Towards Synchronizing Linear Collaborative Objects with Operational Transformation

Abdessamad Imine, Pascal Molli, Gérald Oster, and Michaël Rusinowitch

LORIA-INRIA Lorraine, France
{imine, molli, oster, rusi}@loria.fr

Abstract. A collaborative object represents a data type (such as a text document or a filesystem) designed to be shared by multiple geographically separated users. Data replication is a technology to improve performance and availability of data in distributed systems. Indeed, each user has a local copy of the shared objects, upon which he may perform updates. Locally executed updates are then transmitted to the other users. This replication potentially leads, however, to divergent (*i.e.* different) copies. In this respect, Operational Transformation (OT) algorithms are applied for achieving convergence of all copies, *i.e.* all users view the same objects. Using these algorithms users can apply the same set of updates but possibly in *different orders* since the convergence should be ensured in all cases. However, achieving convergence with the OT approach is still a critical and challenging issue. In this paper, we address an open convergence problem when the shared data has a linear structure such as list, text, ordered XML tree, etc. We analyze the source of this problem and we propose a generic solution with its formal correctness.

1 Introduction

Generally users involved in collaborative and mobile environments work on replicas of shared data. During disconnection periods, they can concurrently execute updates on replicas. This potentially leads to divergent replicas (*i.e.* different states). One of the main issues in such environments is how to maintain *consistency* (or convergence) among replicas after reconnection. Originating from real-time groupware research [2], the *Operational Transformation* (OT) approach provides an interesting solution [3,10]. Using this approach, after reconnection, a user A might get an operation op previously executed during disconnection by some other user B on replica of the shared data. User A does not necessarily integrate op by executing it as is on a replica. Instead, it might execute a variant of op, op' – called a *transformation* of op – that intuitively intends to achieve the same effect as op. When the transformed operations are executed, they create the illusion that all operations were executed in the intended execution context and in the intended order. Compared to other replication systems [12], the advantages of this approach are: (i) *it enables an unconstrained concurrency*, *i.e.* it requires no global order on concurrent operations unlike traditional consistency criteria such as linearizability [4]; (ii) *it transforms operations to run in any order*

F. Wang (Ed.): FORTE 2005, LNCS 3731, pp. 411–427, 2005.

even when they do not naturally commute; (iii) *it produces a convergence state that precisely preserves the intentions of all the operations executed during disconnection periods*. Many collaborative applications are based on OT approach such as CoWord [18] (a collaborative word processor) and CoPowerPoint [15] (a real-time collaborative multimedia slides creation and presentation system).

The OT approach consists of application-dependent transformation algorithm. Thus, for every possible pair of concurrent operations, the application programmer has to define in advance how to merge these operations regardless of reception order. According to Ressel et al. [11], the OT algorithm needs to fulfill two conditions (which will be detailed in Section 2) in order to ensure convergence. Finding such an OT algorithm and proving that it satisfies the convergence conditions was always considered as a very hard task, because this proof is often difficult – even impossible – to produce by hand and unmanageably complicated [16]. To overcome this problem, we have proposed a formal framework to assist the development of correct OT algorithms by using a theorem prover [6,7].

However, although in theory [11], OT approach is able to achieve convergence in the presence of *arbitrary transformation orders*, some types of collaborative object still represent a serious handicap as for the application of the OT approach. Indeed, the convergence property has never been achieved when the collaborative object has a *linear structure* (such as list, text or ordered XML tree) and all proposed OT algorithms [2,11,17,14,5,8] fail to meet this property. In this paper, we analyse thoroughly the source of these failures and we propose an OT algorithm that ensures the convergence. Unlike previous works we have been able to completely give formal proof of its correctness by using a theorem prover. Furthermore, our OT algorithm is generic because it can be applied to any linear structure-based data.

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 the ingredients of our solution giving examples and proofs of correctness. The ingredients of our formalization for the linear collaborative object into a theorem prover language are given in Section 5. Section 6 discusses related work, and section 7 summarizes conclusions.

2 Operational Transformation Approach

2.1 The Model

OT considers n sites, where each site has a copy of the collaborative object. The collaborative object model we take is a *text object* modeled by a sequence of characters, where the position of its first character is zero. It is assumed that the text state 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 model is only an abstract view of many collaborative object 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 $st \odot op = st'$ when an editing operation op is executed on the text state st and produces text state st'. We say that op is *generated* on state st. Notation $[op_1; op_2; \ldots; op_n]$ represents an operation sequence. Applying an operation sequence to a text state st is recursively defined as follows: (i) $st \odot [] = st$, where [] is the empty sequence and; (ii) $st \odot [op_1; op_2; \ldots; op_n] = (((st \odot op_1) \odot op_2) \ldots) \odot op_n$. Two operation sequences seq_1 and seq_2 are *equivalent*, denoted $seq_1 \equiv seq_2$, if $st \odot seq_1 = st \odot seq_2$ for all text states st.

