Exhaustive Search of Puzzles in Operational Transformation

Chengzheng Sun, Yi Xu, and Agustina
School of Computer Engineering, Nanyang Technological University, Singapore

ABSTRACT
Operational Transformation (OT) is a collaboration-enabling technology and increasingly adopted in a wide range of real-world applications. One long-lasting issue in OT research is detecting and resolving puzzles – subtle and characteristic collaborative editing scenarios in which an OT system may fail. After many years of extensive search and research, a variety of intricate puzzles have been detected and resolved. However, it remains open whether all puzzles, under certain well-defined conditions, have been discovered. To address this issue, we set out to devise a system of verification frameworks and a software tool, that are independent of specific OT algorithms and able to exhaustively cover all possible transformation cases in which puzzles (if any) will manifest themselves. With the support of these tools, we verified OT correctness and concluded: *all puzzles, under basic data and operation models and established transformation properties, have been discovered and resolved.* Our discoveries help resolve a number of long-standing mysteries surrounding OT correctness and contribute to the advancement of OT fundamental knowledge and technology.

Author Keywords
Operational Transformation; transformation property; puzzle detection and resolution; real-time collaborative editing.

ACM Classification Keywords
H.5.3 [**Information Systems**]: Group and Organization Interfaces – Synchronous Interaction, Theory and Model.

INTRODUCTION
Operational Transformation (OT) is a collaboration-enabling technology invented by CSCW researchers and has been a technical research subject in CSCW in the past two decades [1-9,11-29]. Due to its unique combination of non-blocking, fine-grained concurrency, and unconstrained interaction properties, OT is particularly suitable for supporting real-time collaboration over the Internet, and has been increasingly adopted in industrial applications, e.g. Google Wave/Doc[1], IBM OpenCoWeb[2], and Codoxware[3].

One long-lasting issue in OT research is detecting and resolving puzzles – subtle and characteristic collaboration scenarios in which an OT system may fail to produce correct results. After many years of extensive search and research, a variety of intricate puzzles and their variations have been discovered and resolved [12,14-16,20-22,24,28]. For over one decade, there has been no new type of puzzle detected despite of continuous efforts by some dedicated researchers.

However, the mere absence of new puzzle discovery is not a proof that no more OT puzzle exists. It remains open whether all puzzles have indeed been detected. Resolving this open issue is important. For researchers, who design and verify OT algorithms, it is important for them to know what correctness issues have been solved or open, so that they can avoid wasting time on issues that had long been solved and devote efforts to open challenges. For practitioners, who adopt and implement OT algorithms to support real-world applications, it is important for them to know which OT algorithms are correct under what conditions, so that they can avoid wasting time in reinventing the wheel or choosing OT techniques that mismatch their target applications. Correctness is particularly important for mission-critical applications that require verification of core algorithms [17].

To solve this open issue, we set out to devise a system of verification frameworks that are independent of specific OT algorithms and able to efficiently and exhaustively cover all possible transformation cases in which puzzles (if any) will manifest themselves. We also developed a software tool to automate the verification process. In this paper, we present our verification frameworks and main research discoveries.

The rest of the paper is organized as follows. First, we briefly introduce prerequisite OT knowledge related to this work. Next, we review prior work on experimental puzzle detection and theoretic verification. Then, we formalize the basic data and operation models, specify combined-effects for concurrent operations, and define a general transformation matrix for verification. Based on these formalizations, we describe verification frameworks and discuss verification discoveries. Finally, we summarize major contributions and suggest future directions.

PREREQUISITE KNOWLEDGE
OT for Real-time Collaborative Editing Systems
Real-time collaborative editing systems allow geographically dispersed people to edit shared documents and see each other's updates instantly over the Internet. Most real-time collaborative editors have adopted the strategy of replicating the shared documents and the co-editing application among multiple collaborating sites. Replication has advantages of masking network latency and helping achieve fast local response, but also has a major problem: consistency maintenance of multiple replicas in the face of concurrent editing, which OT was originally invented to solve [6].

[1] http://docs.google.com
[2] https://github.com/opencoweb/coweb#readme
[3] http://www.codoxware.com

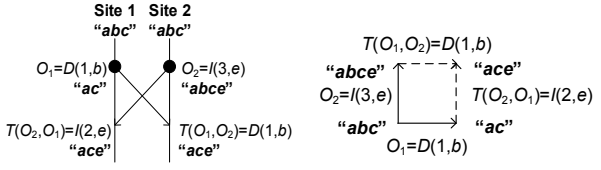
CSCW'14, February 15–19, 2014, Baltimore, MD, USA.

http://dx.doi.org/10.1145/2531602.2531630

In over two decades, OT has evolved from supporting collaborative editing of plain text documents [6, 16, 20, 21], to rich-text documents [23], two-dimensional (2D) spreadsheets [26], and three-dimensional (3D) digital media [1, 2]. A pair of *basic* data and operation models, originated and generalized from text editing, have served as the foundation of most existing OT systems: (1) the basic data model is a sequence of data objects (of any type and size), which can be accessed using positional references; and (2) the basic operation model includes two operations: *insert* and *delete* of a single object, which can be directly manipulated by transformation functions. Past research has invented a core body of OT techniques for the basic data and operation models; and most OT verification work, including this paper, has focused on them due to their foundation role in supporting real-world applications. Core OT techniques, however, provide the foundation but not the complete solution for real-world applications: they can be directly used for applications that are restricted to insert and delete operations on a sequence of objects, but require significant extensions for supporting office productivity tools and 2D and 3D computer-aided design systems, which are frontiers of current OT research and application. Prior research has invented promising techniques, such as *Transparent Adaptation* techniques used in CoWord [23] and CoMaya [1, 2], to bridge the gap between core OT techniques and complex models in real-world applications [1, 2, 23, 26, 27].

Basic Ideas for Consistency Maintenance and Undo

The basic OT idea for consistency maintenance can be illustrated in a *Space-Time-Diagram* (STD) in Figure 1(a). The initial document contains a string *"abc"*, replicated at two collaborating sites; two operations: $O_1 = D(1,b)$ (to delete *"b"* at position *1*), and $O_2 = I(3,e)$ (to insert *"e"* at position *3*) are *concurrently* generated. At site *1*, O_1 is first executed to get the document *"ac"*. Then, O_2 arrives and is transformed against O_1, i.e. $T(O_2,O_1) = I(2,e)$, whose position is decremented to compensate the left-shifting effect by O_1. Executing *I(2,e)* on *"ac"* inserts *"e"* at the end of document to get a new state *"ace"* (note: if O_2 were executed in its original form, it would cause an error as its insert position is beyond the current document length). At site *2*, O_2 is first executed to get the document *"abce"*. Then O_1 arrives and is transformed against O_2, i.e. $T(O_1,O_2) = D(1,b)$, which makes no change because O_2 has no impact on O_1. Executing $D(1,b)$ on *"abce"* deletes *"b"* at position *1* and get the document state *"ace"*, which is identical to the state at site *1* and also achieves the original effects of both O_1 and O_2. In summary, the basic idea is to transform an operation according to the effect of concurrent operations so that the transformed operation can achieve the correct effect and maintain document consistency.



(a) Space-Time Diagram (b) Context-based OT Space

Figure 1. Basic idea of OT for consistency maintenance.

The same scenario can also be illustrated using a different diagram, called *Context-based Operational Transformation Space* (COTS), in Figure 1(b), where a vertex represents (and is labeled by) a document state, and an arrow represents an operation; the starting vertex of an arrow represents the definition state of the operation, and the ending vertex represents the resulting state of the operation. A correct COTS possesses the following property[4]: there is only one label for each vertex or arrow.

Another major OT functionality is supporting user-initiated *undo* – a mechanism for error recovery and alternative exploration, which is commonly available in single-user interactive applications but challenging to support in collaborative systems [14-16, 22, 24]. A user may issue *Undo*(*O*) to reverse the effects of an executed operation *O*. In OT undo solutions, undoing *O* is achieved by executing an *inverse* operation *!O* with the following effect: $S \circ O \circ !O = S$, i.e. applying *O* and *!O* on the document state *S* in sequence would restore *S*. *!O* can be executed as-is if it is executed immediately after executing *O* on *S*; otherwise, *!O* must be transformed and treated in the same way as concurrent operations. In this paper, we verify transformation properties related to both consistency maintenance and undo.

