Operational Transform via Category Theory

david@davidespinosa.net

November 2023

Foozles

“Now largely divorced from its roots in programming languages, Foozle Theory has evolved into a deep and complex subject that lies at the conjunction of category theory, term rewriting, topology, and distributed systems.”

-- Todd Veldhuizen

Foozles: Anatomy of a PL fad (2006)

My category theory teachers

- Y.V. Srinivas
- Dusko Pavlovic
- Jose Meseguer
- Joseph Goguen

Goguen’s career summary:
Tossing algebraic flowers into the great divide

Srinivas’ brilliant PhD thesis:
Pattern matching: a sheaf-theoretic approach

Prior work

- Ellis and Gibbs 1989
  Concurrency control in groupware systems
- Sun and Sun 2009
  Context-based operational transformation in distributed collaborative editing systems

Contribution

- Started from the Sun and Sun algorithm
- Reverse-engineered the theory

Applications

- Collaborative editing
- Version control
- Multiplayer games
- Virtual worlds

Functional consistency

- Each node maintains a local copy of the state
- Each node broadcasts updates to all other nodes
- State is pure function of updates received
- Network is eventual, algorithm is functional

Total order broadcast

- Replicated state machine (Lamport 1977)
- Nodes broadcast updates
- Received updates are sorted and applied
- Various solutions in literature
- Unfortunately, solutions are blocking !

Causal broadcast

- Uses causal “happens before” partial order (Lamport 1977)
- Concurrent updates are unordered
- So concurrent updates must commute
- Non-blocking – entire motivation for OT !
- OT = replicated state machine for causal broadcast

Is OT a good idea ?

- Pro: Offline and peer-to-peer
- Con: N^2 cases for N operations
- Alternative: a real-time protocol
  - Multiplayer games
  - Spanner
  - https://uspto.report/patent/app/20200042199

Outline

- TL ; DR
- Time
- State
- Time → State
- Code
- Related ideas
- Topology
- Building State

Counting – monoid

size : $2^E \rightarrow (\mathrm{Nat}, +)$
size($A \subseteq B$) = $|B - A|$

$$
\begin{tikzcd}
& b \arrow[dl, "\subseteq"'] \arrow[dr, "\subseteq"] & \\
a,b \arrow[dr, "\subseteq"'] & & b,c,d \arrow[dl, "\subseteq"] \\
& a,b,c,d,e &
\end{tikzcd}
$$

$$
\begin{tikzcd}
& {} \arrow[dl, "1"'] \arrow[dr, "2"] & \\
{} \arrow[dr, "3"'] & & {} \arrow[dl, "2"] \\
& {} &
\end{tikzcd}
$$

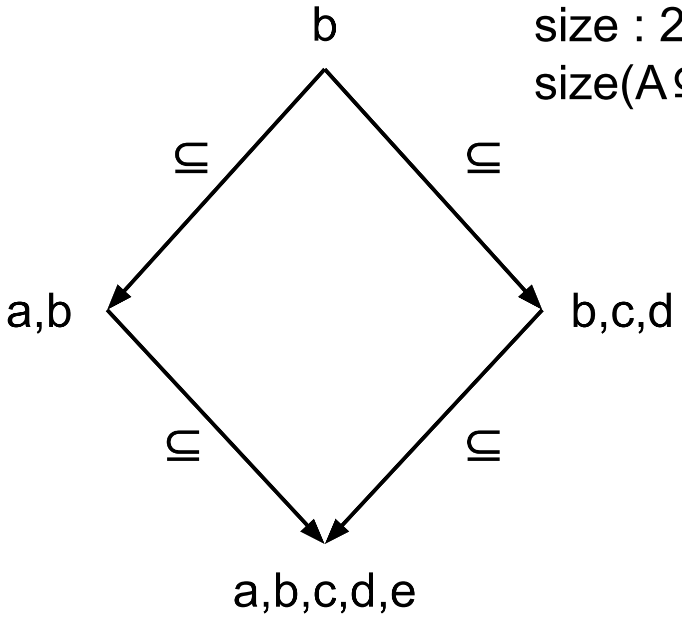
Counting – monoid

$$\begin{tikzcd}
 & b \arrow[dl, "\subseteq"'] \arrow[dr, "\subseteq"] & \\
a,b \arrow[dr, "\subseteq"'] & & b,c,d \arrow[dl, "\subseteq"] \\
 & a,b,c,d &
\end{tikzcd}$$

$$|A \cup B - A| = |B - A \cap B|$$

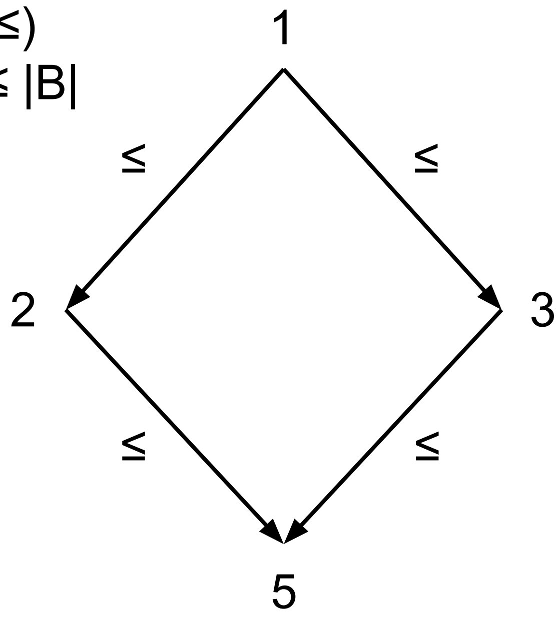
$$\begin{tikzcd}
 & {} \arrow[dl, "1"'] \arrow[dr, "2"] & \\
{} \arrow[dr, dashed, "2"'] & & {} \arrow[dl, dashed, "1"] \\
 & {} &
\end{tikzcd}$$