To detect concurrency, we assume that there exists a Lamport's "happens before" partial ordering between the operations [12]. How this ordering relation is expressed is beyond the scope of this paper.

OT is an optimistic replication which lets many users concurrently update the shared data and next it synchronizes their divergent replicas in order to obtain the same data. The operations of each site are executed on the local replica immediately without being blocked or delayed, and then are propagated to other sites to be executed again. Accordingly, every operation is processed in four steps: (i) *generation* on one site; (ii) *broadcast* to other sites; (iii) *reception* on other sites; (iv) *execution* on other sites.

In the following, we give the conflict relation between two insert operations:

Definition 1. ***(Conflict Relation)*** *Two insert operations* $op_1 = Ins(p_1, c_1)$ *and* $op_2 = Ins(p_2, c_2)$*, generated on different sites,* conflict *with each other iff: (i)* op_1 *and* op_2 *are generated on the same text state; and, (ii)* $p_1 = p_2$*, i.e. they have the same insertion position.*

2.2 Transformation Principle

One of the significant issues when designing collaborative objects with a replicated architecture and an arbitrary communication of messages between sites is the *consistency maintenance* (or *convergence*) of all replicas. To illustrate this problem, consider the following example:

Example 1. Consider the following group text editor scenario (see Figure 1): there are two users (sites) working on a shared document represented by a sequence of characters. These characters are addressed from 0 to the end of the document. Initially, both copies hold the string "`efecte`". User 1 executes operation $op_1 = Ins(1, \text{"}\texttt{f}\text{"})$ to insert the character "`f`" at position 1. Concurrently, user 2 performs $op_2 = Del(5)$ to delete the character "`e`" at position 5. When op_1 is received and executed on site 2, it produces the expected string "`effect`". But, when op_2 is received on site 1, it does not take into account that op_1 has been executed before it and it produces the string "`effece`". The result at site 1 is different from the result of site 2 and it apparently violates the intention of op_2 since the last character "`e`", which was intended to be deleted, is still present in the final string. Consequently, we obtain a *divergence* between sites 1 and 2. It should be pointed out that even if a serialization protocol [2] was used to require that all sites execute op_1 and op_2 in the same order (*i.e.* a global order

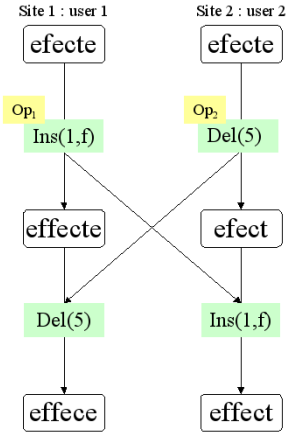

Fig. 1. Incorrect integration

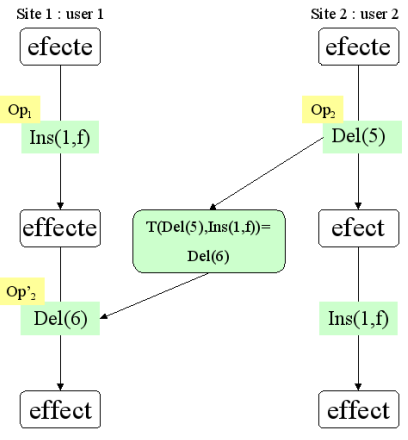

Fig. 2. Integration with transformation

on concurrent operations) to obtain an identical result "`effece`", this identical result is still inconsistent with the original intention of op_2.

To maintain convergence, an OT approach has been proposed by Ellis and Gibbs [2] where a user X might get an operation op that was previously executed by some other user Y on the replica of the shared object. User X does not necessarily integrate op by executing it as it is on its replica. Instead, he might execute a variant of op, denoted by op' (called a *transformation* of op) that *intuitively intends to achieve the same effect as op*. This approach is based on an algorithm which takes two concurrent operations that are defined on the same object state. We denote this algorithm by a function T.

Example 2. In Figure 2, we illustrate the effect of T on the previous example. When op_2 is received on site 1, op_2 needs to be transformed according to op_1 as follows: $T((Del(5), Ins(1, \text{“}\texttt{f}\text{”})) = Del(6)$. The deletion position of op_2 is incremented because op_1 has inserted a character at position 1, which is before the character deleted 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: $T(Ins(1, \text{“}\texttt{f}\text{”}), Del(5)) = Ins(1, \text{“}\texttt{f}\text{”})$; op_1 remains the same because "`f`" is inserted before the deletion position of op_2.

In the OT approach, every site is equipped by two main components [2,11]: the *integration component* and the *transformation component*. The integration component is an algorithm which is responsible for receiving, broadcasting and executing operations. It is *independent* of the semantics of the collaborative objects. Several integration algorithms have been proposed in the groupware area, such as dOPT [2], adOPTed [11], SOCT2,4 [14,19] and GOTO [16]. The transformation component is a set of OT algorithms which is responsible for merging two concurrent operations defined on the same state. Every OT algorithm is

specific to the semantics of a collaborative object (text in our example). Every site generates operations sequentially and stores these operations in a data structure called *history*. When a site receives a remote operation op, the integration component executes the following steps:

1. from the local history it determines the sequence seq of operations that are concurrent to op;
2. it calls the transformation component in order to get operation op' that is the transformation of op according to seq;
3. it executes op' on the current state;
4. it adds op' to local history.

