Formal Verification of Operational Transformation

Yang Liu, Yi Xu, Shao Jie Zhang, and Chengzheng Sun

Nanyang Technological University

Abstract. Operational Transformation (OT) is a technology to provide consistency maintenance and concurrency control in real-time collaborative editing systems. The correctness of OT is critical due to its foundation role in supporting a wide range of real world applications. In this work, we formally model the OT-based collaborative editing systems and establish their correctness, w.r.t. convergence and intention preservation, using a set of well-defined transformation conditions and properties. We then use model checking to verify the transformation properties for basic data and operational models. To the best of our knowledge, this is the first work to conduct a complete verification of OT including control algorithms and transformation functions. Our evaluation confirmed the correctness of existing OT systems and transformation functions with important discoveries.

1 Introduction

Real-time collaborative editing systems allow multiple users to edit shared documents and see each other's updates instantly over the Internet. One major challenge in building collaborative editing systems is consistency maintenance of shared documents in the face of concurrent editing by multiple users. Operational Transformation (OT) was invented to address this challenge [4,11]. Due to its non-blocking, fine-grained concurrency, and unconstrained interaction properties, OT is particularly suitable in high-latency networking environments like the Internet, and has been increasingly adopted in real-world industrial collaborative applications, e.g., Google Wave/Docs, Codoxware, IBM OpenCoWeb and Novell Vibe. As OT is increasingly applied to a wider range of real-world applications and used by more and more people, verifying and ensuring the correctness of its core algorithms become more and more important.

One major challenge in OT verification is the highly dynamic behavior and infinite possibilities of a real-time collaborative editing session: there may exist an arbitrary number of participating users (or sites); each user may generate an arbitrary number of operations; operations may have arbitrary causality/concurrency relationships; each operation may be performed on an arbitrary object in a shared document; the shared document may contain an arbitrary number of data objects. These arbitrary parameters and their combinations result in an infinite number of possible states, which hinders the application of formal methods in both modeling and verification of collaborative editing systems. Past research has directly modeled a collaborative editing session with these arbitrary parameters and used a model checker to explore this inherently infinite state space, which has inevitably encountered the exponential state explosion problem and failed to verify OT correctness (more details in Section 7).

In this paper, we propose a novel OT verification approach based on the following key insights: OT system correctness is defined by a set of well-defined transformation conditions and properties, which are collectively preserved by two core OT components: control algorithms and transformation functions. Following the divide-and-conquer strategy, we apply different techniques (mathematic proof and model checking) to verify different aspects of an OT system. Mathematic induction is used to reduce the problem of verifying OT system correctness under arbitrary operations with arbitrary relationships to the problem of verifying transformation properties under a few operations with simple relationships. Model checking is then used to verify transformation properties under the string data and character operation models of a collaborative editing session. To avoid the state explosion problem, we propose a data abstraction to reduce the verification of arbitrary document sizes and operation positional relations to a finite number of verification cases, which can be automatically and efficiently checked by a model checker. Based on this approach, we have completely verified the OT system without suffering from the state explosion problem. Our approach makes important discoveries on OT system correctness, which were never reported in prior work. In this paper, we report our verification techniques, results and discoveries.

2 Collaborative Editing and OT Systems

2.1 Consistency Requirements in Collaborative Editing

A real-time collaborative editing system is a distributed system that supports human-computer-human interaction and collaboration on shared documents over geographically dispersed collaborating sites. Each collaborating site consists of a human user, an editor and a local replica of the shared document. A user may freely interact with the local editor to view and edit the local replica of the shared document. Operations generated by a user are immediately performed on the local replica to provide quick response to the user; then, local operations are broadcast to all remote collaborating sites and performed there to ensure consistency of multiple replicas of the shared document.

One main challenge in supporting collaborative editing is document consistency maintenance under the following constraints: (1) fast local response: local operations should be responded as quickly as a single-user editor; (2) unconstrained interaction: a user may edit any object in the shared document at any time, i.e., non-blocking, no locking, and no turn-taking among users; (3) real-time and fine-grained operation propagation: users can see each other's edits instantly as they are typing, constrained only by network latency. Three consistency requirements have been identified [4,11,12]:

Causality preservation operations generated in a collaboration session by different users must be executed in their causal order.

Convergence all replicas of a shared document must be identical after the same group of operations have been executed on them.

Intention preservation the effect of executing an operation O on all replicas must be the same as the intention of O, which is defined as the effect of executing O on the local replica from which O was generated.

These three consistency requirements are general for all collaborative editing systems, and can be achieved by using any suitable distributed computing techniques. Causality preservation is commonly achieved by adopting well-established distributed computing techniques, such as vector-clock-based timestamping or central server-based propagation, which are assumed by OT systems and hence not discussed further in this paper. Both convergence and intention preservation can be achieved by using OT techniques, and OT is able to achieve both requirements under the requirements of fast local response, unconstrained interaction, real-time fine-grained operation propagation in collaborative editing [11,12]. This paper aims to provide a formal specification and verification of these two requirements for OT-based collaborative editing systems. In the following discussions, we make no assumption on the communication topology among collaborating sites, e.g., operation broadcasting may be achieved by using a center server or full connections among all sites. However, we assume operation propagation among collaborating sites is reliable, which can be achieved by using standard communication protocols like TCP in the Internet.

2.2 Basic Ideas for Consistency Maintenance in OT

The basic idea of OT consistency maintenance is to transform an operation according to the effect of concurrent operations so that the transformed operation can achieve the correct effect and maintain document consistency. This idea can be illustrated in a Space-Time-Diagram (STD) in Fig. 1. The initial document contains a string "abcd", replicated at two collaborating sites; two operations: $O_1 = D(1, b)$ (to delete "b" at position 1), and $O_3 = D(2, d)$ (to delete "d" at position 2) are sequentially generated at site 1; one operation: $O_2 = I(3, e)$ (to insert "e" at position 3) is concurrently generated at site 2. At site 1, O_1 and O_3 are executed sequentially to get the document "ac". Then, O_2 arrives and is transformed (by using a transformation function T) against O_1 and O_3 in sequence, i.e., $T((T(O_2, O_1), O_3) = I(2, e)$, whose position is decremented to compensate the left-shifting effect by O_1 (but O_3 has no impact on O_2). Executing $I(2, e)$ on "ac" inserts "e" at the end of document to get a new state "ace". If O_2 were executed in its original form, an error would occur as its original insert position 3 is beyond the current document length.

