Achieving Convergence with Operational Transformation in Distributed Groupware Systems

Abdessamad Imine, Pascal Molli, Gérald Oster, Michaël Rusinowitch

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Achieving Convergence with Operational Transformation in Distributed Groupware Systems

Abdessamad Imine — Pascal Molli — Gérald Oster — Michaël Rusinowitch

N° 5188

Mai 2004

Thèmes SYM et COG

ISSN 0249-6399 ISRN INRIA/RR--5188--FR+ENG

Achieving Convergence with Operational Transformation in Distributed Groupware Systems

Abdessamad Imine , Pascal Molli , Gérald Oster , Michaël Rusinowitch

Thèmes SYM et COG — Systèmes symboliques et Systèmes cognitifs
Projets CASSIS et ECOO

Rapport de recherche n° 5188 — Mai 2004 — 19 pages

Abstract: Distributed groupware systems provide computer support for manipulating shared objects by dispersed users. Data replication is used in such systems in order to improve the availability of data. This potentially leads to divergent (or different) replicas. In this respect, the Operational Transformation (OT) approach is employed to maintain convergence of all replicas, *i.e.* all users view the same object. Using this approach, users can exchange their updates in any order since the convergence should be ensured in all cases. However, designing correct OT algorithms is still an open issue. In this paper, we demonstrate that recent OT algorithms are incorrect. We analyse the source of this problem and we propose a generic solution with its formal correctness.

Key-words: Distributed Systems, Optimistic replication, Convergence, Operational Transformation

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 —Télécopie : +33 3 83 27 83 19

Assurer la Convergence des Collecticiels Distribués par l'Approche Transformationnelle

Résumé : Les collecticiels distribués fournissent un support informatique pour la manipulation d'objets partagés par des utilisateurs distants. La réplication de données est utilisée dans de tels systèmes afin d'améliorer la disponibilité des données. Ceci peut potientiellement mener vers des copies divergentes (ou différentes). A cet effet, l'approche transformationnelle est employée pour maintenir la convergence de toutes les copies, *i.e.* tous les utilisateurs observent le même objet. En utilisant cette approche, les utilisateurs peuvent échanger leurs modifications dans n'importe quel ordre puisque la convergence devrait être assurée dans tous les cas. Cependant, la conception d'algorithmes transformationnels corrects reste toujours un problème ouvert. Dans ce rapport, nous montrons d'une part que des algorithmes transformationnels publiés récemment sont incorrects. D'autre part, nous analysons l'origine de ce problème et nous proposons une solution formelle et générique.

Mots-clés : Systèmes distribués, Réplication optimiste, Convergence, Approche Transformationnelle

1 Introduction

Distributed groupware systems allow a group of users to simultaneously manipulate the same object (*i.e.* a text, an image, a graphic, etc.) from physically dispersed sites (or users) that are interconnected by a supposed reliable network [2]. There are two kinds of groupwares: *synchronous* and *asynchronous* systems. In synchronous groupware, people interact with each other at the same time and the response time must be short. Group editors are example of people editing a shared document at the same time [1, 9, 13]. In asynchronous ones, users usually collaborate accessing and modifying shared information without immediate knowledge about the actions of other users (either because users work at different times or simply because they do not have access to each other's actions). Version control systems [11] and data synchronizers [8] are example where users modify a copy of the shared document at different times and have to merge later their modifications in order to obtain the same document.

Data replication is used in distributed groupware systems to improve performance and availability of data [1, 10]. In order to support mobility, offline updates, optimistic replication systems allows replica to diverge. The challenge is then to reconciliate divergent replicas. Originating from real-time groupware research [1], the *Operational Transformation* (OT) approach provides an interesting solution. Compared to optimistic replication systems [10] that only exploit commutativity of operations, OT in some way, allows to transform operations that cannot commute into new equivalent operations that commute. The OT approach consists of two main components:

1. The *integration algorithm* which is responsible of receiving, broadcasting and executing operations. It is independent of the type of replica.
2. The *transformation function* is responsible for merging two concurrent operations.

The integration algorithm calls the transformation function when needed. Correctness of OT approach relies on:

a correct integration algorithm. A lot of integration algorithms [9, 15, 12] have been delivered to the community with their correctness proofs [12, 7].

a correct transformation function. A transformation function is correct if it allows to achieve convergence. Unfortunately, all known transformation functions are incorrect. Without a correct transformation functions, OT approach is useless.

In [4], we have provided couter-examples to OT algorithms that have been there since fifteen years [1, 9, 15] and we have proposed a new OT algorithm for strings conjecturing that it is correct. Recently, Du et al. [6] propose a new solution and claim that it succeeds to achieve convergence. In this paper, we show that our previous solution [4] and the ones of [12, 6] are incorrect by exhibiting tricky counter-examples. We also analyse thoroughly the source of the failures in [4, 12, 6] and we propose an OT algorithm that is radically simplier that all previous ones. Consequently, our new solution avoids the previous pitfalls and (unlike previous works) we have been able to give completely formal proof of its correctness.

The remainder of this paper is organized as follows. We present the operational transformation model in Section 2. Section 3 analyzes convergence problems that still remain and sketches an abstract solution. Section 4 presents our contributions giving proofs of correctness and examples. Section 5 discusses related work, and section 6 summarizes conclusions.

2 Operational Transformation

OT considers n sites, where each site has a copy of the shared document. The shared document model we take is a *text document* modeled by a sequence of characters, indexed from 0 up to the number of characters in the document. It is assumed that the document state (the text) can only be modified by executing the following two primitive editing operations: (i) $Ins(p, c)$ which inserts the character c at position p; (ii) $Del(p)$ which deletes the character at position p.

It should be pointed out that the above text document model is only an abstract view of many document models based on a linear structure. For instance the character parameter may be regarded as a string of characters, a line, a block of lines, an ordered XML node, etc.

We denote $st \bullet op = st'$ when an editing operation op is executed on the document state st and produces document state st'. Notation $[op_1; op_2; \ldots; op_n]$ represents an sequence of operations. Applying an operation sequence to a document state st is recursively defined as follows:

1. $st \bullet [] = st$, where [] is the empty sequence and;
2. $st \bullet [op_1; op_2; \ldots; op_n] = (((st \bullet op_1) \bullet op_2) \ldots) \bullet op_n$.

Definition 2.1 ***(Equivalence sequence.)*** *Two operation sequences seq_1 and seq_2 are equivalent, denoted by $seq_1 \equiv seq_2$, if $st \bullet seq_1 = st \bullet seq_2$ for all document states st.*

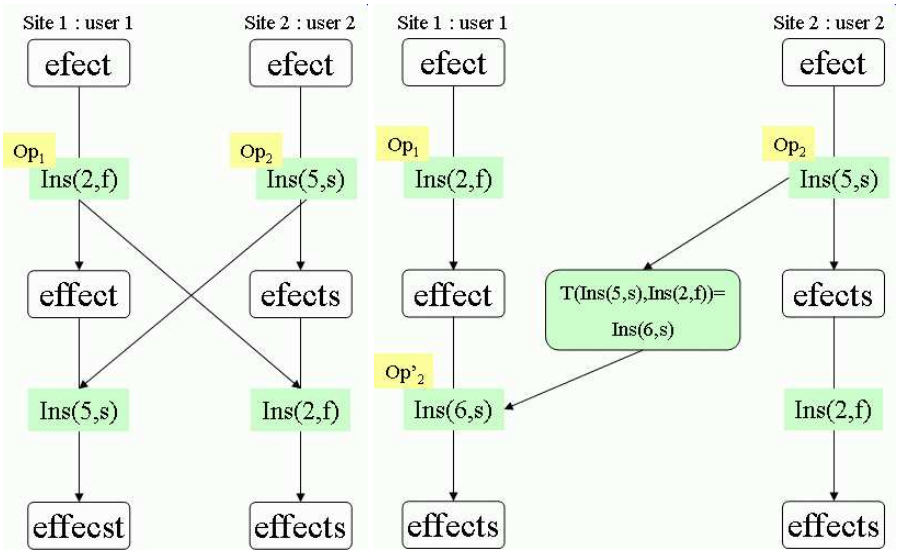
Any operation will go through the process of generation, local execution, propagation and remote execution. Each site generates editing operations sequentially and stores these operations in a data structure called *log* which gives the execution order of operations [1]. We assume that timestamp vector are used to preserve causality between operations [5].

