Operational transform
Set theoretic operations

22 Sep 2008
David Barrett-Lennard

Abstract

This paper provides a solution to the operational transform of set theoretic operations.

Introduction

The document state is assumed to be a set over some element type T. T is a value type, and is assumed to support an equality comparison operator. In this paper we study the operational transform of the following set theoretic operations

- ins(k) - insert value k into the set (if not present)
- del(k) - delete value k from the set (if present)

Each of these operations are idempotent.

We need convergence at quiescence even though operations are executed in different orders at different sites. Specifically we require properties TP1 and TP2 [1].

Discussion

In the following cases, let O_1, O_2 be concurrent operations that are performed on the same document state S.

1. Let operation O_1 insert value k_1, and O_2 inserts value k_2. The set theoretic union operation is associative and commutative so therefore $(S \cup \{k_1\}) \cup \{k_2\} = (S \cup \{k_2\}) \cup \{k_1\}$. Hence operations O_1, O_2 commute and therefore nothing needs to be done under IT.

2. Let operation O_1 delete value k_1, and O_2 delete value k_2. According to set theory $(S \setminus \{k_1\}) \setminus \{k_2\} = (S \setminus \{k_2\}) \setminus \{k_1\}$. Hence operations O_1, O_2 commute and therefore nothing needs to be done under IT.

3. Let operation O_1 insert value k_1, and O_2 delete value k_2. Assuming $k_1 \neq k_2$, $(S \cup \{k_1\}) \setminus \{k_2\} = (S \setminus \{k_2\}) \cup \{k_1\}$. Hence operations O_1, O_2 commute and therefore nothing needs to be done under IT.

4. The only difficult case is where O_1 inserts value k, and O_2 deletes value k. It is proposed that we disable a delete operation when it IT's past an insert operation of the same value, so at quiescence all sites agree that the value was inserted This is implemented using a disable count, so that a delete operation may be disabled multiple times as it IT's past a number of insert operations. Under ET, the delete operation will only be re-enabled when it has ET'd backward past all the insert operations that caused it to be disabled.

Solution

Let an operation contain the following fields

Field	Description

ins	Boolean flag to indicate whether this is an insert or delete operation
d	Disable count, only relevant to a delete operation. Initialised to zero when the operation is first generated.
k	Value to be inserted or deleted

The following C++ code shows the implementation of IT/ET

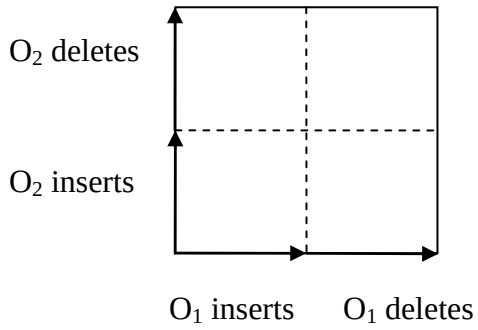
```
void IT(SetOp& O1, const SetOp& O2)
{
    if (O2.ins && !O1.ins && O1.k == O2.k) ++O1.d;
}

void ET(SetOp& O1, const SetOp& O2)
{
    if (O2.ins && !O1.ins && O1.k == O2.k) --O1.d;
}
```

Alternative that supports closure under merging

Consider that a given operation records a set of insertions followed by a set of deletions. ie its deletions are recorded in the context of having already performed the insertions.

This leads to the following picture for dual IT



The only changes occur in the top left and bottom right corner squares - when deletes transform past the other operation's inserts.

So as a group the insertions don't interact with each other under OT. This suggests that we only record the total number of the insertions (which may possibly be zero).

Similarly as a group the deletions don't interact with each other. Also all deletes in the group will have their disable count increased by the same amount (ie the number of insertions in the other operation). For the group of deletions we can assume that only the deletion with the lowest disable count is significant. All the other deletes are redundant. It isn't even necessary to record the number of deletes that share the minimum disable count. Therefore we summarise the deletes by only recording the minimum disable count.

Let an operation contain the following fields

Field	Description
k	Value to be inserted or deleted
int i	Number of insertions of value k
int d	Minimum disable count for deletions of value k.

We assume $i \geq 0$ and $d \geq 0$.

We can generate a pure insertion operation using i=1, d=1, and a pure delete operation with i=0, d=0.

More generally an operation will contain a set of (k,i,d) values for various k, in order to support insertions and deletions for each k. However to keep this exposition simple we assume an operation is associated with only a single (k,i,d). In fact we go even further and henceforth assume that all operations are inserting/deleting the same value of k.