At site 2, O_2 is first executed as-is to get the document "abced". Then O_1 arrives and is transformed against O_2, i.e., $T(O_1, O_2) = D(1, b)$, which makes no change because O_2 has no impact on O_1. Executing $D(1, b)$ on "abced" deletes "b" at position 1 and get the document state "aced". When O_3 arrives, it cannot be transformed directly against O_2 as they were defined on different document states: O_2 was generated from the initial document state, whereas O_3 was generated after O_1 had been executed on the initial document state. The correct way of transforming O_3 is: $T(O_3, T(O_2, O_1)) = D(3, d)$. Executing $D(3, d)$ on "aced" deletes "d" at the end of document to get a new state "ace", which is identical to the state at site 1 and also preserves the original effects of all operations. If O_3 were directly transformed against O_2, i.e., $T(O_3, O_2) = D(2, d)$, the transformation function T would not be able to correctly detect the position shifting effect of O_2 on O_3 and hence fail to change the position of O_3. This scenario is a well-known OT (design) bug [11], which illustrates one intricate OT complication with only three operations, just imaging the complexity under an arbitrary number of operations with arbitrary concurrent and positional relationships in a collaborative editing session.

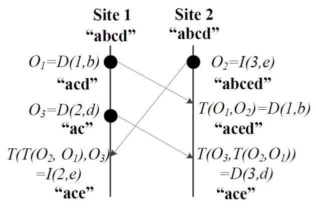

Fig. 1. A Running Example

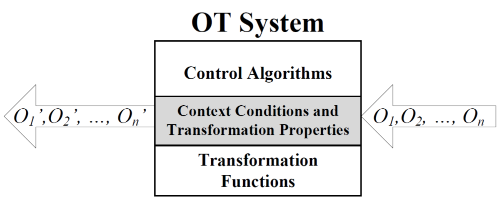

Fig. 2. OT System Overview

2.3 Causal and Concurrent Relations among Operations

Formally, the causal ordering relation of operations is defined as follows [4,6,12].

Definition 1 (Casual Ordering Relation $\rightarrow$). *Given two operations O_1 and O_2 generated at sites i and j, respectively, O_1 is casually before O_2, denoted by $O_1 \rightarrow O_2$, iff (1) $i = j$ and the generation of O_1 happens before the generation of O_2; (2) $i \neq j$ and the execution of O_1 at site j happened before the generation of O_2; or (3) there exists an operation O_x such that $O_1 \rightarrow O_x$ and $O_x \rightarrow O_2$.*

Definition 2 (Concurrent Relation $\|$). *Given two operations O_1 and O_2, O_1 and O_2 are concurrent, denoted by $O_1 \| O_2$, iff neither $O_1 \rightarrow O_2$ nor $O_2 \rightarrow O_1$.*

For example in Fig. 1, O_1 and O_3 are generated at site 1, but the generation of O_1 happens before the generation of O_3, so they are causally ordered, i.e., $O_1 \rightarrow O_3$; O_2 is independently generated from O_1 and O_3, so O_2 is concurrent with both O_1 and O_3, i.e., $O_2 \| O_1$ and $O_2 \| O_3$. It is worth pointing out that causal/concurrent relations are defined only among original operations generated directly by users; transformed operations produced by OT do not have causal/concurrent relationships. To capture essential relationships among all (original and transformed) operations, we need the concept of operation context and context-based conditions, which are defined in the next section.

3 OT System Formalization

3.1 OT Basics

In an OT-based real-time collaborative editing session, the local editor at each site is empowered by an OT system, which takes a sequence of original operations (generated by local or remote users) in their causal orders, and produces a sequence of transformed operations in the same order, as shown in Fig. 2. It is the transformed operations, rather than the original operations, that are actually executed on the local replica of the shared document (a transformed operation may be different from or the same as the original one). The sequences of original operations inputted to OT systems at different sites may have different orders; but the sequences of transformed operations at all sites must produce convergent (identical) and intention-preserved final document states. For example in Fig. 1, original operations are processed in two different sequences: $[O_1, O_3, O_2]$ at site 1 and $[O_2, O_1, O_3]$ at site 2. After processing the two sequences at each site, final document states at both sites are the same and preserve the intentions of all operations.

In this work, we use $\mathcal{O}_o$, $\mathcal{O}_t$ and $\mathcal{O}$ to represent original operations, transformed operations and all possible operations in a given OT system respectively, clearly $\mathcal{O} = \mathcal{O}_o \cup \mathcal{O}_t$. Our modeling and reasoning focus on an (arbitrary) collaborative editing session. We write GO to denote the set of original operations generated by all users during a collaborative editing session. $L = [O_1, \ldots, O_n]$ is used to denote a sequence of operations of size n. $L[i]$ returns the i^{th} operation in L. $L[i, j]$ represents a sub-sequence $[O_i, \ldots, O_j]$ of L. We say that two sequences L_1 and L_2 are equivalent, denoted as $L_1 \equiv L_2$, iff when operations in L_1 and L_2 are sequentially applied on the same initial document state, they produce the same final state. We write $T \in \mathcal{O} \times \mathcal{O} \to \mathcal{O}$ to denote the transformation function, e.g., $T(O_1, O_2)$ transforms operation O_1 according to the impact of operation O_2. We introduce function $LT(O, L)$ to denote repeatedly applying T to transform O against the operations in L sequentially, i.e., $LT(O, L) = T(\ldots T(T(O, L[1]), L[2]), \ldots, L[|L|])$. Given a transformed operation $O \in \mathcal{O}_t$, $org(O)$ represents the original operation of O. For $O \in \mathcal{O}_o$, $org(O) = O$. We write s to denote a document state, which could be concretized if the document model is given (e.g., a string for the string data model in Section 5). The initial document state of a session is denoted by s_0. Given a state s and an operation O, $s' = s \circ O$ represents a new document state s' generated by applying O on s.

In the following, we formally specify the convergence and intention preservation requirements in OT-based collaborative editing systems.

Definition 3 (Convergence and Intention Preservation). *Given a collaborative editing session with the original operation set GO, let L_1 and L_2 be two sequences of causally-ordered original operations from GO at two different sites, and let L_1' and L_2' be the two sequences of transformed operations produced by applying OT to L_1 and L_2, respectively. An OT-based collaborative editor achieves convergence and intention preservation if: (1) Convergence: $L_1' \equiv L_2'$; (2) Intention preservation: the effect of executing any operation O in L_1' or L_2' is the same as the effect of executing the $org(O)$ in the document state from which the $org(O)$ was generated.*

Internally, an OT system consists of two key components [4,7,9,11,12,14]: control algorithms (CA) and transformation functions (TF), as shown in the middle box of Fig. 2. CA determine which operation should be transformed with which other prior operations in the sequence, and invoke TF to perform real transformation between two operations. CA are generic in the sense they work on generic context-based conditions (see Section 3.2) among operations; TF are application-specific such that they work on operation types, positional relations and other parameters.