Control Algorithms and Transformation Functions

An OT system contains two key components: control algorithms and transformation functions [6,12,15,20,21,23,24, 26,27], as shown in Figure 2. Control algorithms are responsible for determining which operations should be transformed with which others and in what orders, and they are *generic* in the sense they work on generic relations (context [24] and/or causality [10]) among operations. On the other hand, transformation functions perform actual transformations between two operations, according to their types, target object relations and other parameters, which may be *application-specific*. The number and types of transformation functions are related to the data and operation models that an OT system is to support. Residing between control algorithms and transformation functions is a collection of transformation properties and conditions that define correctness requirements of an OT system and also divide correctness responsibilities between these two components.

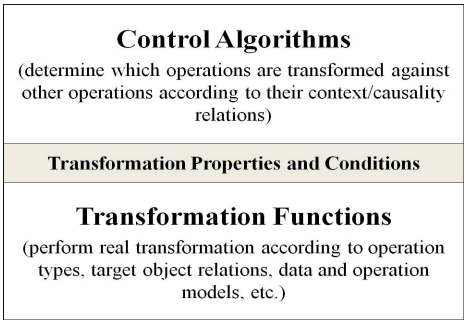


Figure 2. OT system components.

[4] The COTS is an extension of the Interaction Graph in [16] to represent all operations in an OT system supporting both do and undo in an integrated context space [24].

The correctness of an OT system depends on individual correctness of control algorithms and transformation functions and their correct integration. This work focuses on verification of transformation functions. For verification of control algorithms, the reader is referred to [15,21,22,24,29].

Transformation Properties

Past OT research has established a comprehensive set of transformation properties as OT algorithmic correctness criteria [6,14,16,21,22,24,27]. Transformation properties and OT puzzles are intimately related: violation of any transformation property would result in incorrect transformation results, which may lead to inconsistent documents and manifest themselves as puzzles. In other words, transformation property violation is the root of puzzles, whereas puzzles are the phenomena of transformation property violation. In this work, our approach to searching OT puzzles is by verifying transformation properties. We focus on the following properties:

1. **Convergence Property 1 (CP1):** Given O_1 and O_2 defined on the state S, if $O_1^{O2} = T(O_1,O_2)$, and $O_2^{O1} = T(O_2,O_1)$, the following **CP1-equation** must be held:
$$S \circ O_1 \circ O_2^{O1} = S \circ O_2 \circ O_1^{O2,}$$
which means that applying O_1 and O_2^{O1} in sequence on S has the same effect as applying O_2 and O_1^{O2} on S.
2. **Convergence Property 2 (CP2):** Given O_1, O_2 and O_3 defined on the same state, if $O_3^{O2} = T(O_3,O_2)$ and $O_2^{O3} = T(O_2,O_3)$, the following **CP2-equation** must be held:
$$T(T(O_1,O_2),O_3^{O2}) = T(T(O_1,O_3),O_2^{O3})$$
which means transforming O_1 against O_2 and O_3^{O2} equals to transforming O_1 against O_3 and O_2^{O3}. Alternatively, the CP2-equation can be expressed as:
$$T(O_1^{O2},O_3^{O2}) = T(O_1^{O3},O_2^{O3})$$
where $O_1^{O2} = T(O_1,O_2)$, and $O_1^{O3} = T(O_1,O_3)$.
3. **Inverse Property 1 (IP1):** Let $!O$ be the inverse of operation O and S be a document state, then the following **IP1-equation** must be held:
$$S \circ O \circ !O = S$$
which means applying O and $!O$ on S has no effect.
4. **Inverse Property 2 (IP2):** Given O_1 and O_2 defined on the same state, and $!O_2$ as the inverse of O_2, then the following **IP2-equation** must be held:
$$T(T(O_1,O_2),!O_2) = O_1$$
which means that transforming O_1 against O_2 and $!O_2$ in sequence has no effect on O_1.
5. **Inverse Property 3 (IP3):** Given O_1 and O_2 defined on the same state, if $O_1^{O2} = T(O_1,O_2)$, $O_2^{O1} = T(O_2,O_1)$ and $(!O_2)^{O1} = T(!O_2,O_1^{O2})$, then the following **IP3-equation** must be held:
$$(!O2)^{O1} = !(O2^{O1})$$
which means that $(!O2)^{O1}$, the outcome of transforming the inverse $!O_2$ against O_1^{O2}, equals to $!(O2^{O1})$, the inverse of the outcome of transforming O_2 against O_1.

The reason for focusing on these five properties is that CP1 and CP2 are the transformation properties governing convergence, and IP1, IP2, and IP3 are governing transformations related to undo. Formally, an OT puzzle can be defined as a case of violating a transformation property. By exhaustively searching verification cases with respect to these properties, we will be able to detect all possible puzzles related to them and answer the question whether all consistency and undo puzzles have been found. Past research had detected various convergence and undo puzzles under different OT systems [4,9,11,14,15,16,18,20,21,22, 24]. We predict all those known and unknown (if any) puzzles would manifest themselves in our verification frameworks, provided our theory that a puzzle is violation of a property is correct and our verification cases are exhaustive.

CP1 and CP2 were found to be necessary and sufficient conditions to achieve convergence in OT systems that allow operations to be transformed in any orders, such as the adOPTed system [15]. However, they are not unconditionally required. For example, CP1 is not necessary in an OT system, such as GOT [21], that never transforms two operations in different orders; CP2 is not necessary for OT systems that impose special transformation orders among operations [29], including Jupiter [12], Google Docs, NICE [19], SOCT4 [28], and COT [24]. CP1 and CP2 are not sufficient either: transformation functions must meet, in addition to convergence, the *intention-preservation* requirement [20,21]. In this paper, we will formally specify and verify this requirement as *combined-effects* of concurrent operations under the basic data and operation models.

We should point out that T in all transformation properties represents *Inclusion Transformation* (*IT*) functions, which are used in all existing OT systems [1,2,12,13,15,16,19,21, 22,24,26,28]. However, some OT systems also use *Exclusion Transformation* (*ET*) functions [21,22,28], which are reverse functions to *IT*. If both *IT* and *ET* are used in an OT system, they are required to preserve the *Reversibility Property* (RP) [21]: if $O_1^{O2} = IT(O_1,O_2)$, then $ET(O_1^{O2},O_2) = O_1$. Alternatively, RP can be expressed as:
$$ET(IT(O_1,O_2),O_2) = O_1.$$
By substituting T with *IT*, IP2 can be expressed as:
$$IT(IT(O_1,O_2),!O_2) = O_1.$$
By comparing RP and the above IP2, we found RP can be converted into IP2 by replacing *ET* with *IT* and O_2 with $!O_2$. This insight suggests an IP2-preserving *IT* function can be used as an RP-preserving *ET* function by replacing O_2 with $!O_2$. Furthermore, past research has also found an OT system (for both consistency maintenance and undo) can be based on *IT* functions only, without using any *ET* function [15,16,24]. For these reasons, we do not consider *ET* and RP, but focus on verification of *IT* with respect to IP2 (and other transformation properties) in this paper.

Finally, we highlight that all transformation properties share one common condition: *if two operations are transformed by function T, they must be defined on the same state*. This condition is guaranteed by suitable OT control algorithms [12,16,19,20,21,22,24,28]. An OT-based collaborative editing system can support multiple users to generate any number of operations with arbitrary causal relations. Possible scenarios and relations among operations are arbitrary and infinitive. Under correct control algorithms, however, every pair of operations being transformed by T are guaranteed to

be defined on the same state. Therefore, there is no need to model causal relationships among operations for verifying transformation properties and functions, and this insight leads to significant simplifications in our verification frameworks, compared to prior verification work [4].

PRIOR WORK

Ad Hoc Puzzle Detection

An experimental approach, called *puzzle-detection-resolution,* has been commonly adopted in prior research for searching OT puzzles [14-16,21,22,24,27]. This approach suits well with the complex nature of puzzle issues, and has proven its effectiveness in detecting and solving all known puzzles, including the *dOPT* puzzle related to OT control algorithms [6,12,16,20,21], the *False-Tie* (*FT*) puzzle related to violating document convergence [20], and undo puzzles that violate *undo effects* and may cause inconsistency [14,15,22,24]. Despite its effectiveness and success, the ad hoc and informal nature of this approach makes it unsuitable for systematic and formal verification: ad hoc search can detect the existence of puzzles, but cannot prove the absence of puzzles. Another problem is that it requires researchers to have intimate knowledge of specific OT algorithms to be able to detect and resolve puzzles in those algorithms.