Applying an operation

Let S + O denote the state obained by applying operation O to state S. Then

$$\begin{aligned} O.k \in (S+O) \quad &\Leftrightarrow \quad ((O.k \text{ was already present in } S) \ \vee \ (k \text{ was inserted by } O)) \\ &\qquad \wedge (k \text{ wasn't subsequently deleted by } O) \\ &\Leftrightarrow \quad (O.k \in S \ \vee \ O.i > 0) \ \wedge O.d > 0 \end{aligned}$$

where $\vee$ denotes logical OR, and $\wedge$ denotes logical AND.

This can be taken as a definition of how to apply an operation.

This leads to the following procedural implementation:

```
// Let S ← S + O
ApplyOperation(S,O)
{
    if (O.d > 0)
    {
        if (O.i > 0) S.insert(O.k);
    }
    else
    {
        S.delete(O.k);
    }
}
```

Inclusion Transform

Let $O_1 \parallel O_2$. The inclusion transformation of O_1 past O_2 is denoted by $IT(O_1,O_2)$ and satsfies:

$$\begin{aligned} IT(O_1,O_2).i &= O_1.i \\ IT(O_1,O_2).d &= O_1.d + O_2.i \end{aligned}$$

This leads to the following procedural implementation:

```
// Let O1 ← IT(O1,O2)
IT(O1,O2)
{
    O1.d += O2.i;
}
```

Exclusion Transform

Let $O_1 \parallel O_2$ be contextually serialised operations. The exclusion transformation of O_1 backwards past O_2 is denoted by $ET(O_1,O_2)$ and satsfies:

$ET(O_1,O_2).i = O_1.i$
$ET(O_1,O_2).d = O_1.d - O_2.i$

This leads to the following procedural implementation:

```
// Let O2 ← ET(O1,O2)
ET(O1,O2)
{
    O1.d -= O2.i;
}
```

Proof of convergence under IT

In the following we prove the two properties required to ensure convergence under IT, described in [1]:

TP1 $S + [O_1\ IT(O_2, O_1)] = S + [\ O_2\ IT(O_1, O_2)\ ]$

TP2 $IT(\ IT(\ O_3, O_1),\ IT(O_2, O_1)) = IT(\ IT(\ O_3, O_2),\ IT(O_1, O_2))$

Some useful facts needed for the proofs

Given integers $u \geq 0$ and $v \geq 0$, we have

$u+v > 0 \quad \Leftrightarrow \quad u > 0$ or $v > 0$
$\min(u,v) > 0 \quad \Leftrightarrow \quad u > 0$ and $v > 0$

For boolean expressions x,y, let x+y denote $x \vee y$, and xy denote $x \wedge y$.

Note the absorption law: $x + xy = x$.

Proof of TP1

RTP : (TP1) $\forall\ O_1, O_2 \quad S + [O_1\ IT(O_2, O_1)] = S + [\ O_2\ IT(O_1, O_2)\ ]$

Proof:

Let $O_1' = IT(O_1,O_2)$, and $O_2' = IT(O_2,O_1)$.

By definition of IT:

$O_1'.i = O_1.i$
$O_1'.d = O_1.d + O_2.i$
$O_2'.i = O_2.i$
$O_2'.d = O_2.d + O_1.i$

Let boolean expressions a,b,c,d,e be defined as follows

a	$k \in S$
b	$O_1.i > 0$
c	$O_1.d > 0$
d	$O_2.i > 0$
e	$O_2.d > 0$

$k \in S + O_1 + O_2' \quad \Leftrightarrow \quad (k \in S+O_1 \ \vee\ O_2'.i > 0) \ \wedge\ O_2'.d > 0$

$\Leftrightarrow$ $(((k \in S \vee O_1.i > 0) \wedge O_1.d > 0) \vee O_2.i > 0) \wedge (O_2.d + O_1.i > 0)$
$\Leftrightarrow$ $((a + b)\ c\ + d)(e + b)$
$\Leftrightarrow$ $(ac + bc\ + d)(e + b)$
$\Leftrightarrow$ $ace + bce\ + de + abc + bc\ + bd$
$\Leftrightarrow$ $ace + de + bc + bd$ (absorption)
$\Leftrightarrow$ $ace + cde\ + bc + ade + de\ + bd$
$\Leftrightarrow$ $(ae + de + b)(c + d)$
$\Leftrightarrow$ $((a + d)e\ + b)(c + d)$
$\Leftrightarrow$ $(((k \in S \vee O_2.i > 0)\ \wedge O_2.d > 0) \vee O_1.i > 0)\ \wedge (O_1.d + O_2.i > 0)$
$\Leftrightarrow$ $(k \in S+O_2\ \vee\ O_1'.i > 0)\ \wedge O_1'.d > 0$
$\Leftrightarrow$ $k \in S + O_2 + O_1'$