Past research has identified a set of transformation conditions and properties that must be met collectively by OT internal components [9,11,12,14]. CA are responsible for ensuring context-based conditions; TF often ensure transformation properties (see Section 3.2). In this work, we will prove that if CA and TF can collectively meet those transformation conditions and properties, then the OT system as a whole can meet the convergence and intention preservation requirements as specified in Definition 3.

3.2 Context-Based Conditions and Transformation Properties

In OT systems, every operation O is associated with a context, which represents the document state on which O is defined. The significance of operation context is that it provides a ground for interpreting the effect of an operation and reasoning about the relations among operations. The context of O is represented by a set of original operations that have been executed (after transformation) to create the document state on which O is defined.

Definition 4 (Document Context). *The context of a document state s, denoted as $C(s)$, can be calculated as follows: (1) the context of an initial document state s_0 is represented as an empty set $C(s_0) = \emptyset$; (2) after executing an operation O on the document state s, the context of the new document state $s' = s \circ O$ is represented by: $C(s') = C(s) \cup \{org(O)\}$.*

Definition 5 (Operation Context). *The context of an operation O, denoted as $C(O)$, is calculated as follows: (1) for an original operation O, $C(O) = C(s)$, where s is the document state from which O is generated; (2) for a transformed operation $O' = T(O, O_x)$, $C(O') = C(O) \cup \{org(O_x)\}$.*

For example in Fig. 1, O_1 and O_2 are generated from the same initial document state, so $C(O_1) = C(O_2) = \emptyset$; O_3 is generated after executing O_1 on the document state in site 1, so $C(O_3) = \{O_1\}$; after O_2 is transformed with O_1, $C(O_2') = C(O_2) \cup \{O_1\} = \{O_1\}$, where $O_2' = T(O_2, O_1)$; after O_2' is transformed with O_3, $C(O_2'') = C(O_2') \cup \{O_3\} = \{O_1, O_3\}$, where $O_2'' = T(O_2', O_3)$.

Context-based Conditions (CCs) capture essential requirements for correct operation execution and transformation in OT systems. Six context-based conditions have been identified [14] as listed below:

CC1. Given an operation $O \in \mathcal{O}_o$ and a document state s, where $O \notin C(s)$, O can be transformed for execution on document state s only if $C(O) \subseteq C(s)$.

CC2. Given an operation $O \in \mathcal{O}_o$ and a document state s, where $O \notin C(s)$ and $C(O) \subseteq C(s)$, the set of operations that O must be transformed against before being executed on s is $C(s) - C(O)$.

CC3. Given an operation $O \in \mathcal{O}$ and a document state s, O can be executed on s only if: $C(O) = C(s)$.

CC4. Given an operation $O_x \in \mathcal{O}_o$ and an operation $O \in \mathcal{O}$, where $O_x \notin C(O)$, O_x can be transformed to the context of O only if $C(O_x) \subseteq C(O)$.

CC5. Given an operation $O_x \in \mathcal{O}_o$ and an operation $O \in \mathcal{O}$, where $O_x \notin C(O)$ and $C(O_x) \subseteq C(O)$, the set of operations that O_x must be transformed against before being transformed with O is $C(O) - C(O_x)$.

CC6. Given two operations $O_1, O_2 \in \mathcal{O}$, they can be transformed with each other, i.e., $T(O_1, O_2)$ or $T(O_2, O_1)$, only if $C(O_1) = C(O_2)$.

In essence, CC1 and CC4 ensure correct ordering of operation execution and transformation; CC2 and CC5 determine correct transformation reference operations; and CC3 and CC6 ensure correct operation execution and transformation. CC conditions are known to be critical in evaluating and designing OT control algorithms. There have been numerous OT control algorithms capable of ensuring the six context-based conditions, which are called CC-compliant algorithms [7,9,12,11,14].

To achieve convergence, a transformation function T may ensure the following convergence properties:

Convergence Property 1 (CP1). *Given O_1 and O_2 defined on the same document state s, i.e., $C(O_1) = C(O_2) = C(s)$, if $O_1' = T(O_1, O_2)$, and $O_2' = T(O_2, O_1)$, T satisfies CP1 if $s \circ O_1 \circ O_2' = s \circ O_2 \circ O_1'$.*

CP1 means that applying O_1 and O_2' in sequence on s has the same effect as applying O_2 and O_1' in sequence on s. In other words, the list of two operations $[O_1, O_2']$ is equivalent to another list of two operations $[O_2, O_1']$ with respect to the effect in document state, i.e., $[O_1, O_2'] \equiv [O_2, O_1']$.

Convergence Property 1 (CP2). *Given O_x, O_1, and O_2 defined on the same state s, i.e., $C(O_x) = C(O_1) = C(O_2) = C(s)$, if $O_1' = T(O_1, O_2)$, and $O_2' = T(O_2, O_1)$, T satisfies CP2 if $LT(O_x, [O_1, O_2']) = LT(O_x, [O_2, O_1'])$.*

CP2 means that transforming O_x against O_1 and O_2' in sequence equals to transforming O_x against O_2 and O_1' in sequence. In other words, $[O_1, O_2']$ is equivalent to $[O_2, O_1']$ w.r.t. the effect in transformation.

To achieve intention preservation, a transformation function T must meet the following transformation post-condition:

Transformation Post-Condition (TPC). *Given two context-equivalent operations O_1 and O_2, i.e., $C(O_1) = C(O_2)$. After transforming O_1 against O_2 to produce O_1', i.e., $O_1' = T(O_1, O_2)$ and $C(O_1') = C(O_2) \cup \{org(O_2)\}$, T satisfies TPC if the effect of executing O_1' in the document state determined by $C(O_1')$ is the same as the effect of executing O_1 in the document state determined by $C(O_1)$.*

4 Verification of Convergence and Intention Preservation

In this section, we establish that an OT system based on a CC-compliant algorithm (e.g., COT [14]) can achieve convergence and intention preservation for an arbitrary number of operations with arbitrary causal relationships and generated by any number of users in a collaborative editing session, provided that transformation functions can preserve CP1, CP2, and TPC. First, we show that a CC-compliant algorithm possesses the following properties, established as lemmas below.