Counting – total order



size : $2^E \rightarrow (\mathrm{Nat}, \leq)$

size($A \subseteq B$) = $|A| \leq |B|$



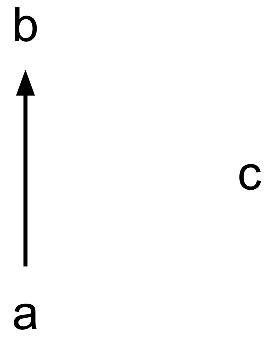
Counting – total order

$$
\begin{tikzcd}
 & b \arrow[dl, "\subseteq"'] \arrow[dr, "\subseteq"] & \\
a,b \arrow[dr, "\subseteq"'] & & b,c,d \arrow[dl, "\subseteq"] \\
 & a,b,c,d &
\end{tikzcd}
$$

$$|A \cup B| = |A| + |B| - |A \cap B|$$

$$
\begin{tikzcd}
 & 1 \arrow[dl, "\leq"'] \arrow[dr, "\leq"] & \\
2 \arrow[dr, dashed, "\leq"'] & & 3 \arrow[dl, dashed, "\leq"] \\
 & 4 &
\end{tikzcd}
$$

Events

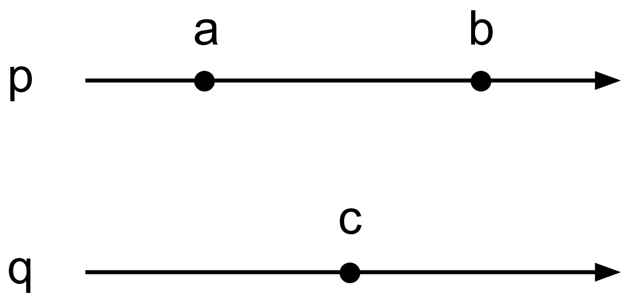


partial order (acyclic)
“happens before”
Lamport 1977

a ≤ b   a || c   b || c

“unrelated”
“concurrent”
“in parallel”
“independent”

Events

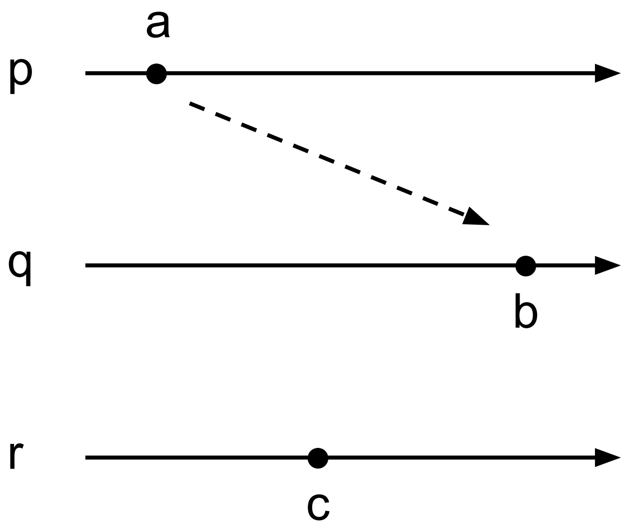


Does NOT mean
a ≤ c ≤ b

Events at different locations
are concurrent

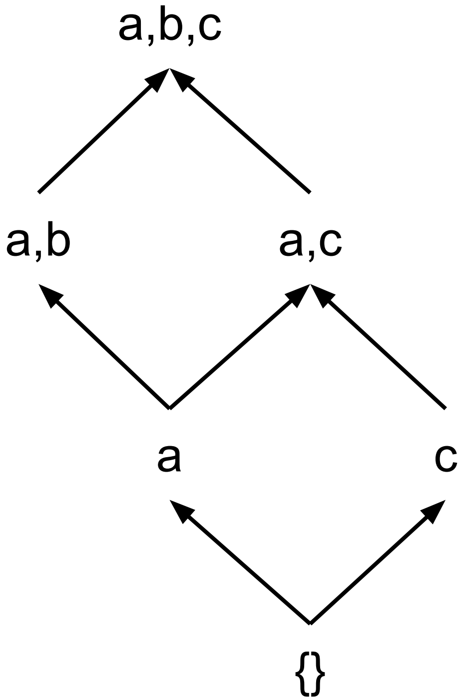
Like special relativity !