In distributed groupware systems editing operations can be generated and executed in arbitrary orders. Local operations are always executed without being delayed but the execution of some remote operations may be delayed until they are causally ready [1, 15]. Thus, there are the following operation relationships [1]:

- *causal ordering* ($op_i \rightarrow op_j$ stands for op_i is executed before op_j);
- *concurrent* ($op_i \parallel op_j$ iff neither $op_i \rightarrow op_j$ nor $op_j \rightarrow op_i$).

2.1 Transformation Principle

The operational transformation is an approach for building distributed groupware systems. This approach is aiming at transforming the remote operation (*i.e.* adjusting its parameters) according to the concurrent ones [1]. As an example, consider the following group text editor scenario (see Figure 1(a)): there are two sites working on a shared document represented by a string of characters. Initially, all the copies hold the string "efect". The document is modified with the operation $Ins(p, c)$ for inserting a character c at position p. Users 1 and 2 generate two concurrent operations: $op_1 = Ins(2, f)$ and $op_2 = Ins(5, s)$ respectively. When op_1 is received and executed on site 2, it produces the expected string "effects". But, when op_2 is received on site 1, it does not take into account that op_1 has been executed before it. Consequently, we obtain a *divergence* between sites 1 and 2.



(a) Incorrect integration (b) Correct integration

Figure 1: Integration of two concurrent operations.

As a solution to divergence problems the OT approach uses an algorithm, $T(op_1, op_2)$, which takes two concurrent operations op_1 and op_2 defined on the same document state and returns op_1' which is equivalent to op_1 but defined on a document state where op_2 has been applied. In Figure 1(b), we illustrate the effect of T. When op_2 is received on site 1, op_2 needs to be transformed according to op_1 as follows: $T(Ins(5, s), Ins(2, f)) = Ins(6, s)$. The insertion position of op_2 is incremented because op_1 has inserted a character at position 2, which

is before the character inserted by op_2. Next, op_2' is executed on site 1. In the same way, when op_1 is received on site 2, it is transformed as follows:

$$T(Ins(2,f), Ins(5,s)) = Ins(2,f)$$

Operation op_1 remains the same because "f" is inserted before "s".

```
T(Ins(p1,c1), Ins(p2,c2)) =
if (p1<p2)
   or (p1 = p2 and C(c1)<C(c2)) then
   return Ins(p1,c1)
elseif (p1>p2)
   or (p1 = p2 and C(c1)>C(c2)) then
   return Ins(p1 + 1,c1)
else return nop
endif;
```

Figure 2: OT for two insert operations.

In [15], the authors have introduced the notion of operation *context* in order to capture the required relationship between operations for correct transformation. Assume all that sites start with the same initial document state.

Definition 2.2 ***(Contextual relations).***
Given any operation op then:

1. *A* context *is an sequence of operations which is executed on the initial document state and leads to the current document state.*
2. *The* definition context $DC(op, i, t)$ *is the context in which op is generated at site i and time t.*
3. *The* execution context $EC(op, i, t)$ *is the context in which op is to be executed at site i and time t.*

For readability, we omit the site and time parameters in the functions DC and EC.

Any operation will go through the process of generation, local execution, propagation, and remote execution. Consequently, due to concurrent execution of operations, $EC(op)$ at a remote site may not match $DC(op)$ at local site. Therefore, op needs to be transformed in op' such that it is defined on a new context different from its generation context, *i.e.* $DC(op') = EC(op)$ at the remote site. Two concurrent operations op_1 and op_2 can be transformed and executed consecutively if and only if they have the same definition context. In other words, $T(op_1, op_2)$ transforms op_1 against op_2 at given site such that $DC(op_1) = DC(op_2)$.

Figure 2 gives the transformation rule for two insert operations. As in [12] we use the character code C to resolve the conflict caused by two Ins operations with the same insertion position and different characters. It enables to serialize them. When both Ins operations are identical – the same position and the same character –, their transformation results in the null operation denoted by nop and which does not affect the document state.

Let seq be an sequence of operations. Transforming any editing operation op along to seq, denoted by $T^*(op, seq)$ is recursively defined as follows:

$$T^*(op, []) = op$$
$$T^*(op, [op_1; op_2; \ldots; op_n]) = T^*(T(op, op_1), [op_2; \ldots; op_n])$$

2.2 Convergence Properties

Using an OT algorithm requires to satisfy two conditions called *convergence properties* [9, 12]:

- The condition C_1 defines a *state identity*. The document state generated by the execution op_1 followed by $T(op_2, op_1)$ must be the same than the document state generated by op_2 followed by $T(op_1, op_2)$. In other words, C_1 is defined as follows:

$$[op_1; T(op_2, op_1)] \equiv [op_2; T(op_1, op_2)]$$

This condition is necessary but not sufficient when the number of concurrent operations is greater than two.

- The condition C_2 ensures that the transformation of an operation according to a sequence of concurrent operations does not depend on the order in which operations of the sequence are transformed:

$$T^*(op_3, [op_1; T(op_2, op_1)]) = T^*(op_3, [op_2; T(op_1, op_2)])$$

In [9, 7], the authors have proved that conditions C_1 and C_2 are sufficient to ensure the convergence property for any number of concurrent operations. It should be pointed out that verifying that a given OT algorithm verifies C_2 is a computationally expensive problem even for a simple document text. Using a theorem prover to automate the verification process is needed and would be a crucial step for building correct distributed groupware systems [4, 3].

3 Convergence Problems

To the best of our knowledge none of the existing OT schemes satisfies the convergence condition C_2. This condition is considered as particularly difficult to meet. For this reason some OT algorithms only require condition C_1 and replace C_2 by some other hypothesis (like global order or locking about operations [17, 16]). In this section we present scenarios in Figures 3 and 5 showing that the condition C_2 is not met by all existing OT algorithms [1, 9, 12, 4, 6].

3.1 Scenarios violating convergence

We consider the OT function illustrated in Figure 2. To illustrate scenarios in which C_2 condition fails, suppose that three users see the word "core" and they want to modify it to "coffe":

1. the first user adds "f" after "r" by generating $op_1 = Ins(3, f)$;
2. the operation generated by the second user, $op_2 = Del(2)$, is intended to delete the "r";
3. the third user generates $op_3 = Ins(2, f)$ to add "f" after "o".

These concurrent operations are broadcast to all sites, then received and finally executed after transformation as illustrated in Figure 3.

