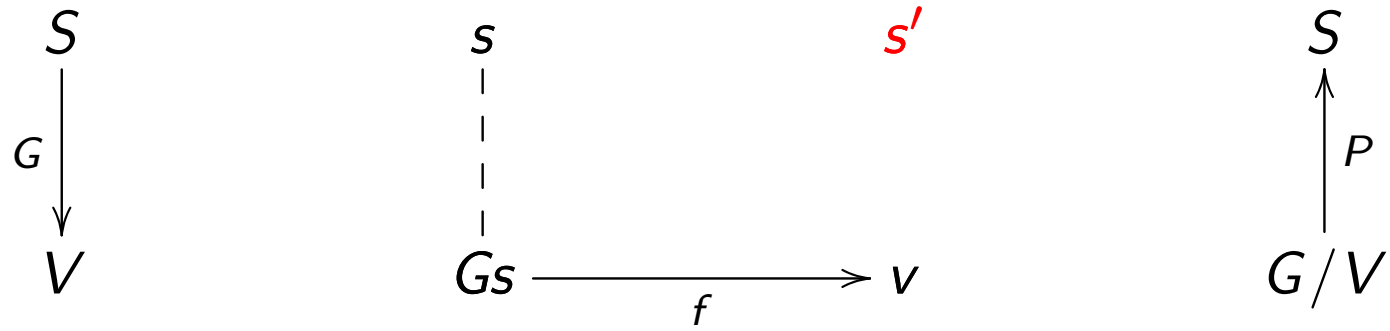
Suppose now G is a functor $S \longrightarrow V$ (not just a function), so G and 1_V are functors with codomain V.

- The comma category $(G, 1_V)$ is sometimes denoted G/V analogously to a slice category
- Objects are triples $(s, v, Gs \longrightarrow v)$
  s-indexed arrows, but we'll sometimes just write $(Gs \longrightarrow v)$
- Arrows are pairs (h, f) making commutative



(G, H)

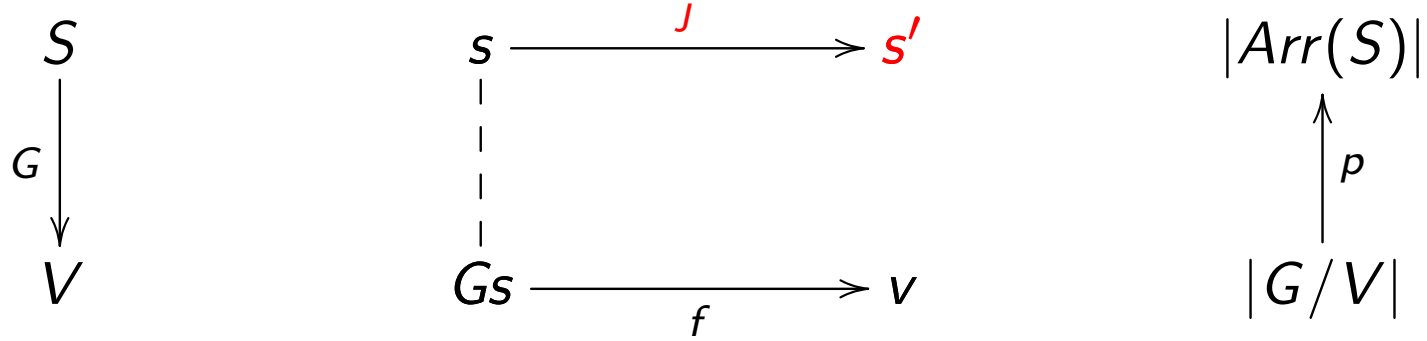
Two types of delta-lenses: 1. c-lens



Satisfing axioms PutGet, GetPut and PutPut.
Such things are exactly the algebras for the monad R on $\mathbf{cat}/V$ which takes an object $G : S \longrightarrow V$ to the object $\pi : G/V \longrightarrow V$.