Events



Send happens before receive

From events to times

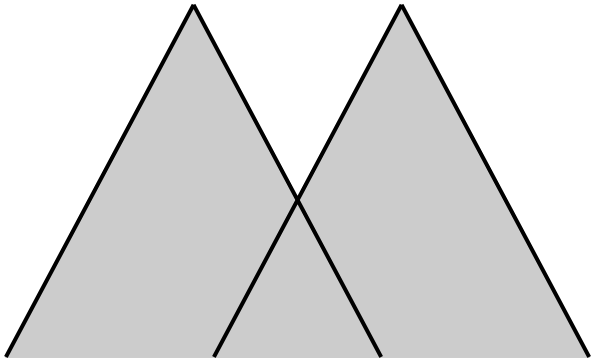


time = set of events that have happened
downward closed
{b} and {b,c} are not dc

Mattern 1988
lattice (union and intersection)
vector clocks

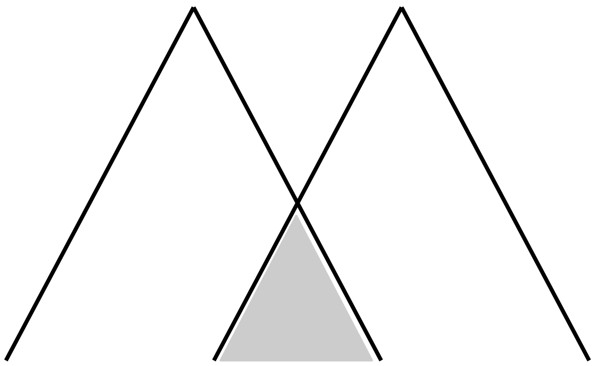
Functors:
$le(b) = \{ a \mid a \leq b \}$
$lt(b) = \{ a \mid a < b \}$

