Context-Based Operational Transformation in Distributed Collaborative Editing Systems

David Sun and Chengzheng Sun, *Member*, *IEEE*

Abstract—Operational Transformation (OT) is a consistency maintenance technique for collaborative editing systems—a special class of distributed applications for supporting human-computer-human interaction and collaboration over communication networks. The theory of causality has been the foundation of all prior OT systems, but it is inadequate to meet essential OT requirements in functionality and correctness. In this paper, we analyze the limitation of the causality theory, propose a novel theory of operation context as the new foundation for OT systems, and present a new OT algorithm—Context-based OT (COT)—which provides uniform and efficient solutions to both consistency maintenance and undo problems. The COT algorithm has been implemented and used for supporting a range of novel collaborative applications. The context theory and context vectors are potentially applicable to other distributed computing applications.

Index Terms—Operational transformation, operation context, causality, consistency maintenance, distributed applications.

1 Introduction

Collaborative editors allow multiple users to view and edit shared documents over communication networks; they are a special class of distributed applications for supporting human-computer-human interaction and collaboration. Operational Transformation (OT) is a technique originally invented for consistency maintenance in collaborative text editors [2]. In more than 20 years, OT has evolved to acquire new capabilities and support an increasing number of new applications, including undo [3], [4], [5], [6], operation notification and compression [7], locking [8], HTML/XML and tree-structured document editing [9], [10], [11], office productivity tools [12], [13], 3D digital media design tools [14], transparent application-sharing environments [15], [16], [11], and replicated mobile computing systems [17].

To effectively and efficiently support existing and emerging applications, we must continue to extend the capability and improve the quality of OT. The soundness of the theoretical foundation for OT is crucial in this process. One theoretical underpinning of all existing OT algorithms is *causality/concurrency* [18], [2]: causally related operations must be executed and transformed in their causal order; concurrent operations must be transformed before their execution. The theory of causality had played an important role in guiding the design of early OT algorithms, but it is limited to capturing causal relationships among user-generated operations, which form only part of the operations in an OT system. This limitation leads to its incapability of capturing essential OT-required relations and conditions among operations of other types, including transformed and inverse operations that are generated by the OT system not by users. These limitations have caused various OT correctness and complexity problems.

The limitation of causality had caused correctness problems from the very beginning of OT history. The dOPT algorithm was the first OT algorithm based on concurrency relationships among operations [2]: a pair of operations is transformable as long as the operations are concurrent. However, later research discovered that the concurrency condition alone is not sufficient, and another essential condition is that the two operations involved in a transformation must be defined on the same document state. In reality, concurrent operations may not be defined on the same state, and not all operations in an OT system have causal/concurrent relationships with each other. The failure to recognize these problems and to capture relevant correctness conditions was the root of the well-known dOPT puzzle [19]. The dOPT puzzle had been solved by using various technical patches, but the theory of causality remained at the core of follow-up OT algorithms.

The limitation of causality became even more prominent when OT was applied to solve the undo problem in collaborative editors. The causal relation is not defined for inverse operations, which are derived from interpreting meta-level undo commands, let alone capturing their OT-related conditions with normal editing operations. To work around this problem, special technical patches were invented to maintain suitable relationships between inverse and other operations under different circumstances, resulting in more intricate and inefficient OT algorithms [5], [6].

After having designed, implemented, and experimented with a series of OT algorithms of increased functionality and complexity, we realized that technical patches and increment improvements to the existing theoretic framework are no longer sufficient to cope with the complexity of existing OT systems and to support the continual evolution of OT.

• *D. Sun is with the Division of Computer Science, Department of Electrical Engineering and Computer Science, University of California, Berkeley, Soda Hall, Berkeley, CA 94702. E-mail: davidsun@cs.berkeley.edu.*
• *C. Sun is with the School of Computer Engineering, Nanyang Technological University, Singapore 639798. E-mail: czsun@ntu.edu.sg.*

Manuscript received 13 June 2007; revised 10 May 2008; accepted 15 Oct. 2008; published online 29 Oct. 2008.
Recommended for acceptance by X. Zhang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2007-06-0190.
Digital Object Identifier no. 10.1109/TPDS.2008.240.

We reflected on what had been learned and set out to develop a new theoretical framework for better understanding and solving OT problems, reducing complexity, verifying correctness, and improving efficiency. In this paper, we report the main results of this effort: the theory of operation context and the Context-based OT (COT) algorithm.

The rest of this paper is organized as follows: First, we present basic definitions of causality and discuss its limitations in Section 2. Then, definitions of operation context and COT conditions are presented in Section 3, followed by context vectors as an efficient representation of contexts in Section 4. In Section 5, we present the basic COT algorithm for supporting consistency maintenance (do) and undo. Then, we discuss properties for transformation functions and their preconditions (PCs) in Section 6 and present COT solutions to these properties by breaking their PCs in Section 7. We discuss operation buffering schemes in the COT framework and analyze their time and space complexity in Section 8. Comparisons of the COT work with prior work are presented in Section 9. Finally, major contributions of this work and future work are summarized in Section 10. Supplemental materials include formal proofs of theorems about the context theory and the COT algorithm in Appendix A, context vector representation of the document state and context-based relations and conditions in Appendix B, and complexity analysis of operation buffering schemes in Appendix C. The appendices can be found on the CS Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2008.240.

2 Basic Concepts of Causality and Its Limitations

2.1 Causal Relationships among Operations and Commands

The theory of causality is central to distributed computing and to the design of prior OT algorithms. Following [18], causal relations among user-generated operations/commands in a collaborative editing session can be defined in terms of their generation and execution orders [2], [20].

Definition 1. Causal-dependency relation "$\rightarrow$." *Given two operations/commands O_a and O_b, generated at sites i and j, O_b is* causal-dependent *on O_a, denoted by $O_a \rightarrow O_b$,* if and only if*: 1) $i = j$ and the generation of O_a* happened before *the generation of O_b, 2) $i \neq j$ and the execution of O_a at site j* happened before *the generation of O_b, or 3) there exists an operation O_x such that $O_a \rightarrow O_x$ and $O_x \rightarrow O_b$.*

Definition 2. Causal-independency relation "$\|$." *Given two operations/commands O_a and O_b, O_a and O_b are* causal independent *or* concurrent, *denoted by $O_a \| O_b$,* if and only if *neither $O_a \rightarrow O_b$ nor $O_b \rightarrow O_a$.*

The above causal relations have been adopted by all existing OT systems. To illustrate causal relationships among operations/commands, consider a real-time collaborative editing session with two sites in Fig. 1. There are four editing operations O_1, O_2, O_3, and O_4 and one undo command $Undo(O_2)$ in this scenario. According to Definitions 1 and 2, we have $O_2 \rightarrow O_3 \rightarrow Undo(O_2)$ because the generation of O_2 happened before the generation of O_3, which in turn

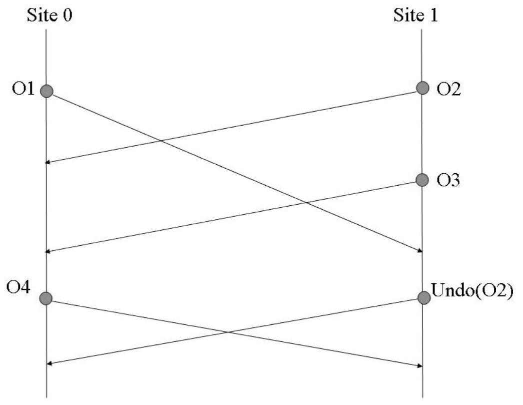


Fig. 1. A real-time collaborative editing scenario.

happened before the generation of $Undo(O_2)$; $O_1 \rightarrow Undo(O_2)$ because the execution of O_1 at site 1 happened before the generation of $Undo(O_2)$; and $O_1 \| O_2$ and $O_1 \| O_3$ because for each pair, neither operation's execution happened before the other's generation.

As *vector logical clocks* are used for capturing causality in distributed systems [21], *state vectors* have been used to time stamp operations for detecting their causal relationships and for modeling document states in OT systems [2], [4], [20].

2.2 Limitations of Causality

In Section 1, we have made some general statements on the limitations of causality in relation to OT for motivating the COT work. In this section, we use the scenario in Fig. 1 to illustrate and elaborate on these general statements. In the process of presenting the COT work, we provide further elaboration on these points as necessary technical backgrounds build up. At the end of the paper, we summarize the limitations of causality and causality-based OT in comparison with the COT work in Section 9.

2.2.1 Lack of Capability of Defining Relations among All Operations in OT Systems

In the following discussions, the term *IT-transformation* is used to mean the invocation of the *Inclusion Transformation (IT)* function $IT(O_a, O_b)$, which transforms operation O_a against operation O_b in such a way that the impact of O_b is effectively included in O_a [20]. This term is to differentiate the invocation of the IT function from other steps involved in a transformation process.

An OT system has at least two types of operation: *original* operations, which are generated by users, and *transformed* operations, which are outcomes of some transformations. For OT systems supporting both *do* and *undo*, another two types of operation can be distinguished: *normal* operations, which are related to *doing* something, and *inverse* operations, which are generated from interpreting metalevel commands for *undoing* executed operations [3], [5], [6]. In combination, there are four types of operation in OT systems supporting both do and undo. For example, all operations O_1, O_2, O_3, and O_4 in Fig. 1 are original normal operations; an original inverse operation $\overline{O_2}$ can be obtained from interpreting command $Undo(O_2)$; a transformed

TABLE 1
Four Types of Operation in an OT System Supporting Both Do and Undo, with Examples from the Scenario in Fig. 1

	Original	**Transformed**
Normal	Operations generated by the user, e.g. O_1, O_2, O_3, O_4	Operations obtained from transforming normal operations, e.g. $O_1' = IT(O_1, O_2)$;
Inverse	Operations derived from user-generated undo commands, e.g. $\overline{O_2}$ from interpreting $Undo(O_2)$	Operations obtained from transforming inverse operations, e.g. $\overline{O_2}' = IT(\overline{O_2}, O_3)$

normal operation O_1' can be obtained by IT-transforming O_1 against O_2, i.e., $IT(O_1, O_2)$; and a transformed inverse operation $\overline{O_2}'$ can be obtained by IT-transforming $\overline{O_2}$ against O_3, i.e., $IT(\overline{O_2}, O_3)$. The four types of operation and examples are summarized in Table 1.

According to Definitions 1 and 2, the causal relation is defined only for original normal operations but not for the other three types of operation. Consequently, the causal relation is inadequate in capturing essential OT-required conditions among all types of operation in an OT system. One such condition is about what relationship two operations must have before being IT-transformed. Under the theory of causality, the commonly used condition is that two operations can be IT-transformed if they have the causal-independency (or concurrency) relationship. However, this condition is neither necessary nor sufficient for two operations to be IT-transformed, which has caused problems in supporting both *do* and *undo* in OT systems.

2.2.2 Problems in Supporting Consistency Maintenance

To make the discussion concrete, we instantiate the initial document state and O_1, O_2, and O_3 in Fig. 1 as follows:

- The initial document state is a string "ABC."[1]
- $O_1 = Insert[2, \text{“}x\text{”}]$, to insert "x" at position 2.
- $O_2 = Insert[1, \text{“}12\text{”}]$, to insert "12" at position 1.
- $O_3 = Insert[3, \text{“}3\text{”}]$, to insert "3" at position 3.

We examine the OT processing of these three operations at both sites. At site 1, after executing O_2 and O_3, the document state becomes "A123BC." When O_1 arrives, it needs to be transformed against concurrent operations O_2 and O_3 according to the causality-based condition. Following the dOPT algorithm [2], O_1 is first IT-transformed against O_2 to produce a transformed operation: $O_1' = IT(O_1, O_2) = Insert[4, \text{“}x\text{”}]$, whose position parameter is 4—an increment by two from the original position of O_1 since the execution of O_2 has shifted the position of O_1 to the right by two positions. Then, O_1' is IT-transformed against O_3 to create $O_1'' = IT(O_1', O_3) = Insert[5, \text{“}x\text{”}]$, whose position is 5—an increment by one from the position of O_1' since the execution of O_3 has shifted the position of O_1' to the right by one position. Finally, the transformed operation O_1'' is executed, and the document state becomes "A123BxC," which correctly preserves the effect of O_1.