(Cf the monad (from Monday) on $\mathbf{set}/V$ which took $G : S \longrightarrow V$ to the object $\pi : V \times S \longrightarrow V$.)

Two types of delta-lenses: 2. d-lens



Satisfing axioms PutGet, GetPut and PutPut
Such things turn out to be algebras for a semi-monad on Cat/V with an extra condition.

Lifting arrows in G/V

- The d-lens Put provides extra information — an arrow j, not just its codomain s'
- But what does it do with the arrows of G/V, and what about functoriality?

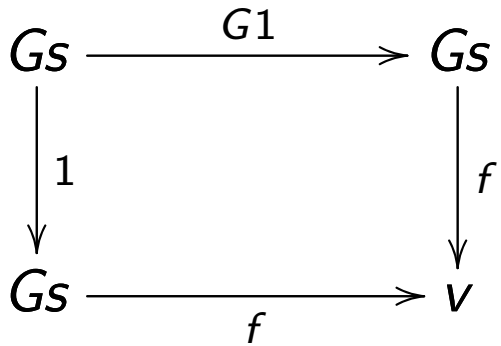
(In fact, a d-lens Put for *some* arrows of G/V, and the corresponding functoriality, follow from d-lens PutPut.)

The extra arrow is important. Do c-lenses really miss out on it?

Full functoriality of P is good, but so is having arrows like j.

Calculus of squares

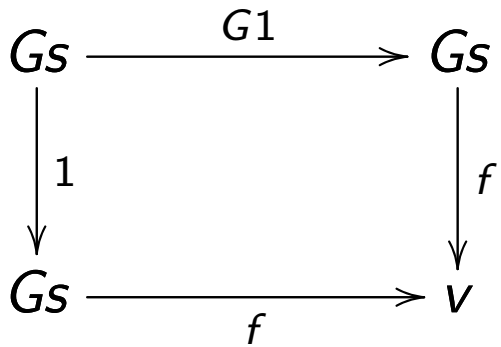
Suppose $f : Gs \longrightarrow v$ is an object of G/V, then $(1, f)$



is an arrow of G/V.

Calculus of squares

Suppose $f : Gs \longrightarrow v$ is an object of G/V, then $(1, f)$

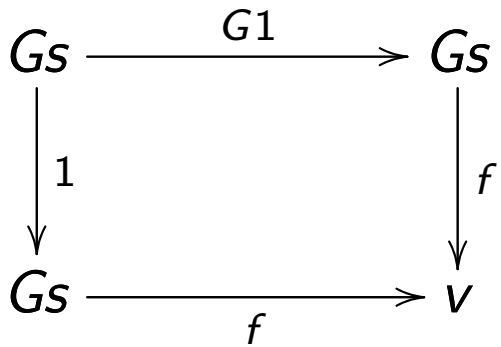


is an arrow of G/V.

What does a c-lens Put P do to that arrow?

Calculus of squares

Suppose $f : Gs \longrightarrow v$ is an object of G/V, then $(1, f)$

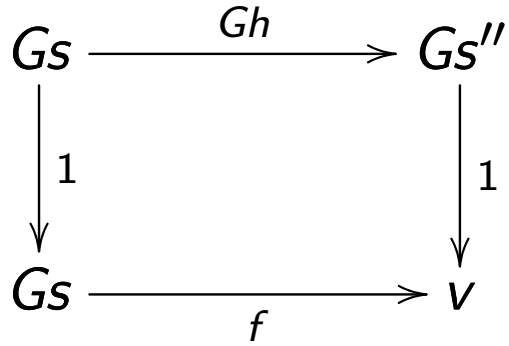


is an arrow of G/V.