In this paper, we only deal with the design of OT algorithm for collaborative objects which have linear structure (such as list, text or ordered XML tree).

2.3 Convergence Conditions

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

$$T^*(op, []) = op \text{ where } [] \text{ is the empty sequence;}$$
$$T^*(op, [op_1; op_2; \ldots; op_n]) = T^*(T(op, op_1), [op_2; \ldots; op_n])$$

Using an OT algorithm requires us to satisfy two conditions [11]. Given two operations op_1 and op_2, let $op_2' = T(op_2, op_1)$ and $op_1' = T(op_1, op_2)$, the conditions are as follows:

- **Condition** C_1: $st \odot [op_1 ; op_2'] = st \odot [op_2 ; op_1']$, for every object state st.
- **Condition** C_2: if $[op_1; op_2'] \equiv [op_2; op_1']$ then $T^*(op, [op_1; op_2']) = T^*(op, [op_2; op_1'])$.

C_1 defines a *state identity* and ensures that if op_1 and op_2 are concurrent, the effect of executing op_1 before op_2 is the same as executing op_2 before op_1. This condition is necessary but not sufficient when the number of concurrent operations is greater than two. As for C_2, it ensures that transforming op along equivalent and different operation sequences will give the same operation. In previous work [11,9], the authors have proved that conditions C_1 and C_2 are sufficient to ensure the convergence property for *any number* of concurrent operations which can be executed in *arbitrary order*.

It should be pointed out that verifying that a given OT algorithm verifies C_1 and 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 collaborative objects based on OT approach [5–7].

3 Convergence Problems

In order to illustrate the convergence problems encountered in building OT algorithm for linear collaborative objects, we present a well known transformation

algorithm designed by Ellis and Gibbs [2] who are the pioneers of the OT approach. This algorithm is used to synchronize a collaborative text object, shared by two or more users. There are two editing operations: $Ins(p, c, pr)$ to insert a character c at position p and $Del(p, pr)$ to delete a character at position p. Operations Ins and Del are extended with another parameter pr[1]. This one represents a priority scheme that is used to solve a conflict occurring when two concurrent insert operations were originally intended to insert different characters at the same position. In Figure 3, we give the four transformation cases for Ins and Del proposed by Ellis and Gibbs. There are two interesting situations in the first case. The first situation is when the arguments of the two insert operations are equal (*i.e.* $p_1 = p_2$ and $c_1 = c_2$). In this case the function T returns the idle operation Nop that has a null effect on text state [2]. The second interesting situation is when only the insertion positions are equal (*i.e.* $p_1 = p_2$). Such conflicts are resolved by using the priority order associated with each insert operation. The insertion position will be shifted to the right $(p_1 + 1)$ when Ins has a higher priority. The remaining cases of T are quite simple.

Using our theorem-proving approach [5,6], we have detected that the function T of Figure 3 contains some not obvious bugs that lead to divergence situations. These situations are detailed in the following.

3.1 Violation of C_1

The scenario violating C_1 is depicted in Figure 4 (for clarity we have omitted the priority parameter). There are two users: (i) $user_1$ inserts x in position 1 (op_1) while $user_2$ concurrently deletes the character at the same position (op_2). (ii) When op_2 is received by site 1, op_2 must be transformed according to op_1. So $T(Del(1), Ins(1, x))$ is called and $Del(2)$ is returned. (iii) In the same way, op_1 is received on site 2 and must be transformed according to op_2. $T(Ins(1, x), Del(1))$ is called and returns $Ins(0, x)$. Condition C_1 is violated. Accordingly, the final results on both sites are different.

The error comes from the definition of $T(Ins(p_1, c_1, pr_1), Del(p_2, pr_2))$. The condition $p_1 < p_2$ should be rewritten $p_1 \leq p_2$. This modification is sufficient to satisfy the condition C_1.

3.2 Violation of C_2

Even having corrected the previous error, we have detected that condition C_2 is not satisfied. Figure 5 presents a scenario for C_2 violation. In this scenario $seq = [op_2; op'_3]$ and $seq' = [op_3; op'_2]$ are two equivalent sequences. Using the function T of Figure 3 we must have $T(op_1, seq) = T(op_1, seq')$:

$$T^*(op_1, seq) = op'_1 = T(T(op_1, op_2), op'_3) = Ins(2, x)$$
$$T^*(op_1, seq') = op''_1 = T(T(op_1, op_3), op'_2) = Ins(3, x)$$

[1] This priority is the site identifier where operations have been generated. Two operations generated from different sites have always different priorities.