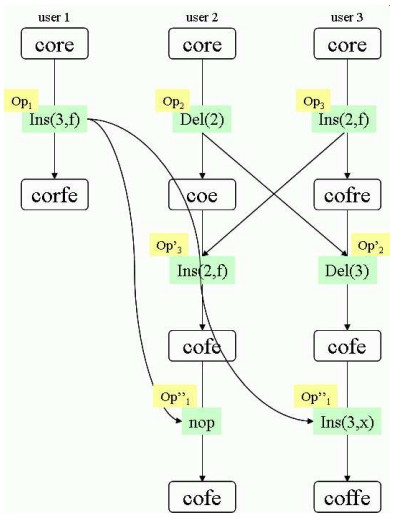


Figure 3: The C_2 puzzle P_1.

On site 3, the local operation op_3 is executed, then op_2 is received. It is transformed against op_3 and becomes $op'_2 = Del(3)$. After op'_2 is executed, the resulting word is "cofe". Arriving at this site, op_1 is integrated according to the sequence $[op_3; op'_2]$ whose transformation result is $op''_1 = Ins(3, f)$. Executing op''_1 gives the word "coffe".

At site 2, op_3 is integrated by transforming it against op_2 producing the same operation $op'_3 = Ins(2, f)$. After op'_3 is executed, the word becomes "cofe". When op_1 arrives, it is integrated according to the sequence $[op_2; op'_3]$. Firstly, it is transformed against op_2 resulting in $op'_1 = Ins(2, f)$. Lastly, it is transformed against op'_3. As op'_1 and op'_3 are identical, their transformation returns the null operation $op''_1 = nop$. The final word is "cofe" which is different to that of site 3. Consequently, there is a divergence problem, *i.e.* two users see different words. This scenario is termed as the *puzzle*. Note OT algorithms proposed in [1], [9] and [15] fail in the scenario of Figure 3.

```
T(Ins(p1,o1,c1), Ins(p2,o2,c2)) =
if (p1<p2)
   or (p1 = p2 and o1<o2)
   or (p1 = p2 and o1 = o2 and C(c1)<C(c2))
      then return Ins(p1,o1,c1)
elseif (p1>p2)
   or (p1 = p2 and o1>o2)
   or (p1 = p2 and o1 = o2 and C(c1)>C(c2))
      then return Ins(p1 + 1,o1,c1)
else return nop
endif;
```

Figure 4: OT for two insert operations with additional information.

Some works have noticed and addressed the puzzle P_1 by trying to propose correct OT algorithms [12, 4, 6]. To avoid the puzzle scenario these works extend the conventional transformation by using additional parameters. For instance, in [4], the insert operation becomes $Ins(p, o, c)$ where p is the actual position and o is the original position defined at the generation of this operation (see Figure 4).

Using this additional information is not sufficient to solve the puzzle problem since, unfortunately, there are scenarios where C_2 condition is still violated. In presence of additional information in OT algorithms, such scenarios are very difficult to guess. Consider for example in Figure 5, five sites that start from the same state "abcd". At site 1, user 1 executes $op_1 = Del(1)$ followed by $op_2 = Ins(3, 3, x)$. Operations $op_3 = Ins(3, 3, x)$ and $op_4 = Del(3)$ are concurrently executed in sites 4 and 5, respectively. Users in sites 2 and 3 do not generate any operation, but receive and execute all remote operations in different orders. As shown in Figure 5, the four operations in this scenario are broadcasted in the following orders: (i) op_1, op_3, op_4 and op_2 at site 2; (ii) op_1, op_4, op_3 and op_2 at site 3.

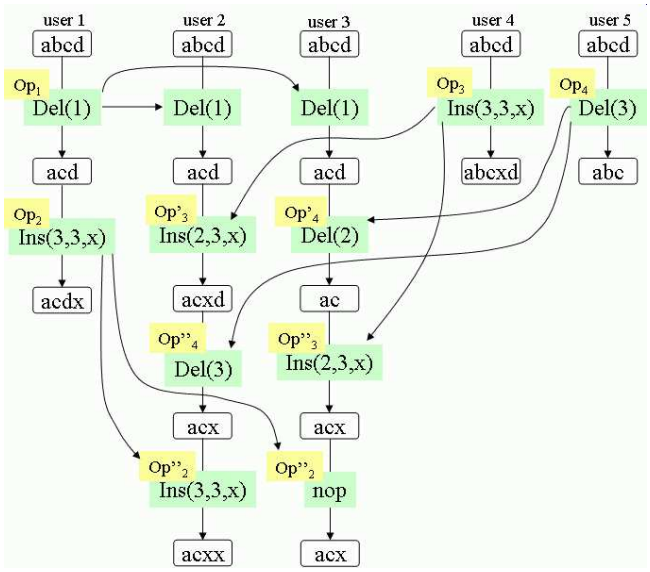
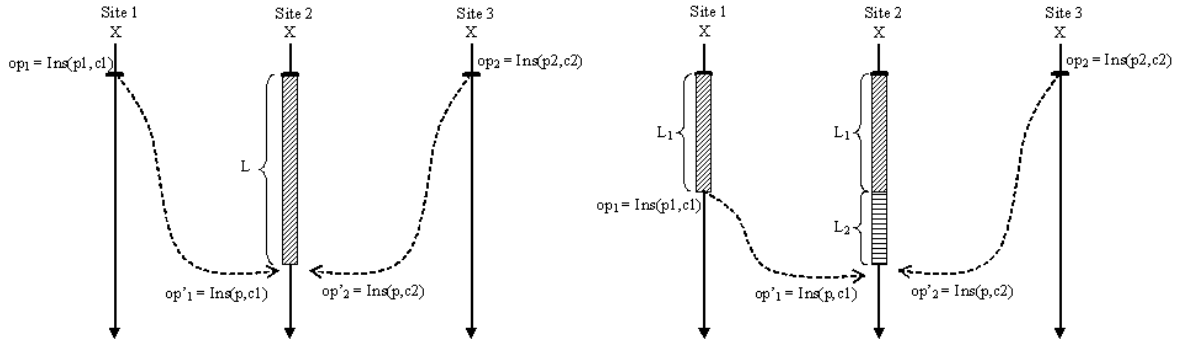


Figure 5: The C_2 puzzle P_2.



(a) Same definition contexts

(b) Different definition contexts

Figure 6: Conflict situations.

Now consider what will happen at sites 2 and 3. At site 2, when op_1 is received it is executed without transformation. When op_3 arrives, it is transformed against op_1 resulting in $op'_3 = Ins(2,3,x)$. Then op_4 arrives and is transformed according to the sequence $[op_1; op'_3]$: transformation against op_1 results in $op'_4 = Del(2)$, then op'_4 against to op'_3 gives $op''_4 = Del(3)$. As op_2 has already seen the execution of op_1 ($op_1 \rightarrow op_2$), then it will be only transformed according to the sequence $[op'_3, op''_4]$. Transforming op_2 against op'_3 produces $op'_2 = Ins(4,3,x)$, and op'_2 against to op''_4 results in $op''_2 = Ins(3,3,x)$. Execution of op''_2 leads to state "acxx".

At site 3, the integration of op_1 is made without transformation. When receiving op_4, it is transformed against op_1 resulting in $op'_4 = Del(2)$. Next, op_3 arrives and is integrated according to $[op_1; op'_4]$: transforming against op_1 gives $op'_3 = Ins(2,3,x)$ and op'_3 against op'_4 results in $op''_3 = Ins(2,3,x)$. In the same way, op_2 will be only transformed according to the sequence $[op'_4; op''_3]$. The first transformation with respect to op'_4 gives $op'_2 = Ins(2,3,x)$ which is next transformed against to op''_3 and becomes *nop* since op'_2 and op''_3 are identical. At the end, the state obtained is "acx".