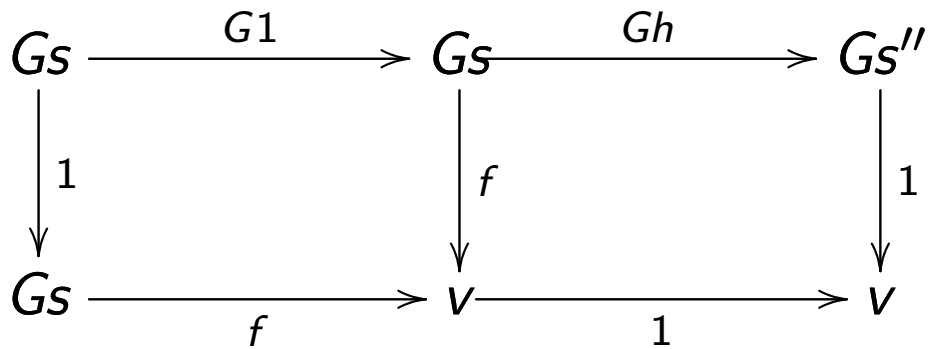
What does a c-lens Put P do to that arrow?
It has to take it to an arrow $k : s \longrightarrow P(s, v, Gs \xrightarrow{f} v) = s'$.

Least change

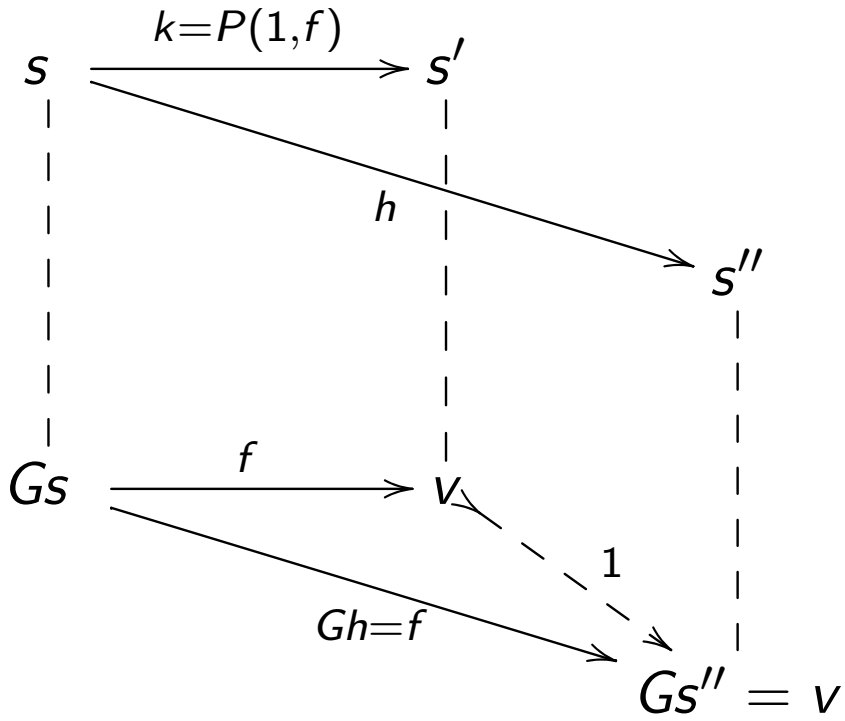
What's more, if $h : s \longrightarrow s''$ was an alternative suggestion for the explicit arrow provided by a d-lense (a different lift), that is if $Gh = f : Gs \longrightarrow v$ (and so $Gs'' = v$), then (h, f) is an arrow of G/V and