[2] The definition of T is completed by: $T(Nop, op) = Nop$ and $T(op, Nop) = op$ for every operation op.

```
T(Ins(p1, c1, pr1), Ins(p2, c2, pr2)) =
if p1 < p2 then return Ins(p1, c1, pr1)
elseif p1 > p2 then return Ins(p1 + 1, c1, pr1)
    elseif c1 == c2 then return Nop()
        elseif pr1 > pr2 then return Ins(p1 + 1, c1, pr1)
            else return Ins(p1, c1, pr1)
endif;

T(Ins(p1, c1, pr1), Del(p2, pr2)) =
if p1 < p2 then return Ins(p1, c1, pr1)
else return Ins(p1 − 1, c1, pr1)
endif;

T(Del(p1, pr1),Ins(p2, c2, pr2)) =
if p1 < p2 then return Del(p1, pr1)
else return Del(p1 + 1, pr1)
endif;

T(Del(p1, pr1),Del(p2, pr2)) =
if p1 < p2 then return Del(p1, pr1)
elseif p1 > p2 then return Del(p1 − 1, pr1)
    else return Nop()
endif;
```

Fig. 3. Transformation function defined by Ellis and Gibbs [2]

As we can see, $op'_1 \neq op''_1$, C_2 is violated; and therefore the convergence is not achieved. The scenario illustrated in Figure 5 is called C_2 *puzzle*.

3.3 Analyzing the Problem

C_2 is considered as particularly difficult to satisfy. To better understand the source of this problem, we consider the previous scenario violating C_2 (see Figure 5). There are three concurrent operations $op_1 = Ins(3, x)$, $op_2 = Del(2)$ and $op_3 = Ins(2, y)$ where the insertion positions (*i.e.* $Pos(Ins(p, c, pr)) = p$) initially have the following relation: $Pos(op_1) > Pos(op_3)$.

According to Definition 1, op_1 and op_3 are not in conflict. In this scenario we have two equivalent operation sequences $S_1 = [op_2; op'_3]$ and $S_2 = [op_3; op'_2]$ where $op'_3 = T(op_3, op_2)$ and $op'_2 = T(op_2, op_3)$. The above relation between op_1 and op_3 is not preserved when transforming op_1 along sequence S_1 since $Pos(T(op_1, op_2)) = Pos(op'_3)$.

The transformation process may lead to two concurrent insert operations (with different original insertion positions) to get into a *false conflict situation* (to have the same insertion position). 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.

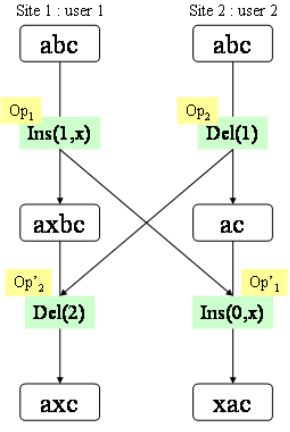

Fig. 4. Scenario violating C_1

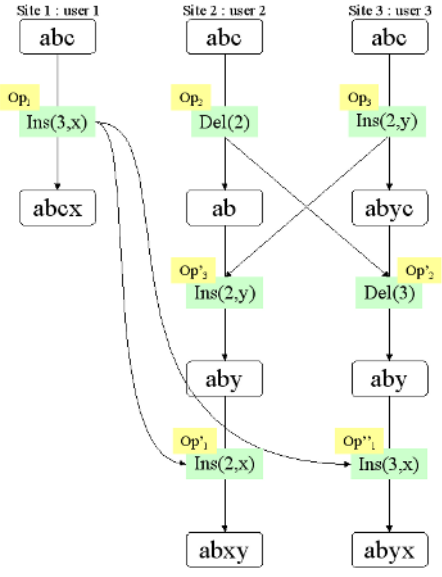

Fig. 5. Scenario violating C_2

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 how this function resolves the divergence problem. Finally, we will show the correctness of our approach.

4.1 Position Words

For any set of symbols Σ called an *alphabet*, Σ^* denotes the set of *words* over Σ. The empty word is denoted by 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. For every $\omega \in \Sigma^*$, such that $|\omega| > 0$, we denote $Base(\omega)$ (resp. $Top(\omega)$) the *last* (resp. *first*) symbol of ω. Thus, $Top(abcde) = a$ and $Base(abcde) = e$. We assume that Σ is totally ordered and denote the strict part of this order by $>$. 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 $Top(u)$ precedes $Top(v)$ in the alphabetic order.

Definition 2. ***(p-word)*** *We consider the natural numbers* $\mathbb{N}$ *as an alphabet. We define the set of p_words* $\mathcal{P} \subset \mathbb{N}^*$ *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* $n - Top(\omega) \in \{0, 1, -1\}$.

We observe immediately that we can concatenate two p-words to get another one if the origin of the first differs of at most 1 from the first letter of the second one:

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* $Base(\omega_1) = Top(\omega_2)$ *or* $Base(\omega_1) = Top(\omega_2) \pm 1$.

