Operational transform
Single character insertion and deletion operations

26 July 2005
David Barrett-Lennard

Abstract
This paper describes a new technique for operational transform of single character insertion and deletion operations on a text document that solves the ERV puzzles and TP2 convergence problems in a far simpler, more efficient and elegant fashion than the SDT [1] or SDTO [2] algorithms.

Like the SDT algorithm, the IT and ET is based on the *effects relation* which is a total ordering on all characters inserted into the text document. The paper provides an easy way to calculate the effects relation.

Introduction
For state S and operation O, S' = S+O denotes the state obtained after executing O on state S. Operation O may only be executed on state S. Therefore we define $state_{in}(O) = S$.

Let id(O) denote the site identifier of the site on which O was originally generated. It is assumed there is a total ordering on site identifiers.

Two operations O_1, O_2 are *equivalent* (written $O_1 \sim O_2$) if $state_{in}(O_1) = state_{in}(O_2) = S$ and $S+O_1 = S+O_2$. Note that this doesn't imply that the operations are equal. For example, it is possible that $id(O_1) \neq id(O_2)$.

$[O_1 ... O_n]$ denotes the list of operations $O_1,...,O_n$ assumed to be contextually serialised - ie intended to be performed in the given order on some initial state S. $S + [O_1...O_n]$ denotes the state $(((S+O_1)+O_2)+...+O_n)$.

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 changes as characters are inserted or deleted. 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 and let s[i,j) denote the sub-string corresponding to the half open interval [i,j).

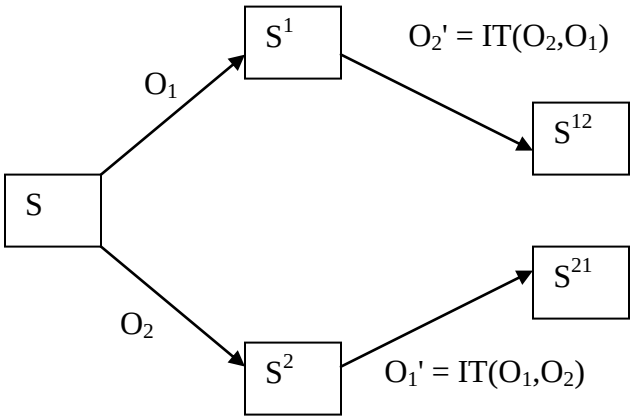
Definition of IT, ET
Let $O_1 \parallel O_2$ and $state_{in}(O_1) = state_{in}(O_2) = S$. Then we define $O_1' = IT(O_1,O_2)$ as being a transformed version of O_1 that maintains the original intention of O_1, whilst being executed in the document state following execution of O_2. ie $state_{in}(O_1') = S+O_2$. Therefore $[O_2\ IT(O_1,O_2]$ is contextually serialised.

ET is defined to be the inverse of IT. ie whenever $O_1' = IT(O_1,O_2)$ is defined, we define $ET(O_1', O_2) = O_1$. Note that ET is only defined on contextually serialised concurrent operations

It has been shown in [3] that if IT satisfies conditions TP1, TP2 (defined below) then sites are able to execute operations in different orders yet achieve convergence at quiescence.

TP1
$\forall\ O_1, O_2$ where $O_1 \parallel O_2$ and $state_{in}(O_1) = state_{in}(O_2) = S$, $S + [O_1\ IT(O_2, O_1)] = S + [\ O_2\ IT(O_1, O_2)\ ]$

In the following diagram we require $S^{12} = S^{21}$.



TP2

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

System 1 : Insert operations

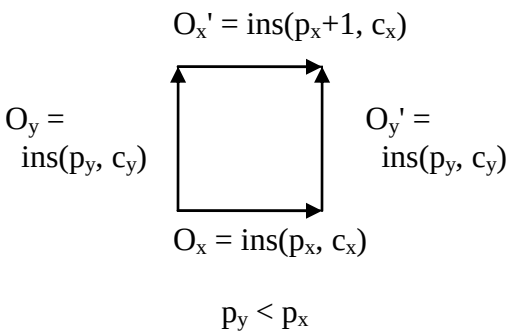
In system 1 we limit ourselves to insert operations that apply to a single text document. Operation O = ins(p,c) inserts character c at position p.

