Primer on Operational Transformation

David Barrett-Lennard

15 Oct 2015

# Operations

Operations represent the updates or deltas to be applied to the database. Let operation O be applied on state S1 to get state S2.

O

S1

S2

An example might be the deletion of a character within a string variable:

"abc"

Delete 'b' at position 1

"ac"

We call S1 the input state and S2 the output state of operation O. This state of affairs is expressed in the following equation:

S1 ⊙ O = S2

input state

operation

output state

Operation application

# Operations don’t necessarily commute

Function composition is associative but not commutative.

Associative: (f∘g)∘h = f∘(g∘h)

Commutative: f∘g = g∘f

Similarly operations don't commute in general, meaning they can't generally be applied in different orders.

Definition: We say O1, O2 commute if

S ⊙ O1 ⊙ O2 = S ⊙ O2 ⊙ O1

# Context equivalent and context serialised operations

Let O1, O2 be concurrent operations (i.e. O1 || O2).

Definition: We say operations O1 ,O2 are context equivalent if they share the same input state.

Definition: We say operations O1 ,O2 are context serialised if the output state of O1 is the input state of O2.

# Transforms

## Inclusion Transformation (IT)

Let O1, O2 be context equivalent operations with input state S. The following diagram depicts the inclusion transform of O2 past O1 to give O2’. O2’ preserves the intention of O2. The input state of O2’ is S⊙O1.

O1

O2

O2’ = IT(O2,O1)

IT of O2 past O1

S

S⊙O1

## Exclusion Transformation (ET)

Let S be the input state of operation O1 and let O1,O2’ be context serialised. The following diagram depicts the exclusion transform of O2’ past O1 to give O2. O2 preserves the intention of O2’. The input state of O2 is S.

O1

O2 = ET(O2’,O1)

O2’

ET of O2’ past O1

S

S⊙O1

## TP1

Transformation property TP1 ensures two sites converge to the same state

O1

O2

IT(O2,O1)

IT(O1,O2)

S⊙O1⊙IT(O2,O1) =

S⊙O2⊙IT(O1,O2)

S

## TP2

Let O1, O2 and O3 be context equivalent operations. Transformation property TP2 means transforming operation O3 either past O1 then IT(O2,O1) or past O2 then IT(O1,O2) gives the same result.

O1

O3

O2

IT(O2,O1)

IT(O1,O2)

IT(IT(O3,O1), IT(O2,O1)) =

IT(IT(O3,O2), IT(O1,O2))

## Sufficient conditions for convergence

TP1+TP2 ⇒ convergence for different paths through the lattice

A

C

B

TP1+TP2 ⇒ convergence

From an inductive definition of the lattice, In 1996 Ressel et al gave an inductive proof of convergence (using induction on the magnitude of the vector time).

## OT approaches which don’t satisfy TP2

There are a number of approaches to OT which fail to satisfy condition TP2 (such as the approach used in Google Docs or Google Wave), and this imposes various limitations on the implementation:

* There can be unintuitive merge results e.g. the case of three operations on text where there’s an ambiguity for concurrent insertion positions on one of the faces. The solutions which fail to meet TP2 typically fail to respect a total order on insertions.
* Not robust:
  + Can’t cope with server failure
  + Requires transaction durability
  + Network topology restrictions
* No arbitrary branching and merging (configuration management)

## Dual IT

Let O1 and O2 be context equivalent operations. DualIT refers to the calculation of both inclusion transforms. In the diamond below the two operations on the left are transformed past each other to give the two operations on the right.

O1

O2

O2’ = IT(O2,O1)

O1’ = IT(O1,O2)

Similarly we can consider the dualIT of context equivalent lists of operations L1, L2 (we write L1 || L2 to mean the lists are concurrent, meaning ∀O1∈L1 ∀O2∈L2 O1||O2)

L1

L2

L2’ = IT(L2,L1)

L1’ = IT(L1,L2)

The following diagram shows how we can calculate IT(L2,L1) and IT(L1,L2). Unfortunately it has quadratic complexity, so it is not suitable for a practical implementation. In fact it implies the whole idea of using linear history buffers isn’t suitable for implementation.