Thus transforming op_2 according to two equivalent sequences, $[op'_3; op''_4]$ and $[op'_4; op''_3]$, results in different operations and leads to different states. Consequently, C_2 is still violated even by using an additional information like the original position. Note that even OT algorithms proposed in [12] and [6] fail to achieve convergence in the scenario of Figure 5.

3.2 Analyzing the problem

3.2.1 Conflict Situations

As seen in the above examples the conflict occurs only when two concurrent insert operations are involved. In Figure 6, we give two conflict situations where there are three sites starting with the same initial document state X. In both situations, there are two concurrent insert operations op_1 and op_2 that are generated at different sites and may have or not the same original insertion positions. In the first one (see Figure 6(a)), op_1 and op_2 have the same generation context, *i.e.* $DC(op_1)$ at site 1 is the same than $DC(op_2)$ at site 3. In site 2, a sequence of operations has been executed and operations are stored in a log denoted by L. The operations of L are concurrent to op_1 and op_2. At site 2, op_1 and op_2 may arrive in different orders after L. As their execution context at site 2 is different from their definition context, they must be transformed against L. Unlike the first situation, op_1 and op_2 do not have the same generation context in the second one (see Figure 6(b)). In site 1, op_1 has already seen a subsequence L_1 of operations executed in site 2. Consider L the log executed in site 2 and which consists of L_1 followed by L_2. After L the arrival order of op_1 and op_2 is arbitrary: op_1 and op_2 must be transformed against L_1 and L respectively. In both situations, this transformation results in two insert operations op'_1 and op'_2 having the same position p, and two possible sequences of operations can be executed after L: $[op'_1; T(op'_2, op'_1)]$ and $[op'_2; T(op'_1, op'_2)]$. Note that op'_1 and op'_2 are in conflict whereas maybe the original operations op_1 and op_2 are not.

The transformation process may lead two concurrent insert operations (with different original insertion positions) to get into conflict. Unfortunately, the original relation between the positions of these operations is lost because of their transformations with other operations. Therefore, we need to know how the insert operations were generated in order to avoid divergence problems.

3.2.2 Using Extra Information

Many OT algorithms [1, 9, 12, 4] use extra informations to resolve the conflict between two insert operations, such that: identifiers of sites [1, 9], character code [12] and original positions [4]. Now we try to fix and analyse this problem. Let $\mathcal{I}$ be the extra information used for every insert operation. Given two equivalent sequences $s_1 = [Ins(1, c_1); Del(2)]$ and $s_2 = [Del(1); Ins(1, c_1)]$, and an insert operation $op = Ins(2, c_2)$ where c_1 and c_2 are any characters. Transforming *op* according to s_1 does not create conflict since the position of *op* is only shifted when it is transformed against $Ins(1, c_1)$ (see Figure 4). Note that in this case no extra information is used. When transforming *op* according to s_2 it generates conflict between two insert operations. The resolution of this conflict is not deterministic and may lead to many results (*nop*, $Ins(1, x)$ or $Ins(2, x)$) depending on the extra information $\mathcal{I}$ that was employed. For instance, if $\mathcal{I}$ is the character code C then we have to compare c_1 and c_2 (see Figure 4).

In the first example (see Figure 3), that corresponds to the first conflict situation, the puzzle P_1 is due to two original insertion operations, which are generated on the same document state and not in conflict because they have not the same insertion positions. $\mathcal{I}$, *i.e.* the character code, fails to avoid the puzzle problem because two insert operations are considered as identical and consequently the position of *op* is not shifted unlike the sequence s_1. If $\mathcal{I}$ is the original position then *op* is transformed against s_2 in the same way than in s_1 (the position of *op* is also shifted).

In the second example (see Figure 5), that corresponds to the second situation of conflict, the puzzle P_2 is due to two original insert operations that are concurrent and not defined on the same document state. Unfortunately, $\mathcal{I}$ (original position) fails to shift *op* position because two insert operations are still considered as identical.

In this paper, we propose a new approach to solve the divergence problem. Intuitively, we notice that storing previous insertion positions for every transformation step is sufficient to recover the original position relation between two insert operations.

4 Our Solution

In this section, we present our approach to achieving convergence. Firstly, we will introduce the key concept of position word for keeping track of insertion positions. Next, we will give our new OT function and show how this function resolves the divergence problem. Finally, we will show the correctness of our approach.

4.1 Position Words

For any set Σ of symbols called an *alphabet*, Σ^* denotes the set of all *words* of symbols over Σ. The empty word is ϵ. For $\omega \in \Sigma^*$, then $|\omega|$ denotes the *length* of ω. If $\omega = uv$, for some $u, v \in \Sigma^*$, then u is a *prefix* of ω and v is a *suffix* of ω. A word $\omega \in \Sigma^*$ can be considered as a function $\omega : \{0, \ldots, |\omega| - 1\} \to \Sigma$; the value of $\omega(i)$, where $0 \leq i \leq |\omega| - 1$, is the symbol in the ith position of ω. For every $\omega \in \Sigma^*$, such that $|\omega| > 0$, we denote $Origin(\omega)$ (resp. $Current(\omega)$) the *last* (resp. *first*) symbol of ω. Thus, $Current(abcde) = a$ and $Origin(abcde) = e$. Assume Σ has an alphabetic (linear) order. If $\omega_1, \omega_2 \in \Sigma^*$, then $\omega_1 \preceq \omega_2$ is the *lexicographic ordering* of Σ^* if: (i) ω_1 is a prefix of ω_2, or (ii) $\omega_1 = \rho u$ and $\omega_2 = \rho v$, where $\rho \in \Sigma^*$ is the longest prefix common to ω_1 and ω_2, and $Current(u)$ precedes $Current(v)$ in the alphabetic order.

Definition 4.1 ***(p-word)***
Let $\Sigma = \mathbb{N}$ be an alphabet over natural numbers. The set $\mathcal{P} \subset \mathbb{N}^*$ of words, called p_words*, is defined as follows: (i) $\epsilon \in \mathcal{P}$; (ii) if $n \in \mathbb{N}$ then $n \in \mathcal{P}$; (iii) if ω is a nonempty p-word and $n \in \mathbb{N}$ then $n\omega \in \mathcal{P}$ iff either $n = Current(\omega)$ or $n = Current(\omega) \pm 1$.*

Theorem 1 *Let ω_1 and ω_2 be two nonempty p-words. The concatenation of ω_1 and ω_2, written $\omega_1 \cdot \omega_2$ or simply $\omega_1\omega_2$, is a p-word iff either $Origin(\omega_1) = Current(\omega_2)$ or $Origin(\omega_1) = Current(\omega_2) \pm 1$.*