From events to times



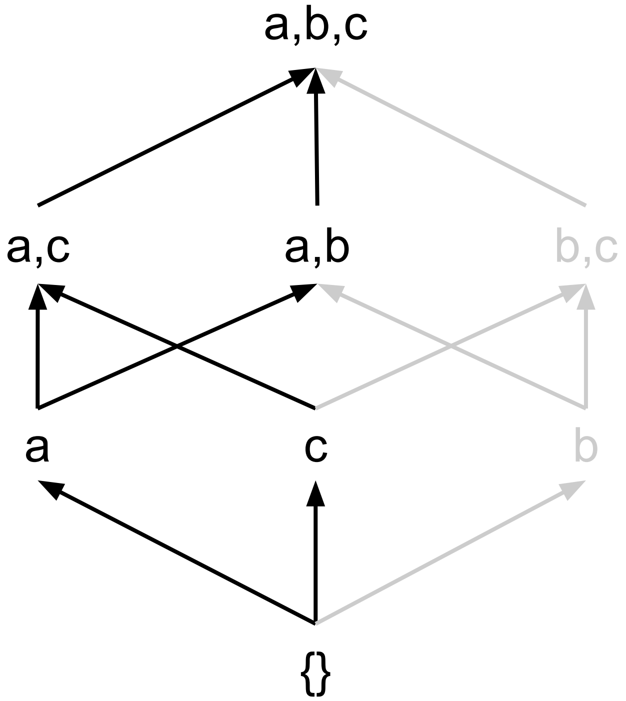
union of dc is dc

From events to times

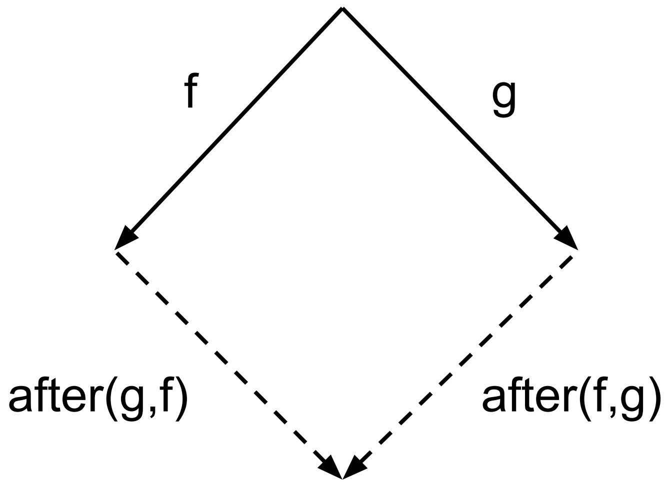


intersection of dc is dc

Hypercube



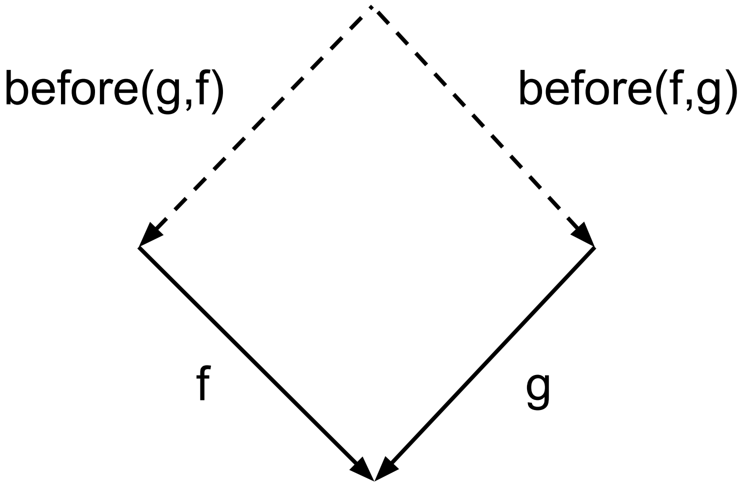
Cocommutative category



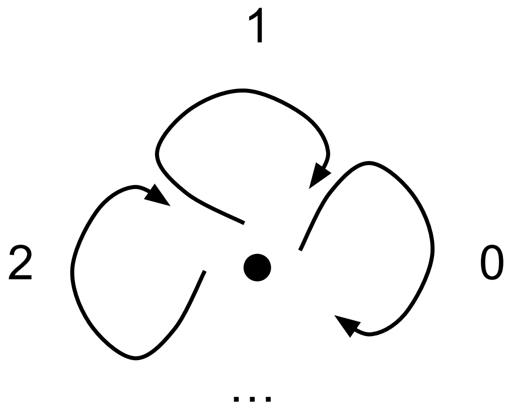
Explains parallel using sequential
Three-way merge on objects

Implicit in:
Gabriel and Zisman 1967
(numerators commute with
denominators)

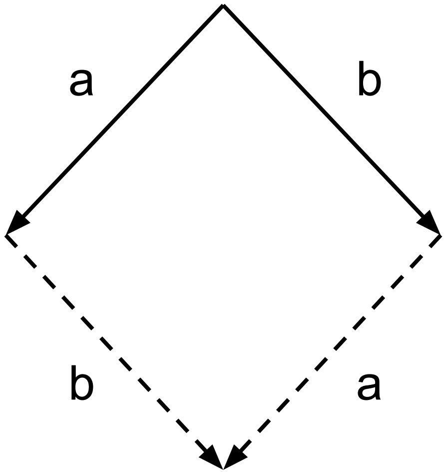
Commutative category



(Nat, +)



Not a pushout !



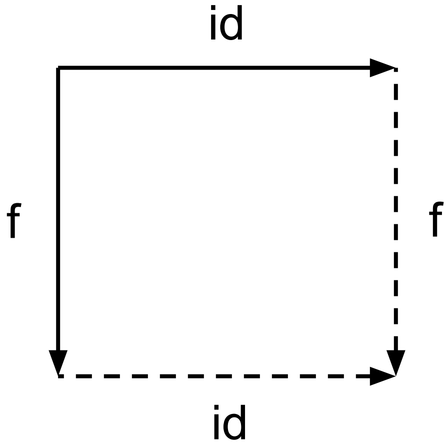
(Nat, ≤)

$$0 \longrightarrow 1 \longrightarrow 2 \longrightarrow \dots$$

$$\begin{tikzcd}
& x \arrow[dl, "\leq"'] \arrow[dr, "\leq"] & \\
a \arrow[dr, dashed, "\leq"'] & & b \arrow[dl, dashed, "\leq"] \\
& a+b-x &
\end{tikzcd}$$

Not a pushout !

Functoriality



Assistance from Antonio Ramirez 👍

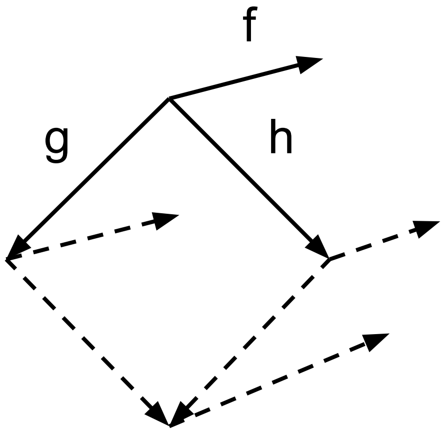
Functoriality

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
{} \arrow[r, "g"] \arrow[d, "f"'] & {} \arrow[r, "h"] \arrow[d, dashed, "A(f{,}g)"] & {} \arrow[d, dashed, "A(A(f{,}g){,}h)"] \\
{} \arrow[r, dashed, "A(g{,}f)"'] & {} \arrow[r, dashed, "A(h{,}A(f{,}g))"'] & {}
\end{tikzcd}
$$

Assistance from Antonio Ramirez 👍

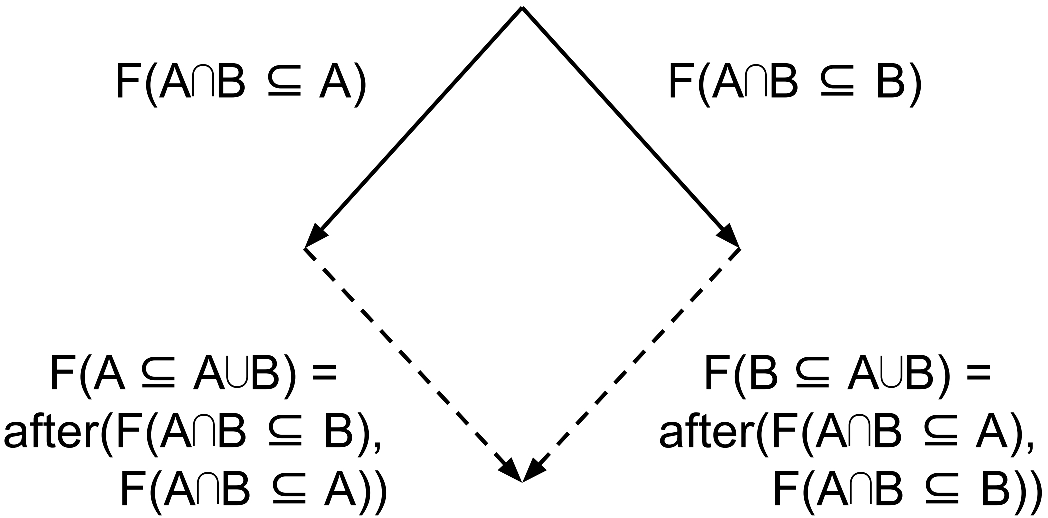
Cube law / T2

A(A(f,g), A(h,g))
= by functoriality
A(f, g ; A(h,g))
= by after
A(f, h ; A(g,h))
= by functoriality
A(A(f,h), A(g,h))



