ISCAS-SKLCS-15-03 **April, 2015**

中国科学院软件研究所
计算机科学国家重点实验室
技术报告

TSO-to-TSO Linearizability is Undecidable

by

Chao Wang, Yi Lv and Peng Wu

State key Laboratory of Computer Science
Institute of Software
Chinese Academy of Sciences
Beijing 100190. China

TSO-to-TSO Linearizability is Undecidable

Chao Wang, Yi Lv, and Peng Wu

State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences

Abstract. *TSO-to-TSO linearizability* is a variant of linearizability for concurrent libraries on the Total Store Order (TSO) memory model. To solve its decision problem, we first show that history inclusion of libraries is the equivalent characterization of linearizability for several memory models, such as sequential consistency (SC) memory model and TSO. Furthermore, extended history inclusion of libraries on TSO is the equivalent characterization of TSO-to-TSO linearizability. The history inclusion problem on SC memory model is proved to be decidable for a bounded number of processes. For TSO memory model, we establish the connection between it and lossy channel system. Thus the trace inclusion problem of a classic-lossy single-channel system, which is known undecidable, can be reduced to the history inclusion problem of specific libraries on the TSO memory model. We prove that linearizability is undecidable for a bounded number of processes on TSO. Based on the equivalence between history inclusion and extended history inclusion for these libraries, we then prove that the extended history inclusion problem of libraries is undecidable on the TSO memory model. By means of extended history inclusion as an equivalent characterization of TSO-to-TSO linearizability, we finally prove that TSO-to-TSO linearizability is undecidable for a bounded number of processes.

1 Introduction

Libraries of high performance concurrent data structures have been widely used in concurrent programs to take advantage of multi-core architectures, such as *java.util.concurrent* for Java and *std::thread* for C++11. It is important but notoriously difficult to ensure that concurrent libraries are designed and implemented correctly. *Linearizability* [10] is accepted as a *de facto* correctness condition for a concurrent library with respect to its sequential specification on the sequential consistency (SC) memory model [11]. It is well known that on the SC memory model linearizability of a concurrent library is decidable for a bounded number of processes [1], but undecidable for an unbounded number of processes [5].

However, modern multiprocessors (e.g., x86 [14], POWER [15]) and programming languages (e.g., C/C++ [4], Java [13]) do not comply with the SC memory model. As a matter of fact, they provide *relaxed memory models*, which allow subtle behaviors due to hardware or compiler optimization. For instance, in a multiprocessor system implementing the total store order (TSO) memory model [14], each processor is equipped with an FIFO store buffer. Any write operation performed by a processor is put into its local store buffer first and can then be flushed into the main memory at any time.

The notion of linearizability has been extended for relaxed memory models, e.g., *TSO-to-TSO linearizability* [7] and *TSO-to-SC linearizability* [9] for the TSO memory model and two variants of linearizability [3] for the C++ memory model. These notions generalize the original one by relating concurrent libraries with their abstract implementations, in the way as shown in [8] for the SC memory model. It is worth mentioning that these notions of linearizability satisfy the abstraction theorem [7,9,3]: if a library is linearizable with respect to its abstract implementation, every observable behavior of any client program using the former can be observed when the program uses the latter instead. Concurrent software developer can benefit from this coincidence in that the library can be safely replaced with its abstract implementation for the sake of optimization or the ease of verification of the client program.

The decision problems for linearizability on relaxed memory models become more complicated. Because of the hierarchy of memory models, it is rather trivial to see that linearizability on relaxed memory models is undecidable for an unbounded number of processes, based on the known undecidability result on the SC memory model [5]. But the decision problem of linearizability on relaxed memory models remains open for a bounded number of processes.

In this paper we investigate the fundamental problems about the algorithmic feasibility of verifying linearizability between libraries running on memory models. Specifically, we mainly study the decision problem for the TSO-to-TSO linearizability of concurrent libraries within a bounded number of processes. TSO-to-TSO linearizability is the first definition of linearizability on relaxed memory models. It relates a library running on the TSO memory model to its abstract implementation running also on the TSO memory model. Histories of method invocations/responses are typically concerned by the standard notion of linearizability. For TSO-to-TSO linearizability, such histories have to be extended to reflect the interactions between concurrent libraries and processor-local store buffers.

The main result of this paper is that TSO-to-TSO linearizability is undecidable for a bounded number of processes. We first show that the extended history inclusion is an equivalent characterization of TSO-to-TSO linearizability. As a byproduct, we also prove that history inclusion of libraries is the equivalent characterization of standard linearizability for several memory models, such as SC and TSO. For SC memory model, standard linearizability is decidable for a bounded number of processes because the operational semantics on SC for a bounded number of processes is a finite state *labelled transition system* (LTS) and its histories is a regular set. For TSO memory model, we prove our undecidability result by reducing the trace inclusion problem between any two configurations of a classic-lossy single-channel system to the extended history inclusion problem between two specific libraries. Recall that the trace inclusion problem between configurations of a classic-lossy single-channel system is undecidable [16]. The reduction is achieved by using as a bridge the history inclusion between these two specific libraries.

Technically, we present a library template that can be instantiated as a specific library for a configuration of a classic-lossy single-channel system. The library is designed with three methods M_i for $1 \leq i \leq 3$. We use two processes P_1 and P_2, calling methods M_1 and M_2, respectively, to simulate traces of the classic-lossy single-channel

system starting from the given configuration. This is based on the observation that on the TSO memory model, a process may miss updates by other processes because multiple flush operations may occur between consecutive read operations of the process [2]. But a channel system accesses the content of a channel always in an FIFO manner; while on the contrary, a process on the TSO memory model always reads the latest updates in its local store buffer (whenever possible). Herein, processes P_1 and P_2 alternatively update their own store buffers, while read only from each other's store buffer. In this way, the labeled transitions of the classic-lossy single-channel system can be reproduced through the interactions between processes P_1 and P_2. Furthermore, we use the third process P_3, calling method M_3 repeatedly, to return each fired transition label repeatedly, so that the traces of the classic-lossy single-channel system starting from a given configuration can be mimicked by the histories of the library exactly. Specially, methods M_1 and M_2 never return, while method M_3 just uses an atomic write operation to return labels in order not to touch process P_3's store buffer. Consequently, we can easily establish the equivalence between the history inclusion and the extended history inclusion between the specific libraries.

By constructing two specific libraries based on the above library template, we show that the trace inclusion problem between any two configurations of a classic-lossy single-channel system can be reduced to the history inclusion problem between the corresponding two concurrent libraries. By means of undecidability of history inclusion for libraries on TSO, we obtain that standard linearizability is undecidable on TSO for a bounded number of processes. While the history inclusion relation and the extended history inclusion relation are equivalent between these two libraries, thus the undecidability result of TSO-to-TSO linearizability for a bounded number of processes follows from its equivalent characterization and the undecidability result of classic-lossy single-channel system. To our best knowledge, this is the first result on the decidability of linearizability of concurrent libraries on relaxed memory models.

Related work Efforts have been devoted on the decidability and model checking of linearizability on the SC memory model [1,5,6,17,12]. The principle of our equivalent characterization for TSO-to-TSO linearizability is similar to that of the characterization given by Bouajjani *et al.* in [6], where history inclusion is proved to be an equivalent characterization of linearizability. Alur *et al.* proved that for a bounded number of processes, checking whether a regular set of histories is linearizable with respect to its regular sequential specification can be reduced to a history inclusion problem, and hence is decidable [1]. Bouajjani *et al.* proved that the problem of whether a library is linearizable with respect to its regular sequential specification for an unbounded number of processes is undecidable, by a reduction from the reachability problem of a counter machine (which is known to be undecidable) [5].

On the other hand, the decidability of linearizability on relaxed memory models is still open for a bounded number of processes. The closest work to ours is [2] by Atig *et al.*, where a lossy channel system is simulated by a concurrent program on the TSO memory model. Our approach for using M_1 and M_2 to simulate classic-lossy single-channel system is inspired by their work. However, in [2] it was the decidable reachability problem of the channel system that was reduced to the reachability problem of the concurrent program on the TSO memory model. Hence, only the start and end con-

figurations of the channel system are needed in their reduction. In this paper, we reduce the trace inclusion problem between any two configurations of a classic-lossy single-channel system, which is undecidable, to the TSO-to-TSO linearizability problem. Our reduction needs to show exactly each step of transitions in the channel system.

2 Concurrent Systems

In this section, we first present the notations of libraries, the most general clients and TSO and SC concurrent systems. Then, we introduce their operational semantics on the TSO and SC memory models.

2.1 Notations

In general, a finite sequence on an alphabet Σ is denoted $l = \alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_k$, where $\cdot$ is the concatenation symbol and $\alpha_i \in \Sigma$ for each $1 \leq i \leq k$. Let $|l|$ denote the length of l, i.e., $|l| = k$, and $l(i)$ denote the i-th element of l for $1 \leq i \leq k$, i.e., $l(i) = \alpha_i$. For an alphabet Σ', let $l \uparrow_{\Sigma'}$ denote the projection of l to Σ'. Given a function f, let $f[x : y]$ be the function that shares the same value as f everywhere, except for x, where it has the value y. We use $\_$ for an item, of which the value is irrelevant.

A *labelled transition system* (LTS) is a tuple $\mathcal{A} = (Q, \Sigma, \rightarrow, q_0)$, where Q is a set of states, Σ is a set of transition labels, $\rightarrow \subseteq Q \times \Sigma \times Q$ is a transition relation and q_0 is the initial state. A state of the LTS $\mathcal{A}$ may be referred to as a *configuration* in the rest of the paper.

A path of $\mathcal{A}$ is a finite transition sequence $q_1 \xrightarrow{\beta_1} q_2 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_k} q_{k+1}$ for $k \geq 0$. A trace of $\mathcal{A}$ is a finite sequence $t = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_k$, where $k \geq 0$ if there exists a path $q_1 \xrightarrow{\beta_1} q_2 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_k} q_{k+1}$ of $\mathcal{A}$. Let *path*$(\mathcal{A}, q)$ and *trace*$(\mathcal{A}, q)$ denote all the paths and all the traces of $\mathcal{A}$ that start from q respectively, and we write *path*$(\mathcal{A})$ and *trace*$(\mathcal{A})$ for short if $q = q_0$.

2.2 Libraries and the Most General Clients

A library implementing a concurrent data structure provides a set of methods for external users to access the data structure. It may contain private memory locations for its own use. A client program is a program that interacts with libraries. For simplicity, we assume that each method has just one parameter and one return value if it returns. Furthermore, all the parameters and the return values are passed via a special register r_f.