Proof. 1 *We proceed by induction on the length of ω_1:*

- *Basis step: $|\omega_1| = 1$ and ω_1 consists of a natural number n. Then $n\omega_2$ is a p-word by Definition 4.1.*
- *Induction hypothesis: if $|\omega_1| \leq l$ then $\omega_1\omega_2$ is a p-word.*
- *Induction step: Let $|\omega_1| = l + 1$. Then $\omega_1 = m\rho$ is a p-word for some $m \in \mathbb{N}$ and $\rho \in \mathcal{P}$ such that $|\rho| = l$. We have: $\omega_1\omega_2 = (m\rho)\omega_2 = m(\rho\omega_2)$ since concatenation is associative. By Definition 4.1 and the induction hypothesis we conclude $\omega_1\omega_2$ is a p-word.*

```
T(Ins(p1,c1,w1),Ins(p2,c2,w2)) =
let α1=PW(Ins(p1,c1,w1)) and
    α2=PW(Ins(p2,c2,w2))

if (α1 ≺ α2 or (α1 = α2 and C(c1)<C(c2)))
then return Ins(p1,c1,w1)
elseif (α1 ≻ α2 or (α1 = α2 and C(c1)>C(c2)))
    then return Ins(p1+1,c1,p1w1)
    else return nop(Ins(p1,c1,w1))
endif;

T(Ins(p1,c1,w1),Del(p2)) =
if p1>p2 then return Ins(p1-1,c1,p1w1)
elseif p1<p2 then return Ins(p1,c1,w1)
    else return Ins(p1,c1,p1w1)
endif;

T(Del(p1),Del(p2)) =
if p1<p2 then return Del(p1)
elseif p1>p2 then return Del(p1-1)
    else return nop(Del(p1))
endif;

T(Del(p1),Ins(p2,c2,w2)) =
if p1<p2 then return Del(p1)
else return Del(p1+1)
endif;
```

Figure 7: New OT function.

□

For example, $\omega_1 = 00$, $\omega_2 = 1232$ and $\omega_1\omega_2 = 001232$ are p-word but $\omega_3 = 3476$ is not.

Definition 4.2 ***(Equivalence of p-words)***
The equivalence relation on the set of p_ words $\mathcal{P}$ is defined by:
$\omega_1 \equiv_{\mathcal{P}} \omega_2 \Leftrightarrow Current(\omega_1) = Current(\omega_2)$ *and* $Origin(\omega_1) = Origin(\omega_2)$, *for* ω_1, $\omega_2 \in \mathcal{P}$.

Proposition 1 ***(Right congruence)***
The equivalence relation $\equiv_{\mathcal{P}}$ is a right congruence, *that is, for all $\rho \in \mathcal{P}$:*

$$\omega_1 \equiv_{\mathcal{P}} \omega_2 \Leftrightarrow \omega_1\rho \equiv_{\mathcal{P}} \omega_2\rho$$

The proof of this proposition is based on Definitions 4.1 and 4.2.

4.2 OT Function

We extend the insert operation with a new parameter that contains a p-word giving the positions occupied before every transformation step. Thus an insert operation becomes: $Ins(p, c, w)$ where p is the insertion position, c the character to be added and w a p-word. Let $Char$ be the set of characters. We define the set of editing operations as follows:

$$\begin{aligned} \mathcal{O} \quad = \quad & \{Ins(p, c, w) \mid p \in \mathbb{N} \text{ and } c \in Char \text{ and } w \in \mathcal{P}\} \\ & \cup \{Del(p) \mid p \in \mathbb{N}\} \end{aligned}$$

We use an undefined function $nop : \mathcal{O} \to \mathcal{O}$ to represent null operation, *i.e.* the operation that has no effect on a document state. We also define a function PW which enables to construct p-words from editing operations.

It is defined as follows:

$$PW \quad : \quad \mathcal{O} \rightarrow \mathcal{P}$$

$$PW(Ins(p,c,w)) \quad = \quad \begin{cases} p & \text{if } w = \epsilon \\ pw & \text{if } w \neq \epsilon \text{ and} \\ & (p = Current(w) \\ & \text{ or } p = Current(w) \pm 1) \\ \epsilon & \text{otherwise} \end{cases}$$

and $PW(Del(p)) = p$. By convention, we pose $PW(nop(op)) = PW(op)$ for every operation $op \in \mathcal{O}$. Initially when an insert operation is generated locally it has an empty p-word parameter; hence such an operation is of type $Ins(p, c, \epsilon)$.

Definition 4.3 ***(Insertion equivalence).***
Given two insert operations $op_1 = Ins(p_1, c_1, w_1)$ and $op_2 = Ins(p_2, c_2, w_2)$. We say that op_1 and op_2 are equivalent iff:

1. *$c_1 = c_2$; and,*
2. *$PW(op_1) \equiv_{\mathcal{P}} PW(op_2)$.*

>From this definition we can deduce that op_1 and op_2 have the same insertion position since their p-words are equivalent.

Based on divergence problems illustrated in Section 3, we redefine OT function by using the p-word concept. In Figure 7, we give all possible transformations regarding Ins and Del. Note the use of PW function is only used to handle insert operations. Given an insert operation op, $PW(op)$ gives a p-word which restores all positions occupied by the insert operation op since it was generation. We first compare the PW values of $Ins(p_1, c_1, w_1)$ and $Ins(p_2, c_2, w_2)$ [1]. If their p-words are equal, then we compare their character codes. When we have the same character to be inserted in the same position then the OT function gives the null operation *nop*, *i.e.* one insert operation must be executed and the other one must be ignored [12]. Moreover, the transformation of *nop* is not mentioned in Figure 7, but this case is very simple. Indeed, for every editing operations $p, p' \in \mathcal{O}$ we complete the definition of T by:

$$\begin{array}{lcl} T(nop(op), op') & = & nop(T(op, op')) \\ T(op', nop(op)) & = & op' \end{array}$$

Definition 4.4 ***(Conflict relation).***
Given op_1 and op_2 two concurrent insert operations defined on the same context ($DC(op_1) = DC(op_2)$). Then op_1 and op_2 are in conflict, *denoted $op_1 \otimes op_2$ iff $PW(op_1) = PW(op_2)$. We denote by $op_1 \odot op_2$ the fact that they do not conflict.*

In other words, according to Definition 4.4, two concurrent insert operations op_1 and op_2 are not in conflict iff their p-words are different, *i.e.* either $PW(op_1) \prec PW(op_2)$ or $PW(op_1) \succ PW(op_2)$.

4.3 Examples

In this section we apply our proposed algorithm on the puzzles previously presented in section 3.

Figure 8 illustrates the replay of C_2 puzzle. When op_3 is received on site 2, it is transformed according to op_2, producing the same operation op_3' with an updated value of p-word parameter equals to [2]. At site 3, op_2 is integrated by being transformed according to op_3. Its deletion position is increased. Thus, the resulting operations is $op_2' = Del(3)$.

When op_1 is received on site 2, it must be transformed according to the sequence $[op_2, op_3']$ as follows :

$$T(\overbrace{Ins(3, f, \epsilon)}^{op_1}, \overbrace{Del(2)}^{op_2}) = \overbrace{Ins(2, f, [3])}^{op_1'}$$

Resulting from the transformation, the insert position decreases and the p-words value is equals to the insert position of op_1.

[1] $Current(PW(op))$ gives the current position of the insert operation op.

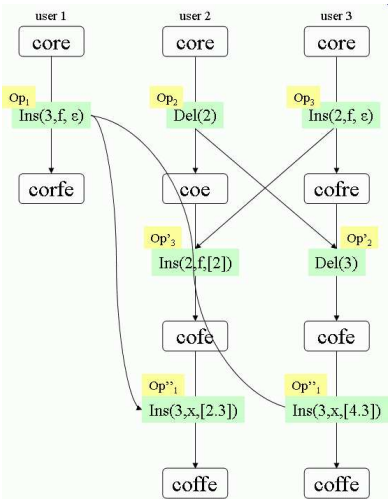


Figure 8: Correct execution of C_2 puzzle P_1.

$$T(\overbrace{Ins(2,f,[3])}^{op'_1},\overbrace{Ins(2,f,[2])}^{op'_3}) = \overbrace{Ins(3,x,[2.3])}^{op''_1}$$

As the insert position are equals, p-words values are compared: p-word of op'_1 is greater than p-word of op'_3, so the insert position must be increased.