Let each operation contain the following fields

Field	Description
id	The site identifier of the site that originally generated the operation.
p	Zero based insertion position
c	Character to be inserted

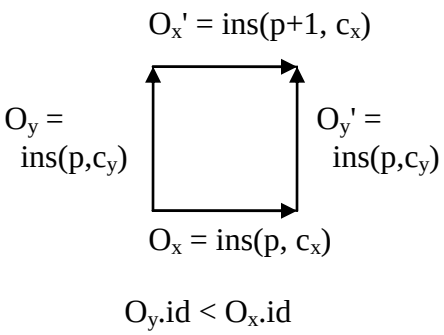
The following diagram shows two cases of transforming operations O_x, O_y against each other. For each cell the initial state is in the bottom left corner and the final state is in the top right corner. There are two paths from the initial state to the final state - either along the bottom and right edges, or along the left and top edges.

Cell 1



$p_y < p_x$

Cell 2



$O_y.id < O_x.id$

From these two cells we get four cases for $IT(O_1,O_2)$

Cell	Binding for O_1	Binding for O_2	Criteria	Action
1	O_x	O_y	$O_2.p < O_1.p$	++ $O_1.p$
1	O_y	O_x	$O_1.p < O_2.p$	-
2	O_x	O_y	$O_2.p = O_1.p \wedge O_2.id < O_1.id$	++$O_1.p$
2	O_y	O_x	$O_2.p = O_1.p \wedge O_1.id < O_2.id$	-

The criteria for these four cases are mutually exclusive and cover all possible cases. Therefore the above cells lead to the following well-defined algorithm for IT.

```
// Assumes O1 || O2
IT(Operation& O1, const Operation& O2)
{
    if (O2.p < O1.p || O2.p == O1.p && O2.id < O1.id) ++O1.p;
}
```

From the two cells we get four cases for $ET(O_1,O_2)$

Cell	Binding for O_1	Binding for O_2	Criteria	Action
1	O_x'	O_y	$O_1.p > O_2.p + 1$	-- $O_1.p$
1	O_y'	O_x	$O_1.p < O_2.p$	-
2	O_x'	O_y	$O_1.p = O_2.p + 1 \wedge O_2.id < O_1.id$	--$O_1.p$
2	O_y'	O_x	$O_1.p = O_2.p \wedge O_1.id < O_2.id$	-

These four criteria are mutually exclusive, but do not cover all cases. The missing cases are

- $O_1.p = O_2.p \ \wedge \ O_2.id < O_1.id$
- $O_1.p = O_2.p + 1 \ \wedge \ O_1.id < O_2.id$

It would seem that these cases can only occur when there is a causal dependence between O_1, O_2. Therefore we argue that this "incompleteness" is acceptable as long as we only ET concurrent operations. Therefore we have the following algorithm for ET

```
// Assumes O1 || O2
ET(Operation& O1, const Operation& O2)
{
    if (O2.p < O1.p) --O1.p;
}
```

We may return to this incompleteness issue in a future paper that addresses the support for *selective undo* - because that requires transpose of causally dependent operations.

Proof of convergence:
TP1: Let the initial document state be S. We want to show that

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

Let $|S| = n$ and S_i denote the i^{th} character in S. Let $O_1 = ins(p_1,c_1)$. $O_2 = ins(p_2,c_2)$. O_1, O_2 both execute on state S. Let $O_1' = IT(O_1,O_2)$ and $O_2' = IT(O_2,O_1)$.

Let $S^1 = S + [O_1] = [S_0\ S_1\ ...\ \ S_{p1-1}\ \ c_1\ \ S_{p1+1}\ ...\ S_{n-1}\ ]$
and $S^2 = S + [O_2] = [S_0\ S_1\ ...\ \ S_{p2-1}\ \ c_2\ \ S_{p2+1}\ ...\ S_{n-1}\ ]$