Although the correct result is achieved in the above transformation process at site 1, this is not obtained by strictly following the causality-based condition: the transformed operation O_1' and the original operation O_3 are IT-transformed, but they do not have any causal relationship. This means that the causal relationship is not a necessary condition for two operations to be IT-transformed. Prior OT algorithms have used other noncausal relationships among operations to determine whether two operations can be IT-transformed, which is nontrivial and ad hoc under complex scenarios [19], [20].

At site 0, after executing O_1, the document state becomes "AxBC." When O_2 arrives, it is IT-transformed against concurrent operation O_1 to produce $O_2' = IT(O_2, O_1) = Insert[1, \text{“}12\text{”}]$. In this IT-transformation, $O_2' = O_2$ because the execution of O_1 does not have any effect on the position parameter of O_2. After executing O_2', the document state becomes "A12BxC," which correctly preserves the effect of O_2. Then, when O_3 arrives, it is IT-transformed against concurrent operation O_1 according to the causality-based condition. After this transformation, we get $O_3' = IT(O_3, O_1) = Insert[4, \text{“}3\text{”}]$ since O_1's position is at the left side of O_3's position by comparing their position parameters. However, executing O_3' would result in the document state "A12B3xC," which does not preserve the effect of O_3 and is also inconsistent with the state "A123BxC" at site 1. This is the well-known dOPT puzzle [19], which demonstrates that the causality-based condition is insufficient for capturing the required operation relation for correct IT-transformation.

The root of the problem is that O_3 (defined on "A12BC") and O_1 (defined on "ABC") are not defined on the same document state so their position parameters are not comparable. The solution to this problem is to first transform O_1 into a suitable form that is defined on the same state as O_3 and then IT-transform O_3 against this new form of O_1. More specifically, O_1 needs to be IT-transformed against O_2 to get $O_1' = Insert[4, \text{“}x\text{”}]$, which is defined on the same document state as O_3; then, O_3 is IT-transformed against O_1' to get $O_3' = Insert[3, \text{“}3\text{”}]$, which correctly detected the position of O_1' is at the right side of O_3's position. Executing the new O_3' would result in a correct document state "A123BxC," which preserves the effect of O_3 and is consistent with the state at site 1.

Although the dOPT puzzle can be and has been solved [19], the required IT-transformation condition involved in the above solution, i.e., the two input operations to the IT function must be defined on the same document state, cannot be captured by the causal relation among operations. To work around this problem, various technical patches were invented to maintain and detect the required IT-transformation condition in different OT systems [19].

2.2.3 Problems in Supporting Undo

To undo an operation O, the user may generate a command $Undo(O)$, which is a metalevel command and has to be interpreted by executing a suitable inverse operation $\overline{O}$. The *undo effect*, adopted by OT-based undo solutions [3], [5], [6], is to eliminate the effect of O but retain the effects of other

1. In OT literature, it is common to use a string of characters to model a document state, but the actual document state to which OT is applicable may contain objects of any type, which may not necessarily be viewed as a sequence from the user interface. The reader is referred to [11] and [14] for discussions on how to map complex document states into the OT data model.

operations in the document. To achieve this undo effect, $\overline{O}$ may have to be transformed with other operations even though it has no causal relationships with those operations.

To illustrate, we focus on command $Undo(O_2)$ and operation O_4 in Fig. 1. The intended effect of undoing O_2 is to eliminate the effect of O_2 but retain the effects of O_1 and O_3. As O_2 is not the last operation executed on the document state, the intended undo effect cannot be achieved by executing the inverse of O_2, denoted as $\overline{O_2}$, on the current document state. We must transform $\overline{O_2}$ against all operations executed after O_2 (i.e., O_3 and O_1). However, the causality-based condition cannot be used to detect these transformation targets since there is no causal relation defined between $\overline{O_2}$ and O_3 and O_1.

The situation becomes even more complex when inverse operations are mixed with normal editing operations. For example, when the normal editing operation O_4 arrives at site 1, it must be transformed against $\overline{O_2}$ since the current document state on which O_4 is to be executed is different from the document state on which O_4 was generated: the former includes the effect of $\overline{O_2}$, but the latter does not. Clearly, this transformation cannot be detected by the causality-based condition because there is no causal relation between O_4 and $\overline{O_2}$. One might be tempted to fix this problem by allowing $\overline{O_2}$ to inherit from $Undo(O_2)$ the causal relation with other operations so that $\overline{O_2}$ would be treated as being concurrent with O_4 (since $Undo(O_2) \| O_4$) and hence be transformed with O_4. However, this quick fix does not work in achieving the intended undo effect: it would incorrectly treat $\overline{O_2}$ as being causally after O_3 and O_1, thus missing the transformation of $\overline{O_2}$ against O_3 and O_1.

The root of this problem is deep and related to the fundamental limitation of the causality theory: an inverse operation has no causal relation with any other operation. This example also illustrates that the relationships between inverse and normal operations, as required for achieving the intended undo effect, cannot be defined by the "happen-before" relationship [18]. To work around these and other undo-related problems/puzzles, various technical patches had been invented to keep track of suitable but noncausal relationships among inverse and other operations under different circumstances, resulting in complex, inefficient, or restrictive OT solutions to the undo problem [3], [5], [6].

In summary, the causal relation is limited to operations/commands directly generated by users but undefined for inverse operations and operations obtained from transformations. The lack of capability of capturing OT-related relationships among all operations has had important impact on the design of OT algorithms. Work-around patches have made prior OT systems complex, intricate, and inefficient, which hinders the understanding, verification, application, and continual evolution of OT. To solve these problems, we need a new theoretic framework that is able to capture essential OT conditions for all types of operation and that allows clean, efficient, and uniformed solutions to both do and undo problems.

3 Operation Context

3.1 Basic Concept of Operation Context

Conceptually, each operation O is associated with a *context*, denoted by $C(O)$, which corresponds to a document state on which the operation is defined. The significance of operation context is that the effect of an operation can be correctly interpreted only in its own context. The ability to determine, compare, and update operation contexts is essential for OT systems to correctly execute and transform operations.

To facilitate comparison and manipulation of operation contexts, it is necessary to *explicitly* represent operation context. Since the context of an operation corresponds to the document state on which the operation is defined, the problem of context representation can be reduced to the problem of document state representation. In an OT-based collaborative editor, a document state can be uniquely characterized by the set of original operations executed so far on the document. These original operations may be executed in different orders or forms at different sites, but they must produce the same document state according to the convergence requirement [20]. Moreover, the difference between two document states can be expressed in terms of different original operations executed on these two states. Since every transformed operation must come from an original operation, we use $org(O)$ to denote the original operation of O. For example, $org(O_1') = O_1$, and $org(O_1'') = O_1$. If O itself is an original operation, then $org(O) = O$. We use a set of original operations, rather than transformed operations, to represent a document state.

Definition 3. Document state representation. *A document state, denoted as DS, can be represented as follows:*

1. *The initial document state is represented as* $DS = \{\ \}$.
2. *After executing an operation O of any type on DS, the new document state is represented by* $DS' = DS \cup \{org(O)\}$.

Based on the above document state representation, the context of an original normal operation should be the same as the document state from which this operation was generated. To achieve the undo effect in [6], an original inverse operation $\overline{O}$ should be defined on the document state $DS = C(O) \cup \{O\}$, which is the state immediately after executing the original operation O on the state $C(O)$. A transformed operation O', where $O' = IT(O, O_x)$, should be defined on the document state $DS = C(O) \cup \{org(O_x)\}$, which is the state achievable by executing O_x on the state $C(O)$. More precisely, the context of an operation of any type is defined as follows:

Definition 4. The context of an operation.

1. *For an original normal operation O,* $C(O) = DS$, *where DS is the document state from which O was generated.*
2. *For an original inverse operation* $\overline{O}$, $C(\overline{O}) = C(O) \cup \{O\}$, *where O is the original operation to be undone.*
3. *For a transformed normal or inverse operation* O', $C(O') = C(O) \cup \{org(O_x)\}$, *where* $O' = IT(O, O_x)$.

For the scenario in Fig. 1, we have $C(O_1) = \{\ \}$, $C(O_2) = \{\ \}$, $C(O_3) = \{O_2\}$, and $C(O_4) = \{O_1, O_2, O_3\}$ by statement 1 in Definition 4; $C(\overline{O_2}) = \{O_2\}$ by statement 2 in Definition 4; and $C(O_2') = \{O_1\}$ $(O_2' = IT(O_2, O_1))$ by statement 3 in Definition 4.

3.2 Context-Dependency/-Independency Relations

Based on operation context, the context-dependency/independency relation among operations can be defined in terms of whether an original operation is included in the context of another operation of any type.

Definition 5. Context-dependency relation "$\xrightarrow{c}$". *Given an original operation O_a and an operation O_b of any type, O_b is* context-dependent *on O_a, denoted by $O_a \xrightarrow{c} O_b$,* if and only if *1) $O_a \in C(O_b)$ or 2) there exists an original operation O_x such that $O_a \xrightarrow{c} O_x$ and $O_x \xrightarrow{c} O_b$.*

It should be noted that the context-dependency relation is defined between an original (normal or inverse) operation and another operation of any type (original or transformed). This is because every operation has a context, but only original operations can be included in a context.

Definition 6. Context-independency relation "$\overset{c}{\|}$". *Given two original operations O_a and O_b, O_a and O_b are* context independent, *denoted by $O_a \overset{c}{\|} O_b$,* if and only if *neither $O_a \xrightarrow{c} O_b$ nor $O_b \xrightarrow{c} O_a$.*

It can be shown that if both O_a and O_b are original normal operations, then $O_a \xrightarrow{c} O_b$ is equivalent to $O_a \rightarrow O_b$, and $O_a \overset{c}{\|} O_b$ is equivalent to $O_a \| O_b$ (Theorem 1 in Appendix A.1). In other words, the causal-dependency/independency relation is a special case of the context-dependency/independency relation.

For original normal operations O_1, O_2, O_3, and O_4 in Fig. 1, we have $O_1 \overset{c}{\|} O_2$ and $O_1 \| O_2$; $O_1 \overset{c}{\|} O_3$ and $O_1 \| O_3$; $O_2 \xrightarrow{c} O_3$ and $O_2 \rightarrow O_3$; $O_1 \xrightarrow{c} O_4$ and $O_1 \rightarrow O_4$; $O_2 \xrightarrow{c} O_4$ and $O_2 \rightarrow O_4$; and $O_3 \xrightarrow{c} O_4$ and $O_3 \rightarrow O_4$. However, for transformed and inverse operations $O_1' = IT(O_1, O_2)$, $O_2' = IT(O_2, O_1)$, and $\overline{O_2}$, we have $O_1 \xrightarrow{c} O_2'$ but $O_1 \not\rightarrow O_2'$; $O_2 \xrightarrow{c} O_1'$ but $O_2 \not\rightarrow O_1'$; and $O_2 \xrightarrow{c} \overline{O_2}$ but $O_2 \not\rightarrow \overline{O_2}$.

3.3 Context-Based OT Conditions

Based on operation context, we have formulated six Context-based Conditions (CCs) to capture essential requirements for correct operation execution and transformation in OT systems.

Context-based Condition 1 (CC1). Given an original operation O and a document state DS, where $O \notin DS$, O can be transformed for execution on DS only if $C(O) \subseteq DS$.