Proof of TP2

RTP: (TP2) $\forall\ O_1, O_2, O_3,$ $IT(\ IT(\ O_3, O_1),\ IT(O_2, O_1)) =\ IT(\ IT(\ O_3, O_2),\ IT(O_1, O_2))$

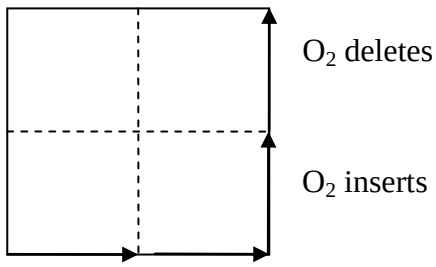
Proof:

$IT(\ IT(O_3, O_1),\ IT(O_2,O_1)).i$ = $IT(\ O_3, O_1).i$
= $O_3.i$
= $IT(\ O_3, O_2).i$
= $IT(\ IT(\ O_3, O_2),\ IT(O_1,O_2)).i$

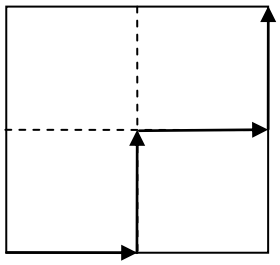
$IT(\ IT(O_3, O_1),\ IT(O_2,O_1)).d$ = $IT(\ O_3, O_1).d\ +\ IT(O_2,O_1).i$
= $O_3.d\ + O_1.i\ +\ O_2.i$
= $O_3.d\ + O_2.i\ +\ O_1.i$
= $IT(\ O_3, O_2).d\ +\ IT(O_1, O_2).i$
= $IT(\ IT(\ O_3, O_2),\ IT(O_1, O_2)).d$

Merge

Let O_1, O_2 be contextually serialised operations.



The merge of O_1,O_2 is denoted by $O_1 \oplus O_2$. It is useful to consider the transpose of the O_1 deletes with the O_2 inserts, which corresponds to the staircase pattern as follows:



This applies an offset of $O_2.i$ to $O_1.d$, and allows for the insertions to be collection together and similarly for the deletions This suggests the following definition:

$(O_1 \oplus O_2).i = O_1.i + O_2.i$
$(O_1 \oplus O_2).d = \min \{ O_1.d + O_2.i,\ O_2.d \}$

This leads to the following procedural implementation:

```
// Assuming contextually serialised operations [O1 O2], let O1 += O2
Merge(O1,O2)
{
    O1.i += O2.i;
    O1.d += O2.i;
    if (O2.d < O1.d) O1.d = O2.d;
}
```

Validation of merge algorithm

In the following sections we prove the four properties required for merge described in [2].

MP1 $(O_1 \oplus O_2) \oplus O_3 = O_1 \oplus (O_2 \oplus O_3)$

MP2 $S + [O_1\ O_2] = S + (O_1 \oplus O_2)$

MP3 $IT(O_1, O_2 \oplus O_3) = IT(O_1, [O_2, O_3])$

MP4 $IT(O_1 \oplus O_2, O_3) = IT(O_1,O_3) \oplus IT(O_2, IT(O_3, O_1))$

RTP: (MP1) $(O_1 \oplus O_2) \oplus O_3 = O_1 \oplus (O_2 \oplus O_3)$
Proof:

$$
\begin{aligned}
((O_1 \oplus O_2) \oplus O_3).i &= (O_1 \oplus O_2).i + O_3.i \\
&= O_1.i + O_2.i + O_3.i \\
&= O_1.i + (O_2 \oplus O_3).i \\
&= (O_1 \oplus (O_2 \oplus O_3)).i
\end{aligned}
$$

$$
\begin{aligned}
((O_1 \oplus O_2) \oplus O_3).d &= \min \{ (O_1 \oplus O_2).d + O_3.i,\ O_3.d \} \\
&= \min \{ \min \{ O_1.d + O_2.i,\ O_2.d \} + O_3.i,\ O_3.d \} \\
&= \min \{ O_3.d, O_2.d + O_3.i,\ O_1.d + O_2.i + O_3.i \} \\
&= \min \{ O_1.d + O_2.i + O_3.i,\ \min \{ O_2.d + O_3.i,\ O_3.d \} \}
\end{aligned}
$$