There are four cases to consider

$p_1 < p_2$	$O_1' = ins(p_1,c_1)$. $O_2' = ins(p_2+1,c_2)$ $S + [O_1\ O_2'] = S^1 + [O_2']$ $= [S_0\ S_1 \ldots\ S_{p1-1}\ c_1\ S_{p1+1} \ldots S_{p2-1}\ c_2\ S_{p2+1}\ \ldots.\ S_{n-1}\ ]$ $= S^2 + [O_1']$ $= S + [O_2\ O_1']$
$p_2 < p_1$	Symmetrical to case of $p_1 < p_2$
$p_1 = p_2$ & $O_1.id < O_2.id$	$O_1' = ins(p_1,c_1)$. $O_2' = ins(p_2+1,c_2)$ $S + [O_1\ O_2'] = S^1 + [O_2']$ $= [S_0\ S_1 \ldots\ S_{p1-1}\ c_1\ c_2\ S_{p1+1} \ldots S_{n-1}\ ]$ $= S^2 + [O_1']$ $= S + [O_2\ O_1']$
$p_1 = p_2$ & $O_2.id < O_1.id$	Symmetrical to case of $p_1 = p_2$ & $O_1.id < O_2.id$

Note that the relative order of characters is preserved under IT. This relates to the requirement of user intention preservation.

TP2:
Need to show $\forall\ O_1, O_2, O_3$, $IT(\ IT(\ O_3, O_1),\ IT(O_2, O_1)) = IT(\ IT(\ O_3, O_2),\ IT(O_1, O_2))$

Continuing the conventions in the proof of TP1, let $O_3 = ins(p_3,c_3)$.

Suppose $p_1 < p_2$. Then there are seven cases to consider. The following table shows the shifts (ie Δp_3) applied to p_3 as O_3 ITs first past O_1 then $O_2' = ins(p_2',c_2)$, or alternatively first past O_2 then $O_1' = ins(p_1',c_1)$. Note that $p_2' = p_2+1$, and $p_1' = p_1$.

	IT past O_1		**IT past O_2'**		**IT past O_2**		**IT past O_1'**	
	p_3 to p_1	Δp_3	p_3' to p_2'	Δp_3	p_3 to p_2	Δp_3	p_3" to p_1'	Δp_3
$p_3 < p_1$	<	0	<	0	<	0	<	0
$p_3 = p_1$ & $O_3.id < O_1.id$	=	0	<	0	<	0	=	0
$p_3 = p_1$ & $O_1.id < O_3.id$	=	+1	<	0	<	0	=	+1
$p_1 < p_3 < p_2$	>	+1	<	0	<	0	>	+1
$p_3 = p_2$ & $O_3.id < O_2.id$	>	+1	=	0	=	0	>	+1
$p_3 = p_2$ & $O_2.id < O_3.id$	>	+1	=	+1	=	+1	>	+1
$p_3 > p_2$	>	+1	>	+1	>	+1	>	+1

There is a consistency in the relationship between p_1,p_3 (irrespective of whether O_3 ITs past O_1 first or O_2 first): In the first five rows, when O_3 ITs first past O_2, p_3 is not changed ($\Delta p_3 = 0$). Also $p_1' = p_1$ so the relationship between p_1,p_3 is preserved. In the last two rows, when O_3 ITs past O_2, p_3 is incremented. This still means $p_3 > p_1$ so again the relationship between p_1,p_3 is preserved.

There is a consistency in the relationship between p_2,p_3 (irrespective of whether O_3 ITs past O_1 first or O_2 first): In the first two rows, when O_3 first ITs past O_1, p_3 is not changed ($\Delta p_3 = 0$). In these cases $p_3 < p_2$ and also $p_3 < p_2' = p_2+1$, so the relationship between p_2, p_3 is preserved. In the last five rows, when O_3 ITs past O_1, p_3 is incremented. However O_2' also has its position p_2 incremented, so the relationship between p_2, p_3 is preserved.

In all cases the total shift applied to p_3 is the same, and therefore condition TP2 is proven. By symmetry we see that TP2 is also proven when $p_2 < p_1$.

Let $p = p_1 = p_2$, WLOG assume $O_1.id < O_2.id$. Therefore $O_1' = ins(p_1,c_1)$ and $O_2' = ins(p_2+1,c_2)$ There are five cases to consider. The following table shows the shifts (ie Δp_3) applied to p_3 as O_3 ITs past O_1 then O_2', or alternatively O_2 then O_1'. In all cases the total shift is the same, and therefore condition TP2 is proven.