Holds in any cocommutative category

Property 1



User model

u : Event * obj(State) → arr(State)

$$s \xrightarrow{u(e,\ s)} t$$

s = state before e
t = state after e

Property 2

$$\begin{tikzcd} \text{Event} \arrow[r, shift left=3, "Ie"] \arrow[r, shift right=3, "It"'] \arrow[r, phantom, "\subseteq \Uparrow"] & \text{Time} \arrow[r, "F"] & \text{State} \end{tikzcd}$$

$$F(It(e) \subseteq Ie(e)) = u(e, F(It(e)))$$

OT proposition

Given

- partial order Event
- cocommutative category State
- function u

There exists unique

- functor $F : \text{Time} \rightarrow \text{State}$
- with properties 1 and 2

Code

F : Time → State

f_state : Time → State

f_transition : Time * Time → Transition

Code

```
t0 = set()
t1 = set([‘a’, ‘b’, ‘c’])

lt_dict = {
    'a': t0,
    'b': set(‘a’),
    'c': t0
}

def lt(e):
    return lt_dict[e]
```

```
event_value_dict = {
    'a': 1,
    'b': 2,
    'c': 3
}

def event_value(e):
    return event_value_dict[e]
```

Code

```
class Monoid:
    def id(self, a):
        return 0

    def compose(self, a, b):
        return a + b

    def dom(self, a):
        return None

    def cod(self, a):
        return None

    def after(self, a, b):
        return a

    f0 = None

    def user(self, e, a):
        return event_value(e)
```

Code

```
class TotalOrder:
    def id(self, a):
        return (a, a)

    def compose(self, t1, t2):
        a, b1 = t1
        b2, c = t2
        assert b1 == b2
        return (a, c)

    def dom(self, t):
        return t[0]

    def cod(self, t):
        return t[1]

    def after(self, t1, t2):
        x1, a = t1
        x2, b = t2
        assert x1 == x2
        return (b, a + b - x1)

    f0 = 0

    def user(self, e, a):
        return (a, a + event_value(e))
```

Code

```
def next_event(s, t):
    es = [e for e in t - s if lt(e) <= s]
    return random.choice(es)

def f_state(cc, t):
    if t == t0:
        return cc.f0
    return cc.cod(f_transition(cc, t0, t))
```

```
def f_transition(cc, s, t):
    result = cc.id(f_state(cc, s))
    while s != t:
        e = next_event(s, t)
        u = cc.user(e, f_state(cc, lt(e)))
        ua = cc.after(u, f_transition(cc, lt(e), s))
        result = cc.compose(result, ua)
        s = s | set(e)
    return result

print(f_transition(Monoid(), t0, t1))
print(f_transition(TotalOrder(), t0, t1))
```

$F(\{\} \subseteq \{\})$

f0

$F(\{\} \subseteq \{a\})$

$$
\begin{tikzcd}
f0 \arrow[r, "u(a,\, f0)"] \arrow[d] & fa \\
f0 \arrow[r] & fa
\end{tikzcd}
$$

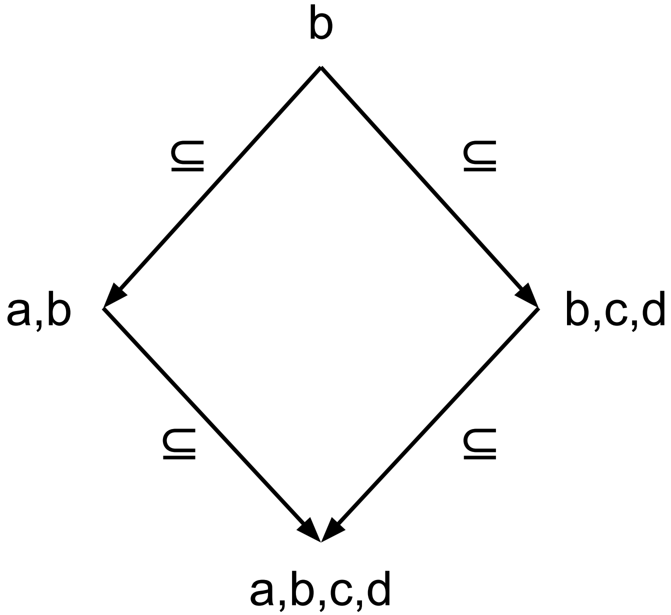
F({} ⊆ {a,c})