$= \quad \min \{ O_1.d + (O_2 \oplus O_3).i,\ (O_2 \oplus O_3).d \}$
$= \quad (O_1 \oplus (O_2 \oplus O_3)).d$

RTP: (MP2) $S + [O_1\ O_2] = S + (O_1 \oplus O_2)$
Proof:

Let boolean expressions a,b,c,d,e be defined as follows

a	$k \in S$
b	$O_1.i > 0$
c	$O_1.d > 0$
d	$O_2.i > 0$
e	$O_2.d > 0$

$k \in S + O_1 + O_2 \quad \Leftrightarrow \quad (k \in S + O_1 \ \vee\ O_2.i > 0) \ \wedge O_2.d > 0$
$\Leftrightarrow \quad (((k \in S \ \vee\ O_1.i > 0) \ \wedge O_1.d > 0) \ \vee\ O_2.i > 0) \ \wedge O_2.d > 0$
$\Leftrightarrow \quad ((a+b)c + d)e$
$\Leftrightarrow \quad (ac + bc + d)e$
$\Leftrightarrow \quad (ac + bc + dc + ad + bd + d)e \qquad$ (absorption)
$\Leftrightarrow \quad (a+b+d)(c+d)e$
$\Leftrightarrow \quad (k \in S \ \vee\ O_1.i > 0 \ \vee\ O_2.i > 0) \ \wedge\ (O_1.d + O_2.i > 0) \wedge\ (O_2.d > 0)$
$\Leftrightarrow \quad (k \in S \ \vee\ O_1.i + O_2.i > 0) \ \wedge\ (\min \{ O_1.d + O_2.i,\ O_2.d \} > 0)$
$\Leftrightarrow \quad (k \in S \ \vee\ (O_1 \oplus O_2).i > 0) \wedge (O_1 \oplus O_2).d > 0$
$\Leftrightarrow \quad k \ \in\ S + (O_1 \oplus O_2)$

RTP: (MP3) $IT(O_1, O_2 \oplus O_3) = IT(O_1, [O_2, O_3])$
Proof:

$IT(O_1, O_2 \oplus O_3).i \quad = \quad O_1.i$
$= \quad IT(O_1, [O_2, O_3]).i$

$IT(O_1, O_2 \oplus O_3).d \quad = \quad O_1.d + (O_2 \oplus O_3).i$
$= \quad O_1.d + O_2.i + O_3.i$
$= \quad IT(O_1, O_2).d + O_3.i$
$= \quad IT(\ IT(O_1, O_2),\ O_3).d$
$= \quad IT(O_1, [O_2, O_3]).d$

RTP: (MP4) $IT(O_1 \oplus O_2, O_3) = IT(O_1,O_3) \ \oplus\ IT(O_2, IT(O_3, O_1))$

Proof:

$IT(O_1 \oplus O_2, O_3).i \quad = \quad (O_1 \oplus O_2).i$
$= \quad O_1.i + O_2.i$
$= \quad IT(O_1,O_3).i + IT(O_2, IT(O_3, O_1)).i$
$= \quad (IT(O_1,O_3) \ \oplus\ IT(O_2, IT(O_3, O_1))).i$

$IT(O_1 \oplus O_2, O_3).d \quad = \quad (O_1 \oplus O_2).d + O_3.i$
$= \quad \min \{ O_1.d + O_2.i,\ O_2.d \} + O_3.i$
$= \quad \min \{ O_1.d + O_3.i + O_2.i,\ O_2.d + O_3.i \}$
$= \quad \min \{ O_1.d + O_3.i + O_2.i,\ O_2.d + IT(O_3, O_1).i \}$
$= \quad \min \{ IT(O_1,O_3).d + IT(O_2, IT(O_3, O_1)).i,\ IT(O_2, IT(O_3, O_1)).d \}$
$= \quad (IT(O_1,O_3) \ \oplus\ IT(O_2, IT(O_3, O_1))).d$

Future work

1. In order to support undo it is necessary to flag whether an operation actually inserted or deleted a value. This flag must undergo transformation under IT/ET

References

[1]	David Barrett-Lennard. Operational transform - Control algorithm. Aug 2005
[2]	David Barrett-Lennard. Operational transform - Merging operations Sep 2005