	IT past O_1		**IT past O_2'**		**IT past O_2**		**IT past O_1'**	
	p_3 to p_1	Δp_3	p_3 to p_2'	Δp_3	p_3 to p_2	Δp_3	p_3 to p_1'	Δp_3
$p_3 < p$	<	0	<	0	<	0	<	0
$p_3 = p$ & $O_3.id < O_1.id$	=	0	<	0	=	0	=	0
$p_3 = p$ & $O_1.id < O_3.id$ & $O_3.id < O_2.id$	=	0	=	+1	=	+1	>	0
$p_3 = p$ & $O_2.id < O_3.id$	=	+1	>	+1	=	+1	>	+1
$p_3 > p$	>	+1	>	+1	>	+1	>	+1

Alternative proof of convergence: At quiescence all sites have executed all the insert operations. Therefore all sites agree on the set of characters that have been inserted. Therefore to show convergence, it is sufficient to show that all sites agree on a total ordering of the characters.

Consider that we break up the text document into groups of characters, where characters within a group were tied for insertion position during IT (and therefore site ids were used to break the ties). It is claimed that all the following are true

1. All sites agree on the assignment of characters to these groups
2. There is agreement by all sites on a total ordering of these groups.
3. Within a group, the characters are ordered according to the total ordering on the siteids

Cell 1 relates to the division of the characters into the groups, and cell 2 to the ordering of operations within a group.

It follows that all sites agree on a total ordering of the inserted characters.

System 2 : Insert and track operations

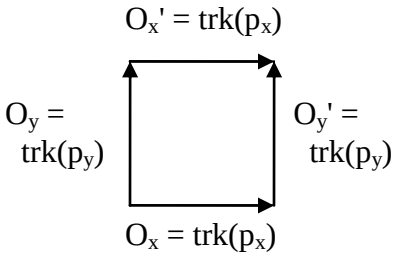
System 2 extends System 1 by allowing for "track" operations, in addition to insert operations. ie System 2 supports the following two operations:-

- ins(p, c) - insert character c at position p
- trk(p) - track character at position p

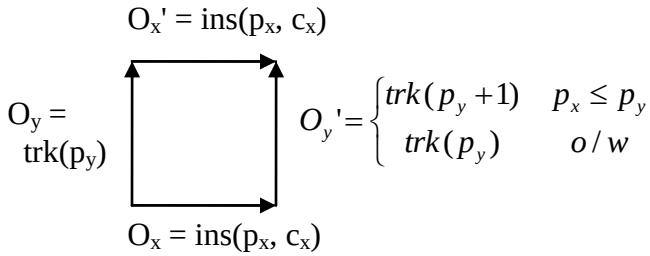
The "track" operation doesn't have any side effects. Its only purpose is to track the location of a character.

The solution is expressed with the following two additional cells

Cell 3



Cell 4



Cell 3 says that tracking operations have no effect on each other. Cell 4 says we need to increment the position of a tracked character when there is an insertion at the same position or to the left.

From these additional cells we can derive the IT and ET algorithms as follows.

```
IT(Operation& O1,const Operation& O2)
{
    if (O2.ins)
    {
        if (O2.p < O1.p || O2.p == O1.p && (O1.trk || O1.id < O2.id)
        {
            ++O1.p;
        }
    }
}

ET(Operation& O1,const Operation& O2)
{
    if (O2.ins && O2.p < O1.p)
    {
        --O1.p;
    }
}
```

Note that ET assumes $O_1 \parallel O_2$, so it never sees the case of $t(O_1) = trk$, $t(O_2) = ins$ and $O_1.p = O_2.p$.

Lemma: Under IT, a tracking operation correctly tracks the position of a character.

Proof: Characters are never deleted, so it makes sense to be able to always track the location of a given character within the document.

Let the given tracking operation be O_1. It is assumed that when O_1 was originally generated, $O_1.p$ was correctly initialised to the location of the character to be tracked.

If O_2 is an insert then $O_1.p$ will be shifted to the right if and only if $O_2.p \le O_1.p$. This indeed ensures that O_1 correctly tracks its character.

Otherwise, if O_2 is a tracking operation then $O_1.p$ is not adjusted, so again O_1 correctly tracks its character.

Proof of convergence: The tracking operations have no effect on the document state; therefore convergence is achieved because we continue to use cells 1,2.