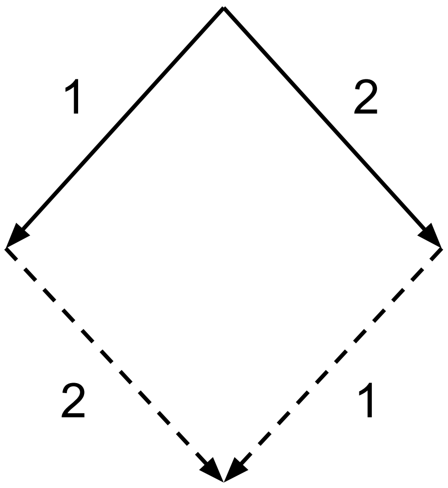
$$\begin{tikzcd}
f0 \arrow[r, "{u(a, f0)}"] \arrow[d] & fa & \\
f0 \arrow[r] & fa \arrow[r] & fac \\
& f0 \arrow[u] \arrow[r, "{u(c, f0)}"'] & fc
\end{tikzcd}$$

$F(\{\} \subseteq \{a,b,c\})$

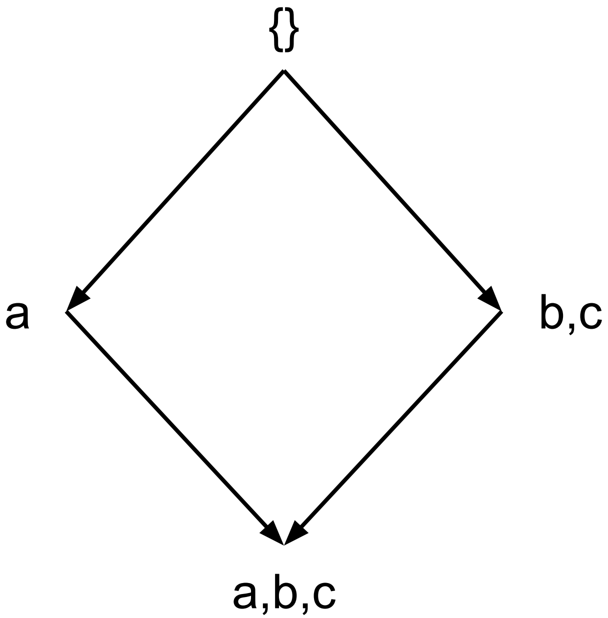
$$
\begin{tikzcd}
f0 \arrow[r, "{u(a,\ f0)}"] \arrow[d] & fa & fa \arrow[r, "{u(b,\ fa)}"] \arrow[d] & fab \\
f0 \arrow[r] & fa \arrow[r] & fac \arrow[r] & fabc \\
 & f0 \arrow[r, "{u(c,\ f0)}"'] \arrow[u] & fc &
\end{tikzcd}
$$

Is OT just counting ?

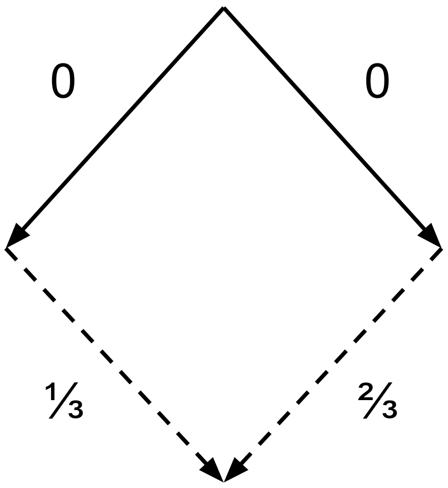




Is OT just probability ?



P(A|B)



Fails – can't compute bottom from top !

Assistance from Jon Rowlands 👍

Is OT just probability ?

$$
\begin{tikzcd}
& \{\} \arrow[dl] \arrow[dr] & \\
a \arrow[dr] & & b,c \arrow[dl] \\
& a,b,c &
\end{tikzcd}
\qquad
\begin{tikzcd}
& {} \arrow[dl, dashed, "0"'] \arrow[dr, dashed, "0"] & \\
{} \arrow[dr, "\frac{1}{3}"'] & & {} \arrow[dl, "\frac{2}{3}"] \\
& {} &
\end{tikzcd}
$$

$$(\tfrac{1}{3} + \tfrac{2}{3} - 1) / (\tfrac{1}{3}) = 0$$

$$(\tfrac{1}{3} + \tfrac{2}{3} - 1) / (\tfrac{2}{3}) = 0$$

Is OT just Seifert - van Kampen (1931) ?

π_1 : Sub(X) → Groupoid

Fundamental groupoid
of a topological space

$$
\begin{tikzcd}
& \pi_1(A \cap B) \arrow[dl] \arrow[dr] & \\
\pi_1(A) \arrow[dr, dashed] & & \pi_1(B) \arrow[dl, dashed] \\
& \pi_1(A \cup B) &
\end{tikzcd}
$$

Finally, a pushout !

Is OT just CRDTs ?

- Conflict-free replicated data types
- Op-based version (commutative monoid, 2007)
- State-based version (semilattice, 2009)
- Special cases of OT 🙂
- OT removes duplicates in the framework, not in the datatype

Is OT just term rewriting ?

- Term rewriting starts with $\to^1$ and $\downarrow^1$
- Then builds $\to^*$ and $\downarrow^*$
- That's a cocommutative category !
- But it's a preorder – paths don't matter
- OT could do labeled term rewriting (State)
- OT manages confluence process (Time)

Is OT just topological sort ?