In this work, we generalize this ad hoc puzzle detection approach into a *systematic and exhaustive puzzle search* approach, which is grounded on established transformation properties, without using the knowledge of specific OT algorithms. This generalized approach has enabled us to not only detect all puzzles already discovered by the traditional approach, but also prove the absence of any new puzzle.

Theoretic Verification Work

Theoretic verification plays an important role in OT correctness study and may be conducted manually or by using computer-aided tools. Prior work had used automatic *theorem-provers* [9] and *model checkers* [4] to assist the verification of convergence properties CP1 and CP2 [4, 9]. Using generic verification tools necessitates the mapping between OT verification problems and theorem-proving or model-checking problems in mathematical logic formalisms. People who are acquainted with such formalisms may not have adequate knowledge of OT, and vice versa. This knowledge gap may cause modeling and specification errors, which may lead to false verification results, as reported in [4, 9, 11]. Another challenge is result interpretation, e.g. whether an unproven logic formula reveals an OT puzzle or the incompleteness of the proving system, and how to establish connections between formal verification results and OT puzzles (if any), which is crucial for any verification approach to be useful for studying OT correctness.

One often-cited reason for using computer-assisted verification tools is that transformation property verification is an *exponential explosion* problem that is too complex and "even impossible" for manual checking [4,9]. A state explosion problem was reported in a model checker, based on the following modeling [4]:

1. A collaborating editing session is modeled as an arbitrary number of users; each user may generate an arbitrary number of operations at any time; operations may have arbitrary concurrency/causality relationships.
2. A document is modeled as a string of characters with a length $L = 2 \times n$, where n is the total number of operations in a simulated co-editing session; and each character in the string may take one of 2 values: 0 or 1.
3. An operation may take one of 2 types: *insert* or *delete* (character-wise); each operation may take one of L positions; and an *insert* has an additional character parameter, which may take one of 2 values: 0 or 1.

The following formula gives the total number of states the model checker has to check to assess whether convergence is preserved in a simulated session with n operations [4]:

$$((2 + 1) \times L)^n = (6 \times n)^n .$$

The model checker has to check $5832 = (6 \times 3)^3$ states for a session with 3 operations, and $331776 = (6 \times 4)^4$ states for 4 operations, which had exceeded the capability of the used model-checking system [4]. As the number of operations n can be arbitrarily large in this modeling, it is impossible to exhaustively check all possible states. On the other hand, limiting $L = 2 \times n$ and character values to be 0 or 1 makes this modeling highly restrictive as well.

The ability to cover all verification cases is essential to answer the question whether all puzzles have been found. As will become clear in this paper, our verification frameworks can efficiently model the verification problem with less than 100 verification cases; and exhaustively cover all verification cases without suffering from exponential case explosion. Moreover, our exhaustive coverage is achieved without imposing restrictions on the data and operation modeling: a document string may have an *arbitrary* length (unrelated to the number of operations); each character may take an *arbitrary* value from any valid set of characters (unlimited to two values: 0 or 1); and an operation may also take an *arbitrary* position in a document. Furthermore, our verification frameworks are based on original OT data and operation formalisms and transformation properties (e.g. transformation property equations are *directly* used in our verification frameworks), which help achieve correct specification and result interpretation. Last but not least, our verification covers not only CP1 and CP2, but also combined-effects (to be defined later) and undo-related properties (IP1/IP2/IP3), which were never addressed in prior work.

FORMALIZATION OF DATA AND OPERATION MODELS

We have chosen character strings and character-wise operations as the data and operation formalisms as they are precise enough for the purpose of verification and directly correspond to the basic OT data and operation models. Conclusions drawn from this formalization are generally applicable to any object sequences.

Data Model Specification

A document state S is represented as a string of characters:

$$S = "c_0\, c_1\, c_2 \ldots c_{p-1}\, c_p\, c_{p+1} \ldots c_{n-1}"$$

where c_p, $0 \le p \le n-1$, represents a character. The following notations are used for convenience of expression:

1. $|S|$ is the length of the string S.

2. $S[p]=c_p$, $0 \le p \le |S|-1$, is the character at position p in S.
3. $S[i, j]="c_i c_{i+1} \ldots c_{j-1} c_j"$, $0 \le i \le j \le |S|-1$, represents a sub-string of characters with positions from i to j, inclusively, in S. For notation convenience in state expressions, $S[i, j]$ represents an empty string in case that $i>j$.
4. $S[i, j] + S[m, n]="c_i c_{i+1} \ldots c_{j-1} c_j c_m c_{m+1} \ldots c_{n-1} c_n"$, represents the concatenation of two sub-strings.

Operation Model Specification

The operation model consists of two operations:

1. $I(p, c)$: to insert a single character c at position p;
2. $D(p, c)$: to delete a single character at position p. The parameter c is used to record the deleted character for the purpose of supporting undo.

Insert and delete operations are inverses of each other, i.e.

1. $!I(p, c) = D(p, c)$;
2. $!D(p, c) = I(p, c)$.

From the basic operations and their inverses, we have:

1. $S \circ I(p, c) \circ !I(p, c) = S \circ I(p, c) \circ D(p, c) = S$;
2. $S \circ D(p, c) \circ !D(p, c) = S \circ D(p, c) \circ I(p, c) = S$.

which means that *IP1*-equation: $S \circ O \circ !O = S$, is preserved by definition for any O in the basic operation model.

Specification 1: Effect of a Single Insert Operation

Given an insert operation $I(p, c)$ defined on a state S, where $0 \le p \le |S|$. The effect of $I(p, c)$ on S is to convert S into the following new state S':

$$S' = S \circ I(p, c) = S[0, p-1] + "c" + S[p, |S|-1],$$

with the following $S' \leftrightarrow S$ mapping relationship:

1. $S'[0, p-1] = S[0, p-1]$, when $p \ne 0$;
2. $S'[p] = c$;
3. $S'[p+1, |S'|-1] = S[p, |S|-1]$, when $p \ne |S|$.

Informally, the effect of an insert operation $I(p, c)$ is to insert a new character c at position p in S, which will *right-shift* characters *at* and *on the right* of $S[p]$ by one position, but keep characters *on the left* of $S[p]$ unchanged.

Specification 2: Effect of a Single Delete Operation

Given a delete operation $D(p, c)$ defined on the state S, where $0 \le p \le |S|-1$. The effect of $D(p, c)$ on S is to convert S into the following new state S':

$$S' = S \circ D(p, c) = S[0, p-1] + S[p+1, |S|-1],$$

with the following $S' \leftrightarrow S$ mapping relationship:

1. $S'[0, p-1] = S[0, p-1]$, when $p \ne 0$;
2. $S'[p, |S'|-1] = S[p+1, |S|-1]$, when $p \ne |S|-1$.

Informally, the effect of a delete operation $D(p, c)$ is to delete a character at position p in S, which will *left-shift* characters *on the right* of $S[p]$ by one position, but keep characters *on the left* of $S[p]$ unchanged.

Exhaustive Coverage of Arbitrary Positions and States

Under the basic data and operation models, a document state S may have an arbitrary (and theoretically unlimited) length $|S|$, and each position in S may contain an arbitrary character from the ASCII character set; and an operation O may have an arbitrary position p in the range $0 \le p \le |S|$, and may affect the positions of an arbitrary number of characters in S, as defined in Specifications 1 and 2. These arbitrary parameters and their combinations present major challenges to achieving exhaustive coverage of all possibilities in OT verification, which caused exponential explosion problems as discussed in the "Prior Work" section.

The solution to this challenge lies in the following key insights in our data and operation modeling. First, $S' \leftrightarrow S$ mappings establish one-to-one positional correspondences between S' and S, and serves as a formalism of representing arbitrary lengths and characters in S' in terms of S (and vice versa), without knowing specific lengths and values of those characters in S. This formalism provides the foundation for checking the equivalence of two states, without knowing specific contents of these states.