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

Definition 3. ***(Equivalence of p-words)*** *The equivalence relation on the set of p_words* $\mathcal{P}$ *is defined by:* $\omega_1 \equiv_{\mathcal{P}} \omega_2$ *iff* $Top(\omega_1) = Top(\omega_2)$ *and* $Base(\omega_1) = Base(\omega_2)$*, where* ω_1*,* $\omega_2 \in \mathcal{P}$.

We can also show that this relation is a congruence using Definitions 2 and 3:

Proposition 1. ***(Right congruence)*** *The equivalence relation* $\equiv_{\mathcal{P}}$ *is a* right congruence*, that is, for all* $\rho \in \mathcal{P}$*:* $\omega_1 \equiv_{\mathcal{P}} \omega_2$ *iff* $\omega_1\rho \equiv_{\mathcal{P}} \omega_2\rho$

4.2 OT Algorithm

In order to preserve the order relation between two insert operations, we propose to keep all different positions occupied by a character during the transformation process. It means that instead of the single position we maintain a stack of positions called a *p-word*. Each time an operation is transformed we push the last position before transformation in the p-word. The size of the stack is proportional to the number of concurrent operations. In Figure 6 we give the details of our new OT function. When two insertion operations insert two different characters at the same position (they are in conflict), a choice has to be made: which character must be inserted before the other? The solution that is generally adopted consists of associating a priority to each character (*i.e.*, the character's code or the site identifier). In our OT function, when a conflict occurs, the character whose code $Code(c)$ is the highest is inserted before the other.

If two p-words are identical it means that the two associated insert operations are equal. Otherwise the p-word allows to track the order relation between the two operations. We shall therefore redefine the insert operation as $Ins(p, c, w)$ where p is the insertion position, c the character to be added and w a p-word. When an operation is generated, the p-word is empty, *i.e.* $Ins(3, x, \epsilon)$. When an operation is transformed and the insertion position is changed, the original position is pushed to the p-word. For example, $T(Ins(3, x, \epsilon), Del(1)) = Ins(2, x, [3])$ and $T(Ins(2, x, [3]), Ins(1, x, \epsilon)) = Ins(3, x, [2 \cdot 3])$.

T($Ins(p_1, c_1, w_1)$,$Ins(p_2, c_2, w_2)$) =
let α_1=$PW(Ins(p_1, c_1, w_1))$ **and** α_2=$PW(Ins(p_2, c_2, w_2))$
if ($\alpha_1 \prec \alpha_2$ **or** ($\alpha_1 = \alpha_2$ **and** $Code(c_1) < Code(c_2)$))
then return $Ins(p_1, c_1, w_1)$
elseif ($\alpha_1 \succ \alpha_2$ **or** ($\alpha_1 = \alpha_2$ **and** $Code(c_1) > Code(c_2)$))
  then return $Ins(p_1 + 1, c_1, p_1 w_1)$
  else return Nop
endif;

T($Ins(p_1, c_1, w_1)$,$Del(p_2)$) =
if $p_1 > p_2$ **then return** $Ins(p_1 - 1, c_1, p_1 w_1)$
elseif $p_1 < p_2$ **then return** $Ins(p_1, c_1, w_1)$
  else return $Ins(p_1, c_1, p_1 w_1)$
endif;

T($Del(p_1)$,$Del(p_2)$) =
if $p_1 < p_2$ **then return** $Del(p_1)$
elseif $p_1 > p_2$ **then return** $Del(p_1 - 1)$
  else return Nop
endif;

T($Del(p_1)$,$Ins(p_2, c_2, w_2)$) =
if $p_1 < p_2$ **then return** $Del(p_1)$
else return $Del(p_1 + 1)$
endif;

Fig. 6. New OT function

We define a function PW which enables to construct p-words from editing operations. It takes an operation as parameter and returns its p-word:

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

$$PW(Del(p)) = p$$

Figure 7 shows how the p-words solve the C_2 puzzle depicted in Figure 5. When op_1 is transformed according to op_3, $3 > 2$, so op_1 is inserted after op_3. This order relation must be preserved when $op'_1 = T(Ins(3, x, \epsilon), Del(2)) = Ins(2, x, [3])$ will be transformed according to op'_3. To preserve the relation detected between op_1 and op_3, we must observe $PW(op'_1) \succ PW(op'_3)$. As $[2; 3] \succ [2; 2]$ is true, the order relation is preserved.