- OT performs operations in topological order (Time)
- But OT also transforms operations (State)

Also:

- Not enough to know the order “acb”
- Need: b depends on a but not on c

Topological sort

Given
E partial order, M monoid
$u : E \to M$
$a \parallel b \Rightarrow u(a) \parallel u(b)$

concurrent
operations
commute

There exists unique functor:
$F : \mathrm{Time} \to M$
$F(A \subseteq A \cup B) = F(A \cap B \subseteq B)$
$F(\mathrm{lt}(e) \subseteq \mathrm{le}(e)) = u(e)$

Topological sort

$E = a \rightarrow b$
$\quad\quad c$

$M = \text{String} \times \text{String}$

u(a) = (“a”, “”)
u(b) = (“b”, “”)

u(c) = (“”, “c”)

Topological sort

$$
\begin{tikzcd}
\{\} \arrow[r] \arrow[d] & a \arrow[r] \arrow[d] & ab \arrow[d] \\
c \arrow[r] & ac \arrow[r] & abc
\end{tikzcd}
\qquad
\begin{tikzcd}
{} \arrow[r, "a,"] \arrow[d, ",c"] & {} \arrow[r, "b,"] \arrow[d, ",c"] & {} \arrow[d, ",c"] \\
{} \arrow[r, "a,"] & {} \arrow[r, "b,"] & {}
\end{tikzcd}
$$

$$F(\{\} \subseteq \{a,b,c\}) = (\text{“ab”}, \text{“c”})$$

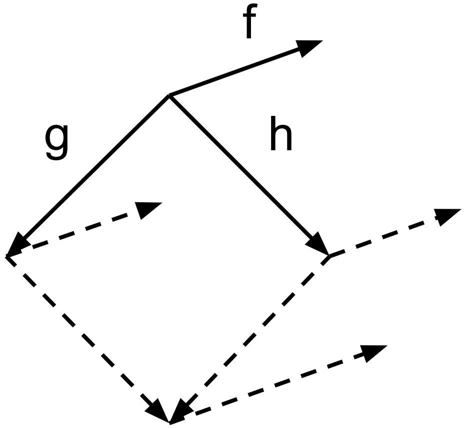
Is OT just a commutative diagram ?

- Build shape
- Add labels
- Show that diagram commutes

Advantages:

- after() only acts on single operations, not sequences
- Don’t need a cocommutative category

Is F well-defined ?



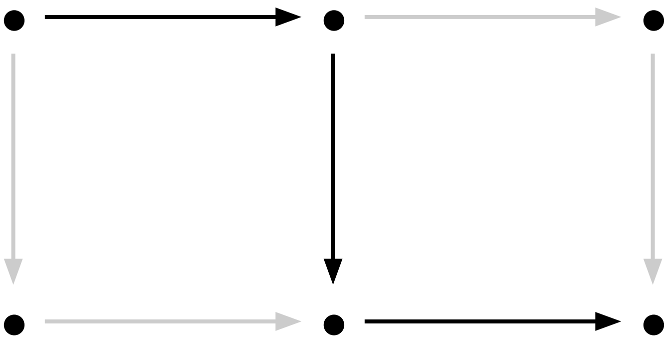
$$A(A(f,g), A(h,g)) = A(A(f,h), A(g,h))$$

cube law

Pasting lemma

$$\begin{tikzcd}
\bullet \arrow[r] \arrow[d] & \bullet \arrow[r] \arrow[d] & \bullet \arrow[d] \\
\bullet \arrow[r] & \bullet \arrow[r] & \bullet
\end{tikzcd}$$

Pasting lemma



Pasting lemma

$$\begin{tikzcd}
\bullet \arrow[r] \arrow[d] & \bullet \arrow[r] \arrow[d] & \bullet \arrow[d] \\
\bullet \arrow[r] & \bullet \arrow[r] & \bullet
\end{tikzcd}$$

Commutativity theorem

If all parallel paths homotopic
and all faces commute
then diagram commutes

global condition on shape
+ local condition on labels
= global condition on diagram

Commutativity theorem

For a 16-dimensional hypercube:

\# paths = $n! = 2\times10^{13}$

\# faces = $C(n,2)\ 2^{n-2} = 2\times10^{6}$

\# paths / # faces = 10^{7}

Homotopy theory

~~Classify topological spaces~~

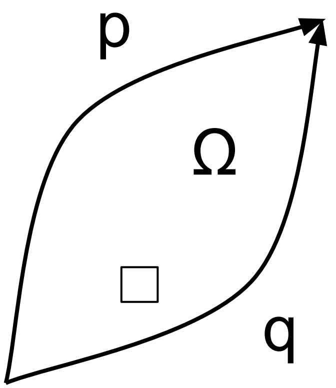
Show that diagrams commute

Is OT just Stokes' theorem (1850) ?