It is important to note, using p-words that the transformation function has been able to detect that the two operations were different at their originating sites. Thus, the resulting transformation is not a null operation *nop* contrary to the execution in subsection section 3.1.

In the same way, at site 3, op_1 is integrated by computing the following transformations according to the sequence $[op_3; op'_2]$:

$$T(\overbrace{Ins(3,f,\epsilon)}^{op_1},\overbrace{Ins(2,f,\epsilon)}^{op_3}) = \overbrace{Ins(4,f,[3])}^{op'_1}$$

The transformation increases the insert position and updates the p-word value to the old insert position.

$$T(\overbrace{Ins(4,f,[3])}^{op'_1},\overbrace{Del(3)}^{op'_2}) = \overbrace{Ins(3,x,[4.3])}^{op''_1}$$

The insert position is decreased, and the old insertion position is prepended to the p-word value.

Finally, we obtain the same operations on site 2 and site 3; p-words are equals according to Definition 4.2.

Figure 9 shows replay of another more complex puzzle, that has never been solved by other algorithms before. According to subsection 3.1, the mistake appeared on site 3, from the transformation $T(op_2, [op'_4; op''_3])$ which was evaluated to *nop*. Now, with our solution, this transformation is computed as follows:

$$T(\overbrace{Ins(3,x,\epsilon)}^{op_2},[\overbrace{Del(2)}^{op'_4};\overbrace{Ins(2,x,[2.3])}^{op''_3}])$$

$$= T(Ins(2,x,[3]),\overbrace{Ins(2,x,[2.3])}^{op''_3}) = \overbrace{Ins(3,x,[2.3])}^{op''_2})$$

Execution of resulting op''_2 gives the correct state "acxx" on site 3, which converge with the state obtained in site 2.

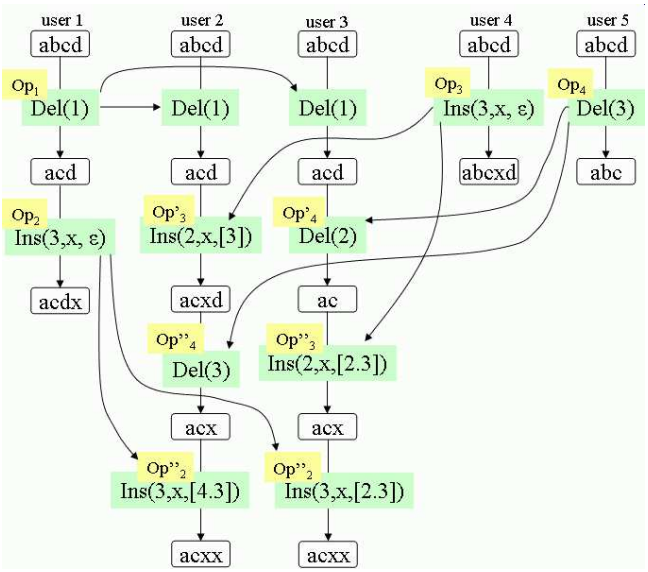


Figure 9: Correct execution of C_2 puzzle P_2.

4.4 Correctness

In the following, we give the correctness of our approach by proving that:

1. our OT function does not lose track of insertion positions;
2. original relation between two insert operations is preserved by transformation;
3. conditions C_1 and C_2 are satisfied.

4.4.1 Conservation of p-words

In the following, we show that our OT function does not lose any information about position words.

Lemma 1 *Given an insert operation $op_1 = Ins(p_1, c_1, w_1)$ such that $PW(op_1) \neq \epsilon$. For every editing operation $op \in \mathcal{O}$, $PW(op_1)$ is a suffix of $PW(T(op_1, op))$.*

Proof. 2 *Let $op'_1 = T(op_1, op)$ and $PW(op_1) = p_1 w_1$. Then, we consider two cases:*

1. *$op = Ins(p, c, w)$: Let $\alpha_1 = PW(op_1)$ and $\alpha_2 = PW(op)$.*
   - *if $(\alpha_1 \prec \alpha_2$ or $(\alpha_1 = \alpha_2$ and $C(c_1) < C(c)))$ then $op'_1 = op_1$;*
   - *if $(\alpha_1 \succ \alpha_2$ or $(\alpha_1 = \alpha_2$ and $C(c_1) > C(c)))$ then $op'_1 = Ins(p_1 + 1, c_1, w_1)$ and $p_1 w_1$ is a suffix of $PW(op'_1)$;*
   - *if $C(c_1) = C(c)$ then $op'_1 = nop(op_1)$ and $PW(op_1) = PW(nop(op_1))$.*
2. *$op = Del(p)$*
   - *if $p_1 > p$ then $op'_1 = Ins(p_1 - 1, c_1, p_1 w_1)$ then $p_1 w_1$ is a suffix of $PW(op'_1)$;*
   - *if $p_1 < p$ then $op'_1 = op_1$;*
   - *if $p_1 = p$ then $op'_1 = Ins(p_1, c_1, p_1 w_1)$ and $p_1 w_1$ is a suffix of op'_1.*

□

The following theorem stipulates that the extension of our OT function to sequence, *i.e.* T^*, does not lose any informations about position words.

Theorem 2 *Given an insert operation $op_1 = Ins(p_1, c_1, w_1)$ such that $PW(op_1) \neq \epsilon$. For every operation sequence seq, $PW(op_1)$ is a suffix of $PW(T^*(op_1, seq))$.*

Proof. 3 *Assume n is the length of seq. We proceed by induction on n.*

- *Basis step: $n = 0$. Then seq is empty and we have $T^*(op_1, []) = op_1$.*
- *Induction hypothesis: for $n \geq 0$, $PW(op_1)$ is a suffix of $PW(T^*(op_1, seq))$.*
- *Induction step: Let $n+1$ the length of seq. Then $seq = [seq'; op]$ is sequence formed by a sequence seq' whose the length is n and some editing operation $op \in \mathcal{O}$. We have $T^*(op_1, [seq'; op]) = T(T^*(op_1, seq'), op)$. By using Lemma 1, $PW(T^*(op_1, seq'))$ is a suffix of $PW(T^*(op_1, [seq'; op])) = PW(T(T^*(op_1, seq'), op))$. By induction hypothesis and the transitivity of suffix relation, we conclude that $PW(op_1)$ is a suffix of $PW(T^*(op_1, seq))$ for every sequence of operations seq.*

□

4.4.2 Position Relations

We can use the position relations between insert operations as an *invariance property* which must be preserved when these operations are transformed and executed in all remote sites. We define this property as follows:

Definition 4.5 ***(Invariance property).*** *Given two concurrent insert operations op_1 and op_2, and let op'_1 and op'_2 be their transformation forms respectively at given remote site:*

- *if $PW(op_1) \succ PW(op_2)$ then $PW(op'_1) \succ PW(op'_2)$;*
- *if $PW(op_1) \prec PW(op_2)$ then $PW(op'_1) \prec PW(op'_2)$.*

According to Definition 4.4, we can deduce that two insert operations do not conflict if their position words are different. In the following, we show the correctness of our OT function (Figure 7) with respect to the invariance property.