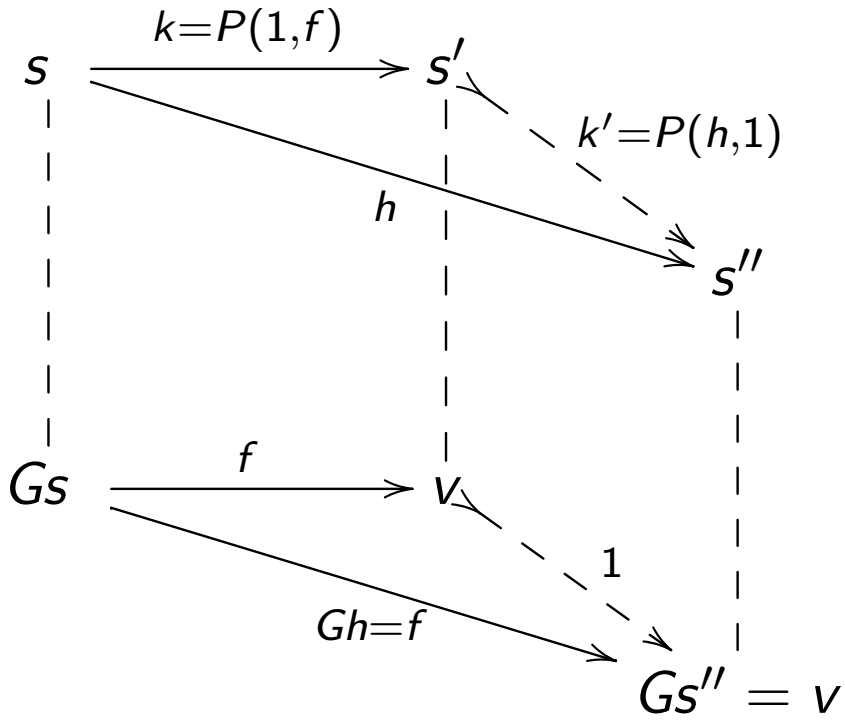
factors as $(1, f)\,;(h, 1)$



A (relabelled) diagram from Wednesday



A (least change) diagram from Wednesday



Summarising this first part

1. Categories and codiscrete categories (sets)
   - Sets are usually thought of as discrete categories
   - Sets in classical lenses are really codiscrete categories

   More in Part 3 about this
2. d-lenses
   - Pay proper attention to state spaces
   - Can have a lot of freedom to choose

   More in Part 3 about this too
3. c-lenses seemed to provide less info than d-lenses
   - In fact, same (k),
   - and even least change (more)
   - and even uniquely so (opfibrations)

Take note of

1. The calculus of squares and its power
2. Universal updates are an emergent property
3. PutPut is fundamental (more in Parts 2 and 3)
4. Lens versus pragmatic updates (Wednesday)
5. Least change

And avoid being confused by the use of the term “delta”.

Part 2

Yes, in a way, I'm going to start again, again

Can we work with oriented state spaces (eg info order)?

PutPut State Space Version

We've had some good discussions this week about PutPut. I noted down that

- PutPut is not in general the same as history ignorance
- But PutPut usually doesn't remember everything
- A PutPut framework:

$$Put(Put(s, \alpha), \alpha') = Put(s, \alpha \cdot \alpha')$$

(where of course the operation $\cdot$ and the precise parameters depend upon the context (the type of lens for example))

PutPut State Space Version

We've had some good discussions this week about PutPut. I noted down that

- PutPut is not in general the same as history ignorance
- But PutPut usually doesn't remember everything
- A PutPut framework:

$$Put(Put(s, \alpha), \alpha') = Put(s, \alpha \cdot \alpha')$$

(where of course the operation $\cdot$ and the precise parameters depend upon the context (the type of lens for example))

When we have $Gs \xrightarrow{\alpha} v \xrightarrow{\alpha'} v'$,

PutPut State Space Version

We've had some good discussions this week about PutPut. I noted down that

- PutPut is not in general the same as history ignorance
- But PutPut usually doesn't remember everything
- A PutPut framework:

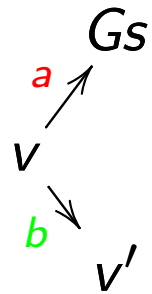
$$Put(Put(s, \alpha), \alpha') = Put(s, \alpha \cdot \alpha')$$

(where of course the operation $\cdot$ and the precise parameters depend upon the context (the type of lens for example))

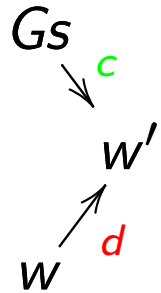
When we have $Gs \xrightarrow{\alpha} v \xrightarrow{\alpha'} v'$, or indeed $v' \xrightarrow{\alpha'} v \xrightarrow{\alpha} Gs$, we say the situation is *monotonic*