System 3 : Insert and delete operations
System 3 extends System 1 by allowing for single character delete operations, in addition to single character insert operations. ie System 3 supports the following two operations:-

- ins(p, c) - insert character c at position p
- del(p) - delete character at position p

A correct solution to IT and ET is surprisingly difficult and there have been many attempts in the last 15 years by researchers that have failed to correctly provide both properties TP1 and TP2. Some researchers have avoided the problem by employing a control algorithm that doesn't require TP2 for convergence. However, [1] points out that it is necessary to address ERV puzzles to preserve user intention correctly, and only the solutions in [1] and [2] have addressed the ERV puzzles.

The solutions that appear in [1] (SDT) and [2] (SDTO) appear correct but are quite complex and this paper describes a far simpler solution. Importantly the proposed approach leads to a far more economical implementation, and doesn't require a change to the control algorithm as required by both [1] and [2].

Note firstly that (in System 1 or 2) we have a working solution that satisfies both TP1 and TP2 when we limit ourselves to insertion operations only. It seems that delete operations are the "fly in the ointment". This leads to the following central idea: to imagine that all delete operations on the document have been disabled! The result would be that the document contains all characters that have ever been inserted into the document. The useful thing here is that we get a total ordering on all characters. This forms the basis of correctly determining the effects relation between any two operations.

Definition: Let the *Effects Document* refer to the (hypothetical) document obtained at a given site assuming all delete operations have been disabled so that the document contains all characters that have ever been inserted by operations.

The Effects Document is only a theoretical tool - it is not actually stored. For each operation we keep track of both the position p in the normal document (in which deletes are allowed to take effect) as well as the position q in the Effects Document (in which all delete operations are ignored). The effects relation is then as simple as comparing q positions instead of p positions. The "false ties" according to [1] are precisely the cases where p positions are tied but q positions are not.

So by comparing q positions, we correctly determine when one operation is to the left of another, avoiding the problem of prematurely resorting to site ids to break the tie. This is used to update both the p and q positions as required. Note that because delete operations never occur on the Effects Document, a q position is never shifted to the left.

Let each operation contain 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.
ins	Boolean flag where true indicates an insertion operation and false indicates a

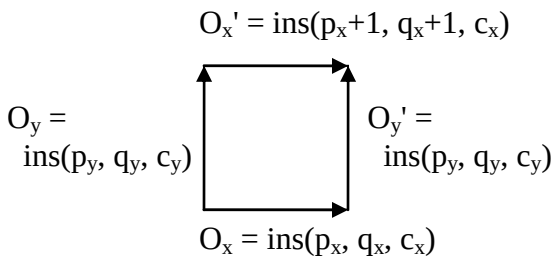
	delete operation.
enabled	Boolean flag for whether the operation is enabled or disabled. Note that only delete operations can be disabled.
p	Zero based position in the real document (in which deletes are allowed)
q	Zero based position in the Effects document (in which deletes are always disabled)
c	Character to be inserted or deleted

Let ins(p,q,c) represent the operation to insert character c at the given p-position and q-position. Let del(p,q) represent the operation to delete the character at the given p-position and q-position. Let ~del(p,q) represent a disabled deletion operation. Insertion operations cannot be disabled.

Analysis using cells
The following sections provide an analysis using our "cells".

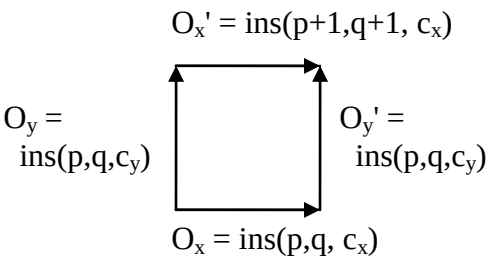
Insert with insert

Cell 5



$q_y < q_x$

Cell 6

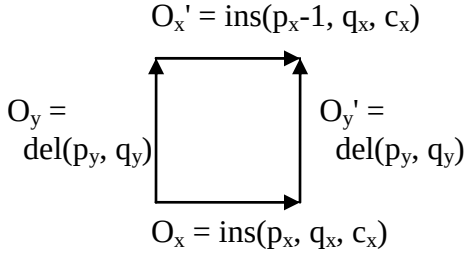


$O_y.id < O_x.id$

Cells 5,6 account for transforming insertion operations with insertion operations, and are analogous to cells 1,2 described previously. Note that only the q-position is tested, while both the p-position and q-position are shifted.