Lemma 1. *Given two original operations O_1 and O_2, under a CC-compliant algorithm, O_1 is executed before O_2 only if $O_1 \rightarrow O_2$ or $O_1 \parallel O_2$.*

Proof: This is directly derived from CC1. □

Lemma 2. *Given three original operations O_1, O_2, and O_3, under a CC-compliant algorithm, if O_1 is executed before O_2, and $O_2 \rightarrow O_3$ and $O_1 || O_3$, then $O_1 \parallel O_2$.*

Proof: First, it is impossible to have $O_2 \rightarrow O_1$ because O_1 is executed before O_2 by Lemma 1. Second, it is impossible to have $O_1 \rightarrow O_2$ because it contradicts to $O_2 \rightarrow O_3$ and $O_1 \parallel O_3$. The lemma follows from Definition 2. □

Lemma 3. *Under a CC-compliant algorithm, two operations O_1 and O_2 are transformed with each other if and only if they are originally concurrent.*

Proof: This can be derived from CC1, CC2, CC4, CC5 and CC6. □

The following corollary of lemma 3 establishes that when an operation is executed under a CC-compliant algorithm, it must have been transformed with operations that are executed before and originally concurrent with this operation.

Corollary 1. *Let O be any operation in a sequence of operations L produced by an OT system based on a CC-compliant OT algorithm. O must have been transformed with all operations (if any) that are originally concurrent with O and positioned before O in L.*

Lemma 4. *Let L_1 and L_2 be two sequences of transformed operations produced by applying a CC-compliant algorithm to the same group of n operations, $O = L_1[n]$ be the last operation in L_1, and $O' = L_2[i]$ be the corresponding operation in L_2 (i.e., $org(O) = org(O')$), where $i < n$. For any operation O_x in the range of $L_2[i+1, n]$, inclusively, it must be that O_x is originally concurrent with O', i.e., $org(O_x) \parallel org(O')$.*

Proof: According to Lemma 1, it is impossible that $org(O_x) \rightarrow org(O')$ because O' is executed before O_x in L_2; it is also impossible that $org(O') \rightarrow org(O_x)$ because $org(O') = org(O)$ and O is executed after all operations, including O_x, in L_1. The lemma follows from Definition 2. □

The following lemmas are based on a CC-compliant algorithm and transformation properties CP1 and CP2.

Lemma 5. *Given a sequence of operations L produced by applying a CC-compliant OT algorithm and CP1-preserving transformation function T to a sequence of original operations, if two adjacent operations in L are originally concurrent, then they can be transposed to make a new L', such that $L' \equiv L$.*

Proof: Assume that $L = [O_1, \ldots, O_{i-1}, O_i, O_{i+1}, O_{i+2}, \ldots, O_n]$, and O_i and O_{i+1} are the two adjacent operations that are originally concurrent. By Corollary 1, O_{i+1} must have been obtained by transformation. Also there must exist an operation O_x, such that $T(O_x, O_i) = O_{i+1}$. Therefore, we can transpose these two operations and make a new L' as follows: $L' = [O_1, \ldots, O_{i-1}, O_x, T(O_i, O_x), O_{i+2}, \ldots, O_n]$. Since T preserves CP1, we have $[O_x, T(O_i, O_x)] \equiv [O_i, O_{i+1}]$ (by CP1). From the fact that other operations in L' are the same as in L, we have $L' \equiv L$. □

Definition 6 (Transpose-reducible sequence). *An operation sequence L_1 is said to be transpose-reducible to another operation sequence L_2, if (1) L_2 can be obtained from L_1 by transposing two adjacent and originally concurrent operations in L_1; or (2) L_1 is transpose-reducible to a list L_3 and L_3 is transpose-reducible to L_2.*

Lemma 6 (Generalization of CP1). *If L_1 is transpose-reducible to L_2, then $L_1 \equiv L_2$.*

Proof: It follows from Definition 6 and CP1. □

Lemma 7 (Generalization of CP2). *Given two operation sequences L_1 and L_2, if L_1 is transpose-reducible to L_2, then, for any operation O, $LT(O, L_1) = LT(O, L_2)$.*

Proof: First, suppose L_2 is obtained by transposing one pair of adjacent and originally concurrent operations in L_1. Let $L_1 = [O_1, \ldots, O_{i-1}, O_i, O_{i+1}, O_{i+2}, \ldots, O_n]$, and $L_2 = [O_1, \ldots, O_{i-1}, O'_{i+1}, O'_i, O_{i+2}, \ldots, O_n]$, where $[O_i, O_{i+1}] \equiv [O'_{i+1}, O'_i]$. The lemma is true because:

1. $O' = LT(O, L_1[1, i-1]) = LT(O, L_2[1, i-1])$ (by $L_1[1, i-1] = L_2[1, i-1]$)
2. $O'' = LT(O', [O_i, O_{i+1}]) = LT(O', [O'_{i+1}, O'_i])$ (by CP2)
3. $O''' = LT(O'', L_1[i+2, n]) = LT(O'', L_2[i+2, n])$ (by $L_1[i+2, n] = L_2[i+2, n]$)

Since transpose-reducibility is transitive (see Definition 6), it can be shown by an induction argument that the lemma is true if L_2 is obtained by transposing any number of pairs of adjacent and originally concurrent operations in L_1. □

Theorem 1. *Given an OT system based on a CC-compliant algorithm and CP1-CP2-preserving transformation functions, and a group of operations GO generated during a collaborative editing session, if L_1 and L_2 are two sequences of transformed operations produced by this OT system at any two different sites, respectively, then $L_1 \equiv L_2$, i.e., convergence is preserved at these two sites in a collaborative editing session.*

Proof: We apply induction on the number of operations in GO. When $|GO| = 1$, the theorem obviously holds since the only operation in L_1 and L_2 produced by this OT system must be in its original form and must be the same in both L_1 and L_2.

As the induction hypothesis, assume the theorem holds for $1 \leq |GO| \leq m$. We show that the theorem holds for $|GO| = m + 1$. Let O be the last operation at $L_1[m+1]$, and O' at $L_2[i]$ be the operation corresponding to O, i.e., $org(O) = org(O')$, where $i \leq m + 1$. In case that $i \neq m + 1$, those operations in the range of $L_2[i+1, m+1]$ must be originally concurrent with $org(O')$ according to Lemma 4. Therefore, O' can be repeatedly transposed and shifted from $L_2[i]$ to $L_2[m+1]$ to produce a new L'_2, and $L'_2 \equiv L_2$ by Lemma 6. Let O'' be the last operation of L'_2. Both O and O'' must be positioned at the end of L_1 and L'_2, respectively; and $L_1[1, m]$ and $L'_2[1, m]$ must contain the same group of m operations. According to the induction hypothesis, we have $L_1[1, m] \equiv L'_2[1, m]$. To prove $L_1 \equiv L'_2 (\equiv L_2)$, we only need to show $L_1[m+1] = L'_2[m+1]$, i.e., $O = O''$.