Compound Operations

Taking Tony's approach of labelling deletes red and inserts green, but orienting the arrows always in the information order we obtain appropriate domains for puts shaped like



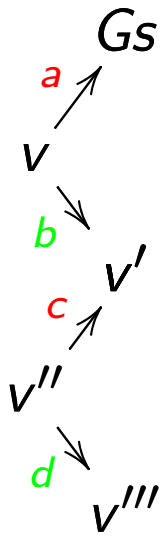
(an object of the domain of RLG) and the following (an object of the domain of LRG)



(The other compounds involving two arrows compose and satisfy monotonic PutPut: $Gs \longrightarrow v \longrightarrow v'$ or $v' \longrightarrow v \longrightarrow Gs$.)

Composing Compound Operations

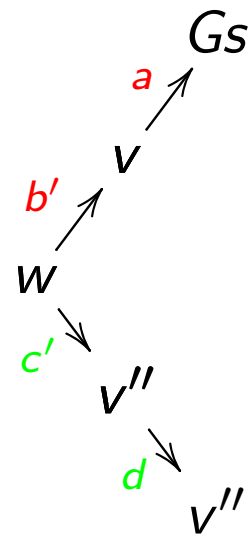
An object of the domain of *RLRLG* is a "zig-zag":



This provides the ingredients for composing mixed operations (and ultimately for another PutPut law).
The composite span is formed by taking the pullback of the cospan in the middle, assuming that pullbacks exist in **V**.

The Composed Spans

The composite span is



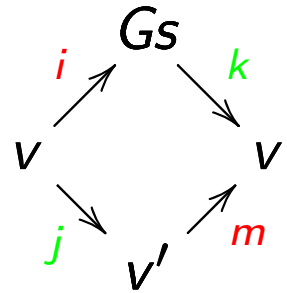
where w is a pullback of b and c. And then the legs themselves are composed in the usual way

(using the multiplication μ^L for the top leg (which is an instance of LLG), and the multiplication μ^R for the bottom leg (which is by then an instance of $RRLG$)).

The Short Story

If monotonic PutPut holds (for both information increasing and information decreasing updates along the information order) then the test for PutPut is how updates are propagated in the two ways around a pullback.

If the result is the same either way around, as it is for universal updates for keyed databases, then PutPut holds for atomic updates of the span form. (Cf Tony's work yesterday)



Notice what this means: The result of propagating an insert followed by a delete is the same as the result of propagating a particular delete followed by an insert (the least changing one – viz the pullback (or what Tony called yesterday "the one that stays furthest from empty")).

The longer story follows (six slides for aficionados)

A distributive law

We seek conditions on $G : \mathbf{S} \longrightarrow \mathbf{V}$ to guarantee that G supports compatible updates for *both* (formal) inserts and deletes.

Now assume that $\mathbf{V}$ *has pullbacks.*

We define a distributive law

$$LR \xrightarrow{\lambda} RL$$

between the monads making RL a monad

The algebras for RL are lenses whose updates treat a span of ordered updates as an atomic update.

The composites LR and RL

Both R and L are endofunctors on $\mathbf{cat}/\mathbf{V}$.

For $G : \mathbf{S} \longrightarrow \mathbf{V}$ the domain of $RG : (G, 1_{\mathbf{V}}) \longrightarrow \mathbf{V}$ has objects $GS \xrightarrow{a} V$ in $\mathbf{V}$ with $RG(GS \xrightarrow{a} V) = V$.

Apply L to RG giving $LR(G) : (1_{\mathbf{V}}, RG) \longrightarrow \mathbf{V}$ which has objects of the form $GS \xrightarrow{a} V \xleftarrow{b} V'$ in its domain, i.e. cospans (a, b) from GS to V'. Furthermore $LR(G)(a, b) = V'$

Similarly, RLG is a functor $(LG, 1_{\mathbf{V}}) \longrightarrow \mathbf{V}$. Objects in its domain are spans (c, d) from GS to W' in $\mathbf{V}$ having form $GS \xleftarrow{c} W \xrightarrow{d} W'$ and $RL(G)(c, d) = W'$.