CC1 imposes a general constraint on the execution order of operations in an OT system and must be satisfied before an operation is given to the OT system for transformation. It can be shown that meeting CC1 is equivalent to executing operations in their context-dependency orders (Theorem 2 in Appendix A.1). When O is an original normal operation, all operations that are causally before O must be included in $C(O)$ (Theorem 1 and Corollary 1 in Appendix A.1). Therefore, CC1 covers the causality-based condition that original normal operations should be executed in their causal orders [2], [19]. When O is an original inverse operation, $C(O)$ must include the operation to be undone by O (see statement 2 of Definition 4), so CC1 preserves the *do-undo* ordering among normal and inverse operations [6].

Context-based Condition 2 (CC2). Given an original operation O and a document state DS, where $O \notin DS$ and $C(O) \subseteq DS$, $DS - C(O)$[2] is the set of operations that O must be transformed against before being executed on DS.

CC2 determines the group of target operations that O should be transformed against in order to be correctly executed on DS. It can be shown that target operations in $DS - C(O)$ are context-independent of O (Theorem 3 in Appendix A.1). When O is an original normal operation, $DS - C(O)$ must include all executed original operations that are concurrent with O (according to Theorem 1). Therefore, CC2 covers the causality-based transformation condition that O should be transformed against concurrent operations [19]. When O is an inverse operation, $DS - C(O)$ must include those operations that are generated after the operation to be undone by O [6]. Thus, CC2 correctly determines the set of target operations for an inverse operation as well. In summary, CC2 gives a uniformed condition in determining target operations for both original normal and inverse operations.

Context-based Condition 3 (CC3). Given any operation O and a state DS, O can be executed on DS only if $C(O) = DS$.

CC3 imposes a *context-state-equivalence* condition for executing all types of operation in OT systems. Every OT algorithm must ensure CC3 for correctness.

Context-based Condition 4 (CC4). Given an original operation O_x and an operation O of any type, where $O_x \notin C(O)$, O_x can be transformed to the context of O only if $C(O_x) \subseteq C(O)$.

CC4 ensures that O_x is transformable to the context of O. If $C(O_x) \not\subseteq C(O)$, then there must be an original operation $O_y \in C(O_x)$, but $O_y \notin C(O)$. Under such circumstances, O_x cannot be transformed to the new context $C(O)$ since IT-transformation cannot remove O_y from $C(O_x)$ (see statement 3 of Definition 4).

When O_x is one of target operations for O to be transformed against, as determined by CC2 and CC5 (see below), CC4 imposes an ordering condition for O_x to be selected from a group of target operations for transformation with O. It can be verified that meeting CC4 is equivalent to selecting target operations in their context-dependency order (Theorem 8 in Appendix A.3).

Context-based Condition 5 (CC5). Given an original operation O_x and an operation O of any type, where $O_x \notin C(O)$ and $C(O_x) \subseteq C(O)$, $C(O) - C(O_x)$ is the set of operations that O_x must be transformed against before being IT-transformed with O.

CC5 determines the target operations that O_x must be transformed against in order to be defined on the context of O.

Context-based Condition 6 (CC6). Given two operations O_a and O_b, they can be IT-transformed with each other, $IT(O_a, O_b)$ or $IT(O_b, O_a)$, only if $C(O_a) = C(O_b)$.

CC6 imposes a *context-equivalence* condition on applying IT-transformation between two operations of any type [20]. Every OT algorithm must ensure CC6 for correctness.

It can be shown that under CC1 and CC6, O_x is in the context of O if and only if O is context-dependent on O_x (Theorem 4 in Appendix A.1). Furthermore, if O_y is a

2. $DS - C(O) = \{O_x | O_x \in DS \text{ and } O_x \notin C(O)\}$, which is the set difference of DS and $C(O)$.

TABLE 2
Summary of COT Conditions

Context-based Conditions	Brief Description
CC1: $C(O) \subseteq DS$	a condition for O to be transformed for execution on DS
CC2: $DS - C(O)$	target operations for O to be transformed to the state DS
CC3: $C(O) = DS$	a condition for O to be executed on DS
CC4: $C(O_x) \subseteq C(O)$	a condition for O_x to be transformed to the same context of O
CC5: $C(O) - C(O_x)$	target operations for O_x to be transformed to the context $C(O)$
CC6: $C(O_a) = C(O_b)$	a condition for $O_a \& O_b$ to be IT-transformed

transformation target of O_x determined by CC5, then O_x and O_y must be context-independent (Theorem 5 in Appendix A.1).

Table 2 gives a summary of context-based conditions: CC1 and CC4 are required for ensuring correct ordering of operation execution and transformation, CC2 and CC5 are required for determining correct transformation target operations, and CC3 and CC6 are required for ensuring correct operation execution and transformation. Apart from CC1, which must be ensured by external schemes/protocols before invoking the OT algorithm, CC2-CC6 must be ensured by the OT algorithm. These context-based conditions are described declaratively and impose no specific way of implementation. They form the foundation for the COT algorithm in Sections 5 and 7.

4 Context Vectors

To represent the set of operations in a context efficiently, we devised the *context vector*, which can be used for time stamping operations in propagation. The context vector is an important element of the operation context theory.

4.1 Representing Original Normal Operations in a Context

For notational convenience, we assume that a collaborative editing session consists of N collaborating sites, identified by $0, 1, \ldots, N-1$. The original normal operations generated at each site are strictly sequential, so each of them can be uniquely identified by a pair of integers (sid, ns), where sid is the site identifier, and ns is the local sequence number of this operation.

Let O_{ij} be an original normal operation generated at site i with a sequence number j. If O_{ij} is included in a context $C(O)$, then $O_{i1}, O_{i2}, \ldots, O_{ij-1}$ must also be included in $C(O)$ according to Definition 3 and Definition 4. Therefore, all normal operations generated at the same site can be sufficiently characterized by the largest sequence number of these operations.

According to their generation sites, original normal operations in a context can be partitioned into N groups, so we can use N integers to represent all original normal operations in a context.

4.2 Representing Original Inverse Operations in a Context

An original inverse operation can be generated to undo an original normal operation or to redo an undone operation. The second case may occur when an undone operation is redone later or when multiple concurrent undo commands are generated to undo the same operation (see details in Section 4.4). Under all circumstances, each original inverse operation directly or indirectly corresponds to exactly one original normal operation. For example, an original inverse $\overline{O}$ may be generated to undo a normal operation O, and another original inverse $\overline{\overline{O}}$ may be generated to undo $\overline{O}$. Both $\overline{O}$ and $\overline{\overline{O}}$ correspond to the same normal operation O. Based on this observation, all original inverse operations in an operation context can be grouped by their corresponding original normal operations: one inverse group for each undone original normal operation.

Inverse operations in the same inverse group can be further differentiated by a sequence number based on their execution order within this group. For example, $\overline{O}$ and $\overline{\overline{O}}$ are in the same inverse group corresponding to O and executed in this order, so $\overline{O}$ has a sequence number "1," and $\overline{\overline{O}}$ has a sequence number "2." In general, an original inverse can be identified by a triple (sid, ns, is), where sid and ns are the site identifier and sequence number of the corresponding original normal operation, and is is the inverse sequence number within the group.

Inverse groups can be further partitioned into N inverse clusters according to the site identifiers of their corresponding normal operations. The inverse cluster ic_i at site i can be expressed as

$$ic_i = \left[(ns^0, is^0), (ns^1, is^1), \ldots, (ns^{k-1}, is^{k-1})\right],$$

where each pair (ns^j, is^j), $0 \leq j < k$, represents an inverse group with is^j inverse operations, which correspond to the same original normal operation whose sequence number is ns^j and site identifier is i. If no normal operation at site i has been undone, ic_i is empty.

4.3 Representing Both Normal and Inverse Operations in a Context

To represent an operation context with both original normal and inverse operations, we use an N-dimensional context vector, which is defined below.

Definition 7. Context vector. *Given an operation O, its context $C(O)$ can be represented by the following context vector $CV(O)$:*

$$CV(O) = [(ns_0, ic_0), (ns_1, ic_1), \ldots, (ns_{N-1}, ic_{N-1})],$$

where for $0 \leq i \leq N-1$,

1. *ns_i represents all original normal operations generated at site i, and*
2. *$ic_i = [(ns^0, is^0), (ns^1, is^1), \ldots, (ns^{k-1}, is^{k-1})]$ represents all inverse operations for undoing normal operations generated at site i, where (ns^j, is^j), $0 \leq j < k$, represents an inverse group with is^j inverses related to the normal operation with sequence number ns^j.*

To refer to various elements in $CV(O)$, the following notations shall be used: $CV(O)[i]$ refers to (ns_i, ic_i), $CV(O)[i].ns$ refers to ns_i, and $CV(O)[i].ic$ refers to ic_i. In the absence of inverse operations in an operation context, all

ic_i, $0 \leq i \leq N-1$, would be empty, and a context vector would be reduced to a state vector [2].

Based on Definition 7, it is straightforward to derive the scheme for maintaining the vector representation for the document state after executing each operation (according to Definition 3). Moreover, the vector representation of operation context can also be used to efficiently detect context-based relations and conditions. These technical details are provided in Appendix B.

In this paper, we have focused on how to define and use context vectors to correctly represent operation contexts, without concern about the size of context vectors. The context vector size may become an issue if the number of users in a session is very large (e.g., over hundreds/thousands) and/or if the number of inverse operations accumulated in the document state representation becomes very large. One efficient way of representing inverse operations in context vectors is to use the sequence-length encoding method: a large number of inverse operations associated with a consecutive sequence of normal operations generated by the same user can be represented by two integers, i.e., the lowest sequence number of those normal operations and a length count of the sequence. This compression method is most effective when operations are undone chronologically, which often occurs in real undo usage [3], [5], [22]. Another way is to keep the number of accumulated inverse operations under control by regularly performing garbage collection on operations in the document state representation. The reader is referred to [20] for operation garbage collection and to [23] for state vector compression in OT systems, which can be adopted to COT systems.

4.4 Inverse Representation and Multi-Undo Effect

The inverse representation in the context vector is related to the definition of multi-undo effects for concurrent undo commands targeting the same normal operation. In the proposed context vector definition, a single sequence counter is used to record the total number of inverse operations related to the same undone normal operation. The inverse sequence counter is able to capture the knowledge of how many times the original normal operation has been undone, but unable to differentiate whether those undo commands are concurrent or sequential. This inverse representation is able to represent the document state and context in systems that support the same multi-undo effect for both concurrent and sequential undo commands targeting a common normal operation, and this multi-undo effect is independent of the order of interpreting these undo commands. We call the above multi-undo as the Serialized and Order-independent Multi-undo Effect (SOME).

SOME can support the eXclusive-OR (XOR) logic effect: an operation is undone if the number of undo commands performed on it is odd and not undone if the number is even. This XOR effect can be achieved by having a later undo command reverse the effect of the previous undo command. Alternatively, SOME can also support the Inclusive-OR (IOR)logic effect: an operation is undone if there is one undo command performed on it, which can be achieved by having the first undo command take effect, but later undo commands take no effect. Both XOR and IOR effects have well-defined semantics and may be equally easy to understand by users.

The XOR multi-undo effect was first proposed and implemented in the REDUCE system [22] and has also been implemented in the Generic Collaboration Engine for CoWord [11]. From our experience, the XOR effect is easy to implement since it allows a uniformed treatment to both concurrent and sequential undo and to both undo and redo (implemented as undoing an undone); it is also easy to use thanks to its unique feature of changing the undo status of an operation with a click of any undo button. If multiple users simultaneously issue undo commands targeting the same normal operation and if the resulting XOR effect is not what these users wanted, just one more click of any undo button by any user would fix it. This XOR multi-undo effect is similar to the XOR logic effect commonly offered by the multiuser interface-multiple power switch buttons located at different places of a building for controlling a single lamp: the lamp state (light on or off) can be changed by pushing any button. In case that multiple users (unintentionally) simultaneously push different buttons and the resulting lamp state is not what these users wanted, any user can push any button to fix it.

