Operational transform
Assignment operations

Aug 2005
David Barrett-Lennard

Abstract
This paper provides a solution to the operational transform of assignment operations. There doesn't appear to be an existing solution in the literature. As it turns out, the problem is more difficult that it would first appear, and the work in [1] provides the basis of both a correct and efficient solution.

Introduction

An assignment operation sets a new value on a given attribute. It is assumed that the attribute has identity, so it is easy to tell whether two assignment operations are assigning to the same attribute. The previous value is completely overwritten - ie there is no merging of edits on the attribute. Rather it is written atomically. To support undo it is necessary for the operation to store the previous value.

We need convergence at quiescence even though operations are executed in different orders at different sites. It is necessary for all sites to agree on a unique "winner" site that in some sense dominates all other sites. It seems that a total ordering on site identifiers will serve this purpose.

The basic idea is to disable an operation when it is inclusion transformed past a site with a larger site identifier. This allows the dominant site to set the final value of the attribute despite the subsequent execution of other (disabled) operations.

Properties required to achieve convergence

TP1

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

TP2

$\forall\ O_1, O_2, O_3, \qquad IT(\ IT(\ O_3, O_1),\ IT(O_2, O_1)) = IT(\ IT(\ O_3, O_2),\ IT(O_1, O_2))$

Discussion

In this section we first review some potential solutions that at first seem like they should work, but in fact do not. Then we present a correct approach.

Consider firstly that an assignment operation stores a boolean flag for whether the operation is enabled. During IT, an assignment operation is disabled if it has a smaller site identifier. This suggests the following algorithm

```
void IT(AssignOp& O1, const AssignOp& O2)
{
    if (O1.attrib == O2.attrib && O1.id < O2.id) O1.enable = false;
}
void ET(AssignOp& O1, const AssignOp& O2)
{
    if (O1.attrib == O2.attrib && O1.id < O2.id) O1.enable = true;
}
```

Let operations O_1,O_2,O_3 be concurrent, and assign different values to the same attribute. Let $O_1.id < O_2.id < O_3.id$. This means O_3 dominates O_1 and O_2.

Let $O_3' = IT(O_3,O_2)$ which is enabled because O_3 dominates O_2. It is a requirement that ET be the inverse of IT, so therefore

O_1 = ET(ET(IT(IT(O_1,O_2), O_3'), O_3'), O_2)

Clearly the use of a simple enable flag can't allow for a correct ET to be written. It seems that a *disable count* is needed so that O_1 is re-enabled if and only if it has ET'd backward past all the operations that caused it to be disabled.

Let O_2' = IT(O_2,O_3) which is disabled because O_2 is dominated by O_3. By TP2 we require

IT(IT(O_1,O_2), O_3') = IT(IT(O_1,O_3), O_2')

We surmise that O_1 should end up with a disable count of 2 in both cases. We therefore conclude that the disable count must be incremented when ITing past an operation with a larger site id, irrespective of whether either operation is disabled or not.

This suggests the following algorithm

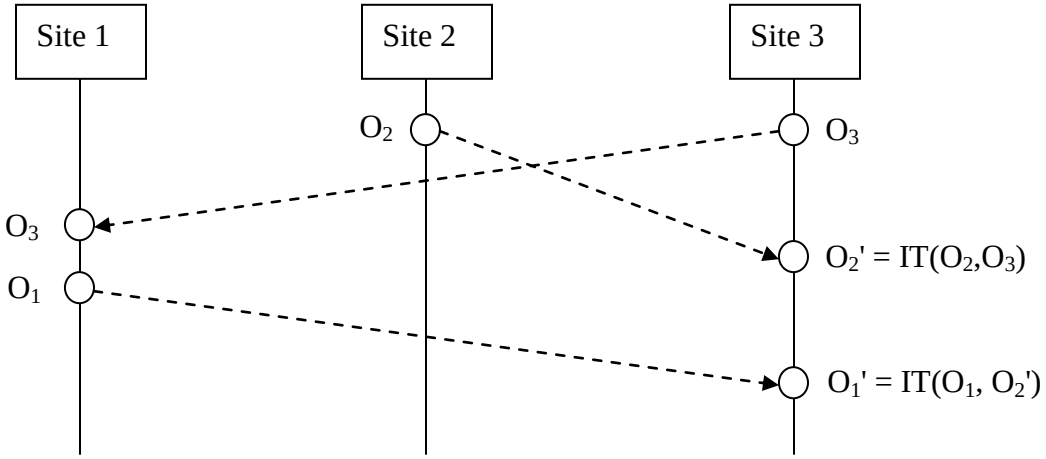
```
void IT(AssignOp& O1, const AssignOp& O2)
{
    if (O1.attrib == O2.attrib && O1.id < O2.id) ++O1.dc;
}
void ET(AssignOp& O1, const AssignOp& O2)
{
    if (O1.attrib == O2.attrib O1.id < O2.id) --O1.dc;
}
```

where field dc is a disable count initialised to zero when the operation is first generated. The operation is enabled if and only if dc is zero.