Formally, let $\mathcal{X}$ be a finite set of memory locations, $\mathcal{M}$ be a finite set of method names, $\mathcal{D}$ be a finite data domain, $\mathcal{R}$ be a finite set of register names and $\mathcal{RE}$ be a finite set of register expressions over $\mathcal{R}$. Then, a set *PCom* of primitive commands considered in this paper includes:

- Register assign commands in the form of $r = re$;
- Register reset commands in the form of *havoc*;
- Read commands in the form of *read*(x, r);

- Write commands in the form of *write*(r, x);
- *Cas* commands in the form of $r_1 = cas(x, r_2, r_3)$;
- Assume commands in the form of *assume*(r);
- Call commands in the form of *call*(m);

where $r, r_1, r_2, r_3 \in \mathcal{R}$, $re \in \mathcal{RE}$, $x \in \mathcal{X}$. Herein, the notations of registers and register expressions are similar to those used in [7].

A *cas* command compresses a read and a write commands into a single one, which is meant to be executed atomically. It is often implemented with the compare-and-swap or load-linked/store-conditional primitive at the level of multiprocessors. This type of commands is widely used in concurrent libraries. A *havoc* command [7] assigns arbitrary values to all registers in $\mathcal{R}$.

A control-flow graph is a tuple $CFG = (N, L, T, q_i, q_f)$, where N is a finite set of program positions, L is a set of primitive commands, $T \subseteq N \times L \times N$ is a control-flow transition relation, q_i is the initial position and q_f is the final position.

A library $\mathcal{L}$ can then be defined as a tuple $\mathcal{L} = (Q_{\mathcal{L}}, \rightarrow_{\mathcal{L}}, InitV_{\mathcal{L}})$, such that (1) $Q_{\mathcal{L}} = \bigcup_{m \in \mathcal{M}} Q_m$ is a finite set of program positions, where each Q_m is the program positions of method m, and it has an unique initial position i_m and an unique final position f_m, (2) $\rightarrow_{\mathcal{L}} = \bigcup_{m \in \mathcal{M}} \rightarrow_m$ is a control-flow transition relation, where for each $m \in \mathcal{M}$, $(Q_m, PCom, \rightarrow_m, i_m, f_m)$ is a control-flow graph and (3) $InitV_{\mathcal{L}} : \mathcal{X} \rightarrow \mathcal{D}$ is an initial valuation for memory locations.

The most general client is a special client program that is used to exhibit all possible behaviors of a library. Formally, the most general client $\mathcal{MGC}$ is defined as a tuple $(\{q_c, q_c'\}, \rightarrow_c)$, where q_c and q_c' are two program positions, $\rightarrow_c = \{(q_c, havoc, q_c')\} \cup \{(q_c', call(m), q_c) | m \in \mathcal{M}\}$ is a control-flow transition relation and $(\{q_c, q_c'\}, PCom, \rightarrow_c, q_c, q_c)$ is a control-flow graph. Intuitively, the most general client repeatedly calls an arbitrary method with an arbitrary argument for arbitrarily many times.

2.3 Operational Semantics on TSO and SC

Assume a concurrent system consists of n client processes, each of which runs the most general client program on a separate processor. Then, the operational semantics of a library can be defined in the context of the concurrent system.

For a library $\mathcal{L}$=$(Q_{\mathcal{L}}, \rightarrow_{\mathcal{L}}, InitV_{\mathcal{L}})$, its operational semantics on the TSO memory model is defined as an LTS $[\![\mathcal{L}, n]\!]_{te}$[1] = $(Conf_{te}, \Sigma_{te}, \rightarrow_{te}, InitConf_{te})$, where $Conf_{te}$, Σ_{te}, $\rightarrow_{te}$, $InitConf_{te}$ are defined as follows.

Each configuration of $Conf_{te}$ is a tuple (p, d, u, r), where

- $p : \{1, \ldots, n\} \rightarrow \{q_c, q_c'\} \cup Q_{\mathcal{L}}$ represents control states of each process;
- $d : \mathcal{X} \rightarrow \mathcal{D}$ represents values at each memory location;
- $u : \{1, \ldots, n\} \rightarrow (\{(x, a) | x \in \mathcal{X}, a \in \mathcal{D}\} \cup \{call(m, a) | m \in \mathcal{M}, a \in \mathcal{D}\} \cup \{return(m, a) | m \in \mathcal{M}, a \in \mathcal{D}\})^*$ represents contents of each processor-local store buffer; each processor-local store buffer may contain a finite sequence of pending write, pending call or pending return operations;

[1] "t" represents TSO memory model. "e" represents that the operational semantics in this paper extends standard TSO operational semantics [14] similarly as [7].

- $r : \{1, \ldots, n\} \to (\mathcal{R} \to \mathcal{D})$ represents values of the registers of each process.

Σ_{te} consists of the following subsets of operations as transition labels.

- Internal operations: $\{\tau(i) | 1 \leq i \leq n\}$;
- Read operations: $\{\mathit{read}(i, x, a) | 1 \leq i \leq n, x \in \mathcal{X}, a \in \mathcal{D}\}$;
- Write operations: $\{\mathit{write}(i, x, a) | 1 \leq i \leq n, x \in \mathcal{X}, a \in \mathcal{D}\}$;
- *Cas* operations: $\{\mathit{cas}(i, x, a, b) |\ 1 \leq i \leq n, x \in \mathcal{X}, a, b \in \mathcal{D}\}$;
- Flush operations: $\{\mathit{flush}(i, x, a) |\ 1 \leq i \leq n, x \in \mathcal{X}, a \in \mathcal{D}\}$;
- Call operations: $\Sigma_{cal} = \{\mathit{call}(i, m, a) | 1 \leq i \leq n, m \in \mathcal{M}, a \in \mathcal{D}\}$;
- Return operations: $\Sigma_{ret} = \{\mathit{return}(i, m, a) | 1 \leq i \leq n, m \in \mathcal{M}, a \in \mathcal{D}\}$;
- Flush call operations: $\Sigma_{fcal} = \{\mathit{flushCall}(i, m, a) | 1 \leq i \leq n, m \in \mathcal{M}, a \in \mathcal{D}\}$;
- Flush return operations: $\Sigma_{fret} = \{\mathit{flushReturn}(i, m, a) | 1 \leq i \leq n, m \in \mathcal{M}, a \in \mathcal{D}\}$.

The initial configuration $\mathit{InitConf}_{te} \in \mathit{Conf}_{te}$ is a tuple $(p_{init}, \mathit{InitV}_{\mathcal{L}}, u_{init}, r_{init})$, where $p_{init}(i) = q_c$, $u_{init}(i) = \epsilon$ (representing an empty buffer) and $r_{init}(i)(r) = \mathit{regV}_{init}$ (a special initial value of a register) for $1 \leq i \leq n$, $r \in \mathcal{R}$;

The transition relation $\to_{te}$ is the least relation satisfying the transition rules shown in Fig. 1. Our operational semantics for the internal, read, write, flush, call, return, flush call and flush return operations are similar to the one presented in [7], except that the xlock and xunlock operations in [7] are replaced with the *cas* operations here.

- *Register-Assign* rule: A function $f_{re} : (\mathcal{R} \to \mathcal{D}) \times \mathcal{RE} \to \mathcal{D}$ is used to evaluates register expression *re* under register valuation *rv* of current process, and its value is assigned to register r_1.
- *Library-Havoc* and $\mathcal{MGC}$*-Havoc* rules: *havoc* commands are executed by current process for libraries and the most general clients respectively.
- *Assume* rule: If value of register r_1 is *true*, current process can execute *assume* command. Otherwise, the process must wait.
- *Read* rule: A function $\mathit{lookup}(u, d, i, x)$ is used to search for the latest value of x from its processor-local store buffer or the main memory, i.e.,

$$\mathit{lookup(u,d,i,x)} = \begin{cases} a & \text{if } u(i) \uparrow_{\Sigma_x} = (x, a) \cdot l, \textit{ for some } l \in \Sigma_x^* \\ d(x) & \text{otherwise} \end{cases}$$

  where $\Sigma_x = \{(x, a) | a \in \mathcal{D}\}$ is the set of pending write operations for x.
  Read operation will takes the latest value of x from processor-local store buffer if possible, otherwise, it looks up the value in memory.
- *Write* rule: A write operation will insert a pair of location and value to the tail of its processor-local store buffer.
- *Cas-Success* and *Cas-Fail* rules: A *cas* command can only be executed when the processor-local store buffer is empty and thus forces current process to clear its store buffer in advance. A successful *cas* command will change the value of memory location x immediately. The result of whether this *cas* command is successful is stored in register r_1.
- *Flush* rule: The memory system may decide to flush the entry at the head of processor-local store buffer to memory at any time.

$$\frac{p(i) = q_1, q_1 \xrightarrow{r_1 = re}_{\mathcal{L}} q_2, r(i) = rv, f_{re}(rv, re) = a}{(p, d, u, r) \xrightarrow{\tau(i)}_{te} (p[i : q_2], d, u, r[i : rv[r_1 : a]])} \textit{Register-Assign}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{havoc}_{\mathcal{L}} q_2, rv \in \mathcal{R} \to \mathcal{D}}{(p, d, u, r) \xrightarrow{\tau(i)}_{te} (p[i : q_2], d, u, r[i : rv])} \textit{Library-Havoc}$$

$$\frac{p(i) = q_c, rv \in \mathcal{R} \to \mathcal{D}}{(p, d, u, r) \xrightarrow{\tau(i)}_{te} (p[i : q_c'], d, u, r[i : rv])} \mathcal{MGC}\textit{-Havoc}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{assume(r_1)}_{\mathcal{L}} q_2, r(i)(r_1) = true}{(p, d, u, r) \xrightarrow{\tau(i)}_{te} (p[i : q_2], d, u, r)} \textit{Assume}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{read(x, r_1)}_{\mathcal{L}} q_2, r(i) = rv, lookup(u, d, i, x) = a}{(p, d, u, r) \xrightarrow{read(i, x, a)}_{te} (p[i : q_2], d, u, r[i : rv[r_1 : a]])} \textit{Read}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{write(r_1, x)}_{\mathcal{L}} q_2, r(i)(r_1) = a, u(i) = l}{(p, d, u, r) \xrightarrow{write(i, x, a)}_{te} (p[i : q_2], d, u[i : (x, a) \cdot l], r)} \textit{Write}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{r_1 = cas(x, r_2, r_3)}_{\mathcal{L}} q_2, r(i) = rv, rv(r_2) = d(x) = a, rv(r_3) = b, u(i) = \epsilon}{(p, d, u, r) \xrightarrow{cas(i, x, a, b)}_{te} (p[i : q_2], d[x : b], u, r[i : rv[r_1 : true]])} \textit{Cas-Success}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{r_1 = cas(x, r_2, r_3)}_{\mathcal{L}} q_2, r(i) = rv, rv(r_2) = a, rv(r_3) = b, rv(r_2) \neq d(x), u(i) = \epsilon}{(p, d, u, r) \xrightarrow{cas(i, x, a, b)}_{te} (p[i : q_2], d, u, r[i : rv[r_1 : false]])} \textit{Cas-Fail}$$