As the proposed inverse representation in the context vector does not capture the concurrency relationship among undo commands targeting the same normal operation, it cannot represent document states for systems that support different multi-undo effects for concurrent and sequential undo commands, e.g., supporting the IOR effect for concurrent undo commands but the XOR effect for sequential undo commands. It is unclear what benefit the user may gain from being offered different multi-undo effects for concurrent and sequential undo commands and whether the benefit (if any) is worth the complexity and cost in differentiating concurrent and sequential undo commands in inverse representation. Usability study is needed to evaluate different multi-undo effects.

5 The Basic COT Algorithm

The COT algorithm has two entries: the COT-DO entry for consistency maintenance (do) and the COT-UNDO entry for supporting undo. Operation context and context-based conditions are at the core of the whole COT algorithm.

In the COT algorithm description, we use the context set representation $C(O)$, rather than the context vector representation $CV(O)$. This is because the context set representation is not only concise in expression but also directly implementable. Moreover, a document state DS is expressed as a set of original operations as well. By using original operation set expressions, we keep the COT algorithm independent of internal operation buffering schemes. We shall discuss operation buffering schemes in the COT framework in Section 8.

When an operation O is propagated from the local site to remote sites, however, it is the context vector, not the operation set, that is actually piggybacked on O. Based on the information in $CV(O)$, operations in $C(O)$ can be easily determined from operations in DS. The mapping between context-set-based and context-vector-based conditions can be found in Appendix B (Table 8).

5.1 COT-DO

COT-DO takes two parameters: an original operation O to be executed and the current document state DS. Before COT-DO is invoked, the following condition must met: $C(O) \subseteq DS$ (CC1). Methods for ensuring CC1 shall be discussed in Section 5.3.1.

```
Algorithm 1: COT-DO(O, DS)
  1) $transform(O, DS - C(O))$;
  2) Execute $O$; $DS := DS \cup \{org(O)\}$.
```

```
Procedure 1: transform$(O, \mathrm{CD})$
  Repeat until $CD = \{\ \}$:
  1) Select and remove $O_x$ from $CD$, where $C(O_x) \subseteq C(O)$;
  2) $transform(O_x, C(O) - C(O_x))$;
  3) $O := IT(O, O_x)$; $C(O) := C(O) \cup \{org(O_x)\}$.
```

COT-DO first invokes $transform(\ )$ to transform O against operations in $DS - C(O)$ (CC2). This is to upgrade the context of O to DS. In step 2, CC3 ($C(O) = DS$) must be met, so O is executed, and the original of O is added to DS (according to statement 2 of Definition 3).

The heart of COT-DO is $transform(O, CD)$, which is to transform O against operations in the context difference CD between $C(O)$ and a new context on which O is to be defined. This procedure repeats the following steps until CD becomes empty:

1. Select and remove an operation O_x from CD, where $C(O_x) \subseteq C(O)$ (CC4). One convenient and efficient way to ensure CC4 is to select operations in CD in their execution order determined by CC1 (see Section 5.3.2 for more details).
2. The procedure $transform(\ )$ is recursively invoked to transform O_x against operations in $C(O) - C(O_x)$ (CC5). This is to upgrade O_x to the context of O, so that they can be used for IT-transformation in the next step.
3. After the recursive call to $transform(\ )$, CC6 ($C(O) = C(O_x)$) must be met, so O is IT-transformed against O_x, and the context of O is updated by adding the original of O_x (according to statement 3 of Definition 4).

As an example, we show how COT-DO works in processing O_1, O_2, and O_3 at site 0 in Fig. 1. Let $DS_0 = \{\ \}$:

1. After the generation of O_1, since $C(O_1) = DS_0$, O_1 is executed as is, and DS_0 is updated to $DS_1 = \{O_1\}$.
2. When O_2 arrives with $C(O_2) = \{\ \}$, $transform(O_2, DS_1 - C(O_2))$ is called, where $DS_1 - C(O_2) = \{O_1\}$.

   Inside $transform(O_2, \{O_1\})$, since $C(O_1) = C(O_2)$, we have $O_2' := IT(O_2, O_1)$,[3] and $C(O_2') = \{O_1\}$.

   Returning from $transform(O_2, \{O_1\})$, we have $C(O_2') = DS_1$, so O_2' is executed, and DS_1 is updated to $DS_2 = \{O_1, O_2\}$, where $O_2 = org(O_2')$.
3. When O_3 arrives with $C(O_3) = \{O_2\}$, $transform(O_3, DS_2 - C(O_3))$ is called, where $DS_2 - C(O_3) = \{O_1\}$.

   Inside $transform(O_3, \{O_1\})$, since $C(O_1) \neq C(O_3)$, $transform(O_1, C(O_3) - C(O_1))$ is recursively called, with $C(O_3) - C(O_1) = \{O_2\}$ (which is the key step in detecting the dOPT puzzle).

   In the recursive $transform(O_1, \{O_2\})$, since $C(O_2) = C(O_1)$, we have $O_1' := IT(O_1, O_2)$, and $C(O_1') = \{O_2\}$.

   Returning from the recursion, we have $C(O_1') = C(O_3)$, so $C(O_3') := IT(O_3, O_1')$ (the dOPT puzzle resolved here), and $C(O_3') = \{O_1, O_2\}$, where $O_1 = org(O_1')$.

   After returning from $transform(O_3, \{O_1\})$, $C(O_3') = DS_2$, so O_3' is executed, and DS_2 is updated to $DS_3 = \{O_1, O_2, O_3\}$, where $O_3 = org(O_3')$.

3. In the $transform(O, CD)$ function definition, the IT-transformation result is assigned back to the original operation variable: $O := IT(O, O_x)$. In the example, however, we use the symbol O' to represent the IT-transformation result to highlight the fact that O has been transformed into a new form.

5.2 COT-UNDO

To undo an operation O, a command $Undo(O)$ must be issued by a user. How to select O and generate the undo command is part of the *undo policy* [6]. This paper is confined to the *undo mechanism*, which determines how to undo the selected operation in a given context.

In COT-UNDO, $Undo(O)$ is interpreted as an inverse $\overline{O}$ with $C(\overline{O}) = C(O) \cup \{O\}$. COT-UNDO takes two input parameters: the command $Undo(O)$ and the current document state DS, where O can be any operation in DS.

```
Algorithm 2: COT-UNDO(Undo(O), DS)
  1) $\overline{O} := makeInverse(O)$; $C(\overline{O}) := C(O) \cup \{O\}$;
  2) $COT\text{-}DO(\overline{O}, DS)$.
```

COT-UNDO first creates an inverse $\overline{O}$ by invoking $makeInverse(O)$,[4] with its context $C(\overline{O}) := C(O) \cup \{O\}$ (statement 2 in Definition 4), and then invoking COT-DO to handle $\overline{O}$.

For example, to interpret $Undo(O_2)$ at site 1 in Fig. 1, COT-UNDO is invoked with parameters $Undo(O_2)$ and $DS = \{O_1, O_2, O_3\}$. First, $\overline{O_2}$ and $C(\overline{O_2}) = \{O_2\}$ are created. Then, COT-DO is invoked with parameters $\overline{O_2}$ and DS. Inside COT-DO, $transform(\overline{O_2}, DS - C(\overline{O_2}))$ shall be invoked, and $\overline{O_2}$ shall be transformed against O_1 and O_3 since $CD = DS - C(\overline{O_2}) = \{O_1, O_3\}$. This example shows that an inverse operation can be directly handled by COT-DO and CC1-CC6 are uniformly applied to both normal and inverse operations.

5.3 COT Correctness with Respect to Context-Based Conditions

5.3.1 Ensuring CC1

For $COT\text{-}DO(O, DS)$ to work properly, the precondition $C(O) \subseteq DS$ (CC1) must be met before COT-DO is invoked. For a local operation O, we have $C(O) = DS$ according to statement 1 of Definition 4, which means that CC1 is automatically met. For remote operations, there are various ways of ensuring CC1, depending on the system communication topology. In systems with a full-connection communication topology, each collaborating site may directly propagate operations (time stamped with context vectors) to other sites. When a remote operation O arrives, its context $C(O)$ must be compared with the local DS. If CC1 is met, then

4. The reader is referred to [13] for precise definitions of the three primitive operations *Insert*, *Delete*, and *Update* and their corresponding inverses. The $makeInverse(O)$ procedure directly follows those definitions.

$COT\text{-}DO(O, DS)$ can be called; otherwise, O should be kept in an internal waiting queue. After executing a remote operation and saving its original in DS, the internal waiting queue should be examined to process waiting operations that are now meeting the requirement of CC1.

In systems with a star-like communication topology, each collaborating site has one first-in, first-out (FIFO) communication channel (e.g., a TCP connection) with a central notifier, which propagates operations among all collaborating sites. Since operations are serialized at the central notifier and each communication channel has the FIFO property, they are guaranteed to arrive at all sites in their causal-dependent order [8], which is also a context-dependent order (according to Theorem 1). Therefore, CC1 is automatically guaranteed if remote operations are processed in their order of arrival.

It is worth pointing out that the context relation covers inverse operations but not undo commands. The causal relationship between an undo command and other operations may not be reflected by the context relationship among the corresponding inverse operation and other operations. However, such causal relationships may be meaningful to users, e.g., users may wish to see that the undo effect takes place after all operations causally before the undo command have been executed. To address such issues, suitable causality-preservation schemes should be used to enforce an inverse operation to be executed after those context-independent operations that are causally before the corresponding undo command. In systems with a star-like communication topology, this order can be automatically enforced if remote operations are processed in their order of arrival.

5.3.2 Ensuring CC4

For $transform(O, CD)$ to work properly, the condition $C(O_x) \subseteq C(O)$ (CC4) must be met when selecting O_x from CD for transformation with O. One method of ensuring CC4 is to select operations in CD in an order that respects their context-dependency relation: for O_x, $O_y \in CD$, if $O_x \xrightarrow{c} O_y$, then O must be transformed against O_x before O_y; if $O_x \parallel_c O_y$, then O can be transformed against them in any order. It can be shown that meeting CC4 is equivalent to selecting operations for transformation in an order that respects their context-dependency relationships (Theorem 8 in Appendix A.3).

There potentially exist many different operation orders that are consistent with their context-dependency order and hence meet CC4. The execution order determined by CC1 is one of them (Corollary 2 in Appendix A.3). The orders required by CC1 and CC4 do not have to be the same, but it is convenient to use the operation execution order determined by CC1 as the operation transformation order required by CC4.

5.3.3 COT Correctness with Respect to CC3 and CC6

From the definitions of COT procedures, it can be verified that COT is able to ensure that 1) an operation O is executed on a document state DS only if $C(O) = DS$ (Theorem 9 in Appendix A.3); and 2) two operations O_a and O_b are IT-transformed only if $C(O_a) = C(O_b)$ (Theorem 10 in Appendix A.3).

6 Transformation Properties and Preconditions

COT is a high-level control algorithm responsible for determining which operation should be transformed against other operations according to context-based conditions. Another important component of an OT system is the low-level transformation functions responsible for transforming operations according to their types and parameters. Past research has identified a number of transformation properties that must be maintained by either control algorithms or transformation functions for ensuring the correctness of an OT system. Different OT systems may have different control algorithms, different transformation functions, and different divisions of responsibilities among these components.

The basic COT algorithm is simple yet powerful—capable of doing and undoing any operations at any time. Among all prior OT systems, only the combination of GOTO and ANYUNDO, referred to as GOTO-ANYUNDO, has similar capabilities [19], [6]. Different from GOTO-ANYUNDO, the basic COT algorithm does not use Exclusion Transformation (ET) functions, thus avoiding the requirement of the *Reversibility Property* (RP) between IT and ET functions [6].

