Operational transform
Single character move operations

Aug 2005
David Barrett-Lennard

Abstract
This paper extends the work in [3] by applying the approach to *single character move operations*. Move operations have richer semantics than insert and delete operations in the sense that move operations can easily model insertions or deletions through the use of additional text buffers. However, the converse is false! Even though a move seems to be nothing more than a deletion followed by an insertion, the semantics are different in the case where a conflict occurs - when two users concurrently try to move the same character to different places. If this is modeled as a delete and insert then the result is that the character is duplicated. This is often undesirable, particularly when the operational transform is used for an array of objects rather than an array of characters.

For example, in an application written by the author, a jigsaw program implemented the z-order of the pieces using an ordered array of pieces, and clicking on a piece caused it to move to the top in the z-order. When this was implemented as a delete + insert it was found that pieces on the jigsaw could be duplicated when multiple users concurrently worked on the jigsaw.

Move operations have the nice feature of *conserving matter* in the system, under operational transform.

Introduction
As in [1], by definition characters in text documents have identity. A character is originally inserted by a particular user at a particular site. The character keeps its unique identity even though its index position in the document needs to be adjusted according to continued editing. Note that a character doesn't simply relate to its appearance (eg its ASCII code) - for example each appearance of the letter 'A' in a document represents a different character.

We take the convention that strings use zero based index positions. Given string s, let s[i] be the i^{th} character. s[i,j) denotes the sub-string corresponding to the half open interval [i,j).

Properties required to achieve convergence

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

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

Move operations
A move operation O = move(c,sb,sp,sq,db,dp,dq) represents a move of the single character c from document sb at p-position sp, q-position sq, to document db at p-position dp, q-position dq. A move 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.
dc	Disable count. Zero means operation is enabled
sb	Identifies the source buffer

sp	Zero based source buffer p-position
sq	Zero based source buffer q-position
db	Identifies the destination buffer
dp	Zero based destination buffer p-position
dq	Zero based destination buffer q-position
c	Character to be moved

Algorithm for IT

```
void DualIT_DaSb(Operation& a, Operation& b)
{
    if (a.db == b.sb)
    {
        if (a.dq <= b.sq)
        {
            if (a.Enabled()) ++b.sp;
            ++b.sq;
        }
        else
        {
            if (b.Enabled()) --a.dp;
        }
    }
}

void DualIT(Operation& a, Operation& b)
{
    if (a.sb == b.sb)
    {
        if (a.sq < b.sq)
        {
            if (a.Enabled()) --b.sp;
        }
        else if (b.sq < a.sq)
        {
            if (b.Enabled()) --a.sp;
        }
        else
        {
            if (a.Enabled())
            {
                b.sb = a.db;
                b.sq = a.dq;
                b.sp = a.dp;
            }
            else
            {
                if (a.db == b.sb)
                {
                    if (a.dq <= b.sq)
                    {
                        ++b.sq;
                    }
                }
            }

            if (b.Enabled())
            {
                a.sb = b.db;
                a.sq = b.dq;
                a.sp = b.dp;
            }
            else
            {
                if (b.db == a.sb)
                {
                    if (b.dq <= a.sq)
                    {
                        ++a.sq;
                    }
                }
            }

            if (a.db == b.db)
```

```
            {
                if (a.dq < b.dq || a.dq == b.dq && a.id < b.id)
                {
                    ++b.dq;
                }
                else
                {
                    ++a.dq;
                }
            }

            if (b.id < a.id) ++b.dc; else ++a.dc;

            return;
        }
    }

    DualIT_DaSb(a,b);
    DualIT_DaSb(b,a);

    if (a.db == b.db)
    {
        if (a.dq < b.dq || a.dq == b.dq && a.id < b.id)
        {
            if (a.Enabled()) ++b.dp;
            ++b.dq;
        }
        else
        {
            if (b.Enabled()) ++a.dp;
            ++a.dq;
        }
    }
}
```

Transforming a list with a list

Let L_1, L_2 be lists of operations. It can be shown that the following algorithm can be used to IT each list past the other.

```
void DualIT(L1, L2)
{
    for (int i = 0 ; i < |L1| ; ++i)
        for (int j=0 ; j < |L2| ; ++j)
            DualIT(L1[i], L2[j])
}
```

This is an in-place algorithm. Note that each L_1[i] is transformed against each L_2[j]. This can't be done in any order - it corresponds to moving through the array of cells in an ordered fashion.

Convergence of the effects document

In this section we demonstrate that the above IT algorithm leads to convergence of the effects document at all sites at quiescence.

For the most part, a move can be treated as an extraction followed by an insertion. We write this as $M = [X\ I]$. Therefore the problem of transforming a move $M_1 = [X_1\ I_1]$ against a move $M_2 = [X_2\ I_2]$ is usually equivalent to transforming a pair of operations with a pair of operations. The basic strategy therefore is to perform the following four steps

1. Dual IT X_1 and X_2
2. Dual IT X_1 and I_2
3. Dual IT X_2 and I_1
4. Dual IT I_1 and I_2

Step 1 must be done first, Steps 2,3 can be done in either order. Step 4 must be done last. These are all in-place transformations.

This assumes that M_1 and M_2 do not conflict - ie attempt to move the same character. In that case a different strategy is require to transform them.