$$\frac{u(i) = l \cdot (x, a)}{(p, d, u, r) \xrightarrow{flush(i, x, a)}_{te} (p, d[x : a], u[i : l], r)} \textit{Flush}$$

$$\frac{p(i) = q_c', r(i)(r_f) = a, u(i) = l}{(p, d, u, r) \xrightarrow{call(i, m, a)}_{te} (p[i : i_m], d, u[i : call(m, a) \cdot l], r)} \textit{Call}$$

$$\frac{p(i) = f_m, r(i)(r_f) = a, u(i) = l}{(p, d, u, r) \xrightarrow{return(i, m, a)}_{te} (p[i : q_c], d, u[i : return(m, a) \cdot l], r)} \textit{Return}$$

$$\frac{u(i) = l \cdot call(m, a)}{(p, d, u, r) \xrightarrow{flushCall(i, m, a)}_{te} (p, d, u[i : l], r)} \textit{Flush-Call}$$

$$\frac{u(i) = l \cdot return(m, a)}{(p, d, u, r) \xrightarrow{flushReturn(i, m, a)}_{te} (p, d, u[i : l], r)} \textit{Flush-Return}$$

Fig. 1. Transition Rules of $\rightarrow_{te}$

- *Call* and *Return* rules: To deal with *call* command, a call marker is added into the tail of processor-local store buffer and current process starts to execute the initial position of method m. When the process comes to final position of method m it can launch a *return* operation, add a return marker to the tail of processor-local store buffer and start to execute the most general client.
- *Flush-Call* and *Flush-Return* rules: The call and return marker can be discarded when they are at the head of processor-local store buffer. Such operations are used to define TSO-to-TSO linearizability only.

For a library $\mathcal{L}$=$(Q_{\mathcal{L}}, \rightarrow_{\mathcal{L}}, \mathit{InitV}_{\mathcal{L}})$, its operational semantics on SC memory model is defined as an LTS $[\![\mathcal{L}, n]\!]_{sc}$ = $(\mathit{Conf}_{sc}, \Sigma_{sc}, \rightarrow_{sc}, \mathit{InitConf}_{sc})$, where Conf_{sc}, Σ_{sc}, $\rightarrow_{sc}$ and $\mathit{InitConf}_{sc}$ are defined as follows.

- Each configuration of Conf_{sc} is a tuple (p, d, r), where p, d and r are same to that of $[\![\mathcal{L}, n]\!]_{te}$.
- Σ_{sc} is a subset of Σ_{te} and does not contain flush call and flush return operations.
- The initial configuration $\mathit{InitConf}_{cs} \in \mathit{Conf}_{te}$ is a tuple $(p_{\mathit{init}}, \mathit{InitV}_{\mathcal{L}}, r_{\mathit{init}})$.
- The transition relation $\rightarrow_{sc}$ is the least relation satisfying the transition rules shown in Fig. 2.

3 Correctness Conditions and Equivalent Characterizations

3.1 Correctness Conditions

Linearizability [10,8] is accepted as a standard correctness condition for concurrent libraries. The behavior of a library is typically represented by histories of interactions between the library and the clients calling it (through call and return operations). A finite sequence $h \in (\Sigma_{\mathit{cal}} \cup \Sigma_{\mathit{ret}})^*$ is a history of an LTS $\mathcal{A}$ if there exists a trace t of $\mathcal{A}$ such that $t \uparrow_{(\Sigma_{\mathit{cal}} \cup \Sigma_{\mathit{ret}})} = h$. Let $\mathit{history}(\mathcal{A})$ denote all the histories of $\mathcal{A}$, and $h|_i$ the projection of h to the operations of the i-th process. Two histories h_1 and h_2 are equivalent, if for each $1 \leq i \leq n$, $h_1|_i = h_2|_i$.

Definition 1 (*linearizability* [8]). *Given two histories h_1 and h_2 of libraries, h_1 is* linearizable *to h_2, if*

- *they are equivalent;*
- *there is a bijection $\pi : \{1, \ldots, |h_1|\} \rightarrow \{1, \ldots, |h_2|\}$ such that for any $1 \leq i \leq |eh_1|$, $h_1(i) = h_2(\pi(i))$;*
- *for any $1 \leq i < j \leq |h_1|$, if $h_1(i) \in \Sigma_{\mathit{ret}} \wedge h_1(j) \in \Sigma_{\mathit{cal}}$, then $\pi(i) < \pi(j)$.*

For two libraries $\mathcal{L}_1$ and $\mathcal{L}_2$, we say that $\mathcal{L}_1$ is linearizable with respect to $\mathcal{L}_2$ on TSO, if for any $h_1 \in \mathit{history}([\![\mathcal{L}_1, n]\!]_{te})$, there exists $h_2 \in \mathit{history}([\![\mathcal{L}_2, n]\!]_{te})$, such that h_1 is linearizable to h_2. We say that $\mathcal{L}_1$ is linearizable with respect to $\mathcal{L}_2$ on SC, if for any $h_1 \in \mathit{history}([\![\mathcal{L}_1, n]\!]_{sc})$, there exists $h_2 \in \mathit{history}([\![\mathcal{L}_2, n]\!]_{sc})$, such that h_1 is linearizable to h_2.

$$\frac{p(i) = q_1, q_1 \xrightarrow{r_1 = re}_{\mathcal{L}} q_2, r(i) = rv, f_{re}(rv, re) = a}{(p, d, r) \xrightarrow{\tau(i)}_{sc} (p[i : q_2], d, r[i : rv[r_1 : a]])}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{havoc}_{\mathcal{L}} q_2, rv \in \mathcal{R} \to \mathcal{D}}{(p, d, r) \xrightarrow{\tau(i)}_{sc} (p[i : q_2], d, r[i : rv])}$$

$$\frac{p(i) = q_c, rv \in \mathcal{R} \to \mathcal{D}}{(p, d, r) \xrightarrow{\tau(i)}_{sc} (p[i : q_c'], d, r[i : rv])}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{assume(r_1)}_{\mathcal{L}} q_2, r(i)(r_1) = true}{(p, d, r) \xrightarrow{\tau(i)}_{sc} (p[i : q_2], d, r)}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{read(x, r_1)}_{\mathcal{L}} q_2, r(i) = rv, d(x) = a}{(p, d, r) \xrightarrow{read(i, x, a)}_{sc} (p[i : q_2], d, r[i : rv[r_1 : a]])}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{write(r_1, x)}_{\mathcal{L}} q_2, r(i)(r_1) = a}{(p, d, r) \xrightarrow{write(i, x, a)}_{sc} (p[i : q_2], d[x : a], r)}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{r_1 = cas(x, r_2, r_3)}_{\mathcal{L}} q_2, r(i) = rv, rv(r_2) = d(x) = a, rv(r_3) = b}{(p, d, r) \xrightarrow{cas(i, x, a, b)}_{sc} (p[i : q_2], d[x : b], r[i : rv[r_1 : true]])}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{r_1 = cas(x, r_2, r_3)}_{\mathcal{L}} q_2, r(i) = rv, rv(r_2) = a, rv(r_3) = b, rv(r_2) \neq d(x)}{(p, d, r) \xrightarrow{cas(i, x, a, b)}_{sc} (p[i : q_2], d, r[i : rv[r_1 : false]])}$$

$$\frac{p(i) = q_c', r(i)(r_f) = a}{(p, d, r) \xrightarrow{call(i, m, a)}_{sc} (p[i : i_m], d, r)}$$

$$\frac{p(i) = f_m, r(i)(r_f) = a}{(p, d, r) \xrightarrow{return(i, m, a)}_{sc} (p[i : q_c], d, r)}$$

Fig. 2. Transition Rules of $\rightarrow_{sc}$

TSO-to-TSO linearizability is a variant of linearizability on the TSO memory model. It additionally concerns the behavior of a library in the context of processor-local store buffers, i.e, the interactions between the library and store buffers through flush call and flush return operations. A finite sequence $eh \in (\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret})^*$ is an extended history of an LTS $\mathcal{A}$ if there exists a trace t of $\mathcal{A}$ such that $t \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret})} = eh$. Let $ehistory(\mathcal{A})$ denote all the extended histories of $\mathcal{A}$, and $eh|_i$ the projection of eh to the operations of the i-th process. Two extended history eh_1 and eh_2 are equivalent, if for each $1 \leq i \leq n$, $eh_1|_i = eh_2|_i$.

Definition 2 (*TSO-to-TSO linearizability* [7]). *Given two extended histories eh_1 and eh_2 of libraries, eh_1 is TSO-to-TSO linearizable to eh_2, if*

- *eh_1 and eh_2 are equivalent;*

- *there is a bijection* $\pi : \{1, \ldots, |eh_1|\} \to \{1, \ldots, |eh_2|\}$ *such that for any* $1 \leq i \leq |eh_1|$, $eh_1(i) = eh_2(\pi(i))$;
- *for any* $1 \leq i < j \leq |eh_1|$, *if* $(eh_1(i) \in \Sigma_{ret} \cup \Sigma_{fret}) \wedge (eh_1(j) \in \Sigma_{cal} \cup \Sigma_{fcal})$, *then* $\pi(i) < \pi(j)$.

For two libraries $\mathcal{L}_1$ *and* $\mathcal{L}_2$, *we say that* $\mathcal{L}_2$ *TSO-to-TSO linearizes* $\mathcal{L}_1$, *if for any* $eh_1 \in$ *ehistory*$(\llbracket \mathcal{L}_1, n \rrbracket_{te})$, *there exists* $eh_2 \in$ *ehistory*$(\llbracket \mathcal{L}_2, n \rrbracket_{te})$, *such that* eh_1 *is TSO-to-TSO linearizable to* eh_2.