Similar to GOTO-ANYUNDO, the basic COT algorithm assumes that transformation functions are capable of preserving the following properties [2], [3], [4], [20], [6]:

1. **Convergence Property 1** (CP1).[5] Given a document state DS and two context-equivalent operations O_a and O_b, if $O'_a = IT(O_a, O_b)$ and $O'_b = IT(O_b, O_a)$, then
$$DS \circ O_a \circ O'_b = DS \circ O_b \circ O'_a,$$
which means that applying O_a and O'_b in sequence on DS is equivalent to applying O_b and O'_a in sequence on DS.
2. **Convergence Property 2** (CP2). Given three context-equivalent operations O_a, O_b, and O_c, if $O'_b = IT(O_b, O_c)$ and $O'_c = IT(O_c, O_b)$, then
$$IT\big(IT(O_a, O_b), O'_c\big) = IT\big(IT(O_a, O_c), O'_b\big),$$
which means that the outcome of transforming O_a against O_b and O'_c in sequence equals the outcome of transforming O_a against O_c and O'_b in sequence.
3. **Inverse Property 2** (IP2).[6] Given any operation O and a pair of operations O_x and $\overline{O_x}$, it must be that
$$IT\big(IT(O, O_x), \overline{O_x}\big) = IT(O, I) = O,$$
which means that the outcome of transforming O against O_x and $\overline{O_x}$ in sequence equals the outcome of transforming O against the identity operation I.
4. **Inverse Property 3** (IP3). Given two context-equivalent operations O_a and O_b, if $O'_a = IT(O_a, O_b)$, $O'_b = IT(O_b, O_a)$, and $\overline{O_a}' = IT(\overline{O_a}, O'_b)$, then
$$\overline{O_a}' = \overline{O'_a},$$

5. **CP1 and CP2** in this paper (and in [6]) are the same as **Transformation Properties 1 and 2** in [4].

6. There is an **Inverse Property 1** (IP1) that is required in an OT system for achieving the correct undo effect [6], but IP1 is not related to IT functions and, hence, not included here.

which means that the transformed inverse operation $\overline{O_a}'$ equals the inverse of the transformed operation $\overline{O'_a}$.

The above transformation properties are important discoveries of past research, but they are not unconditionally required. The Pre-Conditions (PCs) for requiring them, however, were never explicitly stated in their specifications, which has unfortunately caused quite some confusion in OT literature. To clarify the situation and help explore alternative solutions, we explicitly state the PCs for CP1, CP2, IP2, and IP3 below:

1. **PC-CP1**. CP1 is required only if the OT system allows any two operations to be executed in different orders.
2. **PC-CP2**. CP2 is required only if the OT system allows two operations O_a and O_b to be IT-transformed in different contexts: $IT(O_a, O_b)$ and $IT(O'_a, O'_b)$, where $C(O_a) = C(O_b) \neq C(O'_a) = C(O'_b)$.
3. **PC-IP2**. IP2 is required only if the OT system allows an operation O to be transformed against a pair of do and undo operations O_x and $\overline{O_x}$ one by one.
4. **PC-IP3**. IP3 is required only if the OT system allows an inverse operation $\overline{O_a}$ to be transformed against O_b, which is context-independent of O_a.

PC-PC1, PC-IP2, and PC-IP3 are derived directly from definitions of CP1, IP2, and IP3. PC-CP2 does not directly correspond to the definition of CP2. It has been verified that if PC-CP2 is broken, CP2 would not be required since CP1 would be sufficient to ensure convergence (Theorem 11 in Appendix A.4); also, no operation would be transformed against the same group of operations in different orders (Theorem 12 in Appendix A.4).

Generally, there are two ways to achieve OT correctness with respect to these transformation properties: one is to design transformation functions capable of preserving these properties; the other is to design control algorithms capable of breaking their pre-conditions. Past research has shown that it is relatively easy to design transformation functions capable of preserving CP1 but nontrivial to design and formally verify transformation functions capable of preserving CP2, IP2, and IP3. Counterexamples illustrating the violation of these properties in published transformation functions can be found in [20], [6], [24], and [25]. IT functions capable of preserving IP2 and IP3 had been devised in the context of ANYUNDO [6], but our experience in implementing these functions revealed that those solutions are quite intricate, and inefficient (more analysis in Section 9). Therefore, solving CP2, IP2, and IP3 at the control algorithm level has the benefits of simplifying the design of transformation functions and improving system efficiency. In the following section, we extend the basic COT algorithm to provide cleaner and more efficient solutions to CP2, IP2, and IP3 at the control algorithm level.

7 COT Solutions to CP2, IP2, and IP3

A distinctive feature of COT is that in each invocation of $transform(O, CD)$, the whole set of target operations have been provided in CD in advance. With all target operations known at the start, we are able to properly arrange these operations so that precondtions for CP2, IP2, and IP3 can be broken.

7.1 Extended $transform(O, CD)$ Procedure

We extend $transform(O, CD)$ to take advantage of the information in CD. The extended $transform(\ )$ (Procedure 2) retains the structure and main elements of Procedure 1 but add new procedures $ensure\_TPsafety(O, CD)$, $XIT(O, O_x)$, and $make\_IP3safe\_Inverse(\overline{O}, CD))$ for solving CP2, IP2, and IP3.

```
Procedure 2: $transform(O, \mathrm{CD})$
  1) If $CD \neq \{\ \}$, $ensure\_TPsafety(O, CD)$;
  2) Repeat until $CD = \{\ \}$:
      a) Remove the first operation $O_x$ from $CD$;
      b) $transform(O_x, C(O) - C(O_x))$;
      c) $XIT(O, O_x)$;
```

```
Procedure 3: $ensure\_TPsafety(O, CD)$
  1) Ensure CP2-safety: sort operations in $CD$ in a total
      order that respects their context-dependency order, as
      specified in Definition 9.
  2) Ensure IP2-safety: for any $O_x \in CD$, if $\overline{O_x} \in CD$, mark
      $O_x$ as a do-undo pair, and remove $\overline{O_x}$ from $CD$.
  3) Ensure IP3-safety: if $O$ is inverse, invoke $make\_$
      $IP3safe\_Inverse(O, CD)$.
```

```
Procedure 4: $make\_IP3safe\_Inverse(\overline{O}, CD)$
  1) $O := makeInverse(\overline{O})$; $C(O) := C(\overline{O}) - \{O\}$;
  2) $NCD := \{O_x | O_x \in CD$ and $O_x \parallel O\}$;
  3) $transform(O, NCD)$;
  4) $\overline{O} := makeInverse(O)$; $C(\overline{O}) := C(O) \cup \{org(O)\}$;
  5) $CD := CD - NCD$.
```

```
Procedure 5: $XIT(O, O_x)$
  1) If $O_x$ is a do-undo pair,
  2) then $C(O) := C(O) \cup \{org(O_x), org(\overline{O_x})\}$;
  3) else $O := IT(O, O_x)$; $C(O) := C(O) \cup \{org(O_x)\}$.
```

7.2 Breaking the Precondition for CP2

The basic idea is to force the order of remote operation execution (governed by CC1) and the order of operation transformation (governed by CC4) to be the same total ordering that respects the context-dependency relation among original operations. The CC1-related execution order control must be supported by external protocols before invoking COT, whereas the CC4-related transformation order control is built in COT in step 1 of Procedure 3.

To elaborate, let "$\overset{c}{\Rightarrow}$" denote a total ordering of original operations that respects their context-dependency relationships. We specify the following operation execution ordering scheme to ensure CC1 and to control operation execution orders.

Definition 8. "$\overset{c}{\Rightarrow}$"-based operation execution ordering. *An operation O can be executed on document state DS only if 1) O is a local editing operation or 2) for any operation O_x, if $O_x \overset{c}{\Rightarrow} O$, then O_x has been executed, i.e., $O_x \in DS$.*

It should be stressed that under Definition 8, a local editing operation O may be executed immediately after generation, even if there is a remote operation O_x such that $O_x \overset{c}{\Rightarrow} O$. For a group of N context-equivalent original operations, there are only N *permissible* execution orders, each starting from one of these N operations. As an example, for a group of three context-equivalent operations $\{O_a, O_b, O_c\}$, if $O_a \overset{c}{\Rightarrow} O_b \overset{c}{\Rightarrow} O_c$, three permissible execution orders are $<O_a, O_b, O_c>$, $<O_b, O_a, O_c>$, and $<O_c, O_a, O_b>$.

Furthermore, we use the following transformation ordering scheme to ensure CC4 and to force a group of operations to be selected for transformation in the same "$\overset{c}{\Rightarrow}$" order.

Definition 9. "$\overset{c}{\Rightarrow}$"-based operation transformation ordering. *In $transform(O, CD)$, for any two operations O_a and O_b in CD, if $O_a \overset{c}{\Rightarrow} O_b$, then O_a is selected for transformation with O before O_b, where "$\overset{c}{\Rightarrow}$" is the same as the one used in Definition 8.*

It should be noted that CD becomes an ordered set after the sorting. The first O_x in CD must meet the condition $C(O_x) \subseteq C(O)$ in step 2a of $transform(O, CD)$ (Procedure 1), so this condition is no longer explicitly specified in Procedure 2.

Based on Definitions 8 and 9, it can be verified the COT algorithm eliminates the possibility of transforming a pair of operations under different contexts, thus breaking PC-CP2 (Lemma 2 and Theorem 13 in Appendix A.5).

There are various ways to enforce the "$\overset{c}{\Rightarrow}$"-based execution ordering before invoking COT. For systems with an existing central document repository and notification server (such as REDUCE [6] and CoWord [11]), the operation broadcasting order determined by the central server is a total order that can be directly used in the "$\overset{c}{\Rightarrow}$"-based operation execution and transformation ordering schemes. This method is simple and obtained for free (since the server already exists for other purposes) and also most suitable and adequate for real-time collaborative editing sessions typically involving less than 10 users. For systems without a central server or unsuitable to use a central server, distributed protocols can be used to achieve such a total ordering [18], [26]. The COT solution to PC-PC2 is independent of whether the "$\overset{c}{\Rightarrow}$"-based execution ordering is achieved by a distributed or centralized protocol.

7.3 Breaking the Precondition for IP2

The basic idea is to make sure that an operation is never transformed against a pair of do and undo operations one by one. This solution consists of two parts: 1) in step 2 of $ensure\_TPsafety(O, CD)$, operations are coupled with their corresponding inverses if they are included in the same context difference CD, and 2) in the new *eXtended IT* procedure $XIT(O, O_x)()$, if O_x is found to be a *do-undo pair*, the IT-transformation of O against O_x is skipped (effectively treating this pair as an identity operation), and the context of O is updated by adding two operations, i.e., $\{org(O_x), org(\overline{O_x})\}$.

It can be verified that the extended COT algorithm eliminates the possibility of transforming an operation against a pair of normal and inverse operations one by one, thus breaking PC-IP2 (Theorem 14 in Appendix A.5).

It is instructive to examine how the COT solution resolves some well-known IP2 puzzles—representative scenarios in which the OT system may produce incorrect results if transformation functions fail to preserve IP2. Two kinds of IP puzzles have been identified in the context of GOTO-ANYUNDO [6]: one is the *coupled do-undo-pair trap*, in which an operation is transformed against a pair of do and undo operations in a single sequence; another is the *uncoupled do-undo-pair trap*, which occurs when an operation is transformed against a pair of do and undo operations in two different sequences.

The *coupled do-undo-pair trap* is clearly resolved by the combination of step 2 in $ensure\_TPsafety(CD)$ and $XIT(O, O_x)$. The *uncoupled do-undo-puzzle trap* can never occur in the COT algorithm. This is because 1) every transformation process in $transform(O, CD)$ always starts from an original operation O and 2) if O needs to be transformed against both O_x and $\overline{O_x}$, this pair of operations must always be included in CD at the same time and be coupled.