There is still a problem. This solution leads to the convergence (*i.e.* the same states), but C_2 is not respected. Indeed, we can verify in Figure 7 that:

$$T^*(op_1, [op_2; op'_3]) \neq T^*(op_1, [op_3; op'_2])$$

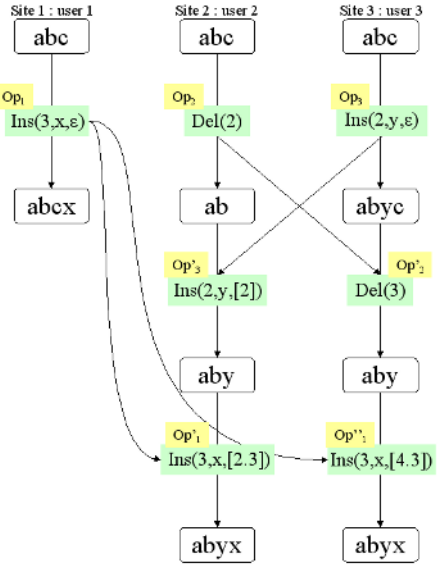

Fig. 7. Correct execution of C_2 puzzle

When two identical insertions operations are transformed according to two equivalent operation sequences, their p-words may get different. If they are different, they can be considered as equivalent if the top and the base of their p-words are equal. With the equivalence of p-words, we give the equivalence of two editing operations.

Definition 4. ***(Operation equivalence)*** *Given two editing operations op_1 and op_2, we say that op_1 and op_2 are equivalent and we denote it also by $op_1 \equiv_{\mathcal{P}} op_2$ iff one of the following conditions holds: (i) $op_1 = Ins(p_1, c_1, w_1)$, $op_2 = Ins(p_2, c_2, w_2)$, $c_1 = c_2$ and $PW(op_1) \equiv_{\mathcal{P}} PW(op_2)$; (ii) $op_1 = Del(p_1)$, $op_2 = Del(p_2)$ and $p_1 = p_2$.*

With the above operation equivalence we propose a weak form of the condition C_2 that still ensures the state convergence. This condition is called C_2'.

Definition 5. ***(Condition C_2')*** *For every editing operations op, op_1 and op_2, if the function T satisfies C_1 then:*

$$T^*(op, [op_1\,;\, T(op_2, op_1)]) \equiv_{\mathcal{P}} T^*(op, [op_2\,;\, T(op_1, op_2)])$$

4.3 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. the original relation between two insert operations is preserved by transformation;
3. the conditions C_1 and C_2' are satisfied.

Most of the proofs have been automatically checked by the theorem prover SPIKE [1].

Let $Char$ be the set of characters. We define the set of editing operations as follows: $\mathcal{O} = \{Ins(p, c, w) \mid p \in \mathbb{N} \text{ and } c \in Char \text{ and } w \in \mathcal{P}\} \cup \{Del(p) \mid p \in \mathbb{N}\}$.

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

Proof. 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 $Code(c_1) < Code(c)$) then $op'_1 = op_1$;
   - if $\alpha_1 \succ \alpha_2$ or ($\alpha_1 = \alpha_2$ and $Code(c_1) > Code(c)$) then $op'_1 = Ins(p_1 + 1, c_1, p_1 w_1)$ and $p_1 w_1$ is a suffix of $PW(op'_1)$;
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 states that the extension of our OT function to sequences, *i.e.* T^*, does not lose any information about position words.

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

Proof. By induction on n, the length of seq.

- *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 $seq = [seq'; op]$ where seq' is a sequence of length n and $op \in \mathcal{O}$. We have $T^*(op_1, [seq'; op]) = T(T^*(op_1, seq'), op)$. By 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 the suffix relation, we conclude that $PW(op_1)$ is a suffix of $PW(T^*(op_1, seq))$ for every sequence of operations seq. □

Position Relations. We can use the position relations between insert operations as an *invariant* which must be preserved when these operations are transformed and executed in all remote sites.

Lemma 2. *Given two concurrent insert operations op_1 and op_2. For every editing operation $op \in \mathcal{O}$ such that $op \neq op_1$ and $op \neq op_2$: $PW(op_1) \prec PW(op_2)$ implies $PW(T(op_1, op)) \prec PW(T(op_2, op))$*

Proof. We have to consider two cases: $op = Ins(p, c, w)$ and $op = Del(p)$.

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:* $PW(op_1) \prec PW(op_2)$ *implies* $PW(T^*(op_1, seq)) \prec PW(T^*(op_2, seq))$.

Proof. By induction on the length of *seq*. □

Convergence Properties. Recall that the condition C_2' is a relaxed form of C_2. Indeed C_2' means that transforming an operation along two equivalent operation sequences will not give the same result but two equivalent operations. 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. The complete proofs of Theorems 4 and 5 below have been automatically checked by the theorem prover SPIKE. Due to lack of space we only give some representatives cases of the proofs.

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 object state st we have:* $st \odot [op_1; T(op_2, op_1)] = st \odot [op_2; T(op_1, op_2)]$.