Dual IT X1, X2
The following algorithm is used to dual IT X_1 and X_2.

```
void DualIT_XX(Operation& a, Operation& b)
{
    if (a.sb == b.sb)
    {
        if (a.sq < b.sq)
        {
            if (a.Enabled()) --b.sp;
        }
        else if (b.sq < a.sq)
        {
            if (b.Enabled()) --a.sp;
        }
        else
        {
            < conflict >
        }
    }
}
```

First a check is made for whether the operations extract characters from the same source buffer. If not then the extractions can't interact. It also shows that the operations do not conflict.

Otherwise the operations are extracting characters from the same source buffer, and therefore it is necessary to compare the extraction q-positions. If the q-positions are identical then the operations must be conflicting (ie attempting to move the same source character). This scenario is discussed later in this paper.

Assuming the operations do not conflict, we use the algorithm from [3] for IT of delete with delete. According to [3], we never adjust the q-position because of a delete. However, the p-position needs to be decremented if the operations is enabled and the q-position is on the left.

DualIT X, I
The following algorithm is used to transform an insertion I with an extraction X.

```
void DualIT_IX(Operation& a, Operation& b)
{
    if (a.db == b.sb)
    {
        if (a.dq <= b.sq)
        {
            if (a.Enabled()) ++b.sp;
            ++b.sq;
        }
        else
        {
            if (b.Enabled()) --a.dp;
        }
    }
}
```

Note that if the extraction and insertion are in different text buffers then they don't interact.

According to [3], I is on the left of X if and only if $I.q \leq X.q$. Assuming I is enabled and I is on the left of X, it is necessary to increment both X.p and X.q. Conversely, assuming X is enabled and X is on the left of I, it is necessary to decrement I.p. I.q must not be decremented.

This approach is taken in the above algorithm, apart from one rather strange caveat : it is necessary to increment X.q when the insertion is on the left, regardless of whether I is enabled or not! It turns out that we regard a disabled move operation as always performing an insertion in the effects document. The reason for this is discussed in the section regarding conflicting operations.

DualIT I, I

The following algorithm is used to dual IT insertions I_1 and I_2.

```
void DualIT_II(Operation& a, Operation& b)
{
    if (a.db == b.db)
    {
        if (a.dq < b.dq || a.dq == b.dq && a.id < b.id)
        {
            if (a.Enabled()) ++b.dp;
            ++b.dq;
        }
        else
        {
            if (b.Enabled()) ++a.dp;
            ++a.dq;
        }
    }
}
```

Note that if the insertions are in different text buffers then they don't interact.

According to [3], we test q positions to see whether I_1 is on the left of I_2, and if q-positions tie then we use site identifiers to break the tie. Assuming I_1 is on the left of I_2, and I_1 is enabled then it is necessary to increment both I_2.p and I_2.q. The condition for when to increment I_1.p, I_2.q is symmetrical.

This approach is taken in the above algorithm, apart from one rather strange caveat : it is necessary to increment the q position of an insertion when the other insertion is on the left, regardless of whether the insertion on the left is enabled or not! It turns out that we regard a disabled move operation as always performing an insertion in the effects document. The reason for this is discussed in the section regarding conflicting operations.

Conflicting move operations

When performing dual IT of extractions X_1, X_2 it may be found that the operations are trying to move the same character. This is a conflict that must be handled in a special way.

The strategy is to somewhat arbitrarily pick a winner and a loser based on the ordering of site identifiers. We choose the operation with the larger site id to be the winner. In other words, at quiescence it will be found that the winning move operation has taken effect, and it's as though the loser operation had never occurred.

It is important to consider the effects document, and in particular we require convergence of the effects document at all sites at quiescence. Otherwise we can't expect q-positions to remain a reliable judge of the effects relation.

Consider the site where the loser operation is applied first. The effect is that the character has been moved to the "wrong" place - possibly even to the wrong text buffer. To fix this, we transform the winner operation so that it tracks the character to where it went, and moves the character from that source position instead. In other words we simply assign the loser's destination position to the winner's source position.

Conversely, consider that the winner operation went first. Clearly the loser operation needs to be disabled because the character is already in the right position. To support this each operation has a disable count. This is incremented for the loser operation. Then when the loser is executed, no character will be moved. This should provide convergence at all sites.

Now consider what happens in the effects document : Here we see the character that was inserted by the loser operation! Therefore a disabled loser operation is in fact not disabled from the point of view of the effects document! The result is that a disabled operation may cause q-positions of other operations to be shifted to the right as they are IT'd against it.

TODO: Explain whether disable count is actually need or else it is sufficient to simply use a boolean flag.

TODO: Explain why tracking is only performed when the other operation is enabled. If not then it is treated as nothing more than an insertion in the effects document. This may require adjustment of the source q-position.

TODO: Explain why it is necessary to perform something similar to Dual IT of the insertion against the insertion, in order to adjust destination q-positions. This even includes use of site ids to break a tie.

References

[1]	Du Li and Rui Li. Ensuring consistency in real-time group editors. *ACM Transactions on Computer-Human Interaction*, April 2004. Under review.
[2]	Du Li and Rui Li. An Operational Transformation Algorithm and Performance Evaluation. *Journal of CSCW*, July 2005. Under review.
[3]	David Barrett-Lennard. Operational transform - Single character insert and delete operations.