7.4 Breaking the Precondition for IP3

The basic idea is to make an inverse $\overline{O}$ to be *IP3-safe* with respect to the context difference CD to which $\overline{O}$ is to be transformed. An IP3-safe inverse is defined below.

Definition 10. *An inverse $\overline{O}$ is IP3-safe with respect to a context difference CD if $\overline{O}$ is made from a transformed version of $org(O)$, which has included all operations in CD that are context-independent of $org(O)$.*

The IP3 solution is encapsulated in $make\_IP3safe\_Inverse\ (\overline{O}, CD)$, which works as follows:

1. Create operation O by making the inverse of $\overline{O}$ and get $C(O) = C(\overline{O}) - \{O\}$.
2. Select all operations from CD that are context-independent of O and create a new context difference NCD.
3. Transform O against operations in NCD (by recursively invoking *transform*( )).
4. Create a new inverse from the transformed O.
5. Create a new CD by subtracting NCD from the old CD (the new CD must preserve the total order as required for solving CP2).

This new inverse $\overline{O}$ must be *IP3-safe* because it is created from a transformed operation whose context has included all operations in NCD. The *IP3-safe* inverse $\overline{O}$ shall never be transformed against the operations in NCD since these operations have been removed from the new CD in step 5.

It can be verified that the extended COT algorithm eliminates the possibility of transforming an inverse operation against operations that are context-independent of the operation to be undone, thus breaking PC-IP3 (Theorem 15 in Appendix A.5).

8 Operation Buffering in the COT Framework

Another distinctive feature of the COT work is the separation of the COT algorithm from the underlying operation buffering organization. The high-level algorithm focuses on the correctness aspect of the system and is based on operation contexts represented by original operations, without any concern about the internal organization of operation contexts and document states. This feature has resulted in not only a clean and simple algorithm, but also a

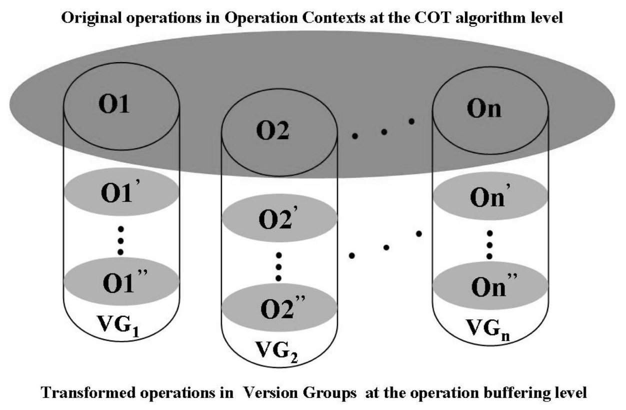

Original operations in Operation Contexts at the COT algorithm level
O1
O2
On
O1'
O1''
VG1
O2'
O2''
VG2
On'
On''
VGn
Transformed operations in Version Groups at the operation buffering level

Fig. 2. Correspondence between original operations at the COT algorithm level and *VGs* at the operation buffering level.

flexible operation buffering organization, which allows efficient implementation of the whole system. In this section, we present an operation buffering organization that saves both original and transformed operations and uses saved operations to achieve a time and space-efficient COT system, while retaining the correctness and cleanliness of the high-level algorithm.

8.1 Operation Version Group

A key idea in the COT buffering organization is the notion of *operation version group (VG)*. In an OT system, every operation is a *version* of an original operation: a transformed operation is a *transformed version* of its associated original operation; an original operation can be regarded as the *original version* of itself. We can organize all versions of the same original operation into an operation VG, and all operations in an OT system into a collection of VGs. Since each original operation has a unique identifier (see Definition 11), we can use this identifier to identify the corresponding VG as well. One operation belongs to one and only one VG, so we use notation $VG(O)$ to denote the VG to which operation O belongs.

For each original operation in an operation context at the algorithm level, there is a corresponding VG at the operation buffering level, as shown in Fig. 2. A version group can be regarded as an augmented representation of an original operation: one original operation can be defined on only one context, but one VG may have multiple versions defined on different contexts, thus being capable of matching different contexts. This *multi-context-matching* capability of a VG allows us to use saved versions to *replace* the original operation at the algorithm level according to the context requirement: when an original operation is used in a specific context at the algorithm level (e.g., at the beginning and step 2a of $transform(O, CD)$), the corresponding VG is searched for a version that meets the context requirement; if such a version exists in the VG, it is used to replace the original operation in the algorithm, thus saving the work to transform the original operation into this version.

Different versions in a VG may be used to represent the original operation in different contexts, but their effects on operation contexts or document states are the same, which is solely determined by the corresponding original operation (see Definition 4). This *single-context-effect* property of a VG allows the use of one original operation to *represent* all its versions in operation contexts and document states at the algorithm level.

The notion of VG provides the foundation for saving both original and transformed operations in the COT framework. The remaining technical issues are the following: which transformed versions should be saved and how to select correct versions from a VG for transformation. These are discussed in the following sections.

8.2 The Need for Saving Transformed Operations

To show the need of saving transformed operations, we first analyze the time complexity of the COT system that saves original operations only.

8.2.1 *Time Complexity in Transforming a Set of Operations*

Consider a scenario with three operations O_1, O_2, and O_3 defined on the same initial document state $DS_0 = \{\ \}$ and assume that they are executed in the order of O_1, O_2, and O_3:

1. After executing O_1, we have $DS_1 = \{O_1\}$.
2. When O_2 arrives, we have $DS_1 - C(O_2) = \{O_1\}$. Since $C(O_2) = C(O_1) = \{\ \}$, O_2 is directly IT-transformed against O_1, i.e., $O_2' = IT(O_2, O_1)$, and $C(O_2') = \{O_1\}$. Then, O_2' is executed on DS_1, and $DS_2 = \{O_1, O_2\}$. In processing O_2, one IT-transformation is performed.
3. When O_3 arrives, we have $CD = DS_2 - C(O_3) = \{O_1, O_2\}$. O_1 is first selected from CD; since $C(O_3) = C(O_1) = \{\ \}$, O_3 is directly IT-transformed against O_1, i.e., $O_3' = IT(O_3, O_1)$, and $C(O_3') = \{O_1\}$. Then, O_2 is selected from CD; since $C(O_3') - C(O_2) = \{O_1\}$, $transform(O_2, O_1)$ is recursively invoked to recreate $O_2' = IT(O_2, O_1)$. Finally, O_3' is IT-transformed against O_2' to create O_3'', i.e., $O_3'' = IT(O_3', O_2')$. In processing O_3, three IT-transformations are performed, one of which is recreating O_2', which repeats the work in processing O_2.

In general, for a set of n operations $O_1, O_2, O_3, \ldots, O_n$ defined on the same context, suppose they are executed in the order of their sequence numbers. The total number of IT-transformations performed in processing O_n is given by the following recurrence:

$$T(n) = \begin{cases} 0, & \text{if } n = 1, \\ \sum_{i=1}^{n-1} T(i) + (n-1), & \text{if } n > 1. \end{cases}$$

Solving the above recurrence relation, we obtain its closed-form expression: $T(n) = 2^{n-1} - 1$. The number of transformations required to process O_n is exponential in n. As n becomes large, we can obtain a tight bound on the asymptotic growth of this expression with $T(n) = \Theta(2^n)$. Observe that we must perform at least $n - 1$ transformations in creating the desired new version of O_n, since O_n is context-independent with all previously executed $n - 1$ operations. However, in this process, we incurred the extra cost of performing $T(1) + T(2) + \ldots + T(n-1)$ transformations to recreate operation versions for $O_1, O_2, \ldots, O_{n-1}$. Substantial savings can be attained by eliminating the recreation of operations (which dominate the time complexity expression) by saving previously created versions.

8.2.2 Time Complexity in Transforming Multiple Streams of Operations

Consider the following two streams of operations:

$$S_1 = O_{1,1}, O_{1,2}, \ldots, O_{1,n}; \quad S_2 = O_{2,1}, O_{2,2}, \ldots, O_{2,m},$$

where $C(O_{1,1}) = C(O_{2,1})$; $C(O_{1,i}) \cup \{O_{1,i}\} = C(O_{1,i+1})$, $1 \leq i < n$; and $C(O_{2,j}) \cup \{O_{2,j}\} = C(O_{2,j+1})$, $1 \leq j < m$.

Under the assumption that no transformed operation is saved, the following expression gives the exact number of IT-transformations performed in transforming $O_{2,m}$ against $O_{1,1}, O_{1,2}, \ldots, O_{1,n}$:

$$T(n,m) = \frac{(n+m)!}{(n! \times m!)} - 1.$$

Using Stirling's approximation [27], we can bound the expression's asymptotic growth, yielding $T(r) = \Theta(2^r/\sqrt{r})$, where $r = n + m$. Note that the growth is subexponential in the total number of operations involved and is strictly less than the case with r context-equivalent operations for $r > 1$ (see Section 8.2.1). Among the total $T(n,m)$ transformations, m transformations must be performed to create new n versions, but $T(n,m) - n$ transformations were performed to recreate versions. A detailed illustration of processing these two streams and the derivation of the above expressions are provided in Appendix C.1.

In general, with an operation buffering scheme that saves original operations only, the time complexity in transforming multiple streams of operations in the COT system would be exponential to the total number of operations involved. Most of these IT-transformations recreate versions, and the number of new and necessary IT-transformations is actually linear to the total number of context-independent operations involved.

With an operation buffering scheme that saves every operation version ever created during the execution of the COT system, it would be possible to completely eliminate recreation of versions and achieve linear complexity. The problem with this approach, however, is its high space complexity: each VG would potentially have to accommodate an unlimited number of versions.

The design goal of the COT operation buffering scheme is to selectively save transformed versions to eliminate retransformations in most common operation sequence patterns, while keeping the number of saved versions small.

8.3 Operation Buffering Schemes Saving Transformed Operations

8.3.1 General Considerations

To devise schemes for saving and using transformed operations in the VG-based buffering framework, the following points should be considered:

1. Transformed versions are not essential for the correctness of the algorithm. So if a transformed version is not saved, it has no impact on the correctness of the algorithm.
2. Not all transformed versions may be reused by later transformations. So, saving more transformed versions increases space complexity, but may not decrease time complexity.
3. Different transformed versions may have different likelihoods of being reused by later transformations. So the key is to selectively save those transformed versions that are mostly likely to be reused.

Due to the nondeterministic nature of operation generation by interactive users, it appears impossible to develop a deterministic algorithm to predict whether an operation version, at the time of its creation, will be reused later. However, by analyzing common patterns of operation sequences generated by users, we are able to discover heuristics for predicting the likelihood of a transformed version to be reused by future transformations. We can use these heuristics to selectively save transformed versions to maximize their reuse and minimize their space usage.

8.3.2 Saving the Most-Recently Created Versions

By experimentation, we have discovered some heuristics that are effective in eliminating retransformations for common operation sequence patterns. One key heuristics is that the most recently created versions (MRCVs) are likely to be reused by later transformations. The buffering scheme based on this heuristics is named as the MRCV scheme, which is sketched as follows:

1. Initially, a VG contains an original operation O.
2. Whenever a new version of O is created during the execution of the COT algorithm, this version is saved in the corresponding $VG(O)$; if there is another nonoriginal version in $VG(O)$, that version is removed.

The MRCV scheme maintains at most two versions in each VG: the original version plus the MRCV.

To make this scheme work effectively, we extend $transform(O, CD)$ (Procedure 2) by replacing $XIT(O, O_x)$ with a *Symmetric IT (SIT)* procedure, which is defined as follows:

$$SIT(O, O_x) = (O', O'_x),$$

where $O' = XIT(O, O_x)$, and $O'_x = XIT(O_x, O)$. With SIT, O and O_x shall be IT-transformed with each other to create two new versions in their corresponding VGs.