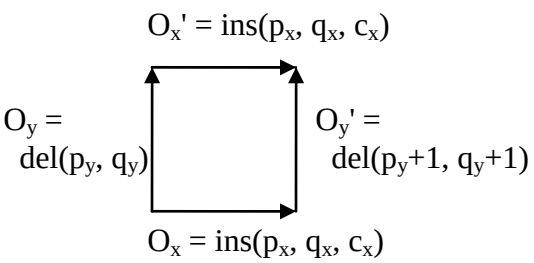
Delete with insert

Cell 7



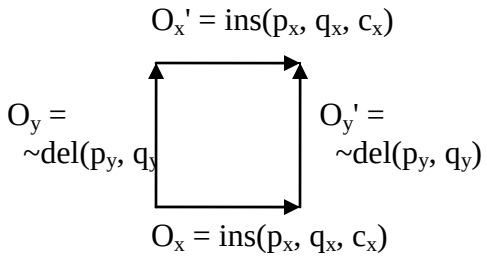
$q_y < q_x$

Cell 8



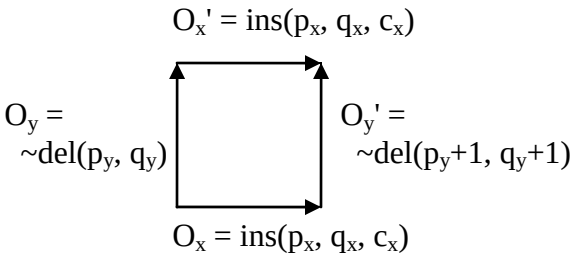
$q_x \leq q_y$

Cell 9



$q_y < q_x$

Cell 10



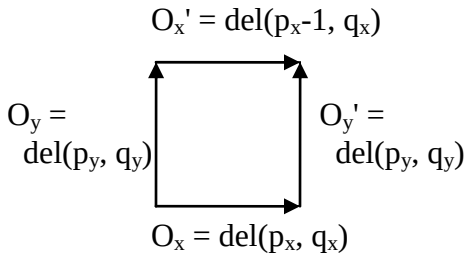
$q_x \leq q_y$

In cell 7 we have a delete on the left of an insert, causing the insertion position to be shift to the left. Note that the p-position is adjusted, but not the q-position.

In cell 8 we have an insert on the left of a delete, causing the deletion position to be shifted to the right. Note that both the p-position and q-position are shifted.

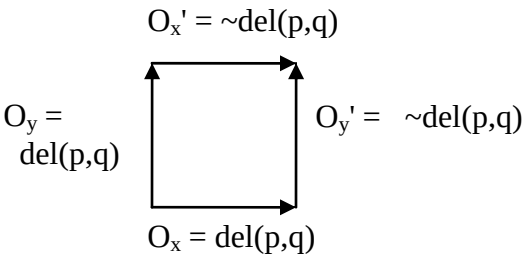
Delete with delete

Cell 11



$q_y < q_x$

Cell 12



In cell 11 we have a delete on the left of an delete, causing the deletion on the right to shift to the left. Note that the p-value is adjusted, but not the q-value.

In cell 12 we have two delete operations that delete the same character. Both of the operations are disabled after transformation.

TODO: Need cells for delete against delete when one or both of the operations are already disabled.

Algorithm

The following tables provide the implementations of IT, ET (in C++) based on the type of operations O_1 and O_2.

O_1	O_2	$IT(O_1,O_2)$

ins	ins	`if (O2.q < O1.q || O2.q == O1.q && O2.id < O1.id) { ++O1.q; ++O1.p; }`
del	ins	`if (O2.q <= O1.q) { ++O1.q; ++O1.p; }`
ins	del	`if (O2.enabled && O2.q < O1.q) --O1.p;`
del	del	`if (O2.enabled && O2.q < O1.q) --O1.p;` `if (O2.enabled && O2.q == O1.q) O1.enabled = false;`

O_1	O_2	$ET(O_1,O_2)$
ins	ins	`if (O2.q < O1.q) { --O1.q; --O1.p; }`
del	ins	`if (O2.q < O1.q) { --O1.q; --O1.p; }`
ins	del	`if (O2.enabled && O2.q < O1.q) ++O1.p;`
del	del	`if (O2.enabled && O2.q < O1.q) ++O1.p;` `if (O2.enabled && O2.q == O1.q) O1.enabled = true;`

Comments on use of enable flag

This algorithm makes use of an enable flag to avoid deleting a character twice. Note that the first operation in a linear history buffer to delete the character is enabled, and all subsequent delete operations are disabled. Therefore, sites may disagree about which operation actually deleted the character! This is reasonable because we are only interested in convergence of the final document state.