Informally speaking, if eh_1 is *TSO-to-TSO Linearizable* to eh_2, then eh_2 keeps all the non-overlapping pairs of call/flush call and return/flush return operations in eh_1 in the same order.

3.2 Equivalence Characterizations

To handle the decision problem of TSO-to-TSO linearizability, we show that the extended history inclusion is an equivalent characterization of TSO-to-TSO linearizability. This is presented formally as the following lemma.

Lemma 1. *For any two libraries* $\mathcal{L}_1$ *and* $\mathcal{L}_2$, $\mathcal{L}_2$ *TSO-to-TSO linearizes* $\mathcal{L}_1$ *if and only if* *ehistory*$(\llbracket \mathcal{L}_1, n \rrbracket_{te}) \subseteq$ *ehistory*$(\llbracket \mathcal{L}_2, n \rrbracket_{te})$.

Proof. (Sketch) The *if* direction is obvious by the definition of TSO-to-TSO linearizability (Definition 2).

The *only if* direction can be proved with a transformation relation $\Rightarrow_{ER}$ and a distance function $eWitSum$ we define upon extended histories. Given extended histories eh_1 and eh_2, we say eh_1 can be transformed in one step to eh_2, written $eh_1 \Rightarrow_{ER} eh_2$, if eh_2 can be obtained by swapping two adjacent elements of eh_1 and eh_1 is TSO-to-TSO linearizable to eh_2. The non-negative distance function $eWitSum(eh_1, eh_2)$ for two equivalent extended histories eh_1 and eh_2 is used to measure the difference between eh_1 and eh_2. Through the well-defined transformation relation and distance function, we can first show that if an extended history eh_1 is TSO-to-TSO linearizable to another extended history eh_2 and $eh_1 \neq eh_2$, then there exists a third extended history eh_3, such that

- eh_1 is TSO-to-TSO linearizable to eh_3;
- $eh_3 \Rightarrow_{ER} eh_2$;
- eh_3 is TSO-to-TSO linearizable to eh_2;
- the distance between eh_1 and eh_3 is strictly less than the one between eh_1 and eh_2.

In this way, if an extended history $eh_1 \in$ *ehistory*$(\llbracket \mathcal{L}_1, n \rrbracket_{te})$ is TSO-to-TSO linearizable to an extended history $eh_2 \in$ *ehistory*$(\llbracket \mathcal{L}_2, n \rrbracket_{te})$, then eh_1 can be transformed to eh_2 by a finite number of $\Rightarrow_{ER}$ transformations. We further show that eh_1, eh_2 and all the intermediate extended histories along the transformations are indeed the extended histories of $\llbracket \mathcal{L}_2, n \rrbracket_{te}$. Please refer to Appendix A.1 for the detailed proof of this lemma. □

Similar to above case, we can prove that history inclusion is an equivalent characterization of linearizability on TSO and SC memory model. The proof to this proposition is simpler than proof for Lemma 1 and can be found in Appendix A.2.

Proposition 1. *For any two libraries $\mathcal{L}_1$ and $\mathcal{L}_2$, $\mathcal{L}_1$ is linearizable with respect to $\mathcal{L}_2$ on TSO (SC) if and only if history$(\llbracket \mathcal{L}_1, n \rrbracket_{te}) \subseteq$ history$(\llbracket \mathcal{L}_2, n \rrbracket_{te})$ (history$(\llbracket \mathcal{L}_1, n \rrbracket_{sc}) \subseteq$ history$(\llbracket \mathcal{L}_2, n \rrbracket_{sc})$).*

It is obvious that for each library $\mathcal{L}$ and positive integer n, $\llbracket \mathcal{L}, n \rrbracket_{sc}$ is a finite state LTS and *history*$(\llbracket \mathcal{L}, n \rrbracket_{sc})$ is a regular set. Based on the decidability of inclusion between two regular sets and equivalence between history inclusion and standard linearizability from Lemma 1, it is not hard to see that the problem of standard linearizability between libraries on SC memory model is decidable for a bounded number of processes.

Corollary 1. *For any two libraries $\mathcal{L}_1$ and $\mathcal{L}_2$, it is decidable whether $\mathcal{L}_1$ is linearizable with respect to $\mathcal{L}_2$ for a bounded number of processes on SC.*

4 Undecidability of TSO-to-TSO Linearizability

As the main result of this paper, we present in this section that the TSO-to-TSO linearizability of concurrent libraries is undecidable for a bounded number of processes. We first reduce the trace inclusion problem between any two configurations of a classic-lossy single-channel system to the history inclusion problem between two specific concurrent libraries. Then, our main undecidability result follows from the equivalence between the history inclusion and the extended history inclusion for these two libraries. Recall that the latter is equivalent to TSO-to-TSO linearizability between the two libraries based on the above Lemma 1. We also prove that standard linearizability of libraries on TSO is undecidable for a bounded number of processes.

4.1 Classic-Lossy Single-Channel Systems

A classic-lossy single-channel system [16] is a tuple $\mathcal{S} = (Q_{cs}, \Sigma_{cs}, \{c_{cs}\}, \Gamma_{cs}, \Delta_{cs})$, where Q_{cs} is a finite set of control states, Σ_{cs} is a finite alphabet of messages, c_{cs} is the name of the single channel, Γ_{cs} is a finite set of transition labels and $\Delta_{cs} \subseteq Q_{cs} \times \Sigma_{cs}^* \times \Gamma_{cs} \times Q_{cs} \times \Sigma_{cs}^*$ is a transition relation.

Given two finite sequences $l_1 = \alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_u$ and $l_2 = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_v$, we say that l_1 is a *subword* of l_2, denoted $l_1 \sqsubseteq l_2$, if there exists $1 \leq i_1 < \ldots < i_u \leq v$ such that for any $1 \leq j \leq u$, $\alpha_j = \beta_{i_j}$. Then, the operational semantics of $\mathcal{S}$ is given by an LTS $\mathcal{CL}(\mathcal{S}) = (Conf_{cs}, \Gamma_{cs}, \rightarrow_{cs}, initConf_{cs})$, where $Conf_{cs} = Q_{cs} \times \Sigma_{cs}^*$ is a set of configurations with $initConf_{cs} \in Conf_{cs}$ as the initial configuration. The transition relation $\rightarrow_{cs}$ is defined as follows: $(q_1, W_1) \xrightarrow{\alpha}_{cs} (q_2, W_2)$ if there exists $(q_1, U, \alpha, q_2, V) \in \Delta_{cs}$ and $W' \in \Sigma_{cs}^*$ such that $U \cdot W' \sqsubseteq W_1$ and $W_2 \sqsubseteq W' \cdot V$.

It is known that for two configurations $(q_1, W_1), (q_2, W_2) \in Conf_{cs}$ of a classic-lossy single-channel system $\mathcal{S}$, the trace inclusion between (q_1, W_1) and (q_2, W_2) is undecidable [16].

4.2 Simulation on the TSO Memory Model

On the TSO memory model flush operations are launched nondeterministically by the memory system. Therefore, between two consecutive *read*$(x, \_)$ operations, more than one flush operations to x may happen. The second read operation can only read the latest flush operation to x, while missing the intermediate ones. These missing flush operations are similar to the missing messages that may happen in a classic-lossy single-channel system. This makes it possible to simulate a classic-lossy single-channel system with a concurrent program running on the TSO memory model. We implement such simulation through a most general client and a library $\mathcal{L}_{\mathcal{S},q,W}$ specifically constructed based on a classic-lossy single-channel system $\mathcal{S}$ and a given configuration $(q, W) \in$ $Conf_{cs}$.

For a classic-lossy single-channel system $\mathcal{S}$ =(Q_{cs}, Σ_{cs}, $\{c_{cs}\}$, Γ_{cs}, Δ_{cs}), assume the finite data domain $\mathcal{D}_{cs}$ = $Q_{cs} \cup \Sigma_{cs} \cup \Delta_{cs} \cup \{\sharp, start, end, \bot, true, false, regV_{init}, rule_f\}$, where $Q_{cs} \cap \Sigma_{cs} = \emptyset$, $Q_{cs} \cap \Delta_{cs} = \emptyset$, $\Sigma_{cs} \cap \Delta_{cs} = \emptyset$, and the symbols $\sharp$, *start*, *end*, $\bot$, *true*, *false*, $regV_{init}$ and $rule_f$ do not exist in $Q_{cs} \cup \Sigma_{cs} \cup \Delta_{cs}$. Given a configuration $(q, W) \in Conf_{cs}$ of $\mathcal{S}$, the library $\mathcal{L}_{\mathcal{S},q,W}$ is constructed with three methods M_1, M_2 and M_3, and three private memory locations x, y and z. x is used to transmit the channel contents from M_2 to M_1, while y is used to transmit the channel contents from M_1 to M_2. z is used to transmit the transition labels of $\mathcal{CL}(\mathcal{S})$ from M_2 to M_3, and it is also used to synchronize M_2 and M_3. The symbol $\sharp$ is used as the delimiter to ensure that one element will not be read twice. The symbols *start* and *end* represent the start and the end of the channel contents, respectively. $\bot$ is the initial value of x, y and z. The symbol $rule_f$ is an additional transition rule that is used to indicate the end of a simulation.

We now present the three methods in the pseudo-code, shown in Methods 1, 2 and 3. The *if* and *while* statements used in the pseudo-code can be easily implemented by the *assume* commands as well as other commands in our formation of a library. For the sake of brevity, the following macro notations are used. For sequence $l = a_1 \cdot \ldots \cdot a_m$, let *writeSeq(x,l)* denote the commands of writing $a_1, \sharp, \ldots, a_m, \sharp$ to x in sequence, and *readSeq(x,l)* denote the commands of reading $a_1, \sharp, \ldots, a_m, \sharp$ from x in sequence. We use $v :=$ *readOne*(x) to represent the commands of reading $e, \sharp$ from x in sequence for some $e \neq \sharp$ and then assigning e to v. If *readSeq*(x, l) or *readOne*(x) fails to read the specified content, then the calling process will no long proceed. We use *writeOne*$(x,$ *reg*$)$ to represent the commands of writing $a, \sharp$ to x in sequence where a is the current value of register *reg*. In the pseudo-code, r is a register in $\mathcal{R}$.