To illustrate the effectiveness of this scheme, we examine how it works in processing the two streams of operations discussed in Section 8.2.2 (see details in Appendix C.2). With a space cost of two versions in each VG, the basic MRCV scheme is able to completely eliminate retransformations and achieve the optimal linear time complexity for processing these two streams.

To achieve linear time complexity for handling an arbitrary number of streams of operations, the basic MRCV scheme can be extended to maintain a maximum of N versions in each VG, where N is the number of collaborating sites in a session. The N versions consist of one original version and $N - 1$ MRCVs corresponding to $N - 1$ remote collaborating sites.

The scenario of n context-equivalent operations is a special case of n streams of operations, where each stream has only one operation. Without saving any transformed operation, the number of IT-transformations required for transforming the nth operation against the other $n - 1$ operations would be $\Theta(2^n)$ (see Section 8.2.1). Under the extended MRCV scheme, the number of IT-transformations can be reduced to $2(n - 1)$ (the constant factor 2 is due to the use of SIT), which is an exponential to linear reduction with a maximum of n versions saved in each VG, where $n \leq N$.

8.3.3 Processing a Stream of Undo Commands under the MRCV Scheme

To examine how the MRCV scheme works in processing inverse operations, we continue with the prior example scenario with two streams of operations. After both S_1 and S_2 have been executed, suppose one user issues a stream of undo commands

$$Undo(O_{1,n}), \ldots, Undo(O_{1,2}), Undo(O_{1,1})$$

to undo operations in S_1 in a reverse chronological order. A detailed illustration of processing this stream of undo commands can be found in Appendix C.3. Under the MRCV scheme, for each undo command, an IP3-safe inverse can be generated without any IT-transformation since the VG associated with the operation to be undone has already a version that contains in its context all context-independent operations. Then, the IP3-safe inverse needs to be transformed against operations that are context dependent on the operation to be undone. A total of $(n - i)$ IT-transformations are needed to undo the ith operation in a reverse chronological order. If the user selectively undoes the ith operation without undoing $O_{1,i+1}, \ldots, O_{1,n}$, then the total number of IT-transformations shall be $2(n - i)$.

When there are only two streams of operations, the MRCV scheme is able to eliminate all retransformations involved in creating any IP3-safe inverse. However, when more than two streams of operations are involved in a scenario, there is no guarantee that every VG associated with an operation to be undone has an MRCV that contains all context-independent operations in its context. To create the IP3-safe inverse for undoing such an operation, we have to transform this operation against its context-independent operations, which has a quadratic time complexity with respect to the number of operations involved.

It should be pointed out that the VG-based operation buffering framework is not bound with the MRCV scheme but is generic and applicable to any buffering scheme. This framework provides a flexible environment for discovering and experimenting with new buffering schemes. Finally, when saved operations are no longer needed for future transformations or undo, they can be collected as garbage [20].

9 Discussions

9.1 Comparison to Causality and Prior Work on Operation Context

Based on the theory of causality, prior OT algorithms have used state vectors to capture causal-dependency relationships among original normal operations and to represent document states with effects of original normal operations. However, causal-dependency relationships are not defined for inverse or transformed operations, and state vectors cannot represent document states with original inverse operations. The theory of causality is unable to capture essential OT conditions for all types of operation—original and transformed normal and inverse operations.

To overcome the limitation of the causality theory, we first proposed the notion of operation context and used it in conjunction with the theory of causality in the GOT algorithm [20]. In early work, the context of an operation O was defined as a *sequence of transformed operations* that can be executed to bring the document from its initial state to the state on which O is defined, and there was no explicit representation of an operation context. The old definition of context was directly coupled to the sequential history buffering strategy, which saved operations in their execution forms. In the absence of explicit context representation, context relationships among operations had to be derived from their ordering relationships in the history buffer *plus* causal relationships between their original versions [20]. However, this way of deriving context relationships does not work with inverse operations since they have no causal relationship with any operation (see Section 3). To work around this problem, special technical patches had to be invented to maintain suitable relationships among inverse and other operations under different circumstances.

Based on the insights from prior work on operation context, we have redefined an operation context in this article as *a set of original operations* corresponding to the document state on which this operation is defined. This new concept of operation context is independent of the underlying operation buffering strategy and can be explicitly represented as an operation set. Based on the set representation of operation context, we have formulated CC1-CC6 to capture essential OT requirements that are uniformly applicable to all types of operation. Moreover, we have devised the context vector to efficiently represent both normal and inverse operations in a context. The context vector is more general than the state vector and potentially applicable to other distributed computing systems as well.

The main differences between the causality and context theories and between the old and new context theories are summarized in Tables 3 and 4, respectively.

TABLE 3
Comparison of the Causality Theory and the Context Theory

Issues	Causality Theory	Context Theory
Defined Operation Types	1. Original normal	1. Original normal 2. Original inverse 3. Transformed normal 4. Transformed inverse
OT-related Conditions	1. Causal-dependent ordering 2. Causal-independent targeting	1. Context-dependent ordering (CC1 and CC4) 2. Context-independent targeting (CC2 and CC5) 3. Context-equivalent execution/transformation (CC3 and CC6)
Document State Representation	A set of original normal operations	A set of original normal and inverse operations
Vector Representation	State Vector: 1. N integers for original normal operations	Context Vector: 1. N integers for original normal operations 2. N inverse-clusters for original inverse operations

9.2 Comparison to Prior OT Algorithms

9.2.1 COT versus Prior OT Algorithms

Most prior OT algorithms support consistency maintenance (do) only, including dOPT [2], CCU [28], Jupiter [29], GOT [20], GOTO [19], SOCT3/4 [30], NICE [7], treeOPT [10], TIBOT [31], and Mark & Retrace [32]. None of these OT systems addressed issues related to IP2 and IP3 since they do not support undo, but some of them are capable of solving CP2 at the control algorithm level, including the

TABLE 4
Comparison of the Old and New Context Theories

Issues	Old Context Theory	New Context Theory
Definition	A sequence of executed operations	A set of original operations
Representation	Implicit	Explicit
Relation to Causality	Complementary	Replacement
Relation to Buffering	Dependent	Independent
Formulation of OT-related Conditions	Context-equivalent execution (CC3) and IT-transform (CC6)	CC1 – CC6
Context Vector	No	Yes

Jupiter system by a central transformation-based server [29], the GOT system by an undo/redo scheme based on total ordering [20], the SOCT3/4 system by a control strategy based on a continuous global sequencing [30], the NICE system by a central transformation-based notifier [7], and the TIBOT system by a time-interval-based distributed synchronization protocol [31]. The COT work contributes the explicit specification and formal verification of PC-CP2 and is unique in decoupling schemes for breaking PC-CP2 from the core algorithm. As pointed out before, CP2 could be solved at either the OT algorithm or function level, and an OT designer may favor one over the other by assessing their relative benefits and complexities. The reader is referred to [24] and [25] for alternative approaches to solving CP2 at the function level.

Undo has long been recognized as a complex issue [33], [34], [35], [36], [37], [38]. Designing an OT algorithm for supporting both do and undo is significantly more challenging than designing an OT algorithm for supporting do only. DistEdit [3] is the first OT-based system supporting undo, but its consistency maintenance is based on locking, and its selective undo algorithm is unable to undo an operation if it is adjacent to or overlapping with another operation (conflict). The adOPTed algorithm [4], [5] is the first OT system capable of supporting both do and undo, but its undo capability is restricted to chronological undo. ANYUNDO [6] is the first undo algorithm capable of undoing any operation in any order and is combined with GOTO to support both do and undo. For detailed comparison of these prior OT-based undo solutions, the reader is referred to [6]. The following discussion focuses on the comparison of COT and GOTO-ANYUNDO since they are the only pair of OT systems with similar capabilities.

9.2.2 COT versus GOTO-ANYUNDO

Both COT and GOTO-ANYUNDO are capable of doing and undoing any operations at any time. The main difference is that COT achieves this capability without using ET functions (thus eliminating the RP requirement for IT functions) and without requiring IT functions to preserve CP2, IP2, and IP3. Avoiding RP and breaking pre-conditions for CP2, IP2, and IP3 have significantly simplified the design of transformation functions and the OT system as a whole.

COT is simpler than GOTO-ANYUNDO because COT is based on a single theory of operation context, which is uniformly applicable to all types of operation, whereas GOTO-ANYUNDO is based on a mixture of operation causality and history buffer ordering relationships and is further complicated by intricate operation reordering/coupling/decoupling for maintaining suitable operation relationships under various circumstances.

COT is independent of underlying buffering schemes, whereas GOTO-ANYUNDO is tightly coupled with the linear history buffering scheme. COT is more efficient than GOTO-ANYUNDO in supporting both do and undo. When combined with the MRCV buffering scheme, COT is able to achieve a linear time complexity for transforming a normal operation (for do) in the face of multiple streams of operations. To transform an inverse operation (for undo), the number of IT-transformations is linear to the number of operations generated after the operation to be undone. The number of IT-transformations involved in creating an IP3-safe inverse version is linear in common operation sequence patterns but exponential in the worst case to the number of operations that are context-independent of the operation to be undone. The maximum space cost is N versions in each VG, where N is the number of collaborating sites in a session.

In contrast, GOTO-ANYUNDO uses an IT-ET-based *transposition* procedure to reorder executed operations in the history buffer in order to enforce context-related conditions. For the transposition to work properly, each operation in the history buffer has to maintain two versions: one original version and one execution version. This is because the original version is needed in recursive calls to GOTO to achieve IT-ET reversibility in the face of lossy IT transformations [6]. To transform a normal operation (for do) in the face of multiple streams of operations, the worst case time complexity is exponential to the total number of operations involved. For more detailed discussions on time complexity and performance of prior OT algorithms, the reader is referred to [5], [6], and [39].

For the purpose of preserving IP2, the do part (a normal operation) and the undo part (an inverse operation) need to be coupled in both COT and GOTO-ANYUNDO. In GOTO-ANYUNDO, an *eager* coupling strategy was adopted: an inverse operation is coupled with its corresponding normal operation immediately after its execution. Under this scheme, inverse operations are not explicitly represented in the history buffer. When a normal operation is to be executed, however, it may need to be transformed against only the undo part of a *do-undo pair*. To cope with this problem, an extra *decouple-GOTO-recouple* scheme has to be used to decouple a *do-undo pair* before invoking GOTO and then recouple them afterward [6], which causes unnecessary retransformations. In contrast, a *lazy* coupling strategy is adopted in the COT algorithm: the coupling of a *do-undo pair* occurs not immediately after executing each inverse, but only when both the do part and the undo part appear in the same transformation process at some later stage. These strategies avoid retransformations caused by the eager coupling scheme and the decouple-recouple scheme.

In the GOTO-ANYUNDO-based system, the solution to IP3 is encapsulated in an IP3-preserving IT function (*IP3P-IT* [6]). Inside this function, an extended *ET* function has to be used, which may invoke the GOTO algorithm to ensure RP with the corresponding *IT* function. In contrast, the COT solution to IP3 is encapsulated in the high-level procedure $make\_IP3safe\_Inverse(\overline{O}, CD)$, which is more efficient since 1) it avoids converting $\overline{O}$ to O back and forth multiple times for each $O_x \in NCD$ (if $IP3P\text{-}IT(\overline{O}, O_x)$ were used instead) and 2) the *transform()* procedure is much cheaper than GOTO.

TABLE 5
Comparison of the COT and GOTO-ANYUNDO Systems

Issues	GOTO-ANYUNDO	COT
Ability to Do and Undo Any Operation Anytime	Yes	Yes
Inclusion Transformation	Used	Used
Exclusion Transformation	Used	Not Used
Reversibility Property	Required	Not Required
Convergence Property 1	By trans. functions	By trans. functions
Convergence Property 2	By trans. functions	By control algorithm
Inverse Property 1	By operation def.	By operation def.
Inverse Property 2	By trans. functions	By control algorithm
Inverse Property 3	By trans. functions	By control algorithm
Relation to Buffering	Dependent	Independent
Time Complexity (Do)	Exponential	Linear [a]
Time Complexity (Undo)	Exponential	Exponential
Max Number of Saved Versions per Operation	2	N (Number of Collaborating Sites)

[a] *Scheme which saves k operations, $1 \leq k \leq N$, where N is the total number of collaborating sites. If only a single original operation is buffered, COT has an exponential time complexity (see Section 8.2).*

The main similarities and differences between COT and GOTO-ANYUNDO are summarized in Table 5.