Defining λ

The G'th component of the natural transformation λ will be a (pseudo-)functor $\lambda_G : (1_{\mathbf{V}}, RG) \longrightarrow (LG, 1_{\mathbf{V}})$ (fibred over $\mathbf{V}$).

At an object $GS \xrightarrow{a} V \xleftarrow{b} V'$ in the domain $(1_{\mathbf{V}}, RG)$ of LRG

(LRG is a functor from the iterated comma category to $\mathbf{V}$)

take the pullback in $\mathbf{V}$ (and on arrows take the induced maps).

Thus $\lambda_G(a, b)$ is a span in $\mathbf{V}$ from GS to V', an object in the domain of RLG.

Theorem

The transformation $LR \xrightarrow{\lambda} RL$ is a pseudo-distributive law.

Algebras for *RL*

If $G : \mathbf{S} \longrightarrow \mathbf{V}$ is the carrier for both an R-algebra and an L-algebra then both universal inserts and universal deletes are available.

BUT we have no information about how the delete and insert liftings interact. However, if G is an RL algebra then all is well.

Recall that G is an RL algebra exactly if G is an L-algebra *and* an R-algebra *and* the L algebra structure is a homomorphism of R algebras.

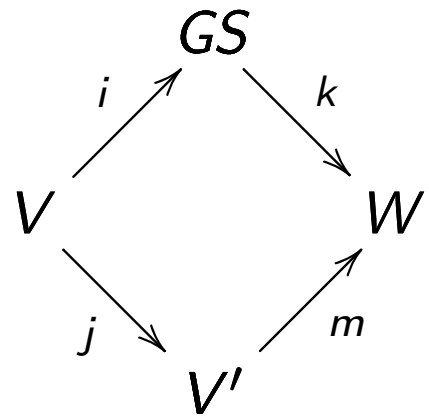
We can also use condition what is essentially Beck-Chevalley:

A BC Condition

Suppose $G : \mathbf{S} \longrightarrow \mathbf{V}$ and $RG \xrightarrow{r} G$, $LG \xrightarrow{\ell} G$ are R-, L-algebras.

For $k : GS \longrightarrow W$ an object of $G/\mathbf{V}$, denote r image by $r(S, k)$.
For $i : V \longrightarrow GS$ an object of $\mathbf{V}/G$, denote ℓ image by $\ell(S, k)$.

Condition $(*)$ *is satisfied* if
for all S an object of $\mathbf{S}$ and for any pullback (in $\mathbf{V}$)



it is the case that $r(l(i, S), j) \cong l(r(S, k), m)$.

Updating an insert along k and then a delete along m is equivalent to updating a delete along i and then an insert along j.

Algebra criterion

Theorem

If $RLG \xrightarrow{\xi} G$ *is an RL-algebra then* $r = \xi R \eta^L G$ *and* $\ell = \xi \eta^R LG$ *define R-, L-algebras satisfying condition* $(*)$.

Conversely, suppose $RG \xrightarrow{r} G$ *and* $LG \xrightarrow{\ell} G$ *are R-, L-algebras satisfying* $(*)$. *Define*

$$\xi : RLG \longrightarrow G \text{ by } \xi(GS \xleftarrow{i} V \xrightarrow{j} V') = r(l(i, S), j).$$

Then ξ *is an RL algebra structure on* G.

In other words, insert updates satisfy monotonic PutPut, delete updates satisfy monotonic PutPut, and jointly they satisfy $(*)$ if and only if we have an algebra of mixed updates satisfying span PutPut. In that case atomic updates are spans of directed (eg information order) updates.

Part 3

Yes I'm going to start again, but for the last time...