Second, the effect of O on S can be partitioned into three *uniform effect ranges*:

1. $S[0,p-1]$, the *left* effect range of p;
2. $S[p]$, the effect range at p; and
3. $S[p+1,|S|-1]$, the *right* effect range of p.

The effect of O in each range is *uniform* in the sense that all positions in the same range are affected by O in the same way, though the effect in different ranges may be different. This effect uniformity provides the foundation for reducing arbitrary possibilities to a fixed number of possibilities.

The significance of these insights will become clear when we define combined-effects of concurrent operations and derive exhaustive verification cases.

UNION-EFFECTS AND TRANSFORMATION MATRIX

Combined-Effects of Concurrent Operations

In the absence of concurrency, the single-operation effect specification is adequate for a user to understand the behavior of an editor (e.g. a single-user editor). In the presence of concurrency in collaborative editing, however, we need to define *combined-effects* for concurrent operations, independent of their execution orders.

To specify combined-effects, we need first determine all possible cases. Given two operations defined on the same state, they may have three possible position relationships: $p_1 < p_2$, $p_1 = p_2$, $p_1 > p_2$, which directly correspond to three uniform effect ranges of each operation. Under each specific position relationship, two operations may affect each other in a uniform way, and their combined-effect can be uniformly specified in a single effect expression. Based on this observation, an exhaustive set of combined-effect cases for two operations are derived as follows. First, there are three different operation type combinations: I-I, D-D, and I-D (D-I is the same type combination). Second, for each type combination, there are three position relationships based on the uniform effect ranges of each operation. In total, there are $3 \times 3 = 9$ combined-effect cases for two operations.

The second issue is to specify concrete combined-effects for all possible cases. Different collaborative applications may choose different combined-effects for meeting specific needs. Most OT systems have (implicitly) adopted a special combined-effect, named *Union Effect* (*UE*) in this paper.

We specify union effects under all *9* possible cases in the column $UE(O_1,O_2)$ in Table 1. The $UE(O_1,O_2)$ is a formal specification of the general *intention-preservation* requirement under the basic data and operation models [21,20]. The $UE(O_1,O_2)$ has the merits of retaining the *original-effect* of individual operations, and preserving *all-operation-effects* under all circumstances. Such union effects have been commonly supported by OT systems, but this work is the first to give it a formal specification.

Two cases in $UE(O_1,O_2)$ deserve special attention. One is Case 2 (*insert-insert-tie*): $I(p_1,c_1)$ and $I(p_2,c_2)$ with $p_1 = p_2$. In this case, two possible combined-effects: *"c_1c_2"* and *"c_2c_1"* are specified as valid union effects. An OT system is free to support any of them, so this combined-effect introduces *non-determinism* to $UE(O_1,O_2)$. Such a union effect is not the only possibility for the *insert-insert-tie* case. Other possibilities are: only one result is accepted as valid; none of them is accepted; or two characters are merged into one if they are the same [6]. Nearly all OT systems have chosen the same effect as in $UE(O_1,O_2)$ due to its ability of preserving all operation effects and intuitiveness to end-users.

O_1 O_2	PR	$UE(O_1,O_2)$	#
$I(p_1,c_1)$ $I(p_2,c_2)$	$p_1<p_2$	$S[0,p_1-1]+"c_1"+S[p_1,p_2-1]+"c_2"+S[p_2,\|S\|-1]$	1
	$p_1=p_2$	$S[0,p_1-1]+"c_1c_2"+S[p_1,\|S\|-1]$ $S[0,p_1-1]+"c_2c_1"+S[p_1,\|S\|-1]$	2
	$p_1>p_2$	$S[0,p_2-1]+"c_2"+S[p_2,p_1-1]+"c_1"+S[p_1,\|S\|-1]$	3
$I(p_1,c_1)$ $D(p_2,c_2)$	$p_1<p_2$	$S[0,p_1-1]+"c_1"+S[p_1,p_2-1]+S[p_2+1,\|S\|-1]$	4
	$p_1=p_2$	$S[0,p_1-1]+"c_1"+S[p_1+1,\|S\|-1]$	5
	$p_1>p_2$	$S[0,p_2-1]+S[p_2+1,p_1-1]+"c_1"+S[p_1,\|S\|-1]$	6
$D(p_1,c_1)$ $D(p_2,c_2)$	$p_1<p_2$	$S[0,p_1-1]+S[p_1+1,p_2-1]+S[p_2+1,\|S\|-1]$	7
	$p_1=p_2$	$S[0,p_1-1]+S[p_1+1,\|S\|-1]$	8
	$p_1>p_2$	$S[0,p_2-1]+S[p_2+1,p_1-1]+S[p_1+1,\|S\|-1]$	9

Table 1. Union effect specification. PR: Position Relationship.

The other special union effect is Case 8 (*delete-delete-tie*): $D(p_1,c_1)$ and $D(p_2,c_2)$ with $p_1 = p_2$ and $c_1 = c_2$. In this case, only one character c_1 is deleted, which is the same as a single delete operation effect. It is also possible to define other combined-effects for this case, e.g. none of the deletes has an effect (so the character remains). However, all OT systems have adopted the same union effect for the delete-delete-tie case, based on similar reasons as the insert-insert-tie case. This introduces *non-serialisability* to $UE(O_1,O_2)$ as no serialization scheme is able to achieve such a combined-effect. As will become clear later in this paper, *non-determinism* and *non-serialisablilty* properties in $UE(O_1,O_2)$ have intricate relationships with all puzzles related to transformation properties.

Integrating UE and CP1 Requirements

Given two operations O_1 and O_2 defined on the same state S, transforming O_1 against O_2 is to produce $O_1^{O2} = T(O_1,O_2)$ as follows: (1) assume O_2 has been executed; (2) assess the execution effect of O_2 on the state S; and (3) derive O_1^{O2} so that executing O_1^{O2} after O_2 on S would produce the union effect specified in $UE(O_1,O_2)$, i.e.

$$UE(O_1,O_2) = S \circ O_2 \circ O_1^{O2}.$$

In addition, transformation must satisfy the CP1-equation:

$$S \circ O_1 \circ O_2^{O1} = S \circ O_2 \circ O_1^{O2}.$$

For transformation to be meaningful (in terms of $UE(O_1,O_2)$) and consistent (by CP1), the following new property equation, denoted as *UE-CP1*, must be satisfied:

$$UE(O_1,O_2) = S \circ O_1 \circ O_2^{O1} = S \circ O_2 \circ O_1^{O2}.$$

Transformation Matrix as Verification Target

We use the Transformation Matrix (TM) in Table 2 to represent all results of transforming O_1 against O_2. The TM consists of four rows corresponding to four operation type permutations[5]: I-I, I-D, D-I and D-D. Each type permutation is divided into three sub-rows according to three position relationships. Entries in column $TM(O_1,O_2)$ represent results of transforming O_1 against O_2 for individual cases.

O_1 O_2	PR	$TM(O_1,O_2)$	#
$I(p_1,c_1)$ $I(p_2,c_2)$	$p_1<p_2$	$I(p_1,c_1)$	1
	$p_1=p_2$	$I(p_1,c_1)$ if priority(O_1)> priority(O_2) $I(p_1+1,c_1)$ if priority(O_1)<priority(O_2)	2
	$p_1>p_2$	$I(p_1+1,c_1)$	3
$I(p_1,c_1)$ $D(p_2,c_2)$	$p_1<p_2$	$I(p_1,c_1)$	4
	$p_1=p_2$	$I(p_1,c_1)$	5
	$p_1>p_2$	$I(p_1-1,c_1)$	6
$D(p_1,c_1)$ $I(p_2,c_2)$	$p_1<p_2$	$D(p_1,c_1)$	7
	$p_1=p_2$	$D(p_1+1,c_1)$	8
	$p_1>p_2$	$D(p_1+1,c_1)$	9
$D(p_1,c_1)$ $D(p_2,c_2)$	$p_1<p_2$	$D(p_1,c_1)$	10
	$p_1=p_2$	NULL	11
	$p_1>p_2$	$D(p_1-1,c_1)$	12

Table 2. Transformation Matrix $TM(O_1, O_2)$.

We have filled entries in $TM(O_1,O_2)$ with transformation results derived from $UE(O_1,O_2)$ and UE-CP1. In most cases, convergence is guaranteed as long as the union effect is achieved. In two cases, however, special measures have been taken to achieve UE-CP1. In the insert-insert-tie case, a priority-based tie-breaking rule, which ensures a total ordering among all operations, is used. In the delete-delete-tie case, the result is the special operation NULL[6].