The pseudo-code of method M_1 is shown in Method 1. M_1 contains an infinite loop that never returns (Lines 1-3). At each round of the loop, it reads a new update from x and writes it to y.

The pseudo-code of method M_2 is shown in Method 2. M_2 first guesses a transition rule $rule_1$, puts $rule_1$ and W into the processor-local store buffer by writing them to x (Lines 1-2). Then, it begins an infinite loop that never returns (Lines 3-16). At each round of the loop, it reads the current transition rule $rule_2 \in \Delta_{cs}$ of $\mathcal{S}$ (Line 4) and guesses a transition rule $rule_3 \in \Delta_{cs} \cup \{rule_f\}$ (Line 5). If M_2 guesses the rule $rule_f$ in Line 5, then in the next round of this loop it will be blocked at Line 4 and the simulation terminates. M_2 does not confirm $rule_2$ until it reads $start \cdot U_1$ from y at Line 6 (intermediate values of y may be lost). At Line 7, it writes $rule_3 \cdot start$ to y. Then, it

reads the remaining contents of method M_1's processor-local store buffer (intermediate values of y may be lost) and writes them and V_1 to x (Lines 8-13). In Lines 14-16, it transmits the transition label α_1 to method M_3.

The pseudo-code of method M_3 is shown in Method 3. M_3 first waits for M_2 to transmit transition label to it though z by a non $\perp$ value (Lines 1-4). Then it acknowledges M_2 at Line 5 and returns this transition label z at Line 6. M_3 uses a *cas* command to communicate with M_2. It never puts a pending write operation to its processor-local store buffer.

Method 1: M_1

Input: an arbitrary argument

```
1  while true do
2  |  r := readOne(x);
3  |  writeOne(y, r);
```

Method 2: M_2

Input: an arbitrary argument

```
1  guess a transition rule rule_1 = (q, _, _, _, _) ∈ Δ_cs ∪ {rule_f};
2  writeSeq(x, rule_1 · start · W · end);
3  while true do
4  |  r := readOne(y) for some rule rule_2 = (q_1, U_1, α_1, q_2, V_1) ∈ Δ_cs;
5  |  guess a transition rule rule_3 that is either some (q_2, _, _, _, _) ∈ Δ_cs or rule_f;
6  |  readSeq(y, start · U_1);
7  |  writeSeq(x, rule_3 · start);
8  |  while true do
9  |  |  r := readOne(y);
10 |  |  if r = end then
11 |  |  |  break;
12 |  |  writeSeq(x, r);
13 |  writeSeq(x, V_1 · end);
14 |  z := α_1;
15 |  while z ≠ ⊥ do
16 |  |  ;
```

Method 3: M_3

Input: an arbitrary argument

Output: transition label for one step in $\mathcal{CL}(\mathcal{S})$

```
1  while true do
2  |  r := z;
3  |  if r ≠ ⊥ then
4  |  |  break;
5  cas(z, r, ⊥);
6  return v;
```

4.3 Undecidability of History Inclusion

In this subsection we show that given a classic-lossy single-channel system $\mathcal{S}$ and a configuration $(q, W) \in Conf_{cs}$, the histories of library $\mathcal{L}_{\mathcal{S},q,W}$ simulate exactly the paths of $\mathcal{S}$ start from (q, W).

A path $p_{\mathcal{S}} = (q_1, W_1) \xrightarrow{\alpha_1}_{cs} (q_2, W_2) \xrightarrow{\alpha_2}_{cs} \dots \xrightarrow{\alpha_k}_{cs} (q_{k+1}, W_{k+1}) \in \mathit{path}(\mathcal{CL}(\mathcal{S}), (q_1, W_1))$ is *conservative*, if the following two conditions hold: (1) it contains at least one transition, (2) assume the *i-th* step uses rule $r_i = (q_i, U_i, \alpha_i, q_{i+1}, V_i)$ for each $1 \leq i \leq k$, then for each $1 \leq i \leq k$, there exists $W_i', W_i'' \in \Sigma_{cs}^*$ such that $U_i \cdot W_i' \sqsubseteq W_i$, $W_i'' \sqsubseteq W_i'$ and $W_{i+1} = W_i'' \cdot V_i$. Intuitively, each i-th step of a conservative path does not lose any element in V_i.

A trace $t_{\mathcal{L}} \in \mathit{trace}(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$ is *effective*, if $t_{\mathcal{L}}$ contains at least one operation $\mathit{return}(\_, M_3, \_)$. Otherwise, it is *ineffective*.

There is actually a close connection between the conservative paths of $\mathcal{CL}(\mathcal{S})$ and the effective traces of $\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te}$. An effective trace $t_{\mathcal{L}} \in \mathit{trace}(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$ and a conservative path $p_{\mathcal{S}} \in \mathit{path}(\mathcal{CL}(\mathcal{S}), (q, W))$ *correspond*, if the sequence of return values of M_3 in $t_{\mathcal{L}}$ is the same as the sequence of transition labels of $p_{\mathcal{S}}$.

Fig. 3 shows an example of generating a corresponding effective trace of $\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te}$ from a conservative path of $\mathcal{CL}(\mathcal{S})$. Note that many possible executions of $\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te}$ can get into deadlock due to the operational semantics and the pseudo-code of each method. Herein, we consider only the executions where M_3 always manages to output return labels accordingly. In Fig. 3, contents of a store buffer are written from left to right, while the time progresses from left to right, too. Assume $(q, W) = (q_1, a \cdot a)$ and there is a conservative path $p_{\mathcal{S}} = (q_1, a \cdot a) \xrightarrow{\alpha_1}_{cs} (q_2, b \cdot c) \xrightarrow{\alpha_2}_{cs} (q_3, a)$, where the first step uses rule $\mathit{rule}_1 = (q_1, a, \alpha_1, q_2, b \cdot c)$ (loses a in the channel), and the second one uses rule $\mathit{rule}_2 = (q_2, b, \alpha_2, q_3, a)$ (loses c in the channel). For this path, we can get a corresponding effective trace of $t_{\mathcal{L}}$ as follows:

1. Run M_1, M_2 and M_3 in processes P_1, P_2 and P_3 respectively. Recall that M_1 and M_2 never return, while each invocation of M_3 is associated with an interval shown in Fig. 3.
2. At Line 2 of Method 2, M_2 puts (x, rule_1), $(x, \sharp)$, (x, start), $(x, \sharp)$, (x, a), $(x, \sharp)$, (x, a), $(x, \sharp)$, (x, end), $(x, \sharp)$ into the store buffer of process P_2.
3. By several loops of Lines 1-3, M_1 captures the updates of x in a lossy manner, and puts (y, rule_1), $(y, \sharp)$, (y, start), $(y, \sharp)$, (y, a), $(y, \sharp)$, (y, end), $(y, \sharp)$ into the store buffer of process P_1.
4. At Line 4 of Method 2, M_2 captures the updates of y in a lossy manner. M_2 guesses an applicable transition rule rule_2, and then puts (x, rule_2) ,$(x, \sharp)$, (x, start), $(x, \sharp)$, (x, b), $(x, \sharp)$, (x, c), $(x, \sharp)$, (x, end), $(x, \sharp)$ into the store buffer of process P_2, according to transition rule rule_1.
5. M_2 sends the transition label α_1 to M_3 at Line 14 of Method 2. Then, M_3 returns α_1 and we finish simulating the first transition in $p_{\mathcal{S}}$.
6. By several loops of Lines 1-3, M_1 captures the updates of x in a lossy manner, and puts (y, rule_2), $(y, \sharp)$, (y, start), $(y, \sharp)$, (y, b), $(y, \sharp)$, (y, end), $(y, \sharp)$ into the store buffer of process P_1.

7. At Line 4 of Method 2, M_2 captures the updates of y. Then, M_2 decides to terminate the simulation and puts (x, rule_f),$(x, \sharp)$, (x, start), $(x, \sharp)$, (x, a), $(x, \sharp)$, (x, end), $(x, \sharp)$ into the store buffer of process P_2, according to transition rule rule_2.
8. M_2 sends the transition label α_2 to M_3 at Line 14 of Method 2. Then, M_3 returns α_2 and we finish simulating the second transition in $p_{\mathcal{S}}$.

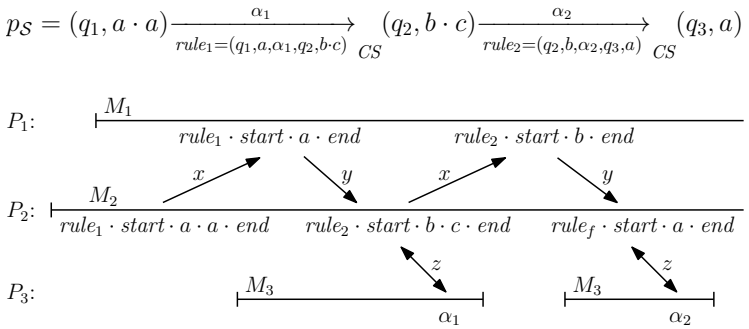


Fig. 3. A Conservative Path and its Corresponding Effective Trace

It can be seen that $t_{\mathcal{L}}$ and $p_{\mathcal{S}}$ correspond in this example. We further prove that for any conservative path $p_{\mathcal{S}} \in \mathit{path}(\mathcal{CL}(\mathcal{S}), (q, W))$, there exists an effective trace $t_{\mathcal{L}} \in \mathit{trace}(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$ such that $t_{\mathcal{L}}$ and $p_{\mathcal{S}}$ correspond, and vice versa, as stated in Lemmas 21 and 22 in Appendix B.2.

The following lemma shows that the history inclusion between concurrent libraries is undecidable on the TSO memory model.

Lemma 2. *For any two libraries $\mathcal{L}_1$ and $\mathcal{L}_2$, it is undecidable whether* $\mathit{history}(\llbracket \mathcal{L}_1, 3 \rrbracket_{te}) \subseteq \mathit{history}(\llbracket \mathcal{L}_2, 3 \rrbracket_{te})$.

Proof. (Sketch) By Lemmas 21 and 22, it can be proved that for any two configurations (q_1, W_1), $(q_2, W_2) \in \mathit{Conf}_{cs}$ of a classic-lossy single-channel system $\mathcal{S}$, if $\mathit{history}(\llbracket \mathcal{L}_{\mathcal{S},q_1,W_1}, 3 \rrbracket_{te}) \subseteq \mathit{history}(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_{te})$, then $\mathit{trace}(\mathcal{CL}(\mathcal{S}), (q_1, W_1)) \subseteq \mathit{trace}(\mathcal{CL}(\mathcal{S}), (q_2, W_2))$. In Appendix B.3, we prove that the other way around also holds. Therefore, the undecidability result follows from that the trace inclusion problem between any two configurations of a classic-lossy single-channel system is undecidable [16]. □

By Lemma 1, history inclusion is an equivalent characterization of standard linearizability on TSO. Therefore, it is obvious that standard linearizability between libraries on TSO is undecidable for a bounded number of processes.