Can we systematise and simplify through mathematical foundations?

An Alpabet Soup of Lenses

	Origin
c-lens	Category
d-lens	Delta
e-lens	Edit
...	
p-lens	Projection
q-lens	Quotient
r-lens	Relational
...	
u-lens	Update
v-lens	Very Well Behaved
w-lens	Well Behaved

Many come in at least two variants (symmetric and asymmetric).

Known Relationships

1. State spaces can all be seen as categories.
2. Among those lenses
   - Symmetrics can be derived from asymetrics via the span construction
   - Edit lenses can be converted to Delta lenses by the category of elements construction
   - Set-based lenses are special cases of Delta lenses using codiscrete categories of states
   - And c-lenses are a special case of d-lenses (Part 1 today)
3. The Puts are usually functorial
   - That is they satisfy appropriately phrased GetPut and PutPut laws

A "Grand" Unification (well..., a pretty good one)

Lens	Origin	Asymmetric	Symmetric	PutPut	Type
c-lens	Category	$\rightarrow$		Yes	delta
d-lens	Delta	$\rightarrow$	$\not\rightleftarrows$	Yes	delta
e-lens	Edit	$\rightarrow$	$\not\rightleftarrows$	Yes	edit
p-lens	Projection				extant
q-lens	Quotient				extant
r-lens	Relational				extant
u-lens	Update	$\rightarrow$		Yes	edit
v-lens	VeryWB	$\rightarrow$	$\not\rightleftarrows$	Yes	set-based
w-lens	WB'haved	$\rightarrow$	$\not\rightleftarrows$	No	set-based

NULLs

Exercises

You don't have to do them, but if you want to learn this stuff you need to play

Would you like some?

Exercises

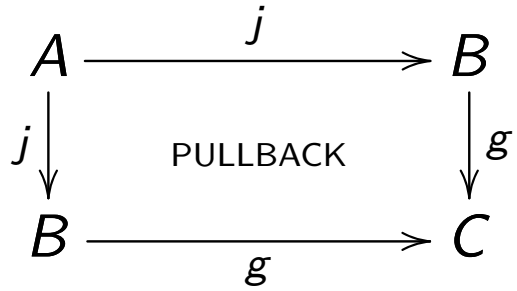
1. "Prove" the folk theorem that g is surjective in any asymmetric set-based lens satisfying the three axioms. Which axioms do you really need in your proof?
2. Now fix it — the folk theorem is false. Why? State the correct theorem, and check that your proof is really a proof now.
3. What is the free category on a directed graph? Compare it with the free monoid on a set. How is the free category a "set of terms"?
4. What is the free category on a directed graph which has only one object? It's something you know already.

Exercises continued

5. First, describe a finite monoid M of your choice. The counit of the free monoid adjunction $F \dashv U$ is a monoid morphism from the free monoid on the elements of M to M itself. Describe the effect of the counit in your case explicitly. If M did not have cardinality greater than 1 then do the question again.
6. Show that, as claimed, $\Sigma \dashv \Delta$. Is that true even when V is empty?
7. State the axioms PutGet, GetPut and PutPut for d-lenses.
8. Repeat the previous question for c-lenses.

Exercises continued further

9. Prove that if



is a pullback, then g is monic (that is, whenever $gh = gk$ it necessarily follows that $h = k$).
This is how pullback squares can be used to specify that arrows in EA-sketches are required to be monic in models.

10. Prove that a natural transformation between models of keyed EA-sketches has components which are all monic.
This demonstrates that keyed databases have categories of models which are partial orders (there exists at most one arrow (one update) between any two states). The ordering is the *information order*.

Exercises continued further further

11. Choose your favourite kind of asymmetric lens. Consider the pullback of the Gets of two such lenses and show how the given lenses determine lens structures on the pullback projections.
(Hint: Use the fact that a pullback in **set** or **cat** can be represented as a collection of pairs (of elements, or of objects and of arrows, respectively)

End of these lectures

Would you like some morning tea?