From $TM(O_1,O_2)$, it is straightforward to design concrete transformation functions. One possible set of transformation functions capable of producing the same transformation results as $TM(O_1,O_2)$ is given in APPENDIX. Conversely, we can also derive $TM(O_1,O_2)$ from existing transformation functions. In our verification frameworks, $TM(O_1,O_2)$, rather than specific transformation functions, is used for verification, which allows us to verify both existing and new transformation functions whose transformation results are captured in $TM(O_1,O_2)$. This approach is in contrast to prior verification work, which all used specific existing transformation functions for verification [4,9].

VERIFICATION FRAMEWORKS

Based on the formalization of data and operations models, $UE(O_1,O_2)$, and $TM(O_1,O_2)$, we devised a system of frameworks to verify UE-CP1, CP2, IP2 and IP3 (IP1 is preserved by the operation model definition), respectively.

[5] Type permutations (in which orders matter) are different from type combinations (in which operation orders do not matter), e.g. D-I and I-D are different type permutations, but the same type combination.

[6] *NULL* has the following properties: $S \circ NULL = S$; $T(NULL, O) = NULL$; $T(O, NULL) = O$.

Each framework contains two key components: (1) an exhaustive set of verification cases; and (2) a verification process based on the corresponding property equation.

UE-CP1 Verification Framework

Exhaustive UE-CP1 verification case derivation. UE-CP1 involves two independent operations O_1 and O_2. The two operations may have three different operation type combinations: I-I, D-D, and I-D (D-I is the same type combination). For each combination, two operations have three possible position relationships. In total, there are $3 \times 3 = 9$ cases.

Specification 3: UE-CP1 Verification Process

Given all *UE-CP1* verification cases, the verification process works on each case one-by-one as follows:

1. Derive $S_{12} = S \circ O_1 \circ O_2^{O1}$:
   (1) Derive $S_1 = S \circ O_1$ and $S_1 \leftrightarrow S$ mappings, according to Specifications 1&2.
   (2) Derive $O_2^{O1} = T(O_2, O_1)$, according to types and position relationships of O_1 and O_2 by consulting the $TM(O_1, O_2)$.
   (3) Derive $S_{12} = S_1 \circ O_2^{O1}$, according to Specification 1&2.
   (4) Derive $S_{12} = S \circ O_1 \circ O_2^{O1}$, by $S_1 \leftrightarrow S$ mappings associated with O_1.
2. Derive $S_{21} = S \circ O_2 \circ O_1^{O2}$ by following similar steps.
3. Check the UE-CP1-equation: $S_{12} = S_{21} = UE(O_1, O_2)$. If the equation is satisfied, this is a UE-CP1-confirmation case; otherwise, it is a UE-CP1-violation case.

Steps in the UE-CP1 verification process are well defined by the UE-CP1-equation, Specification 1 and 2, $TM(O_1, O_2)$, and $UE(O_1, O_2)$. Step 1-(4)/2-(4) is a special step to convert S_1/S_2-based representations of S_{12}/S_{21} into S-based representations so that they can be literally compared for equivalence. The correctness of this conversion is ensured by the correctness and completeness of $S_1 \leftrightarrow S / S_2 \leftrightarrow S$ mappings. We illustrate one UE-CP1 case verification in Table 3, and provide all verification cases and results in APPENDIX.

UE-CP1 Verification Case: $O_1 = I(p_1, c_1)$, $O_2 = D(p_2, c_2)$, $p_1 < p_2$	
1	**Derive $S_{12} = S \circ O_1 \circ O_2^{O1}$**
(1)	$S_1 = S \circ O_1 = S[0,p_1\text{-}1] + \text{“}c_1\text{”} + S[p_1,\|S\|\text{-}1]$ // by Specification 1 $= S_1[0,p_1\text{-}1] + S_1[p_1] + S_1[p_1\text{+}1,\|S_1\|\text{-}1]$ // $S_1 \leftrightarrow S$
(2)	$O_2^{O1} = T(O_2, O_1) = D(p_2\text{+}1, c_2)$ // derived from row #9 in **TM(O_1,O_2)**
(3)	$S_{12} = S_1 \circ O_2^{O1} = S_1[0,p_2] + S_1[p_2\text{+}2,\|S_1\|\text{-}1]$ // by Specification 2
(4)	$S_{12} = S \circ O_1 \circ O_2^{O1} = S_1[0,p_2] + S_1[p_2\text{+}2,\|S_1\|\text{-}1]$ $= S_1[0,p_1\text{-}1] + S_1[p_1] + S_1[p_1\text{+}1,p_2] + S_1[p_2\text{+}2,\|S_1\|\text{-}1]$ // a middle step $= S[0,p_1\text{-}1] + \text{“}c_1\text{”} + S[p_1,p_2\text{-}1] + S[p_2\text{+}1,\|S\|\text{-}1]$ // by $S_1 \leftrightarrow S$
2	**Derive $S_{21} = S \circ O_2 \circ O_1^{O2}$**
(1)	$S_2 = S \circ O_2 = S[0,p_2\text{-}1] + S[p_2\text{+}1,\|S\|\text{-}1]$ // by Specification 2 $= S_2[0,p_2\text{-}1] + S_2[p_2,\|S_2\|\text{-}1]$ // $S_2 \leftrightarrow S$
(2)	$O_1^{O2} = T(O_1, O_2) = I(p_1, c_1)$ // derived from row #4 in **TM(O_1,O_2)**
(3)	$S_{21} = S_2 \circ O_1^{O2} = S_2[0,p1\text{-}1] + \text{“}c_1\text{”} + S_2[p_1,\|S_2\|\text{-}1]$ // by Specification 1
(4)	$S_{21} = S \circ O_2 \circ O_1^{O2} = S_2[0,p1\text{-}1] + \text{“}c_1\text{”} + S_2[p_1,\|S_2\|\text{-}1]$ // $= S_2[0,p_1\text{-}1] + \text{“}c_1\text{”} + S_2[p_1,p_2\text{-}1] + S_2[p_2,\|S_2\|\text{-}1]$ //a middle step $= S[0,p_1\text{-}1] + \text{“}c_1\text{”} + S[p_1,p_2\text{-}1] + S[p_2\text{+}1,\|S\|\text{-}1]$ // by $S_2 \leftrightarrow S$
3	**Check UE-CP1-Equation: $S_{12} = S_{21} = UE(O_1, O_2)$** by comparing the results in 1-(4) and 2-(4) and row #4 in **$UE(O_1, O_2)$**. The equation is satisfied, so this is a UE-CP1 confirmation case.

Table 3. An example case in UE-CP1 verification.

CP2 Verification Framework

Exhaustive CP2 verification case derivation. CP2 involves three independent operations O_1, O_2 and O_3. As each operation may independently take one of two different types – *insert* or *delete*, there exist $2^3 = 8$ different type permutations. Furthermore, three independent operations may have thirteen position relationships, according to a set of three operators: $\{<, =, >\}$:

1. Six cases where all three positions are different: (1) $p_1 < p_2 < p_3$, (2) $p_2 < p_1 < p_3$, (3) $p_2 < p_3 < p_1$, (4) $p_1 < p_3 < p_2$, (5) $p_3 < p_1 < p_2$, (6) $p_3 < p_2 < p_1$;
2. Six cases where two positions are the same but one position is different: (1) $p_1 = p_2 < p_3$, (2) $p_3 < p_1 = p_2$, (3) $p_1 = p_3 < p_2$, (4) $p_2 < p_1 = p_3$, (5) $p_2 = p_3 < p_1$, (6) $p_1 < p_2 = p_3$;
3. One case where all positions are the same: $p_1 = p_2 = p_3$.

In total, there exist $8 \times 13 = 104$ different combinations of operation type permutations and position relationships. Each of them could make a CP2 verification case by transforming O_1 against O_2 and O_3 in different orders. However, careful examination reveals that nearly half of those CP2-equations are actually redundant. After eliminating redundant cases, there are a total of *58* distinctive CP2 verification cases to be checked.