For any two adjacent operations in the range of $L_1[1, m]$, if the left one is originally concurrent with O but the right one is causally before O, these two operations must be concurrent (by Lemma 2) and hence can be transposed according to Lemma 5. Therefore, operations in $L_1[1, m]$ can be transposed and shifted in such a way that operations that are originally causally before O are positioned at the left side (denoted as $L_1.left$) and operations that are originally concurrent with O are positioned at the right side (denoted as $L_1.right$). Hence, we have $L_1[1, m] \equiv L_1.left + L_1.right$. The same can be done on $L'_2[1, m]$ to get $L'_2[1, m] \equiv L'_2.left + L'_2.right$. $L_1.left$ and $L'_2.left$ must contain the same group of operations that are originally causally before O. According to the induction hypothesis, we have $L_1, left \equiv L'_2.left$, which, together with $L_1[1, m] \equiv L'_2[1, m]$, implies that $L_1.right \equiv L'_2.right$. If $L_1.right$ and $L'_2.right$ are empty lists, which means that no operation is originally concurrent with O, then both O and O'' must be the same original operation, so $O = O''$. Otherwise, according to Corollary 1, O and O'' must have been transformed with concurrent operations in $L_1.right$ and $L'_2.right$, respectively, since they are positioned at the end of each list. Let $O = LT(org(O), L_1.right)$ and $O'' = LT(org(O), L'_2.right)$. Since $L_1.right \equiv L'_2.right$, we have $O = O''$ by Lemma 7. Therefore, we establish $L_1 \equiv L'_2$ for $|GO| = m + 1$.

By induction, we have $L_1 \equiv L'_2$. $L_1 \equiv L_2$ follows from $L_1 \equiv L'_2$ and $L'_2 \equiv L_2$. □

Theorem 2. *Under an OT system based on a CC-compliant algorithm and CP1-CP2-TPC-preserving transformation functions, for any operation O generated in a collaborative editing session, the effect of executing O on any document state is the same as the effect of executing O on the document state from which O was generated, i.e., the intention of O is preserved at all sites in a collaborative editing session.*

Proof: The theorem holds when O is executed at the local site since it is executed immediately on the document state from which O was generated. When O is executed at a remote site, operations that are causally before O must have been executed by CC1. Under this condition, there are two possible cases to be considered:

Case 1: No concurrent operation has been executed before O in this site. In this case, O will be executed as-is without transformation according to Lemma 3. Since both the remote and local sites have executed the same group of operations that are causally before O, the document states at the remote and local sites must be identical by Theorem 1. Therefore, executing O on the remote document state should achieve the same effect as executing O on the local document state. The theorem holds in this case.

Case 2: Some concurrent operations have been executed before O in this site. In this case, O will be transformed against those concurrent operations according to Lemma 3. Let O' be the operation obtained from transforming O against those concurrent operations. Since the transformation function preserves TPC, the effect of O' in the document state determined by $C(O')$ must be the same as the effect of original O in the document state determined by $C(O)$. We know that the document state determined by $C(O')$ is the same as the remote document state according to CC3, and the document state determined by $C(O)$ is the same as the local document state from which O was generated by Definition 4 (1). Thus, the theorem holds in this case as well. □

In summary, Theorems 1 and 2 collectively establish that OT control algorithms capable of ensuring context-based conditions can generally achieve convergence and intention preservation, provided that underlying transformation functions can preserve transformation properties CP1, CP2 and TPC. Verification of transformation properties requires formalization of operation effects and data models, which are discussed in the next section.

5 Verification of Transformation Properties

In this section, we investigate the verification of CP1, CP2 and TPC properties for concrete data and operation models. We choose strings as the data model and character-wise operations as the operation model, as they are the basic and most commonly used models in existing OT systems.

Definition 7 (String Data Model). *A string data model is a formal language $\mathcal{S}$ defined over a set of alphabet Σ, that is, a subset of Σ^*. Each element in $\mathcal{S}$ is written in $\langle c_0 c_1 c_2 \cdots c_{n-1} \rangle$ where each $c_i \in \Sigma$.*

Definition 8 (Character Operation Model). *Character operation model supports two kinds of operations. Given a string $s \in \mathcal{S}$, the insert operation $I(p, c)$ inserts a character c at position p of s where $0 \leq p \leq |s|$; the delete operation $D(p, c)$ deletes the character c at position p of s where $0 \leq p < |s|$.*

Under the string data model and character operation model, a document state s may have an arbitrary length, and each position in s may contain any character from Σ; and an operation O may have an arbitrary position p within its valid range. To formally verify CP1/CP2/TPC, we need to consider all these infinite scenarios, which stop us from directly using automatic verification techniques like model checking.

To solve this problem, we propose a data abstraction of the string data model using symbolic representations. For a document state $s = \langle c_0 c_1 c_2 \cdots c_{n-1} \rangle$ in $\mathcal{S}$, $s[i]$ denotes the character at the i-th position of s, i.e., $s[i] = c_i$; $s[i, j]$ denotes a substring of s from the indices i to j, *i.e.*, $s[i, j] = \langle c_i c_{i+1} c_{i+2} \cdots c_{j-1} c_j \rangle$; $|s|$ denotes the sequence length of s. For notation convenience, $s[i, j]$ means an empty string if $i > j$. $s[i, j] + s[m, n]$ denotes the concatenation of two substrings of s. We define single operation effects based on this symbolic data model representation.

Definition 9 (Effect of a Single Insert Operation). *Given an insert operation $I(p, c)$ defined on a document state s, where $0 \le p \le |s|$. The symbolic effect of $I(p, c)$ on s is to convert s into a new state $s' = s \circ I(p, c) = s[0, p-1] + c + s[p, |s|-1]$.*

Definition 10 (Effect of a Single Delete Operation). *Given a delete operation $D(p, c)$ defined on a document state s, where $0 \le p < |s|$. The symbolic effect of $D(p, c)$ on s is to convert s into the following new state $s' = s \circ D(p, c) = s[0, p-1] + s[p+1, |s|-1]$.*

From the operation effect definitions above, we can see that the effect of O on s can be partitioned into three uniform effect ranges: (1) $s[0, p-1]$, the left effect range of p; (2) $s[p]$, the effect range at p; and (3) $s[p+1, |s|-1]$, the right effect range of p. The effect of O in each range is uniform in the sense that all positions in the same range are affected by O in the same way, though the effect in different ranges may be different. This effect uniformity provides the foundation for reducing arbitrary possibilities to a fixed number of possibilities.