Alternative exposition

Definition 1: For operation O, let shiftp(O) and shiftq(O) be defined as follows

$$shiftp(O) = \begin{cases} +1 & if \quad O = ins(p,q,c) \\ 0 & if \quad O =\sim del(p,q) \\ -1 & if \quad O = del(p,q) \end{cases}$$

$$shiftq(O) = \begin{cases} +1 & if \quad O = ins(p,q,c) \\ 0 & if \quad O =\sim del(p,q) \\ 0 & if \quad O = del(p,q) \end{cases}$$

Definition 2: For operations O_1, O_2, let $O_1 < O_2$ (meaning O_1 is on the left of O_2) be defined as follows

$O_1 < O_2 \;=\; (O_1.q < O_2.q) \;\vee\; (O_1.q = O_2.q \;\wedge\; O_1.ins \;\wedge\; (O_2.del \vee O_1.id < O_2.id))$

Algorithm

```
IT(O1,O2)
{
    if (O2.enabled && !O1.ins && !O2.ins && O1.q == O2.q && O1.enabled)
    {
        O1.enabled = false;
    }
    else if (O2 < O1)
    {
        O1.q += shiftq(O2)
        O1.p += shiftp(O2)
    }
}

ET(O1,O2)
{
    if (O2.enabled && !O1.ins && !O2.ins && O1.q == O2.q && !O1.enabled)
    {
        O1.enabled = true;
    }
```

```
    else if (O2 < O1)
    {
        O1.q -= shiftq(O2)
        O1.p -= shiftp(O2)
    }
}
```

Proof of correctness

There is a useful mapping from system 3 to system 2, where delete operations are mapped to tracking operations, and the effects document is mapped to a document in which delete operations are never performed. This immediately gives us two useful results - that convergence of the effects document is achieved, and delete operations correctly track the location of the character within the effects document.

Lemma 1: $ET(IT(O_1,O_2), O_2) = O_1$

Proof: There are three cases to consider

1. If O_1,O_2 are both enabled delete operations at the same q-position then O1 is disabled by IT, then re-enabled by ET.
2. Otherwise, suppose $O_2 < O_1$. Shifts are applied to $O_1.p$ and $O_1.q$. Now $shiftq(O_2) \geq 0$, therefore $O_1.q$ can only be shifted further to the right. Therefore $O_2 < O_1$ must continue to hold after IT. Therefore under ET the shifts that were applied to $O_1.p$, $O_1.q$ will be reversed.
3. Otherwise not $O_2 < O_1$. Then under IT O_1 is not modified. Similarly under ET O_1 is again not modified.

In all cases we have established the result.

Note also, that ET doesn't upset the order relation between operations either. Proof?

Lemma 2: Assuming operations on a single text document, the **<** relation on operations is anti-symmetric. ie $O_1 < O_2 \Rightarrow \neg (O_2 < O_1)$

Proof: Suppose not ie $O_1 < O_2$ and $O_2 < O_1$. Now $O_1.q < O_2.q$ contradicts $O_2 < O_1$. Similarly $O_2.q < O_1.q$ contradicts $O_1 < O_2$. Therefore $O_1.q = O_2.q$. Hence $O_1.ins \wedge (O_2.del \vee O_1.id < O_2.id)$ and also $O_2.ins \wedge (O_1.del \vee O_2.id < O_1.id)$. So $O_1.id < O_2.id$ and $O_2.id < O_1.id$. This contradicts a total ordering on the site identifiers.

Lemma 3: Assuming operations on a single text document and O_1,O_2 are not deleting then same character then < relation defines a total ordering.