L1

L2

L1’

L2’

## Transpose

The transpose of context serialised operations O1,O2 is associated with applying them in the reverse order. In the following diagram the two operations on the top of the diamond are transformed to give the two operations on the bottom of the diamond.

O1

O2

O2’ = ET(O2,O1)

O1’ = IT(O1, ET(O2,O1))

Similarly we can consider the transpose of context serialised lists of operations L1, L2

L1

L2

L2’ = ET(L2,L1)

L1’ = IT(L1, ET(L2,L1))

However just like DualIT the transpose of two lists is quadratic

L1

L2

L1’

L2’

## History buffers of two sites

Consider the following Venn diagram depicting the sets of operations which have been applied so far at sites A and B

Ai

Ci

Bi

Site A

Site B

The state having the Ci preserves causality

* The Ai are the operations unique to site A
* The Bi are the operations unique to site B
* The Ci are the operations they have in common

The set of operations in common represents a state that preserves causality. Also for the union of all operations

Consider that each site records the ordered list of operations it has applied so far (in the order they have been applied at that site). This list is called a History Buffer. For example:

C1 C2 C3 A1 C4 C5 A2 A3 C6 A4 A5

Site A

History Buffers

C5 C3 B1 C2 C4 C1 B2 C6 B3

Site B

Site A has a mixture of the Ai and Ci. Site B has a mixture of the Bi and Ci.

Note that the common operations (the Ci) can be jumbled up with the Ai and Bi. Furthermore the Ci may not be in the same relative order in these two lists.

How do we synchronise the two sites? Consider for example that operation A1 is sent to site B. The context for A1 is the state in which {C1 C2 C3} have been executed. We must somehow transform A1 to A1’ such that it can be appended to the end of site B’s history buffer.

C1 C2 C3 A1 C4 C5 A2 A3 C6 A4 A5

Site A

C5 C3 B1 C2 C4 C1 B2 C6 B3 A1'

Site B

## Factorise

The history buffer of A has a mixture of C’s and A’s. We transpose adjacent pairs of Ai,Cj operations (i.e. where the Ai is to the left of the Cj) so all the C’s form a prefix and the A’s a suffix. This is possible because Ai || Cj. We can’t have Ai→Cj because that contradicts C preserving causality, and we can’t have Cj→Ai because Ai is to the left of Cj and the total order of the operations in a history buffer always respects the precedes partial order.

We call this a factorisation of the history buffer of site A with respect to a vector time V(C) to give C⊙A.

C1 C2 C3 A1 C4 C5 A2 A3 C6 A4 A5

Site A

C1 C2 C3 C4 A1 C5 A2 A3 C6 A4 A5

C1 C2 C3 C4 C5 A1 A2 A3 C6 A4 A5

C1 C2 C3 C4 C5 A1 A2 C6 A3 A4 A5

C1 C2 C3 C4 C5 A1 C6 A2 A3 A4 A5

C1 C2 C3 C4 C5 C6 A1 A2 A3 A4 A5

transpose

prefix C

suffix A

A1||C4

Vector time of C

C⊙A

## Synchronise two sites

In the following diagram the history buffers of both site A and site B are factorised so that the common operations are in a prefix. That means that even though the common operations had been applied in a different order they result in the same state. Therefore the suffixes (the A’s and B’s) are applied in the same context and therefore a DualIT can be performed (noting that this is possible because A || B)

C1 C2 C3 A1 C4 C5 A2 A3 C6 A4 A5

Site A

C1 C2 C3 C4 C5 C6 A1 A2 A3 A4 A5

C5 C3 B1 C2 C4 C1 B2 C6 B3

Site B

C5 C3 C2 C4 C1 C6 B1 B2 B3

Dual IT

A || B

Common operations

Factorise to CA

Factorise to CB

C⊙A⊙IT(B,A)

C⊙B⊙IT(A,B)

## But too inefficient

We have described a solution described in the literature, but it is far too inefficient to be useful in a production system, because dualIT and factorisation are quadratic algorithms.

Fortunately linear approach exist. CEDA uses algorithms that are linear in the number of operations that need to be merged. This typically allows for merging hours of works in a fraction of a second.

These linear algorithms are not described in this document.