Specification 4: CP2 Verification Process

Given all CP2 verification cases, the verification process works on each of these cases as follows:

1. Derive $O_1^{O2,O3} = T(O_1^{O2}, O_3^{O2})$:
   (1) Derive $O_1^{O2} = T(O_1, O_2)$ and $O_3^{O2} = T(O_3, O_2)$, according to types and position relationships among O_1, O_2 and O_3, by consulting the TM.
   (2) Derive position relations between the transformed operations O_1^{O2} and O_3^{O2}, according to the original position relations, and transformed positions.
   (3) Derive $O_1^{O2,O3} = T(O_1^{O2}, O_3^{O2})$, according to the operation types of O_1 and O_3 and the derived position relationship in Step(2), by consulting the TM.
2. Derive $O_1^{O3,O2} = T(O_1^{O3}, O_2^{O3})$ by similar steps.
3. Check the CP2-equation: $O_1^{O2,O3} = O_1^{O3,O2}$. If the equation is satisfied, this is CP2-confirmation case; otherwise, it is a CP2-violation case.

Steps in the CP2 verification process are well defined by the CP2-equation and the TM. Step 1-(2)/2-(2) is special for CP2-verification as it involves deriving new position relationship between transformed operations. Such position relationship is required in using the TM to derive $O_1^{O2,O3} = T(O_1^{O2}, O_3^{O2})$ in Step 1-(3)/2-(3). This position relation derivation is supported by a Position Relationship Derivation Matrix (PRDM) in APPENDIX.

IP2 and IP3 Verification Frameworks

Exhaustive IP2 and IP3 verification case derivation. IP2 and IP3 verification frameworks are closely related and discussed together. IP2/IP3 involves two independent operations O_1 and O_2, plus an inverse $!O_2$, which is not an independent operation but derived from O_2. Each independent operation may take one of two types – insert or delete, so there exist $2^2 = 4$ operation type permutations. Furthermore, each pair of operations may have three different positional relationships. Therefore, there are a total of $4 \times 3 = 12$ distinctive IP2/IP3 verification cases.

Specification 5: IP2 Verification Process

Given the set of IP2 verification cases and $TM(O_1,O_2)$, the IP2 verification process works on each case as follows:

1. Derive $O_1^{O2,!O2}$:
   (1) Derive $O_1^{O2} = T(O_1,O_2)$, according to types and position relationships between O_1 and O_2 by consulting the TM.
   (2) Derive position relationship between transformed operation O_1^{O2} and $!O_2$.
   (3) Derive $O_1^{O2,!O2} = T(O_1^{O2}, !O_2)$ according to types of O_1 and $!O_2$ and derived position relation in 1-(2), by the TM.
2. Check the IP2-equation: $O_1 = O_1^{O2,!O2}$. If the equation is satisfied, this is an IP2-confirmation case; otherwise, it is an IP2-violation case.

Specification 6: IP3 Verification Process

Given the set of IP3 verification cases and $TM(O_1,O_2)$, the IP3 verification process works on each case as follows:

1. Derive $!O_2^{O1}$:
   (1) Derive $O_1^{O2} = T(O_1,O_2)$, using types and position relationships between O_1 and O_2 to consult the TM.
   (2) Derive position relationship between transformed operation O_1^{O2} and $!O_2$.
   (3) Derive $!O_2^{O1} = T(!O_2, O_1^{O2})$ by using types of O_1 and $!O_2$ and derived position relation in Step (2) to consult the TM.
2. Derive $!(O_2^{O1})$:
   (1) Derive $O_2^{O1} = T(O_2,O_1)$, by using operations types and position relationships in the case to consult the TM.
   (2) Create an inverse operation $!(O_2^{O1})$ from O_2^{O1}.
3. Check the IP3-equation: $!O_2^{O1} = !(O_2^{O1})$. If the equation is satisfied, this is an IP3-confirmation case; otherwise, this is an IP3-violation case.

Steps in the IP2/IP3 verification process are well defined by the IP2/IP3-equation and the TM. Step 1-(2) involves deriving position relationships between a transformed operation and an inverse operation, supported by the same PRDM.

VERIFICATION TOOL

Based on the derived verification cases and the verification process specifications, we have developed a software tool, named *OTX* (*OT eXplorer*), to automate the verification process. OTX was originally developed to support development of STD-/COTS-based benchmarking suites for testing and validating real OT implementations [25], and has been extended to support automatic verification of transformation properties. The main benefit of using OTX is not so much about saving time (as the number of verification cases is not unmanageable and each verification case is simple), but more about using OTX to support experimentation with alternative transformation designs, based on different operation models, combined-effects, and transformation matrixes. We have executed all verification processes both manually and by using OTX. We present main verification discoveries in this paper, and provide full verification results in APPENDIX.

VERIFICATION DISCOVERIES

UE-CP1 Verification Discoveries

Based on UE-CP1 verification results, we declare $TM(O_1, O_2)$ *can preserve UE-CP1 under all cases.* This work is the first to formally define and verify both UE and CP1 for transformations defined on the basic data and operation models and to draw the conclusion on UE-CP1 correctness. In fact, convergence can be trivially achieved if UE (or other suitably defined combined-effects) were not considered [16,20,21]. It is essential for OT systems to achieve UE and convergence properties at the same time, and to verify both of them in an integral way.

CP2 Verification Discoveries

Based on CP2 verification for all *58* cases, we discovered $TM(O_1, O_2)$ can preserve CP2 under *57* cases, but violate CP2 in one case: $O_1 = I(p_1, c_1)$, $O_2 = D(p_2, c_2)$, and $O_3 = I(p_3, c_3)$, where $(p_1 - 1) = p_2 = p_3$.

CP2-violation vs. the FT Puzzle

The CP2-violation case is a general description of the well-known *False-Tie (FT)* puzzle [21]. This puzzle involves an *insert-insert-tie,* which is considered as a *false-tie* because the tie was not originally generated by users but created by transformation; this FT may result in inconsistent states if a normal tie-breaking rule is applied.

A concrete FT puzzle can be created by instantiating the general CP2-violation case with specific operations $O_1 = I(2, x)$, $O_2 = D(1, b)$, and $O_3 = I(1, y)$, where $((p_1=2) - 1) = (p_2=1) = (p_3=1)$ generated from the same state *"abc"*, as shown in the COTS in **Figure 3**.

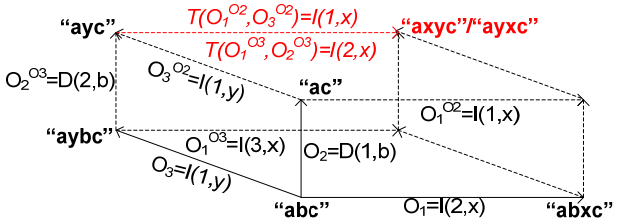


Figure 3. Illustrating the False-Tie puzzle and CP2 violation.

This COTS contains one arrow labeled by two different operations and one vertex labeled by two different states, which are resulted from transforming O_1 against O_2 and O_3 along two different paths:

1. One is transforming O_1 against O_2 and O_3^{O2} in sequence: $T(O_1^{O2}, O_3^{O2}) = I(1, x)$, which involves an FT as the two inserts $O_1^{O2} = I(1, x)$ and $O_3^{O2} = I(1, y)$ are created by transformations, and this FT is resolved by a tie-breaking scheme based on the $priority(O_1) > priority(O_3)$ (so $I(1, x)$ remains unchanged).
2. The other is transforming O_1 against O_3 and O_2^{O3} in sequence: $T(O_1^{O3}, O_2^{O3}) = I(2, x)$, where $O_1^{O3} = I(3, x)$ and $O_2^{O3} = D(2, b)$.

CP2 is violated as $I(1, x) \neq I(2, x)$, which results in divergent final states: *"axyc"* ≠ *"ayxc"*.

Since the discovery of the basic FT puzzle in [21], various CP2-violation scenarios have been reported in different OT systems [4,9,11,18] and CP2-violation phenomena became a main mystery surrounding OT correctness. However, examination of all reported CP2-violation cases revealed that they were just variations of the FT puzzle, i.e. CP2-violation occurs only if an FT is involved. Based on the exhaustiveness

of our CP2 verification, we confirm: *the FT puzzle is the only possible case for CP2-violation*, which means all puzzles in relation to CP2 have been discovered.