10 Conclusions

We have contributed the theory of operation context and the COT algorithm, which provide a new theoretical framework and uniformed solutions to both consistency maintenance and undo problems in distributed collaborative editing systems. With these results, we have achieved the goals to better understand and solve OT problems, reduce complexity, verify correctness, improve efficiency, and support the continual evolution of OT.

We started with identifying limitations of the causality theory as the foundation of OT: the causal relation is not defined for inverse operations for undo or for transformed operations obtained from transformations; causality-based conditions are not applicable to inverse and transformed operations and inadequate for ensuring correct IT-transformation. Technical patches for working around these limitations had resulted in intricate, complex, and inefficient OT algorithms.

We proposed the theory of operation context to replace the theory of causality as the new foundation of OT. Context-based conditions (CC1-CC6) have been formulated to capture essential OT requirements, including context-dependent ordering of operation execution and transformation, context-independent selection of transformation target operations, and context-equivalent execution and IT-transformation for all types of operation. Furthermore, the context vector has been devised to represent operation contexts efficiently for operation time stamping and propagation. The context vector is able to represent states with both normal and inverse operations and potentially applicable to other distributed computing applications.

Based on the theory of operation context, we devised a new OT algorithm—COT—which is distinctive in achieving the capability of doing and undoing any operation at any time without using ET functions and without requiring transformation functions to preserve CP2, IP2, and IP3. The avoidance of ET, CP2, IP2, and IP3 has significantly simplified the design of underlying transformation functions. In addition, we contributed explicit specifications of pre-conditions for CP2, IP2, and IP3 and provided formal verification (proofs) to the correctness of the COT algorithm with respect to CC1-CC6, CP2, IP2, and IP3.

Another distinctive feature of the COT algorithm is its independence of operation buffering structures. We proposed the notion of operation *VG* to bridge the high-level COT algorithm and underlying buffering schemes. We devised an operation buffering scheme that saves the MRCVs. Under this buffering scheme, the COT system is able to achieve the optimal linear time complexity in transforming normal operations (for do) in all cases and inverse operations (for undo) in common operation sequence patterns. The worst case time complexity for undo is exponential, but this is insignificant in practice since the number of independent operations is small in real-time collaborative editing sessions.

The future of OT lies clearly in its applications. Real-world applications give impetus to OT research, which in turn enables novel collaborative applications. Collaborative text editors had inspired the invention of the first OT algorithm and served as vehicles for continuous OT research in the past decade. The COT work was built on prior OT research and was motivated by the need to support advanced collaborative applications such as CoWord and CoPowerPoint.[7] We have implemented COT in a generic collaboration engine, which is being used to support collaborative office productivity tools, digital media design, computer-aided design, and software engineering tools. These and other emerging applications provide exciting opportunities and fresh stimuli to our ongoing and future research.

Acknowledgments

This paper is a substantially extended version of the paper published in the Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW) [1].

References

[1] D. Sun and C. Sun, “Operation Context and Context-Based Operational Transformation,” *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW ’06),* pp. 279-288, Nov. 2006.

[2] C.A. Ellis and S.J. Gibbs, “Concurrency Control in Groupware Systems,” *Proc. ACM SIGMOD ’89,* pp. 399-407, May 1989.

[3] A. Prakash and M. Knister, “A Framework for Undoing Actions in Collaborative Systems,” *ACM Trans. Computer-Human Interaction,* vol. 4, no. 1, pp. 295-330, Dec. 1994.

[4] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser, “An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors,” *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW ’96),* pp. 288-297, Nov. 1996.

[5] M. Ressel and R. Gunzenhäuser, “Reducing the Problems of Group Undo,” *Proc. ACM Conf. Supporting Group Work (GROUP ’99),* pp. 131-139, Nov. 1999.

[6] C. Sun, “Undo as Concurrent Inverse in Group Editors,” *ACM Trans. Computer-Human Interaction,* vol. 9, no. 4, pp. 309-361, Dec. 2002.

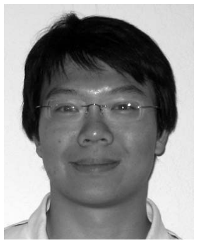
[7] H. Shen and C. Sun, “A Flexible Notification Framework for Collaborative Systems,” *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW ’02),* pp. 77-86, Nov. 2002.

[8] C. Sun, “Optional and Responsive Fine-Grain Locking in Internet-Based Collaborative Systems,” *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 9, pp. 994-1008, Sept. 2002.

[9] A. Davis, C. Sun, and J. Lu, “Generalizing Operational Transformation to the Standard General Markup Language,” *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW ’02),* pp. 58-67, Nov. 2002.

7. CoWord and CoPowerPoint Demo: http://cooffice.ntu.edu.sg.

[10] C. Ignat and M. Norrie, "Customizable Collaborative Editor Relying on treeOPT Algorithm," *Proc. Eighth European Conf. Computer-Supported Cooperative Work (ECSCW '03),* pp. 315-324, Sept. 2003.
[11] C. Sun, Q. Xia, D. Sun, D. Chen, H. Shen, and W. Cai, "Transparent Adaptation of Single-User Applications for Multi-User Real-Time Collaboration," *ACM Trans. Computer-Human Interaction,* vol. 13, no. 4, pp. 531-582, Dec. 2006.
[12] S. Xia, D. Sun, C. Sun, D. Chen, and H. Shen, "Leveraging Single-User Applications for Multi-User Collaboration: The CoWord Approach," *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW '04),* pp. 162-171, Nov. 2004.
[13] D. Sun, S. Xia, C. Sun, and D. Chen, "Operational Transformation for Collaborative Word Processing," *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW '04),* pp. 437-446, Nov. 2004.
[14] Agustina, F. Liu, S. Xia, H. Shen, and C. Sun, "CoMaya: Incorporating Advanced Collaboration Capabilities into 3D Digital Media Design Tools," *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW '08),* Nov. 2008.
[15] J. Begole, M. Rosson, and C. Shaffer, "Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application-Sharing Systems," *ACM Trans. Computer-Human Interaction,* vol. 6, no. 2, pp. 95-132, 1999.
[16] D. Li and R. Li, "Transparent Sharing and Interoperation of Heterogeneous Single-User Applications," *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW '02),* pp. 246-255, Nov. 2002.
[17] R. Guerraoui and C. Hari, "On the Consistency Problem in Mobile Distributed Computing," *Proc. Second ACM Int'l Workshop Principles of Mobile Computing (POMC '02),* pp. 51-57, Oct. 2002.
[18] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM,* vol. 21, no. 7, pp. 558-565, 1978.
[19] C. Sun, C.A. Ellis, "Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements," *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW '98),* pp. 59-68, Nov. 1998.
[20] 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 Trans. Computer-Human Interaction,* vol. 5, no. 1, pp. 63-108, Mar. 1998.
[21] M. Raynal and M. Singhal, "Logical Time: Capturing Causality in Distributed Systems," *IEEE Computer Magazine,* vol. 29, no. 2, pp. 49-56, Feb. 1996.
[22] C. Sun, "Undo Any Operation at Any Time in Group Editors," *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW '00),* pp. 191-200, Dec. 2000.
[23] C. Sun and W. Cai, "Capturing Causality by Compressed Vector Clock in Real-Time Group Editors," *Proc. 16th Int'l Parallel and Distributed Processing Symp. (IPDPS '02),* pp. 59-66, Apr. 2002.
[24] A. Imine, P. Molli, G. Oster, and M. Rusinowitch, "Proving Correctness of Transformation Functions in Real-Time Groupware," *Proc. Eighth European Conf. Computer-Supported Cooperative Work (ECSCW '03),* Sept. 2003.
[25] D. Li and R. Li, "Preserving Operation Effects Relation in Group Editors," *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW '04),* pp. 457-466, Nov. 2004.
[26] C. Sun and P. Maheshwari, "An Efficient Distributed Single-Phase Protocol for Total and Causal Ordering of Group Operations," *Proc. Third IEEE Int'l Conf. High-Performance Computing (HiPC '96),* pp. 295-300, Dec. 1996.
[27] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms,* second ed. MIT Press, p. 55, 2001.
[28] G. Cormack, "A Calculus for Concurrent Update," Report CS-95-06, Dept. of Computer Science, Univ. of Waterloo, Canada, 1995.
[29] D. Nichols, M.D.P. Curtis, and J. Lamping, "High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System," *Proc. Eighth Ann. ACM Symp. User Interface Software and Technology (UIST '95),* pp. 111-120, Nov. 1995.
[30] N. Vidot, M. Cart, J. Ferrié, and M. Suleiman, "Copies Convergence in a Distributed Real-Time Collaborative Environment," *Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW '00),* pp. 171-180, Dec. 2000.
[31] R. Li, D. Li, and C. Sun, "A Time Interval Based Consistency Control Algorithm for Interactive Groupware Applications," *Proc. 10th Int'l Conf. Parallel and Distributed Systems (ICPADS '04),* pp. 429-436, July 2004.
[32] N. Gu, J. Yang, and Q. Zhang, "Consistency Maintenance Based on the Mark & Retrace Technique in Groupware Systems," *Proc. ACM SIGGROUP Conf. Supporting Group Work (GROUP '05),* pp. 264-273, Nov. 2005.
[33] G.D. Abowd and A.J. Dix, "Giving Undo Attention," *Interacting with Computers,* vol. 4, no. 3, pp. 317-342, 1992.
[34] A. Dix, R. Mancini, and S. Levialdi, "The Cube—Extending Systems for Undo," *Proc. Eurographics Workshop the Design Specification and Verification of Interactive Systems (DSV-IS '97),* pp. 473-495, 1997.
[35] T. Berlage, "A Selective Undo Mechanism for Graphical User Interfaces Based on Command Objects," *ACM Trans. Computer-Human Interaction,* vol. 1, no. 3, pp. 269-294, Sept. 1994.
[36] R. Choudhary and P. Dewan, "A General Multi-User Undo/Redo Model," *Proc. Fourth European Conf. Computer-Supported Cooperative Work (ECSCW '95),* pp. 231-246, Oct. 1995.
[37] R. Gordon, G. Leeman, and G. Lewis, "Concepts and Implications of Undo for Interactive Recovery," *Proc. ACM Ann. Conf.,* pp. 150-157, 1985.
[38] W.K. Edwards, T. Igarashi, A. LaMarca, and E.D. Mynatt, "A Temporal Model for Multi-Level Undo and Redo," *Proc. 13th Ann. ACM Symp. User Interface Software and Technology (UIST '00),* pp. 31-40, 2000.
[39] D. Li and R. Li, "A Performance Study of Group Editing Algorithms," *Proc. 12th Int'l Conf. Parallel and Distributed Systems (ICPADS '06),* pp. 300-307, July 2006.



David Sun is a PhD student in the Division of Computer Science, Department of Electrical Engineering and Computer Science, University of California, Berkeley. His research interests include computer-supported cooperative work, distributed computing, ubiquitous computing, and human-computer interaction.

Chengzheng Sun received the PhD degree in computer engineering from the National University of Defense Technology, China, 1987 and the PhD degree in computer science from the University of Amsterdam, The Netherlands, 1992. He is a professor in the School of Computer Engineering, Nanyang Technological University, Singapore. His current research lies in the intersections of computer-supported cooperative work, distributed systems, human-computer interaction, and software engineering. He is a member of the IEEE.