Lemma 2 *Given two concurrent insert operations op_1 and op_2. For every editing operation $op \in \mathcal{O}$:*

$$op_1 \odot op_2 \implies T(op_1, op) \odot T(op_2, op)$$

This lemma shows that our OT function preserves the invariance property.

Proof. 4 *We have to consider two cases: $op = Ins(p, c, w)$ and $op = Del(p)$. For instance, consider the following case : $op_1 = Ins(p_1, c_1, w_1)$ and $op_2 = Ins(p_2, c_2, w_3)$ such that $op = op_1$ and $PW(op_1) \prec PW(op_2)$. We have the following transformations:*

1. *$op'_1 = T(op_1, op) = nop(op_1)$;*
2. *$op'_2 = T(op_2, op) = Ins(p_2 + 1, c_1, p_2 w_2)$*

We can conclude that $PW(op'_1) \prec PW(op'_2)$. Since the p-word order between op_1 and op_1 is arbitrary and p is any editing operation, we have to consider all possible cases. The complete proof is verified by a theorem prover.

□

The following theorem shows that the extension of our OT function to sequence, *i.e.* T^*, preserves also the invariance property.

Theorem 3 *Given two concurrent insert operations op_1 and op_2. For every sequence of operations seq:*

$$op_1 \odot op_2 \implies T^*(op_1, seq) \odot T^*(op_2, seq)$$

Proof. 5 *Suppose n is the length of the sequence of operations seq. It is sufficient to show this theorem by induction on n.*

- *Basis step: for $n = 0$ we have $T^*(op_1, []) = op_1$ and $T^*(op_2, []) = op_2$.*
- *Induction hypothesis: if $n \geq 0$ then $op_1 \odot op_2 \implies T^*(op_1, seq) \odot T^*(op_2, seq)$.*
- *Induction step: Let $n+1$ be the length of seq. Then $seq = seq'; op$ is sequence formed by a sequence seq' whose the length is n and some editing operation $op \in \mathcal{O}$. We have:*

$$T^*(op_1, [seq'; op]) = T(T^*(op_1, seq'), op) \text{ and}$$
$$T^*(op_2, [seq'; op]) = T(T^*(op_2, seq'), op)$$

After this rewriting, we can use induction hypothesis and Lemma 2 for this proof.

□

4.4.3 Convergence Properties

In [9, 12] the authors define convergence conditions and have shown that an OT technique can achieve convergence for arbitrary operation execution order if the OT function satisfies the convergence conditions C_1 and C_2. In the following, we sketch the proof that C_1 and C_2 are verified by our transformations and we can therefore conclude that it achieves convergence.

Based on p-word concept and our OT function, we define a total order on editing operations.

Definition 4.6 ***(Total order).*** *We define the strict part of a total order on the set of operations with the same contexts as follows: Given two concurrent editing operations op_1 and op_1 such that $DC(op_1) = DC(op_2)$. $op_1 \sqsubset op_2$ iff one of the following conditions holds:*

1. $Current(PW(op_1)) < Current(PW(op_2))$;
2. $op_1 = Ins(p_1, c_1, w_1)$, $op_2 = Ins(p_2, c_2, w_2)$ *and* $PW(op_1) \prec PW(op_2)$;
3. $op_1 = Ins(p_1, c_1, w_1)$, $op_2 = Ins(p_2, c_2, w_2)$, $PW(op_1) = PW(op_2)$ *and* $C(c_1) < C(C_2)$;
4. $op_1 = Ins(p_1, c_1, w_1)$, $op_2 = Del(p_2)$ *and* $p_1 = p_2$.

Based on the total order $\sqsubseteq$, we define the function $TPW(op_1, op_2)$ which gives the p-word of $T(op_1, op_2)$:

$$
\begin{array}{rcl}
TPW & : & \mathcal{O} \times \mathcal{O} \rightarrow \mathcal{P} \\
TPW(p_1, p_2) & = & \begin{cases}
PW(op_1) & \text{if } op_1 \sqsubseteq op_2 \\
\omega_1 \cdot PW(op_1) & \text{if } op_2 \sqsubset op_1 \text{ and } op_1 = Ins(p_1, c_1, w_1) \text{ and } op_2 = Ins(p_2, c_2, w_2) \\
\omega_2 \cdot PW(op_1) & \text{if } op_2 \sqsubset op_1 \text{ and } op_1 = Ins(p_1, c_1, w_1) \text{ and } op_2 = Del(p_2) \\
\omega_1 & \text{if } op_2 \sqsubset op_1 \text{ and } op_1 = Del(p_1) \text{ and } op_2 = Ins(p_2, c_2, w_2) \\
\omega_2 & \text{if } op_2 \sqsubset op_1 \text{ and } op_1 = Del(p_1) \text{ and } op_2 = Del(p_2)
\end{cases}
\end{array}
$$

where:

$$
\begin{aligned}
\omega_1 &= Current(PW(op_1)) + 1 \\
\omega_2 &= Current(PW(op_2)) - 1
\end{aligned}
$$

The following theorem shows that our OT function satisfies C_1.

Theorem 4 ***(Condition C_1).***
Given any editing operations $op_1, op_2 \in \mathcal{O}$ and for every document state st:
$st \bullet [op_1; T(op_2, op_1)] = st \bullet [op_2; T(op_1, op_2)]$.

Proof. 6 *Consider the following case: $op_1 = Ins(p_1, c_1, w_1)$, $op_2 = Ins(p_2, c_2, w_2)$ and $op_1 \sqsubset op_2$. According to this order, c_1 is inserted before c_2. If op_1 has been executed then when op_2 arrives it is shifted ($op'_2 = T(op_2, op_2) = Ins(p_2 + 1, c_1, p_2 w_2)$) and op'_2 inserts c_2 to the right of c_1. Now, if op_1 arrives after the execution of op_2, then op_1 is not shifted, i.e. $op'_1 = T(op_1, op_2) = op_1$. The character c_1 is inserted as it is to the left of c_2. Thus executing $[op_1, op'_2]$ and $[op_2, op'_1]$ on the same document state gives also the same document state.* □

Transforming an insert operation along two equivalent sequences produces two insert operations that are equivalent.

Theorem 5 *Given an insert operation $op = Ins(p, c, w)$. For every editing operations $op_1, op_2 \in \mathcal{O}$, $T^*(op, [op_1; op'_2]$ and $T^*(op, [op_2; op'_1])$ are identical where $op'_1 = T(op_1, op_2)$ and $op'_2 = T(op_2, op_1)$.*

Proof. 7 *Consider the case of* $op_1 = Ins(p_1, c_1, w_1)$, $op_2 = Del(p_2)$, $p_1 = p_2$ *and* $p > p_2 + 1$. *Using our OT function (see Figure 7), we have* $op'_1 = T(op_1, op_2) = Ins(p_1, c_1, p_1 w_1)$ *and* $op'_2 = T(op_2, op_1) = Del(p_2 + 1)$. *When transforming op against* $[Ins(p_1, c_1, w_2); Del(p_2 + 1)]$ *we get* $op' = Ins(p, c, (p+1)pw)$ *and when transforming op against* $[Del(p_2); Ins(p_1, c_1, p_1 w_1)]$ *we obtain* $op'' = Ins(p, c, (p-1)pw)$. *Operations* op' *and* op'' *have the same insertion position and the same character. It remains to show that* $PW(op') \equiv_{\mathcal{P}} PW(op'')$. *As* $p(p-1)p \equiv_{\mathcal{P}} p(p+1)p$ *and the equivalence relation* $\equiv_{\mathcal{P}}$ *is a right congruence by Proposition 1 then* op' *and* op'' *are identical.* □

Theorem 6 shows that our OT function also satisfies C_2

Theorem 6 ***(Condition*** C_2***).***
If the OT function T *satisfies* C_1 *then for every concurrent editing operations* op_1, op_2, $op_3 \in \mathcal{O}$:

$$T^*(op_1, [op_2; T(op_3, op_2)]) = T^*(op_1, [op_3; T(op_2, op_3)])$$

This theorem means that if T satisfies conditions C_1 then when transforming op_1 against two equivalent sequences $[op_2; T(op_3, op_2)]$ and $[op_3; T(op_2, op_3)]$ it will produce the same operation.