In the absence of concurrency, single-operation effect definitions are adequate for users to understand the behavior of an editor. In the presence of concurrency in collaborative editing, however, we need to define combined-effects for concurrent operations, independent of their execution orders. Different applications may choose different combined-effects for meeting specific application needs.

Union-Effects (UE) is the most commonly used combined-effects, which is able to retain the original-effects of individual operations, and to preserve all-operation-effects under all circumstances. This UE is a specialization of the general intention preservation requirement under the string data model and character operation model. To specify UE for string data and operation model, we first enumerate all possible operation type combinations: $I - I$, $I - D$, $D - I$ and $D - D$. Second, for each type permutation, there are three position relationships $p_1 < p_2, p_1 = p_2, p_1 > p_2$ based on the uniform effect ranges of each operation. In total, there are 12 cases, as shown under the column $UE(O_1, O_2)$ in Table 1. Each UE defines a concrete transformation matrix (TM) to represent the actual effects of the transformation function.

Two cases in Table 1 deserve special attention. In case 2, which is known as "insert-insert-tie", two possible combined-effects: c_1c_2 and c_2c_1 are valid which introduces non-determinism to $UE(O_1, O_2)$. To achieve such union effect, a priority-based tie-breaking rule for O_1 and O_2, which ensures a total ordering among all operations.

Table 1. Union-Effect and Transformation Matrix

No.	O_1, O_2	Position Rel.	$UE(O_1, O_2)$	$TM(O_1, O_2)$
1	$I(p_1, c_1)$ $I(p_2, c_2)$	$p_1 < p_2$	$s[0, p_1 - 1] + c_1 + s[p_1, p_2 - 1] + c_2 + s[p_2, \|s\| - 1]$	$I(p_1, c_1)$
2_a		$p_1 = p_2$	$s[0, p_1 - 1] + c_1c_2 + s[p_1, \|s\| - 1]$	$I(p_1, c_1)$
2_b		$p_1 = p_2$	$s[0, p_1 - 1] + c_2c_1 + s[p_1, \|s\| - 1]$	$I(p_1 + 1, c_1)$
2_{v1}		$p_1 = p_2$	$s[0, \|s\| - 1]$	$D(p_1, c_2)$
3		$p_1 > p_2$	$s[0, p_2 - 1] + c_2 + s[p_2, p_1 - 1] + c_1 + s[p_1, \|s\| - 1]$	$I(p_1 + 1, c_1)$
4	$I(p_1, c_1)$ $D(p_2, c_2)$	$p_1 < p_2$	$s[0, p_1 - 1] + c_1 + s[p_1, p_2 - 1] + s[p_2 + 1, \|s\| - 1]$	$I(p_1, c_1)$
5		$p_1 = p_2$	$s[0, p_1 - 1] + c_1 + s[p_2 + 1, \|s\| - 1]$	$I(p_1, c_1)$
6		$p_1 > p_2$	$s[0, p_2 - 1] + s[p_2 + 1, p_1 - 1] + c_1 + s[p_1, \|s\| - 1]$	$I(p_1 - 1, c_1)$
7	$D(p_1, c_1)$ $I(p_2, c_2)$	$p_1 < p_2$	$s[0, p_1 - 1] + s[p_1 + 1, p_2 - 1] + c_2 + s[p_2, \|s\| - 1]$	$D(p_1, c_1)$
8		$p_1 = p_2$	$s[0, p_1 - 1] + c_2 + s[p_1 + 1, \|s\| - 1]$	$D(p_1 + 1, c_1)$
9		$p_1 > p_2$	$s[0, p_2 - 1] + c_2 + s[p_2, p_1 - 1] + s[p_1 + 1, \|s\| - 1]$	$D(p_1 + 1, c_1)$
10	$D(p_1, c_1)$ $D(p_2, c_2)$	$p_1 < p_2$	$s[0, p_1 - 1] + s[p_1 + 1, p_2 - 1] + s[p_2 + 1, \|s\| - 1]$	$D(p_1, c_1)$
11		$p_1 = p_2$	$s[0, p_1 - 1] + s[p_1 + 1, \|s\| - 1]$	NULL
11_{v2}		$p_1 = p_2$	$s[0, \|s\| - 1]$	$I(p_1, c_2)$
12		$p_1 > p_2$	$s[0, p_2 - 1] + s[p_2 + 1, p_1 - 1] + s[p_1 + 1, \|s\| - 1]$	$D(p_1 - 1, c_1)$

We denote $O_1 >_p O_2$ if O_1 has higher priority than O_2. Cases 2_a and 2_b represent $O_1 >_p O_2$ and $O_2 >_p O_1$ respectively. One alternative way to solve the insert-insert-tie case, defined as a new UE_{v1}, rejects both inserts as shown in case 2_{v1} and priority is not used in corresponding $TM_{v1}(O_1, O_2)$.

The other special union effect is case 11 "delete-delete-tie". In this case, only one character c_1 is deleted, which is the same as a single delete operation effect. A special operation NULL[1] is used in $T(O_1, O_2)$ to achieve such effect. We define one variation UE_{v2} for this case with the transformation matrix $TM_{v2}(O_1, O_2)$, e.g., none of the deletes has an effect (so the character remains), which is shown in case 11_{v2}.

Guided by the specified UE and CP1, results of transforming O_1 against O_2 under all 12 cases are specified in the column $TM(O_1, O_2)$ in Table 1. Given two operations O_1 and O_2 defined on the same document state s, the basic strategy of transforming O_1 against O_2 is to produce $O_1' = TM(O_1, O_2)$ as follows: (1) assume O_2 has been executed; (2) assess the execution effect of O_2 on the state s; and (3) derive O_1' so that executing O_1' after O_2 on s would produce the union effect specified in $UE(O_1, O_2)$, i.e., $UE(O_1, O_2) = s \circ O_2 \circ O_1'$. The effect of executing O_1' on the state after executing O_2 is the same as executing O_1 on the state before executing O_2. Therefore, $TM(O_1, O_2)$ meets the TPC requirement under the defined data and operation models. In addition, transformation must satisfy CP1. Therefore, we integrate combined effects with CP1, denoted as UE-CP1 as $UE(O_1, O_2) = s \circ O_1 \circ O_2' = s \circ O_2 \circ O_1'$. From $TM(O_1, O_2)$, it is straightforward to design a concrete transformation implementation. Most existing implementations [1,4,9,12] produce the same or similar transformation results as $TM(O_1, O_2)$. In our evaluation, $TM(O_1, O_2)$, rather than specific transformation implementation, is used for verification, so the verification results are generally applicable to all transformation implementations, that are capable of producing the same results as $TM(O_1, O_2)$.