Proof: Suppose not. ie $\neg (O_1 < O_2)$ and $\neg (O_1 < O_2)$. Now $O_1.q < O_2.q$ contradicts $\neg (O_1 < O_2)$. Similarly $O_2.q < O_1.q$ contradicts $\neg (O_1 < O_2)$. Therefore $O_1.q = O_2.q$. So $\neg(O_1.ins \wedge (O_2.del \vee O_1.id < O_2.id))$ and also $\neg(O_2.ins \wedge (O_1.del \vee O_2.id < O_1.id))$. Therefore $O_1.del \vee (O_2.ins \wedge O_2.id < O_2.id)$ and $O_2.del \vee (O_1.ins \wedge O_1.id < O_2.id)$. From the first condition, if O_1 is an insert then O_2 must be an insert. From the second condition if O_2 is an insert then O_1 must be an insert as well. Therefore we can't have an insert and a delete. If O_1, O_2 are both inserts then we obtain a contradiction because of the total ordering on site identifiers. If O_1, O_2 are both deletes then they must be deleting the same character because $O_1.q = O_2.q$ which contradicts assumption.

Lemma 4: Under IT, the q-position of a delete operation tracks the position of the deleted character as it appears in the effects document.

Proof: Note that characters are never deleted from the effects document, so it makes sense to be able to always track the location of a given character within the effects document.

Let the given delete operation be O_1. It is assumed that when O_1 is originally generated, $O_1.q$ is correctly initialised to the location of the character as it appears in the effects document.

Now the definition of $O_2 < O_1$ makes use of q-position which is precisely the index position in the effects document.

$$O_2 < O_1 = (O_2.q < O_1.q) \vee (O_2.q = O_1.q \wedge O_2.ins \wedge (O_1.del \vee O_2.id < O_1.id))$$
$$= (O_2.q < O_1.q) \vee (O_2.q = O_1.q \wedge O_2.ins)$$

because O_1 is a delete.

If O_2 is an insert then $O_1.q$ will be shifted to the right if and only if $O_2.q \leq O_1.q$. This indeed ensures that O_1 correctly tracks its character.

If O_2 is a delete operation (enabled or not) then $O_1.q$ is not adjusted because $shiftq(O_2) = 0$. This indeed ensures that O_1 correctly tracks its character because O_2 doesn't actually delete anything from the effects document.

Lemma 5: At quiescence, convergence of the effects document is achieved.
Proof : Only insert operations cause changes to the effects document. Furthermore, when an insert operation O_1 is transformed against a delete operation O_2, $O_1.q$ is not adjusted. Therefore the presence of delete operations has no effect on the q-positions of the insert operations.

This shows that the solution can be mapped to the simpler system where the document state is related to the effects document, document positions are related to q-positions, and there are only insert operations.

So assuming the simpler system (with only insert operations) achieves convergence, we demonstrate convergence of the effects document as required.

Lemma 6: At quiescence, convergence of the document is achieved.
Proof: This follows from lemma 5 which shows that the effects document converges at all sites, and from lemma 4 which shows that delete operations track correctly in the effects document.

Lemma 7: Characters in the document appears in the same order as the effects document

Lemma 8: IT and ET preserves the intention of the user, according to CSM defined in [1].
Proof: TODO

Generation of operations
When generating an operation it is necessary to correctly initialise both the p-position and q-position stored in the operation. To support this it is necessary to be able to map from p-position to q- position. Therefore a document needs to maintain information about where characters have been deleted. An efficient implementation can store run-length encoded information. However, for the purposes of easily understanding the requirements, consider that an array B[] of booleans flags is stored, indexed by q-position. Let B[q] be false to indicate that the q^{th} character has been deleted, otherwise B[q] = true.

To generate an operation to delete the p^{th} character, we calculate the q-position as follows

```
int Getq(int p)
{
    int n = B.size();
    int pi = 0;
    for (int i=0 ; i < n ; ++i)
    {
```

```
        if (B[i])
        {
            if (pi == p)
            {
                return i;
            }
            ++pi;
        }
    }
    return n;
}
```

When operation O = del(p,q) is executed, B[] is updated as follows

```
void UpdateBForDelete(int p,int q)
{
    B[q] = false;
}
```

When operation O = ins(p,q,c) is executed, B[] is updated as follows

```
void UpdateBForInsert(int p,int q)
{
    B.insert(B.begin() + q, true);
}
```

Note that the q-positions (not the p-positions) stored within the operation are used to update B.

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.