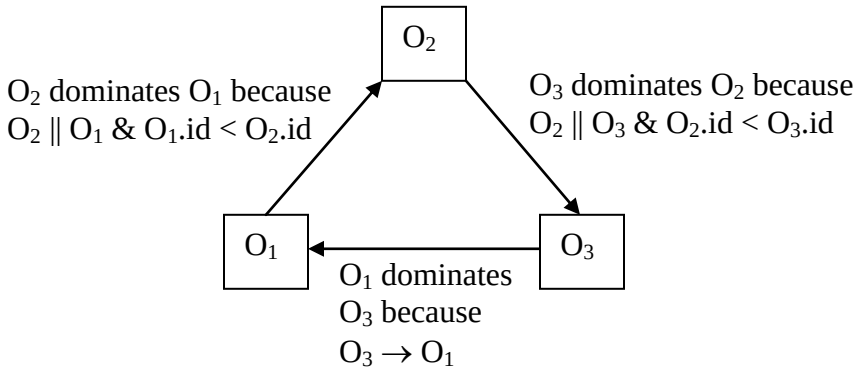
Problem with disable counts

The following example shows that the dominance relation between concurrent operations can't simply involve a comparison of site identifiers. The problem is that the dominance relation must respect causality, and therefore it is wrong to commit too early to using site identifiers to select the winner. The problem is with cycles in the dominance relation.

Consider the following example where O_1, O_2, O_3 are assignments on the same attribute



Note that $O_3 \rightarrow O_1$, $O_2 \parallel O_1$ and $O_2 \parallel O_3$. Let $O_1.id < O_2.id < O_3.id$. Then assuming we use always use site ids to select a winner for concurrent operations, we get the following cycle in the dominance relation.



This is at odds with picking a unique winner, and shows that there is a problem with the whole idea of using a disable count. The fly in the ointment is the causal relation $O_3 \rightarrow O_1$ from which we expect O_1 to always dominate O_3 even though site ids suggest otherwise. The problem is quite tricky because somehow when O_1 IT's past O_2' it should not be disabled despite its smaller site id. It needs to account for the fact that it dominates O_3 by causality, and O_3 dominates O_2 by site ids.

Solution

It seems very difficult to solve this problem! All sites need to somehow agree on a total ordering on all the assignments that have ever been performed. That way, all sites can agree (at quiescence) on the unique winner that dominates all other assignments. This way of thinking leads directly to the solution: employ a hypothetical effects document that holds every value that has ever been "assigned" (or actually inserted into the effects document). The winner is simply the left most element in the effects document (ie at index position 0). Convergence is guaranteed.

Therefore a workable solution simply involves introduction of a q-position in the assignment operation to represent the insertion position in the effects document. This is initialised to 0 when the operation is first generated so it dominates all previous assignments that have been performed on that document state. The q-position is transformed as for insertion operations into a string, as in [1]. When an assignment operation is performed it is regarded as enabled if and only if its q-position is 0.

It would be possible to define the right most character in the effects document to be the winner. However, it would be more difficult to generate an operation with the correct q-position.

Assignment operations

An assignment operation has the following fields.

Field	Description
id	The site identifier of the site that originally generated the operation. It is assumed there is a total ordering on site identifiers.
t	The sequence number assigned by the site when the operation was generated.
attrib	Identifies the attribute being assigned
nv	New value
pv	Previous value
q	The insertion position in the effects document

Algorithm for IT/ET

```
void IT(AssignOp& O1, const AssignOp& O2)
{
    if (O1.attrib == O2.attrib)
    {
        if (O2.q < O1.q || O2.q == O1.q && O2.id < O1.id) ++O1.q;
        if (O2.q == 0) O1.pv = O2.nv;
    }
}

void ET(AssignOp& O1, const AssignOp& O2)
{
    if (O1.attrib == O2.attrib)
    {
        if (O2.q < O1.q) --O1.q;
        if (O2.q == 0) O1.pv = O2.pv;
    }
}

void Do(AssignOp& O)
{
    if (O.q == 0) O.attrib = nv;
}

void Undo(AssignOp& O)
{
    if (O.q == 0) O.attrib = pv;
}
```

References

[1]	David Barrett-Lennard. Operational transform - Single character insert and delete operations. Jul 2005