Proof. Consider the following case: $op_1 = Ins(p_1, c_1, w_1)$, $op_2 = Ins(p_2, c_2, w_2)$ and $PW(op_1) \prec PW(op_2)$. 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 ($op_2' = T(op_2, op_1) = Ins(p_2 + 1, c_1, p_2 w_2)$) 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 object state gives also the same object state. □

Theorem 5 shows that our OT function also satisfies C_2'. This theorem means that if T satisfies condition C_1 then when transforming op_1 against two equivalent sequences $[op_2; T(op_3, op_2)]$ and $[op_3; T(op_2, op_3)]$ we will obtain two equivalent operations according to Definition 4.

Theorem 5. ***(Condition*** C_2'***).*** *If the OT function* T *satisfies* C_1 *then for all* op_1, op_2, $op_3 \in \mathcal{O}$ *we have:* $T^*(op_1, [op_2; T(op_3, op_2)]) \equiv_{\mathcal{P}} T^*(op_1, [op_3; T(op_2, op_3)])$.

Proof. 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 6), 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 equivalent. □

5 Formal Specification

For modelling the structure and the manipulation of data in programs, *abstract data types* (ADTs) are frequently used [20]. Indeed, the *structure* of data is reflected by so called *constructors* (*e.g.*, 0 and successor $s(x)$, meaning $x + 1$, may construct the ADT *nat* of natural numbers). Moreover, all (potential) data are covered by the set of *constructors terms*, exclusively built by constructors. An ADT may have different *sorts*, each characterized by a separate set of constructors. Furthermore, the *manipulation* of data is reflected by *function symbols* (*e.g.*, *plus* and *minus* on *nat*). The value computed by such functions are specified by *axioms*, usually written in equational logic. An *algebraic specification* is a description of one or more such abstract data types [20].

5.1 Collaborative Object Specification

More formally a collaborative object can be considered as a structure of the form $G = (O, T)$ where O is the set of operations applied to the object and T is the transformation function. In our approach, we construct an algebraic specification from a collaborative object. We define a sort Opn for the operation set O, where each operation serves as a constructor of this sort. These constructors are as follows: (i) $Ins(p, c, \omega)$ inserts element c at position p, (ii) $Del(p)$ deletes the element at position p.

We use the *List* ADT for specifying a linear collaborative object. The *List* ADT has two constructors: (i) $\langle\rangle$ (*i.e.*, an empty list); (ii) $l \circ x$ (*i.e.*, a list composed by an element x added to the end of the list l). The data type of *List*'s elements is only a template and can be replaced by each type needed. For instance, an element may be regarded as a character, a paragraph, a page, an XML node, etc. Because all operations are applied to the object structure in order to modify it, we give the following function: $\odot : List \times Opn \rightarrow List$. All appropriate axioms of the function $\odot$ describe the transition between the object states when applying an operation. For example, the operation $Del(p)$ changes *List* as follows:

$$l \odot Del(p) = \begin{cases} \langle\rangle & \text{if } l = \langle\rangle \\ l & \text{if } l = l' \circ c \text{ and } p \geq |l| \\ l' & \text{if } l = l' \circ c \text{ and } p = |l| - 1 \\ (l' \odot Del(p)) \circ c & \text{if } l = l' \circ c \text{ and } p < |l| - 1 \end{cases}$$

where $|l|$ returns the length of the list l.

In the same way, we define $Ins(p, c)$ modifications below:

$$l \odot Ins(p, c, \omega) = \begin{cases} \langle\rangle & \text{if } l = \langle\rangle \text{ and } p \neq 0 \\ l \circ c & \text{if } l = \langle\rangle \text{ and } p = 0 \\ (l' \odot Ins(p, c, \omega)) \circ d & \text{if } l = l' \circ d \text{ and } p < |l| \\ (l' \circ d) \circ c & \text{if } l = l' \circ d \text{ and } p = |l| \\ l & \text{if } l = l' \circ d \text{ and } p > |l| \end{cases}$$

An OT algorithm is defined by the following function: $T : Opn \times Opn \rightarrow Opn$. It takes two operation arguments. For example, the following transformation:

$T(Del(p_1),Ins(p_2,c_2,\omega_2)) =$ **if** $p_1 \geq p_2$ **then return** $Del(p_1+1)$ **else return** $Del(p_1)$

is defined by two conditional equations:

$$p_1 \geq p_2 \Longrightarrow T(Del(p_1), Ins(p_2,c_2,\omega_2)) = Del(p_1+1)$$
$$p_1 \not\geq p_2 \Longrightarrow T(Del(p_1), Ins(p_2,c_2,\omega_2)) = Del(p_1)$$

This example illustrates how it is easy to translate a transformation function into conditional equations. This task is straightforward and can be done mechanically.