Based on the above formalisms, the correctness checking of UE-CP1 and CP2 on arbitrary string states is reduced to a checking on the symbolic representation, which yields a finite state system. For UE-CP1, we define a model with two string variables x and y with initial value s. Then we check for all the possible types of operations O_1

[1] Formally, $s \circ NULL = s; T(NULL, O) = NULL; T(O, NULL) = O$.

Table 2. Counterexamples of CP2 verfication

Transformation Matrix	Case #	O_1	O_2	O_3	Positional Relationship
TM	1	$I(p_1, c_1)$	$D(p_2, c_2)$	$I(p_3, c_3)$	$p_2 = p_3 < p_1$
TM_{v1}	1	$I(p_1, c_1)$	$D(p_2, c_2)$	$I(p_3, c_3)$	$p_1 = p_2 < p_3$
	2	$I(p_1, c_1)$	$D(p_2, c_2)$	$I(p_3, c_3)$	$p_2 = p_3 < p_1$
TM_{v2}	1	$I(p_1, c_1)$	$D(p_2, c_2)$	$I(p_3, c_3)$	$p_2 = p_3 < p_1$
	2	$D(p_1, c_1)$	$D(p_2, c_2)$	$I(p_3, c_3)$	$p_1 = p_2 = p_3$

and O_2 with all the possible relations of the two positions p_1 and p_2 in O_1 and O_2, such that $x \circ O_1 \circ TM(O_2, O_1) = y \circ O_2 \circ TM(O_1, O_2) = UE(O_1, O_2)$ is true. Note that different union effects have different transformation results. UE_{v1} and UE_{v2} are defined similarly as UE in Table 1 with the differences in cases 2 and 11, respectively. One problem of using symbolic representation of the document state is the equivalence comparison since two identical document states may have different symbolic values, e.g., $s[0, p-1] + s[p, |s|-1]$ and $s[0, p] + s[p+1, |s|-1]$. Therefore, we introduce a combination function to convert a symbolic representation to a normalized format if needed, e.g., $s[p_0, p_1] + s[p_2, p_3] = s[p_0, p_3]$ if $p_1 + 1 = p_2$. For CP2, our model will enumerate all possible combinations of operations with all possible position relations, then check for the equivalence of the CP2-equation literally in their symbolic formats. In CP2 verification, we introduce additional symbolic position calculation rules to compare the symbolic value with arithmetics operations (see [2] for details).

6 Evaluation

To evaluate the proposed approach, we perform the verification on the three versions of UE and their corresponding transformations matrices in Table 1 and report our findings in this section. We use the PAT [15] (`www.patroot.com`) as the model checker. A data structure written in C# is used to capture the symbolic state representation and the corresponding operation model. The modeling of the transformation matrices is a direct interpretation of the transformation effects. We altered the reachability verification algorithm so that all counterexamples will be returned by completely exploring the whole state space rather than just returning the first counterexample. The experiments are conducted on a machine with Intel Core i7-2600 CPU at 3.4GHz and 4GM memory. Due to space limitation, the details about the models, PAT tool and evaluation data are provided in [2]. The verification of UE-CP1 and CP2 of the three TM finishes in around 0.1 sec. and 1 sec. respectively. UE-CP1 is satisfied by all three versions of TM. CP2-violation cases are found in all three versions, which are discussed below.

Classic UE. Based on UE-CP1 verification results, we declare $TM(O_1, O_2)$ can preserve UE-CP1 under all cases. One counterexample is discovered in CP2 verification as shown in Table 2. This CP2-violation case is a general description of the well-known False-Tie (FT) bug [12]. On one side of the CP2, $TM(TM(O_1, O_2), TM(O_3, O_2)) = TM(I(p_1 - 1, c_1), I(p_3, c_3)) = I(p_1 - 1, c_1)$, (because $p_1 - 1 = p_3$, generated by symbolic position calculation rules [2]), this is considered as a false-tie because the tie was not originally generated by users but created by transformation, and this FT is resolved by a tie-breaking priority condition $O_1 >_p O_3$. On the other side of the CP2-equation,

$TM(TM(O_1, O_2), TM(O_3, O_2)) = TM(I(p_1 + 1, c_1), D(p_2 + 1, c_2)) = I(p_1, c_1)$ (because $p_1 + 1 > p_2 + 1$). Therefore, CP2 is violated as $I(p_1 - 1, c_1) \neq I(p_1, c_1)$.

Since the discovery of the basic FT bug in [12], various CP2-violation scenarios have been reported in different OT systems [3,5] and CP2-violation phenomena became a main mystery surrounding OT correctness. However, examination of all reported CP2-violation cases revealed that they were just variations of the FT bug, i.e., CP2-violation occurs only if an FT is involved. Based on the exhaustiveness of CP2 verification, we confirm that the FT is the only possible case for CP2-violation under the classic UE.

Based on CP2 verification discoveries, we assert that CP2-preservation can be achieved by solving the FT bug at the transformation matrix level. Past research has also found that the FT bug can be solved by generic control algorithms by avoiding CP2 [16] without an FT-solution in transformation functions.

UE$_{v1}$ $TM_{v1}(O_1, O_2)$ is based on an alternative combined effect (UE_{v1} in Table 1) which achieves a deterministic null-effect in the insert-insert-tie case and our conjecture is that the FT bug will be solved in this way. Surprisingly, more counterexamples are discovered in CP2 verification of the variation, as shown in Table 2. The verification results immediately disapprove our conjecture and shows that UE_{v1} is no better than UE in solving FT bug. However, UE_{v1} is able to achieve a weaker form (but still meaningful) intention preservation and convergence (with a CP2-avoidance control algorithm integrally used).

UE$_{v2}$ $TM_{v2}(O_1, O_2)$ is based on an alternative combined effect (UE_{v2} in Table 2) which achieves a null-effect in the delete-delete-tie case. It preserves UE$_{v2}$-CP1 but fails in CP2 with two counterexamples as shown in Table 2. Case 1 is the same FT case revealed in CP2 verification of UE_{v2} because it is localized. Case 2 is a new FT bug caused by operation type transformation.

Verification of the three versions of TM demonstrates the experimental power of our verification framework. Without actually implementing the system, researchers are able to discover bugs. This model checking approach not only allows researchers to try out different combined-effects and corresponding transformation matrices, but also provides fast and reliable verification results.