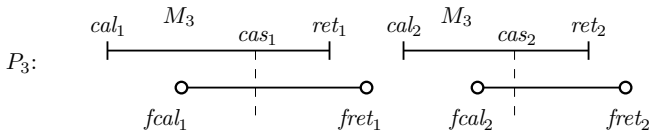
Corollary 2. *For any two concurrent libraries $\mathcal{L}_1$ and $\mathcal{L}_2$, it is undecidable whether $\mathcal{L}_1$ is linearizable with respect to $\mathcal{L}_2$ for a bounded number of processes on TSO.*

4.4 Undecidability of TSO-to-TSO linearizability

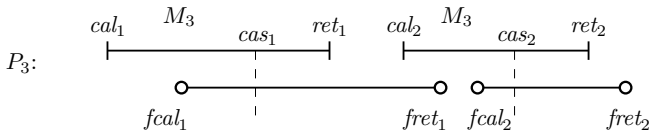
Although we prove above that history inclusion is undecidable on the TSO memory model, there is still a gap between the history inclusion and the extended history inclusion between concurrent libraries. Obviously there exist libraries $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\mathit{history}(\llbracket \mathcal{L}_1, n \rrbracket_{te}) \subseteq \mathit{history}(\llbracket \mathcal{L}_2, n \rrbracket_{te})$ but $\mathit{ehistory}(\llbracket \mathcal{L}_1, n \rrbracket_{te}) \not\subseteq \mathit{ehistory}(\llbracket \mathcal{L}_2, n \rrbracket_{te})$. We show in this subsection that for the two libraries $\mathcal{L}_{\mathcal{S},q_1,W_1}$ and $\mathcal{L}_{\mathcal{S},q_2,W_2}$, corresponding to the configurations (q_1, W_1) and (q_2, W_2) of a classic-lossy single-channel system, respectively, the history inclusion and the extended history inclusion between $\mathcal{L}_{\mathcal{S},q_1,W_1}$ and $\mathcal{L}_{\mathcal{S},q_2,W_2}$ coincides on the TSO memory model.

Without loss of generality, assume M_1 and M_2 of $\mathcal{L}_{\mathcal{S},q,W}$ are called by processes P_1, P_2, respectively; while M_3 of $\mathcal{L}_{\mathcal{S},q,W}$ is repeatedly called by process P_3. Then, an extended history $\mathit{eh} \in \mathit{ehistory}(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$ that contains at least one return operation of M_3 is in the following form:

- The first six operations of *eh* are always call and corresponding flush call operations of M_1, M_2 and M_3, while these operations may occur in any order.
- The projection of *eh* on P_i is exactly $\mathit{call}(i, M_i, \_) \cdot \mathit{flushCall}(i, M_i, \_)$ for $i \in \{1, 2\}$.
- Fig. 4 shows the possible positions of flush call (*fcal*) and flush return (*fret*) operations in eh. Since M_3 always executes a *cas* command before it returns, during each round of a call to M_3 in P_3, the flush call operation must occur before the *cas* operation (see the dashed vertical lines in Fig. 4); hence it can only occur before the return operation of M_3.
  During each round of a call to M_3 in P_3, the flush return operation may occur alternatively at two positions: the first position is after the return operation of M_3 and before the next round of a call operation of M_3, as shown the position of fret_1 in Fig. 4 (a); while the second one is after the next round of a call operation of M_3 and before the consequent flush call operation, as shown the position of fret_1 in Fig. 4 (b).



(a) fret_1 occurs between ret_1 and cal_2



(b) fret_1 occurs between cal_2 and $fcal_2$

Fig. 4. Possible Positions of Flush Call and Flush Return Operations

To prove that the history inclusion and thpe extended history inclusion coincide between libraries $\mathcal{L}_{\mathcal{S},q_1,W_1}$ and $\mathcal{L}_{\mathcal{S},q_2,W_2}$, we need to show that for an extended history

eh_1 of $[\![\mathcal{L}_{\mathcal{S},q_1,W_1},3]\!]_{te}$, if eh_1 contains a return operation in P_3 and $eh_1 \uparrow_{(\Sigma_{cal}\cup\Sigma_{ret})} \in$ *history*$([\![\mathcal{L}_{\mathcal{S},q_2,W_2},3]\!]_{te})$, then $eh_1 \in$ *ehistory*$([\![\mathcal{L}_{\mathcal{S},q_2,W_2},3]\!]_{te})$. Because $eh_1 \uparrow_{(\Sigma_{cal}\cup\Sigma_{ret})}$ is a history of $[\![\mathcal{L}_{\mathcal{S},q_2,W_2},3]\!]_{te}$, there exists a path $p'_{\mathcal{L}}$ of $[\![\mathcal{L}_{\mathcal{S},q_2,W_2},3]\!]_{te}$ corresponding to eh_1. From $p'_{\mathcal{L}}$ we can generate another path $p_{\mathcal{L}}$ of $[\![\mathcal{L}_{\mathcal{S},q_2,W_2},3]\!]_{te}$ such that the extended history along $p_{\mathcal{L}}$ is exactly eh_1.

The path $p_{\mathcal{L}}$ is generated from $p'_{\mathcal{L}}$ by changing the positions of the flush return operations. Recall that during each round of a call to M_3, the flush return operation may occur alternatively at two positions only. Since M_3 does not insert any pending write operation into the process P_3's store buffer, $p'_{\mathcal{L}}$ can be transformed into $p_{\mathcal{L}}$ by swapping each flush return operation in $p'_{\mathcal{L}}$ from its current position to the other possible one if necessary.

An extended history is *effective* if it contains at least one *return*$(\_, M_3, \_)$ operation. Otherwise, it is *ineffective*. The following lemma formalizes the idea describe above and is proved in Appendix B.4.

Lemma 3. *For a classic-lossy single-channel system $\mathcal{S}$ and two configurations (q_1, W_1), $(q_2, W_2) \in$ Conf$_{cs}$, if $eh_1 \in$ ehistory$([\![\mathcal{L}_{\mathcal{S},q_1,W_1},3]\!]_{te})$ is an effective extended history and $eh_1 \uparrow_{(\Sigma_{cal}\cup\Sigma_{ret})} \in$ history$([\![\mathcal{L}_{\mathcal{S},q_2,W_2},3]\!]_{te})$, then $eh_1 \in$ ehistory$([\![\mathcal{L}_{\mathcal{S},q_2,W_2},3]\!]_{te})$.*

With the help of Lemma 3, we can prove that the history inclusion and the extended history inclusion between the specific libraries coincide on the TSO memory model.

Lemma 4. *For two configurations $(q_1, W_1), (q_2, W_2)$ of a classic-lossy single-channel system $\mathcal{S}$, history$([\![\mathcal{L}_{\mathcal{S},q_1,W_1},3]\!]_{te}) \subseteq$ history$([\![\mathcal{L}_{\mathcal{S},q_2,W_2},3]\!]_{te})$ if and only if ehistory $([\![\mathcal{L}_{\mathcal{S},q_1,W_1},3]\!]_{te}) \subseteq$ ehistory$([\![\mathcal{L}_{\mathcal{S},q_2,W_2},3]\!]_{te})$.*

Proof. The *if* direction is obvious.

The *only if* direction can be proved by contradiction. Assume there is an extended history eh_1 such that $eh_1 \in$ *ehistory*$([\![\mathcal{L}_{\mathcal{S},q_1,W_1},3]\!]_{te})$ but $eh_1 \notin$ *ehistory*$([\![\mathcal{L}_{\mathcal{S},q_2,W_2},3]\!]_{te})$.

We can see that the set of ineffective histories of $\mathcal{L}_{\mathcal{S},q_1,W_1}$ and $\mathcal{L}_{\mathcal{S},q_2,W_2}$ are the same (see Lemma in Appendix B.5). By assumption, eh_1 is not an ineffective extended history of $\mathcal{L}_{\mathcal{S},q_2,W_2}$, thus it is not an ineffective extended history of $\mathcal{L}_{\mathcal{S},q_1,W_1}$. However, eh_1 is an extended history of $\mathcal{L}_{\mathcal{S},q_1,W_1}$, so eh_1 must be an effective extended history of $\mathcal{L}_{\mathcal{S},q_1,W_1}$.

Let history $h = eh_1 \uparrow_{(\Sigma_{cal}\cup\Sigma_{ret})}$. It is obvious that $h \in$ *history*$([\![\mathcal{L}_{\mathcal{S},q_1,W_1},3]\!]_{te})$. Then, by assumption, $h \in$ *history*$([\![\mathcal{L}_{\mathcal{S},q_2,W_2},3]\!]_{te})$. By Lemma 3, $eh_1 \in$ *ehistory*$([\![\mathcal{L}_{\mathcal{S},q_2,W_2},3\,]\!]_{te})$, which contradicts the assumption. □

The undecidability of TSO-to-TSO linearizability for a bounded number of processes is a direct consequence of Lemmas 1, 2 and 4.

Theorem 1. *For any two concurrent libraries $\mathcal{L}_1$ and $\mathcal{L}_2$, it is undecidable whether $\mathcal{L}_2$ TSO-to-TSO linearizes $\mathcal{L}_1$ for a bounded number of processes.*

5 Conclusion and Future Work

We have shown that the decision problem of TSO-to-TSO linearizability is undecidable for a bounded number of processes. The proof method is essentially by a reduction from

a known undecidable problem, the trace inclusion problem of a classic-lossy single-channel system. To facilitate such a reduction, we introduced an intermediate notion of history inclusion between concurrent libraries on the TSO memory model. We then demonstrated that a configuration (q, W) of a classic-lossy single-channel system $\mathcal{S}$ can be simulated by a specific library $\mathcal{L}_{\mathcal{S},q,W}$, interacting with three specific processes on the TSO memory model. Although history inclusion does not coincide with extended history inclusion in general, they do coincide on a restricted class of libraries. We prove that $\mathcal{L}_{\mathcal{S},q,W}$ lies within such class. Finally, our undecidability result follows from the equivalence between extended history inclusion and TSO-to-TSO linearizability.