$\int_{p-q} f = \int_{\Omega} df$

If df = 0:

$\int_{p} f = \int_{q} f$



Topology and concurrency

- Herlihy, Kozlov, and Rajsbaum 2014
  Distributed computing through combinatorial topology
- Fajstrup, Goubault, Haucourt, Mimran, and Raussen 2016
  Directed algebraic topology and concurrency

Highly recommended:

- Ghrist 2014
  Elementary Applied Topology

Building State

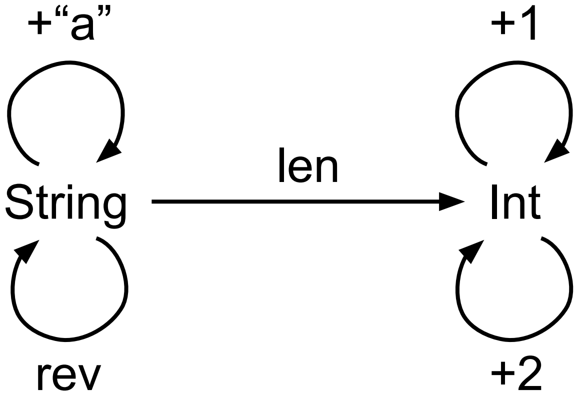
Start with:

- A collection of basic operations
- A definition of after() for each pair

Build:

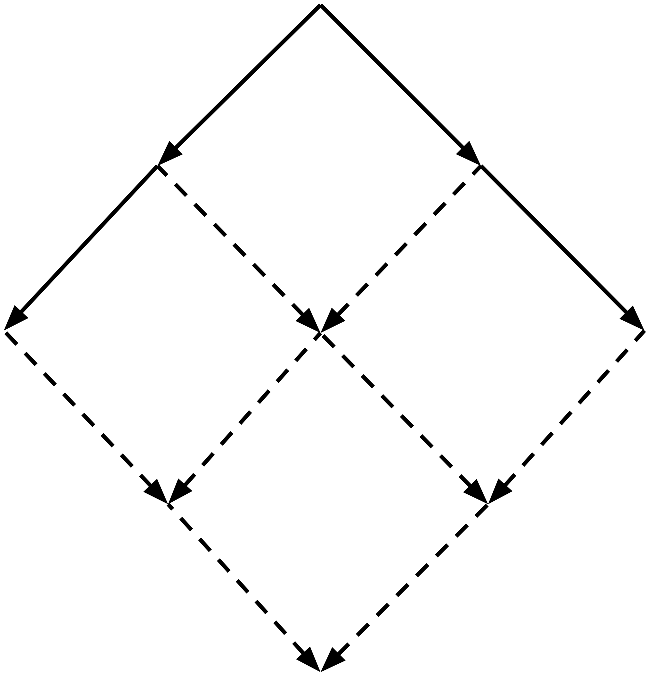
- A cocommutative category

Syntax



Syntax = paths in graph

after() on paths



Commutativity

$$
\begin{tikzcd}
 & \mathrm{Int} \arrow[dl, "1+"'] \arrow[dr, "2+"] & \\
\mathrm{Int} \arrow[dr, "2+"'] & & \mathrm{Int} \arrow[dl, "1+"] \\
 & \mathrm{Int} &
\end{tikzcd}
$$

Does this diagram commute ?

Commutativity

$$\begin{tikzcd} & \mathrm{Int} \arrow[dl, "1+"'] \arrow[dr, "2+"] & \\ \mathrm{Int} \arrow[dr, "2+"'] & & \mathrm{Int} \arrow[dl, "1+"] \\ & \mathrm{Int} & \end{tikzcd}$$

Does this diagram commute ?

In Set, yes.
In Syntax, no !

Syntax mod after

We quotient paths by:

f ; after(g,f) ~ g ; after(f,g)

for every pair f,g of basic operations.

Is after() well-defined ?

A(k, f ; A(g,f))

~ by functoriality

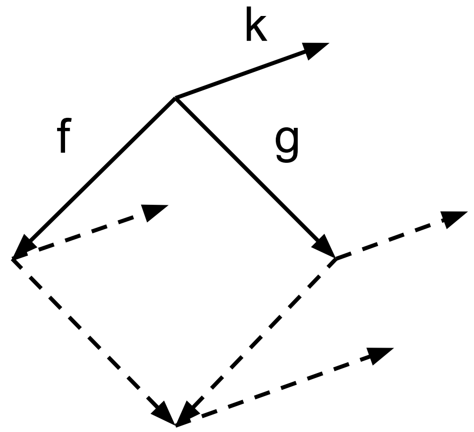
A(A(k,f), A(g,f))

~ by cube law

A(A(k,g), A(f,g))

~ by functoriality

A(k, g ; A(f,g))

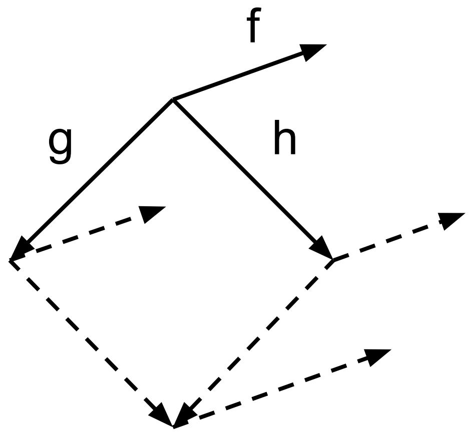


cube law

Cube law

- Holds in any cocommutative category
- Used to label diagram consistently
- Used to build state category from individual operations

Cube law



Assures cube is solid ?
Reduces dimension from N to 2 ?
Need to ask a topologist !

Summary