7 Related Work

Since the introduction of OT, there are various works in the literature to study OT's correctness, as discussed below.

Ad Hoc Bug Detection. Experimental approaches, under the name puzzle-detection-resolution [8,9,12], have been commonly adopted in prior research for searching OT bugs. Trying to detect and resolve various intellectually challenging puzzles has been a major stimulus in OT research and a driving force for OT technical advancement. This experimental approach has proven its effectiveness in detecting and solving all known puzzles, including the dOPT puzzle related to OT control algorithms [4,7,9,11,12], the False-Tie (FT) puzzle related to violating document convergence in transformation functions [12], etc. However, the ad hoc and informal nature of this approach makes it unsuitable for systematic and formal verification.

Manual Proof. Mathematic proof has been effective in verifying correctness of OT control algorithms. The adOPTed control algorithm [9] has been proven for correctness in ensuring document convergence if transformation functions can preserve CP1 and CP2. The GOTO algorithm [11] has been proven for correctness for achieving convergence and intention preservation provided that transformation functions ensure CP1, CP2 and TPC [10]. The COT algorithm [14] has been proved to be CC-compliant. This work is the first to use mathematic proof to establish OT system correctness in both convergence and intention preservation without referring to specific OT control algorithms or transformation functions. This verification result is significant as it is generally valid to all OT systems capable of preserving established context-based conditions and properties.

Computer-Aided Verification. Prior work had used theorem-provers [5] and model checkers [3] to verify CP1 and CP2 for specific transformation functions. However, prior theorem-prover-based approach had used mathematical formalisms that are quite different from OT transformation properties, which hindered correct specification and result interpretation. Consequently, there were repeated specification errors and false verification results, as reported in [3,5]. Prior model checking OT work was hindered by the infinite state space of collaborating editing systems, as reported in [3]: (1) a collaborating editing session is modeled as an arbitrary number n of operations; (2) a document is a string of characters with a fixed length $L = 2 \times n$; each character can be either 0 or 1; and (3) an operation is either to insert or delete in one of L positions. This modeling produces the formula $((2 + 1) \times L)^n = (6 \times n)^n$ to calculate the total number of states. As reported in [3], a session with 4 operations had generated $331776 = (6 \times 4)^4$ states, which had exceeded the capability of the used model checker.

Thanks to our mathematic induction proof which reduces the problem of verifying OT systems under arbitrary operations with arbitrary relationships to the problem of verifying transformation properties under a few (2 for CP1, and 3 for CP2) operations with equivalent context relationships, our model checking approach can focus on verifying CP1 and CP2 under basic data and operation models. To avoid the state explosion problem at the data/operation modeling level, we use novel data and operation formalization and position relationship reduction techniques to exhaustively cover infinite state space with a finite number of verification cases. Moreover, our verification covers not only CP1 and CP2, but also TPC (i.e., intention preservation), which were never addressed in prior work. With this exhaustive coverage of verification cases, we have established the correctness in preserving CP1 and union effects – commonly adopted combined-effects (and TPC) for existing OT systems, and discovered that the FT puzzle is the only CP2-violation case in OT systems based on the union effect. Furthermore, our model checking target is represented by a general transformation matrix, which captures the essential convergence requirement and combined-effects under given data and operation models, without reference to specific transformation implementations; so it is generally applicable to a wide range of transformation implementations. This transformation matrix approach has also enabled us to evaluate alternative combined-effects, proposed but never checked in prior research, without actually implementing them in real OT systems. In parallel with PAT-based modeling checking verification, we have also developed and used another software tool, called OT eXplorer (OTX), to exhaustively search OT puzzles and verify transformation properties [13]. Independent verification results from OTX and PAT are consistent and provide mutual confirmation.

8 Conclusion

This work has contributed a complete verification of OT including control algorithms and transformation functions, by marrying the power of mathematical proof and automatic model checking. We verified both convergence and intention preservation in OT-based collaborative editing systems for the first time. By establishing the correctness of OT system based on a set of well-defined transformation conditions and properties, the design of correct control algorithms and transformation functions could be guided by these conditions and properties. We are extending and applying our model checker to support experimental design and evaluation of various combined-effects under different data and operation models for novel collaborative applications, which will be reported in future publications.

Acknowledgement. This research is partially supported by an Academic Research Fund Tier 1 Grant from Ministry of Education Singapore, and by "Formal Verification on Cloud" project under Grant No: M4081155.020.

References

1. `http://cooffice.ntu.edu.sg/otfaq/`
2. `http://pat.sce.ntu.edu.sg/ot`
3. Boucheneb, H., Imine, A.: On model-checking optimistic replication algorithms. In: Lee, D., Lopes, A., Poetzsch-Heffter, A. (eds.) FMOODS/FORTE 2009. LNCS, vol. 5522, pp. 73–89. Springer, Heidelberg (2009)
4. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. In: SIGMOD, pp. 399–407 (1989)
5. Imine, A., Molli, P., Oster, G., Rusinowitch, M.: Proving Correctness of Transformation Functions in Real-time Groupware. In: ECSCW, pp. 277–293 (2003)
6. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. CACM 21(7), 558–565 (1998)
7. Nichols, D.A., Curtis, P., Dixon, M., Lamping, J.: High-latency, low-bandwidth windowing in the Jupiter collaboration system. In: UIST, pp. 111–120 (1995)
8. Prakash, A., Knister, M.J.: A framework for undoing actions in collaborative systems. TOCHI 1(4), 295–330 (1994)
9. Ressel, M., Nitsche-Ruhland, D., Gunzenhäuser, R.: An integrating, transformation-oriented approach to concurrency control and undo in group editors. In: CSCW, pp. 288–297 (1996)
10. Sun, C.: Undo as concurrent inverse in group editors. TOCHI 9(4), 309–361 (2002)
11. Sun, C., Ellis, C.A.: Operational transformation in real-time group editors: issues, algorithms, and achievements. In: CSCW, pp. 59–68 (1998)
12. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.: Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. TOCHI 5(1), 63–108 (1998)

13. Sun, C., Xu, Y., Agustina: Exhaustive search of puzzles in operational transformation. In: CSCW, pp. 519–529 (2014)
14. Sun, D., Sun, C.: Context-based operational transformation in distributed collaborative editing systems. TPDS 20(10), 1454–1470 (2009)
15. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: Towards flexible verification under fairness. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 709–714. Springer, Heidelberg (2009)
16. Xu, Y., Sun, C., Li, M.: Achieving convergence in operational transformation: conditions, mechanisms and systems. In: CSCW, pp. 505–518 (2014)