The problem of the linearizability between libraries on the SC memory model [8] can be shown to be decidable for a bounded number of processes. This is due to the provable equivalence between history inclusion and linearizability on the SC memory model, while the former is decidable. Thus, our work states clearly a boundary of decidability for linearizability of concurrent libraries on various memory models. In fact, as a by-product of this work, we can show that the standard linearizability on TSO memory model, which is the weaker notion of TSO-to-TSO linearizability without concerning flush call and flush return operations, is already undecidable for a bounded number of processes.

Other relaxed memory models, such as the memory models of POWER and ARM, are much weaker than the TSO memory model. We conjecture that variants of linearizability on these relaxed memory models may also be reduced to some new forms of extended history inclusion, similar to the variants of linearizability for C/C++ memory model in [3], and these variants should also be undecidable. However, we find no clue to the decision problem of TSO-to-SC linearizability for a bounded number of processes, since TSO-to-SC linearizability is between linearizability on the SC memory model and TSO-to-TSO linearizability. As future work, we would like to investigate the decidability of TSO-to-SC linearizability and other variants of linearizability for relaxed memory models.

References

1. Alur, R., McMillan, K., Peled, D.: Model-checking of correctness conditions for concurrent objects. In: LICS 1996, pp. 219–228. IEEE Computer Society (1996)
2. Atig, M.F., Bouajjani, A., Burckhardt, S., Musuvathi, M.: On the verification problem for weak memory models. In: Hermenegildo, M. V., Palsberg, Jens. (eds.) POPL 2010, pp. 7–18. ACM (2010)
3. Batty, M., Dodds, M., Gotsman, A.: Library abstraction for C/C++ concurrency. In: Giacobazzi, R., Cousot, R. (eds.) POPL 2013, pp. 235–248. ACM (2013)
4. Batty, M., Owens, S., Sarkar, S., Sewell, P., Weber, T.: Mathematizing C++ concurrency. In: Ball, T., Sagiv, M. (eds.) POPL 2011, pp. 55–66. ACM (2011)
5. Bouajjani, A., Emmi, M., Enea, C., Hamza, J.: Verifying concurrent programs against sequential specifications. In: Felleisen, M., Gardner, P. (eds.) ESOP 2013. LNCS, vol. 7792, pp. 290–309. Springer (2013)
6. Bouajjani, A., Emmi, M., Enea, C., Hamza, J.: Tractable Refinement Checking for Concurrent Objects. In: Rajamani, S. K., Walker, David. (eds.) POPL 2015, pp. 651–662. ACM (2015)

7. Burckhardt, S., Gotsman, A., Musuvathi, M., Yang, H.: Concurrent library correctness on the TSO memory model. In: Seidl, H. (eds.) ESOP 2012. LNCS, vol. 7211, pp. 87–107. Springer (2012)
8. Filipovic, I., O'Hearn, P., Rinetzky, N., Yang, H.: Abstraction for concurrent objects. In: Castagna, G. (eds.) ESOP 2009. LNCS, vol. 5502, pp. 252–266. Springer (2009)
9. Gotsman, A., Musuvathi, M., Yang, H.: Show no weakness: Sequentially consistent specifications of TSO libraries. In: Aguilera, M. K. (eds.) DISC 2012. LNCS, vol. 7611, pp. 31–45. Springer (2012)
10. Herlihy, M.P., Wing, J.M.: Linearizability: A correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. **12**(3), 463–492 (1990)
11. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess program. IEEE Transactions on Computers **28**(9), 690–691 (1979)
12. Liu, Y., Chen, W., Liu, Y.A., Sun, J., Zhang, S.J., Dong, J.S.: Verifying linearizability via optimized refinement checking. IEEE Trans. Software Eng. **39**(7), 1018–1039 (2013)
13. Manson, J., Pugh, W., Adve, S.V.: The Java memory model. In: Palsberg, J., Abadi, M. (eds.) POPL 2005, pp. 378–391. ACM (2005)
14. Owens, S., Sarkar, S., Sewell, P.: A better x86 memory model: x86-TSO. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 391–407. Springer (2009)
15. Sarkar, S., Sewell, P., Alglave, J., Maranget, L., Williams, D.: Understanding POWER multiprocessors. In: Hall, M. W., Padua, D. A. (eds.) PLDI 2011, pp. 175–186. ACM (2011)
16. Schnoebelen, P.: Bisimulation and other undecidable equivalences for lossy channel systems. In: Kobayashi, N.,Pierce, B. C. (eds.) TACS 2001, pp. 385–399. Springer (2001)
17. Vechev, M.T., Yahav, E., Yorsh, G.: Experience with model checking linearizability. In: Pasareanu, C. S. (eds.) SPIN 2009. LNCS, vol. 5578, pp. 261–278. Springer (2009)

A Proofs In Section 3

A.1 Proof of Lemma 1

In this section we will prove that extended history inclusion is an equivalent characterization of TSO-to-TSO linearizability. It is obvious that extended history inclusion implies TSO-to-TSO linearizability. To prove the opposite direction, we need to prove that if $\mathcal{L}_2$ TSO-to-TSO linearizes $\mathcal{L}_1$, $eh_1 \in$ *ehistory*$(\llbracket \mathcal{L}_1, n \rrbracket_{te})$, $eh_2 \in$ *ehistory*$(\llbracket \mathcal{L}_2, n \rrbracket_{te})$ and eh_1 is TSO-to-TSO linearizable to eh_2, then $eh_1 \in$ *ehistory*$(\llbracket \mathcal{L}_2, n \rrbracket_{te})$. Let us use an example to explain how to prove the opposite direction shown in Figure 5. For simplicity we assume that eh_1 and eh_2 contain only call and return operations. Here, a time axis runs from left to right, and each method is associated with an interval.

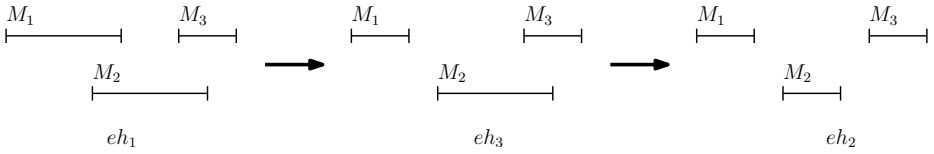


Fig. 5. Transformation from eh_1 to eh_2

It is not hard to see that there are two differences between eh_1 and eh_2. The first one is the order between return of M_1 and call of M_2, and the second one is the order between return of M_2 and call of M_3. We can generate another extended history eh_3 from eh_1 by swapping return of M_1 and call of M_2. It is easy to see that eh_1 can be transformed to eh_3 and eh_3 can be transformed to eh_2 by swapping one pair of adjacent elements and do not violate TSO-to-TSO linearizability. We can obtain eh_3 from eh_2 by delaying return of M_2 safely, since this transformation does not introduce any new overlapping situation between operations. Therefore we can conclude that $eh_3 \in$ *ehistory*$(\llbracket \mathcal{L}_2, n \rrbracket_{te})$. Similarly, $eh_1 \in$ *ehistory*$(\llbracket \mathcal{L}_2, n \rrbracket_{te})$.

Let $\Sigma_{ei} = \{\tau(i)\} \cup \{read(i,x,d) | x \in \mathcal{X}, d \in \mathcal{D}\} \cup \{write(i,x,d) | x \in \mathcal{X}, d \in \mathcal{D}\} \cup \{cas(i,x,d_1,d_2) | x \in \mathcal{X}, d_1, d_2 \in \mathcal{D}\} \cup \{flush(i,x,a) | x \in \mathcal{X}, a \in \mathcal{D}\} \cup \{call(i,m,a) | m \in \mathcal{M}, a \in \mathcal{D}\} \cup \{return(i,m,a) | m \in \mathcal{M}, a \in \mathcal{D}\} \cup \{flushCall(i,m,a) | m \in \mathcal{M}, a \in \mathcal{D}\} \cup \{flushReturn(i,m,a) | m \in \mathcal{M}, a \in \mathcal{D}\}$. Σ_{ei} is the set of process i's operations in Σ_{te}. Let relation $ER = ((\Sigma_{ret} \cup \Sigma_{fret}) \times (\Sigma_{cal} \cup \Sigma_{fcal})) \cup (\Sigma_{e1} \times \Sigma_{e1}) \cup \ldots \cup (\Sigma_{en} \times \Sigma_{en})$. A transformation $\Rightarrow_{ER}$ is a relation between two extended histories and it is defined as follows: $eh_1 \Rightarrow_{ER} eh_2$, if $eh_1 = l_1 \cdot \alpha \cdot \beta \cdot l_2$, $eh_2 = l_1 \cdot \beta \cdot \alpha \cdot l_2$ and $(\alpha, \beta) \notin ER$. We write $\Rightarrow^*_{ER}$ to denote the transition closure of $\Rightarrow_{ER}$.

Given two equivalent extended histories eh_1 and eh_2, we say that they correspond by π, if π is a bijection between $\{1, \ldots, |eh_1|\}$ and $\{1, \ldots, |eh_2|\}$, moreover for all i, j, k, if $j < k \wedge eh_1(j), eh_1(k) \in \Sigma_{ei}$, then $\pi(j) < \pi(k)$. It is not hard to see that given two equivalent extended histories eh_1 and eh_2, there exists only one such bijection. Given two equivalent extended histories eh_1 and eh_2 and assume eh_1 and eh_2 correspond by π, predicate $eWit(eh_1, eh_2, i_1, i_2)$ holds if $i_1 < i_2$ and $\pi^{-1}(i_1) > \pi^{-1}(i_2)$. Given two equivalent extended histories eh_1 and eh_2, $eWitSum(eh_1, eh_2) = |\{(m,n) | eWit(eh_1,$

$eh_2, m, n)\ holds\}|$ is a distance function which is used to measure difference between eh_1 and eh_2.

The following lemma shows that if $eWitSum(eh_1, eh_2) > 0$, there must be a pair of adjacent elements of eh_2 such that the order of them on eh_1 is different from the order of them on eh_2.