CP2-Preservation vs. FT-Solution

Based on CP2 verification discoveries, we assert that CP2-preservation can be achieved by solving the FT puzzle at the transformation function level. The general CP2-violation description also provides hints on how to resolve the FT puzzle: as the FT occurs only under circumstances that involve *delete* and *insert* operations with special position relationships, it is advisable to *localize* an FT solution in transformation functions related to *delete* and *insert* operations only, without disturbing the rest of the system, e.g. transformation functions unrelated to *insert*/*delete,* data and operation models, or control algorithms.

Past research has found that CP2 is not unconditionally required. CP2-violation can be and has been solved, without an FT-solution in transformation functions, by CP2-avoidance schemes in OT control algorithms [12,19,21,24, 28,29].

IP2 and IP3 Verification Discoveries

Based on IP2 and IP3 verification results for all *12* cases, we discovered $TM(O_1, O_2)$ can preserve IP2 and IP3 under ten cases, but violate IP2 and IP3 in following two cases:

1. $O_1=I(p_1,c_1), O_2=D(p_2,c_2)$, and $!O_2=I(p_2,c_2)$, $(p_1-1)=p_2$.
2. $O_1=D(p_1,c_1)$, $O_2=D(p_2,c_2)$, and $!O_2=I(p_2,c_2)$, $p_1=p_2$.

IP2/IP3-violation vs Undo Puzzles

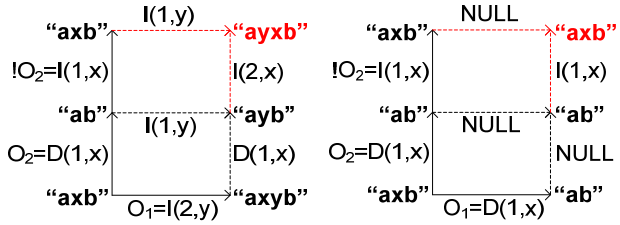
The two IP2 and IP3-violation cases are general descriptions of all known undo puzzles, previously discovered in different OT systems [14,15,16,22]. *DistEdit* was the first OT undo solution [14], which undoes an operation in a history buffer by transposing (by using transformation) this operation to the end of the buffer, then creating an inverse of the transposed operation to achieve the desired undo effect. *DistEdit* is unable to undo an operation if this operation *conflicts* with any operation on the way to the end of the buffer. Two editing operations (*insert* and *delete*) may conflict if they affect *adjacent* or *overlapping* characters in the document [14]. These conflict scenarios were the earliest reported undo puzzles, which were late rediscovered in follow-up OT undo solutions *adOPTed* [15,16] and *ANYUNDO* [22], but under different names. All undo puzzles are just special instances of general IP2/IP3-violation cases discovered in this work. Based on the exhaustiveness of IP2/IP3 verification cases, we confirm: *all undo puzzles in relation to IP2/IP3 have been found*.

Classification of Undo Puzzles

Based on the two general IP2/IP3-violation cases, we can classify all undo puzzles into two groups. One group can be described by IP2/IP3-violation Case 1. In Figure 4(a), we illustrate an instance of this group, where the initial state is state *"axb"*, $O_1=I(2, y)$, $O_2=D(1, x)$, and $!O_2=I(1, x)$, with a position relationship $(p_1 - 1) = p_2$. Using $TM(O_1, O_2)$, we can derive $O_1^{O2} = T(O_1, O_2) = I(1,y)$, and $O_2^{O1} = T(O_2, O_1) = D(1,x)$. After applying O_1 and O_2 (and one of them is transformed) in different orders on "*axb*", the same state *"ayb"* will be reached (guaranteed by the *UE-CP1* preservation of $TM(O_1, O_2)$). Then, if O_2 is undone, the state should become *"axyb"*, in which the effect of O_2 is eliminated but the effect of O_1 is retained, according to the established *undo effect* requirement in [14,15,22]. However, an incorrect state *"ayxb"* could be produced by:

1. executing $I(1, y)$ on state *"axb"*, where $I(1, y)$ is obtained by $T(I(1, y), I(1, x)) = I(1, y))$, which violates IP2 as $I(1, y)=T(T(O_1, O_2),!O_2) \neq O_1$; *or*
2. executing $I(2, x)$ on state *"ayb"*, where $I(2, x)$ is obtained from creating an *inverse* $!O_2 = I(1, x)$ and performing $T(I(1, x), I(1, y)) = I(2, x))$, which violates IP3 as $I(2, x) \neq !T(O_2, O_1) = I(1, x)$.

The above IP2-violation case involves an *insert-insert-tie* ($I(1,x)$ vs $I(1,y)$) and this tie is resolved by using *priority*(O_1)>*priority*(O_2) (assuming the priority of $!O_2$ is the same as O_2). Reversing the priority could solve the puzzle in this scenario, but will fail in another similar scenario where $O_1 = I(1, y)$, which is why this puzzle was named *insert-insert-tie* dilemma in [22]. As the order of *"x"* and *"y"* is reversed in the final state *"ayxb"* (compared to the correct undo effect "*axyb*"), this puzzle is the root of the *order-puzzle* in [15,16]; O_1 and O_2 also have an *adjacent-conflict* relationship as defined in [14].



(a) *Insert-insert-tie dilemma, order-puzzle, adjacent-conflict.* (b) *Overlapping-delete, del-del-puzzle, overlapping-conflict.*

Figure 4. Illustrating undo puzzles and IP2/IP3-violation.

The other group can be described by IP2/IP3-violation Case 2. In Figure 4(b), we illustrate an instance of this group, where the initial state is state *"axb"*, $O_1 = D(1, x)$, $O_2 = D(1, x)$, and $!O_2 = I(1, x)$, with a special position relationship: $p_1=p_2$. After applying O_1 and O_2 (and one of them is transformed) in different orders on "*axb*", the same state *"ab"* will be reached. Then, if O_2 is undone, the state should remain *"ab"* because undoing one overlapping delete O_2 should not recover *"x"* as the effect of O_1 should be retained according to the *undo effect* requirement in [14,15, 22]. However, an OT system may prematurely recover *"x"* to produce *"axb"*, which could be produced by:

1. executing *NULL* on *"axb"*, where *NULL* is obtained by $T(NULL, !O_2)$, which violates IP2 as $NULL=T(T(O_1, O_2),!O_2) \neq O_1$; or
2. executing $I(1, x)$ on *"ab"*, where $I(1, x)$ is obtained by creating $!O_2=I(1, x)$ and performing $T(I(1, x), NULL)$, which violates IP3 as $I(1, x) \neq !T(O_2, O_1)=NULL$.

If this scenario is extended to undo O_1 as well, the final state would become *"axxb"*, which is not the initial state *"axb"* (after undoing both O_1 and O_2) and thus violates the *undo property* as well [22]. In this undo puzzle scenario,

both O_1 and O_2 are *delete* operations and have an *overlapping-conflict* relationship according to [14]; this puzzle was named *del-del-puzzle* [15], and *overlapping-delete puzzle* [22], respectively, for obvious reasons.

Connections between IP2/IP3-violation and CP2-violation

By comparing verification results between IP2/IP3 and CP2, we discovered an interesting connection between undo puzzles (related to IP2/IP3-violation Case 1, as illustrated in Figure 4(a)) and the FT puzzle: one side of the IP2/IP3-equation is a deterministic result, but the other side contains an *insert-insert-tie* that has to be resolved by a tie-breaking rule. This *insert-insert-tie* is an FT because the tie is not original but created by transformation, and it plays exactly the same role as the FT does in the CP2-violation case. From this observation, we deduce: the FT phenomenon is a common cause in CP2-violation and IP2/IP3-violation Case 1, which provides hints for an *integrated* FT solution for preserving CP2, IP2 and IP3.

IP2/IP3-Preservation vs. Undo-Puzzle-Solution

Based on IP2/IP3 verification discoveries, we can assert that IP2/IP3-preservation can be achieved by solving known undo puzzles at the transformation function level. The general IP2/IP3-violation description of undo puzzles and their connections to CP2-violation also provide hints on how to resolve them: an FT solution at the transformation function level may resolve both CP2-violation and IP2/IP3-violation Case 1. Solutions for the IP2/IP3-violation Case 2 should focus on transformation functions related to two *delete* operations only.