Proof. 8 *To prove Theorem 6, we use the total order* $\sqsubseteq$ *and the function TPW defined above. Let* op_1, op_2 *and* op_3 *be three concurrent editing operations. Assume that* $op_1 \sqsubset op_2$ *and* $op_2 \sqsubset op_3$. *Let* $op'_2 = T(op_2, op_3)$, $op'_3 = T(op_3, op_2)$, $op_1^{1\prime} = T(op_1, op_2)$ *and* $op_1^{2\prime} = T(op_1, op_3)$. *Using Lemma 2, we can show that* $op_1^{1\prime} \sqsubset op'_3$ *and* $op_1^{2\prime} \sqsubset op'_2$. *With this order we can determine p-word of* $op_1^{1\prime}$ *and* $op_1^{2\prime}$, *as follows:* $PW(op_1^{1\prime}) = PW(op_1)$ *and* $PW(op_1^{2\prime}) = PW(op_1)$. *Due to* $op_1^{1\prime} \sqsubset op'_3$, *when computing* $op_1^{1\prime\prime} = T(op_1^{1\prime}, op'_3)$ *we have* $PW(op_1^{1\prime\prime}) = PW(op_1)$. *Then when computing* $op_1^{2\prime\prime} = T(op_1^{2\prime}, op'_2)$, *due to* $op_1^{2\prime} \sqsubset op'_2$, *we have* $PW(op_1^{2\prime\prime}) = PW(op_1)$. *Consequently, this case satisfies condition* C_2 *since* $op_1^{1\prime\prime} = op_1^{2\prime\prime}$. □

The complete proofs of Theorems 4, 5 and 6 are automatically checked by a theorem prover.

5 Related Work

Since C_2 puzzle [9] was discovered, a lot of propositions has been proposed to address this problem. Proposed solutions can be categorized in two categories.

The first approach try to avoid C_2 puzzle scenario. This is achieved by constraining the communication among replicas in order to restrict the space of possible execution order. For example, SOCT4 algorithm [17] uses a sequencer, associated with a deferred broadcast and a sequential reception, to enforce a continuous global order on updates. This global order can also be obtained by using an undo/do/redo scheme like in GOTO [14].

The second approach deals with resolution of the C_2 puzzle. In this case, concurrent operations can be executed in any order, but transformation functions require to satisfied the C_2 condition. This approach has been developed in aDOPTed [9], SOCT2 [12], GOT [15]. Unfortunately, we have proved in [4] that all previously proposed transformation functions fails to satisfy this condition.

Recently, Li and al. [6] have tried to analyze the root of the problem behind C_2 puzzle. We have found that there is still a flaw in their solution. Consider three concurrent operations $op_1 = Ins(3, x)$, $op_2 = Del(2)$ and $op_3 = Ins(2, y)$ generated on sites 1, 2 and 3 respectively. They use a function β that compute for every editing operation the original position according to the initial document state. Initially, $\beta(op_1) = 3$, $\beta(op_2) = 2$ and $\beta(op_3) = 2$. We obtain the following transformation:

1. $op'_1 = T(op_1, op_2) = Ins(2, x)$ and,
2. $op'_3 = T(op_3, op_2) = Ins(2, x)$.

The mistake is due to the definition of their β function. Indeed, their definition relies on the exclusion transformation function, which is the reversed function of the "basic" transformation one, *i.e.* T. As this reverse function is not always defined [14], due to the non-inversibility of T, op'_1 and op'_3 are regarded as two insert operations in conflict. Indeed, we lose the original relation between op_1 and op_2 since $\beta(op'_1)$ can be equal to 2 or 3 when it is computed according to op_2. Consequently, the convergence property cannot be achieved in all cases.

6 Conclusion

OT has a great potential for generating non-trivial state of convergence. However, without a correct set of transformation functions, OT is useless. In this paper we have pointed out correctness problems of the existing transformation functions and proposed a formal and generic solution to solve them.

We can now use our generic solution to write transformation functions for more complex object types such as XML trees. We also apply our formal results in the development of a file synchronizer [8]. This tool is able to safely synchronize files and their contents (text, XML, ...), ensuring convergence in all cases.

In future work, we plan:

- to deal with the semantic convergence by using OT approach [16];
- to explore the impact of our approach when undoing operations [9].

References

[1] C. A. Ellis and S. J. Gibbs. Concurrency Control in Groupware Systems. In *SIGMOD Conference*, volume 18, pages 399–407, 1989.

[2] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: some issues and experiences. *Commun. ACM*, 34(1):39–58, 1991.

[3] A. Imine, P. Molli, G. Oster, and M. Rusinowitch. Development of Transformation Functions Assisted by a Theorem Prover. *Fourth International Workshop on Collaborative Editing (ACM CSCW'02), Collaborative Computing in IEEE Distributed Systems Online*, Nov. 2002.

[4] A. Imine, P. Molli, G. Oster, and M. Rusinowitch. Proving Correctness of Transformation Functions in Real-Time Groupware. In *8th European Conference of Computer-supported Cooperative Work*, Helsinki, Finland, 14.-18. September 2003.

[5] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[6] D. Li and R. Li. Ensuring Content Intention Consistency in Real-Time Group Editors. In *the 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, Tokyo, Japan, March 2004. IEEE Computer Society.

[7] B. Lushman and G. V. Cormack. Proof of correctness of ressel's adopted algorithm. *Information Processing Letters*, 86(3):303–310, 2003.

[8] P. Molli, G. Oster, H. Skaf-Molli, and A. Imine. Using the transformational approach to build a safe and generic data synchronizer. In *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 212–220. ACM Press, 2003.

[9] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhauser. An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'96)*, pages 288–297, Boston, Massachusetts, USA, November 1996.

[10] Y. Saito and M. Shapiro. Optimistic Replication. Technical Report MSR-TR-2003-60, Microsoft Research, October 2003.

[11] H. Shen and C. Sun. Flexible merging for asynchronous collaborative systems. In *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002*, pages 304–321. Springer-Verlag, 2002.

[12] M. Suleiman, M. Cart, and J. Ferrié. Concurrent Operations in a Distributed and Mobile Collaborative Environment. In *Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*, pages 36–45. IEEE Computer Society, 1998.

[13] C. Sun. The coword project. `http://www.cit.gu.edu.au/~scz/projects/coword/`, 2003.

[14] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 59–68. ACM Press, 1998.

[15] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving Convergence, Causality-preservation and Intention-preservation in real-time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1):63–108, March 1998.

[16] C. Sun and R. Sosic. Optimal locking integrated with operational transformation in distributed real-time group editors. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 43–52. ACM Press, 1999.

[17] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'00)*, Philadelphia, Pennsylvania, USA, December 2000.

Contents

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399