We now express the convergence conditions as theorems to be proved in our algebraic setting. Both convergence conditions C_1 and C_2 are formulated as follows:

Theorem 6. (Condition C_1) $\forall op_1, op_2 \in Opn$ *and* $\forall st \in List$:

$$(st \odot op_1) \odot T(op_2, op_1) = (st \odot op_2) \odot T(op_1, op_2).$$

Theorem 7. (Condition C_2) $\forall op_1, op_2, op \in Opn$:

$$T(T(op, op_1), T(op_2, op_1)) = T(T(op, op_2), T(op_1, op_2)).$$

5.2 The Theorem Prover: SPIKE

To automatically check the convergence conditions C_1 and C_2 we have used SPIKE [1], an automated induction-based theorem prover. SPIKE was employed for the following reasons: (i) its high automation degree; (ii) its ability to perform case analysis (to deal with multiple methods and many transformation cases); (iii) its ability to find counter-examples; (iv) its incorporation of *decision procedures* (to automatically eliminate arithmetic tautologies produced during the proof attempt) [13].

6 Related Work

Several techniques have been proposed to address C_2. These may be categorized as follows.

The first approach tries to avoid the 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, the SOCT4 algorithm [19] 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 [16].

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 satisfy the C_2 condition. This approach has been developed in adOPTed [11], SOCT2 [14], and GOT [17]. Unfortunately, we have proved elsewhere [5] that all previously proposed transformation functions fail to satisfy this condition.

Recently, Li et al. [8] 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. Let $op_1 =$

$Ins(p+1,x)$, $op_2 = Ins(p,z)$ and $op_3 = Del(p)$ be three concurrent operations generated on sites 1, 2 and 3 respectively. They use a function β that computes for every editing operation the original position according to the initial object state. For this case, it is possible to get: $\beta(op_1) = \beta(op_2) = \beta(op_3)$ whereas $Pos(op_1) > Pos(op_2)$. When a conflict occurs Li et al. use the site identifier to reorder the character to be inserted (see our OT function in Figure 6). Consider the following sequences:

$$S1 = [op_1; T(op_3, op_1)] = [Ins(p+1,z); Del(p)]$$
$$S2 = [op_3; T(op_1, op_3)] = [Del(p); Ins(p,z)]$$

Transforming op_2 against $S1$ does not give the same operation that transforming op_2 against $S2$. This case leads to divergence problem. Note that $Pos(T(op_2, op_3)) = Pos(T(op_1, op_3))$. Thus op_1 and op_2 lose their original relation after transformation according to op_3. The mistake is due to the definition of their β function. Indeed, their definition relies on the exclusion transformation function ET, which is the reversed function of T. For instance, if $T(op_1, op_2) = op_1'$ then $ET(op_1', op_2) = op_1$. Due to the non-inversibility of T, ET is not always defined [16]. Consequently, the convergence property cannot be achieved in all cases.

7 Conclusion

OT has a great potential for generating non-trivial states 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 OT algorithms used to synchronize linear collaborative objects (such as document text or XML trees) and we have proposed a solution based on a weak form of the condition C_2. Using our theorem-proving approach [5,6] we have provided a complete proof for our OT algorithm. Furthermore, our solution is generic because it can be applied to any linear structure-based data.

Although this weak form still ensures the convergence state, we cannot plug our OT algorithm in all integration algorithms based on the condition C_2, such as adOPTed [11] and SOCT2 [14]. So, we consider our work as a first step towards to build a generic integration algorithm based only on conditions C_1 and C_2'. Moreover, we plan to optimize our OT algorithm because the size of p-words increase according to the number of transformation steps.

References

1. A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated Mathematical Induction. *Journal of Logic and Computation*, 5(5):631–668, 1995.
2. C. A. Ellis and S. J. Gibbs. Concurrency Control in Groupware Systems. In *SIGMOD Conference*, volume 18, pages 399–407, 1989.
3. R. Guerraoui and C. Hari. On the consistency problem in mobile distributed computing. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 51–57. ACM Press, 2002.
4. M. P. Herlihy and J. M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990.

5. 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.
6. A. Imine, P. Molli, G. Oster, and M. Rusinowitch. Deductive verification of distributed groupware systems. In *Algebraic Methodology and Software Technology, 10th International Conference, AMAST 2004*, volume 3116 of *Lecture Notes in Computer Science*, pages 226–240. Springer, 2004.
7. A. Imine, M. Rusinowitch, G. Oster, and P. Molli. Formal design and verification of operational transformation algorithms for copies convergence. *Theoretical Computer Science*, to appear (2005).
8. 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.
9. B. Lushman and G. V. Cormack. Proof of correctness of ressel's adopted algorithm. *Information Processing Letters*, 86(3):303–310, 2003.
10. 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.
11. 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.
12. Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.
13. S. Stratulat. A general framework to build contextual cover set induction provers. *Journal of Symbolic Computation*, 32(4):403–445, 2001.
14. 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.
15. C. Sun. The copowerpoint project. `http://reduce.qpsf.edu.au/copowerpoint/`, 2004.
16. 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.
17. 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.
18. D. Sun, S. Xia, C. Sun, and D. Chen. Operational transformation for collaborative word processing. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 437–446, New York, NY, USA, 2004. ACM Press.
19. 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.
20. M. Wirsing. Algebraic Specification. *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 675–788, 1990.