Past research has also found that IP2 and IP3 are not unconditionally required; IP2 and IP3 can be and have been solved by breaking their preconditions at the OT control algorithm level [15,22,24], without requiring undo puzzle solutions in transformation functions.

OT Puzzles vs. Non-determinism and Non-serializability

Last but not least, we found that all OT puzzles, related to transformation properties, are rooted in the two common properties of OT systems:

1. *Non-deterministic* combined-effects for *insert-insert-ties* (some of them are FTs), which is achieved by using a tie-breaking rule. This combined-effect and achieving technique are responsible for CP2-violation and IP2/IP3-violation Case 1.
2. *Non-serialisable* combined-effects for *delete-delete-ties*, which is achieved by using a special operation NULL. This combined-effect and achieving technique are responsible for IP2/IP3-violation Case 2.

The above combined-effects and achieving techniques have been adopted by nearly all OT systems based on basic data and operation models, which explains why the same puzzles occur in different OT systems, independent of their algorithmic designs and functionalities.

CONCLUSIONS

This is the first work to verify OT correctness in both consistency maintenance and undo, with respect to established transformation properties, including UE-CP1, CP2, IP1, IP2, and IP3. In this work, we have contributed a novel system of verification frameworks, a software verification tool, and important verification discoveries.

OT puzzles were notoriously difficult to detect as they were often hidden in intricate scenarios and took varying forms in different OT systems. It is amazing to see all those puzzles manifesting themselves under verification frameworks that are solely based on data and operation models and transformation properties, independent of specific OT algorithms. Furthermore, our verification frameworks are able to exhaustively cover all verification cases, without exponential case explosion, which is essential to achieving exhaustive search of all OT puzzles. Main innovations in our verification frameworks include: formalization of basic data and operation models, formal specification of union effects as a specialization of the intention-preservation requirement, integration of union effects with convergence properties, general transformation matrix for verifying existing and new transformation functions, exhaustive verification case derivation methods based on uniformed effect ranges, and verification processes based on property equations.

The main discovery from this study is: *all OT puzzles, in relation to established transformation properties under the basic data and operation models, have been discovered*. Moreover, we discovered general descriptions and classifications of all puzzles, and connections among seemly unrelated transformation properties and puzzles. These discoveries help resolve a number of long-standing mysteries surrounding OT correctness and contributed to advancement of OT fundamental knowledge and technology. Past research has solved all discovered puzzles and many solutions are based on generic control algorithms [29], which are applicable to a wide range of applications.

We have extended our verification frameworks to cover string-wise operations and implemented this extension in OTX. The extension from char-wise to string-wise operations has increased the number of verification cases (e.g. from less than one hundred to several hundreds for CP2 verification) and operation overlapping cases (e.g. from a single delete-delete-tie case to multiple partial delete-delete overlapping cases and new delete-insert overlapping cases). This extended work has reconfirmed the effectiveness of the general verification framework and main discoveries reported in this paper, and also made interesting new discoveries, which will be reported in the future.

While core OT techniques under basic data and operation models have been thoroughly studied, new and significant challenges lie in applying core OT techniques to advanced real-world applications [1, 2, 23, 26, 27], that are based on complex data and operation models with domain-specific constraints on conflict resolution and consistency requirements. Existing and emerging real-world applications continue to provide exciting opportunities and fresh stimuli to ongoing and future OT research and innovation.

ACKNOWLEDGEMENTS

This research is partially supported by an Academic Research Fund Tier 1 Grant from Ministry of Education Singapore. The authors wish to thank anonymous reviewers for their insightful and constructive comments and suggestions.

REFERENCES

1. Agustina, Liu, F., Xia, S., Shen, H.F. and Sun, C. CoMaya: Incorporating advanced collaboration capabilities into 3D digital media design tools. *ACM CSCW* (2008), 5 – 8.
2. Agustina and Sun, C. Dependency-conflict detection in real-time collaborative 3D design systems. *ACM CSCW* (2013), 715 – 728.
3. Begole, J., Rosson, M.B. and Shaffer, C.A. Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems. *ACM TOCHI* 6, 2 (1999), 95 – 132.
4. Boucheneb, H. and Imine, A. On Model-checking optimistic replication algorithms. *FORTE on Formal Techniques for Distributed Systems* (2009),73 – 89.
5. Davis, A., Sun, C. and Lu, J. Generalizing operational transformation to the standard general markup language. *ACM CSCW* (2002), 58 – 67.
6. Ellis, C. A. and Gibbs, S. J. Concurrency control in groupware systems. *ACM SIGMOD* (1989), 399 – 407.
7. Gu, N., Yang, J. and Zhang, Q. Consistency maintenance based on the mark & retrace technique in groupware systems. *ACM GROUP* (2005), 264 – 273.
8. Ignat, C. and Norrie, M.C. Customizable collaborative editor relying on treeOPT algorithm. *ECSCW* (2003), 315 – 334.
9. Imine, A., Molli, P., Oster, G. and Rusinowitch, M. Proving correctness of transformation functions in real-time groupware. *ECSCW* (2003), 277 – 293.
10. Lamport, L. Time, clocks, and the ordering of events in a distributed system. *CACM 21*, 7 (1978), 558 – 565.
11. Li, D. and Li, R. Preserving operation effects relation in group editors. *ACM CSCW* (2004), 427 – 466.
12. Nichols, D., Curtis, P., Dixon, M. and Lamping, J. High-latency, low-bandwidth windowing in the Jupiter collaboration system. *ACM UIST* (1995), 111 – 120.
13. Palmer, C.R. and Cormack, G.V. Operation transforms for a distributed shared spreadsheet. *ACM CSCW* (1998), 69 – 78.
14. Prakash, A. and Knister, M. A framework for undoing actions in collaborative systems. *ACM TOCHI 1, 4* (1994), 295 – 330.
15. Ressel, M. and Gunzenhauser, R. Reducing the problems of group undo. *ACM GROUP* (1999), 131 – 139.
16. Ressel, M., Ruhland, N. and Gunzenhauser, R. An integrating, transformation-oriented approach to concurrency control and undo in group editors. *ACM CSCW* (1996), 288 – 297.
17. Robert, C. S. Collaborative editing for mission control. *Third Int. Workshop on Collaborative Editing Systems*, in conjunction with *ACM GROUP* (2001).
18. Shao, B., Li, D. and Gu, N. A sequence transformation algorithm for supporting cooperative work on mobile devices. *ACM CSCW* (2010), 159 – 168.
19. Shen, H.F. and Sun, C. Flexible notification for collaborative systems. *ACM CSCW* (2002), 77 – 86.
20. Sun, C. and Ellis, C.A. Operational transformation in real-time group editors: issues, algorithms, and achievements. *ACM CSCW* (1998), 59 – 68.
21. Sun, C., Jia, X., Zhang, Y., Yang, Y, and Chen, D. Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems. *ACM TOCHI 5, 1* (1998), 63 – 108.
22. Sun, C. Undo as concurrent inverse in group editors. *ACM TOCHI 9, 4* (2002), 309 – 361.
23. Sun, C., Xia, S., Sun, D., Chen, D., Shen, H. and Cai, W. Transparent adaptation of single-user applications for multi-user real-time collaboration. *ACM TOCHI 13, 4* (2006), 531 – 582.
24. Sun, D. and Sun, C. Context-based operational transformation in distributed collaborative editing systems. *IEEE TPDS 20, 10* (2009), 1454 – 1470.
25. Sun, C., Agustina, and Xu, Y. Exploring operational transformation: from core algorithms to real-world applications. *ACM CSCW* (2011) *demo.*
26. C. Sun, Wen, H. and Fan, H. Operational transformation for orthogonal conflict resolution in collaborative 2-dimensional document editors. *ACM CSCW (2012),*1391 – 1400.
27. Sun, C. OTFAQ: Operational Transformation Frequently Asked Questions and Answers. http:// www.ntu.edu.sg/home/czsun/projects/otfaq
28. Vidot, N., Cart, M., Ferrie, J. and Suleiman, M. Copies convergence in a distributed real-time collaborative environment. *ACM CSCW* (2000), 171 – 180.
29. Xu, Y., Sun, C. and Li, M. Achieving convergence in operational transformation: conditions, mechanisms and systems. *ACM CSCW* (2014).

APPENDIX of this paper can be found at http:// www.ntu.edu.sg/home/czsun/cscw2014_239_Appendix.pdf.