Lemma 5. *Assume two equivalent extended histories eh_1 and eh_2 correspond by π. If $eWitSum(eh_1, eh_2) > 0$, then there exists positive number $m \in \{1, \ldots, |eh_2|\}, \pi^{-1}(m) > \pi^{-1}(m+1)$.*

Proof. Let $minDis = min\{n - m | eWit(eh_1, eh_2, m, n)\ holds\}$. We know immediately that there exists a positive integer ind such that $\pi^{-1}(ind) > \pi^{-1}(ind + minDis)$. We need to prove that $minDis = 1$. It is obvious that $minDis \geq 1$. Now we claim that $minDis \leq 1$. This can be seen by a case analysis, based on the various positions of $\pi^{-1}(ind)$ and $\pi^{-1}(ind + 1)$. The former case is $\pi^{-1}(ind) < \pi^{-1}(ind + 1)$. We can find that $\pi^{-1}(ind + 1) > \pi^{-1}(ind + minDis)$ and $minDis \leq minDis - 1$. This case does not hold. Thus the latter case of $\pi^{-1}(ind) > \pi^{-1}(ind + 1)$ must hold. Thus $eWit(eh_1, eh_2, ind, ind + 1)$ holds and $minDis \leq 1$. Therefore we can conclude that $minDis = 1$. This completes the proof of Lemma 5. □

With above lemma we can prove Lemma 6.

Lemma 6. *If an extended history eh_1 is TSO-to-TSO linearizable to an extended history eh_2 and $eh_1 \neq eh_2$, then there exists an extended history eh_3 such that: (1) $eh_3 \Rightarrow_{ER} eh_2$, (2) eh_1 is TSO-to-TSO linearizable to eh_3, (3) eh_3 is TSO-to-TSO linearizable to eh_2, (4) $eWitSum(eh_1, eh_2) > eWitSum(eh_1, eh_3)$.*

Proof. Because eh_1 is TSO-to-TSO linearizable to eh_2, eh_1 and eh_2 are equivalent. Assume eh_1 and eh_2 correspond by π_{I2}. It is obvious that $eWitSum(eh_1, eh_2) > 0$. By Lemma 5 there exists a positive integer m such that $\pi_{I2}^{-1}(m) > \pi_{I2}^{-1}(m+1)$. Assume $eh_2 = l_1 \cdot eh_2(m) \cdot eh_2(m+1) \cdot l_2$.

By definition of TSO-to-TSO linearizability, the order of elements in $(\Sigma_{ret} \times \Sigma_{cal}) \cup (\Sigma_{fret} \times \Sigma_{cal}) \cup (\Sigma_{fret} \times \Sigma_{fcal}) \cup (\Sigma_{ret} \times \Sigma_{fcal}) \cup (\Sigma_{e1} \times \Sigma_{e1}) \cup \ldots \cup (\Sigma_{en} \times \Sigma_{en})$ on eh_1 is the same as that on eh_2. Therefore, the process id of $eh_2(m)$ is different from the process id of $eh_2(m+1)$, and it is not the case that $eh_2(m) \in (\Sigma_{cal} \cup \Sigma_{fcal})$ and $eh_2(m+1) \in (\Sigma_{ret} \cup \Sigma_{fret})$.

Let $eh_3 = l_1 \cdot eh_2(m+1) \cdot eh_2(m) \cdot l_2$. It is obvious that eh_1 and eh_3 are equivalent. Assume eh_1 and eh_3 correspond by π_{I3}. Let us prove that eh_1 is TSO-to-TSO linearizable to eh_3 by contradiction. Assume eh_1 is not TSO-to-TSO linearizable to eh_3. Thus there must exist ind_1, ind_2, such that $ind_1 < ind_2, \pi_{I3}(ind_1) > \pi_{I3}(ind_2)$, the process id of $eh_1(ind_1)$ is different from the process id of $eh_1(ind_2)$, $eh_1(ind_1) \in \Sigma_{ret} \cup \Sigma_{fret}$ and $eh_1(ind_2) \in \Sigma_{cal} \cup \Sigma_{fcal}$. Recall that eh_1 is TSO-to-TSO linearizable to eh_2 and the only difference between eh_2 and eh_3 is the order of $eh_2(m)$ and $eh_2(m+1)$. Therefore the only candidate for ind_1 and ind_2 is $\pi_{I2}^{-1}(m+1)$ and $\pi_{I2}^{-1}(m)$. But it is not the case that $eh_2(m) \in (\Sigma_{cal} \cup \Sigma_{fcal}) \wedge eh_2(m+1) \in (\Sigma_{ret} \cup \Sigma_{fret})$. This contradicts our assumption.

Now we prove that eh_3 is TSO-to-TSO linearizable to eh_2 by contradiction. Assume eh_3 is not TSO-to-TSO linearizable to eh_2. Recall that the only difference between eh_2

and eh_3 is the order between $eh_2(m)$ and $eh_2(m+1)$. Thus we can conclude that $eh_2(m) \in (\Sigma_{cal} \cup \Sigma_{fcal}) \wedge eh_2(m+1) \in (\Sigma_{ret} \cup \Sigma_{fret})$. But we already know it is not the case that $eh_2(m) \in (\Sigma_{cal} \cup \Sigma_{fcal})$ and $eh_2(m+1) \in (\Sigma_{ret} \cup \Sigma_{fret})$. This contradicts our assumption.

At last we prove that $eWitSum(eh_1, eh_2) > eWitSum(eh_1, eh_3)$. This holds for two reasons. First, the only difference between eh_2 and eh_3 is the order between $eh_2(m)$ and $eh_2(m+1)$. Second, the order between $eh_2(m)$ and $eh_2(m+1)$ is the same on eh_1 and eh_3. □

With Lemma 6 we can prove that if eh_1 is TSO-to-TSO linearizable to eh_2 then eh_2 can be generated from eh_1 by finite number of $\Rightarrow_{ER}$ transformations.

Lemma 7. *If eh_1 is TSO-to-TSO linearizable to eh_2, then $eh_1 \Rightarrow^*_{ER} eh_2$.*

Proof. Repeatedly apply Lemma 6. This iteration will finally stop because the value of distance function *eWitSum* is a non-negative integer, and it decreases during each step of iteration. □

The following lemma states that given a trace of $[\![\mathcal{L}, n]\!]_{te}$, a call operation β of process i and a corresponding operation α that changes position from q_c to q'_c and uses *havoc* command to set values of register r_f for β on that trace, we can move α and β forward over successive operations that are not in $(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret}) \cap \Sigma_{ei}$ to obtain another trace of $[\![\mathcal{L}, n]\!]_{te}$. We can also move α forward and move β backward over successive operations that are not in $(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret}) \cap \Sigma_{ei}$ to obtain another trace of $[\![\mathcal{L}, n]\!]_{te}$.

Lemma 8. *If $t_1 = l_1 \cdot l_2 \cdot \alpha \cdot l_3 \cdot \beta \cdot l_4 \in trace([\![\mathcal{L}, n]\!]_{te})$, $\beta \in \Sigma_{cal} \cap \Sigma_{ei}$, $\alpha \in \Sigma_{ei}$ is the corresponding operation that uses havoc command to set value of r_f for β, and there is no $(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret}) \cap \Sigma_{ei}$ operation in $l_2 \cdot l_3$, then $t_2 = l_1 \cdot \alpha \cdot \beta \cdot l_2 \cdot l_3 \cdot l_4 \in trace([\![\mathcal{L}, n]\!]_{te})$.*

If $t_3 = l_1 \cdot l_2 \cdot \alpha' \cdot \beta' \cdot l_3 \cdot l_4 \in trace([\![\mathcal{L}, n]\!]_{te})$, $\beta' \in \Sigma_{cal} \cap \Sigma_{ei}$, $\alpha' \in \Sigma_{ei}$ is the corresponding operation that uses havoc command to set value of r_f for β', and there is no $(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret}) \cap \Sigma_{ei}$ operation in $l_2 \cdot l_3$, then $t_4 = l_1 \cdot \alpha' \cdot l_2 \cdot l_3 \cdot \beta' \cdot l_4 \in trace([\![\mathcal{L}, n]\!]_{te})$.

Proof. Assume $\beta = call(i, m, g)$ and the path for t_1 is $path_1 = q_0 \xrightarrow{\alpha_1}_{te} q_1 \ldots \xrightarrow{\alpha_m}_{te} q_m \xrightarrow{\alpha_{m+1}}_{te} \ldots \xrightarrow{\alpha_{m+u}}_{te} q_{m+u} \xrightarrow{\alpha}_{te} q_{m+u+1} \xrightarrow{\alpha_{m+u+1}}_{te} \ldots \xrightarrow{\alpha_{m+u+v}}_{te} q_{m+u+v+1} \xrightarrow{\beta}_{te} q_{m+u+v+2} \xrightarrow{\alpha_{m+u+v+1}}_{te} \ldots \xrightarrow{\alpha_{m+u+v+k}}_{te} q_{m+u+v+k+2}$, where $q_0 = InitConf_{te}$, $\alpha_1 \cdot \ldots \cdot \alpha_m = l_1$, $\alpha_{m+1} \ldots \alpha_{m+u} = l_2$, $\alpha_{m+u+1} \ldots \alpha_{m+u+v} = l_3$ and $\alpha_{m+u+v+1} \ldots \alpha_{m+u+v+k} = l_4$. Let each configuration $q_j = (p_j, d_j, u_j, r_j)$. Assume $u_{m+u+v+1}(i) = l$ and $r_{m+u+1}(i) = rv$. Let us construct a new path $path_2 = q_0 \xrightarrow{\alpha_1}_{te} q_1 \ldots \xrightarrow{\alpha_m}_{te} q_m \xrightarrow{\alpha}_{te} q'_{m+1} \xrightarrow{\beta}_{te} q'_{m+2} \xrightarrow{\alpha_{m+l}}_{te} \ldots \xrightarrow{\alpha_{m+u}}_{te} q'_{m+u+2} \xrightarrow{\alpha_{m+u+1}}_{te} \ldots \xrightarrow{\alpha_{m+u+v}}_{te} q_{m+u+v+2} \xrightarrow{\alpha_{m+u+v+1}}_{te} q_{m+u+v+3} \ldots \xrightarrow{\alpha_{m+u+v+k}}_{te} q_{m+u+v+k+2}$, where $q'_{m+1} = (p_m[i : q'_c], d_m, u_m, r_m[i : rv])$, $q'_{m+2} = (p_m[i : i_m], d_m, u_m[i : call(m, g) \cdot l], r_m[i : rv])$, for each $1 \leq j \leq u$, $q'_{m+j+2} = (p_{m+j}[i : i_m], d_{m+j}, u_{m+j}[i : call(m, g) \cdot l], r_{m+j}[i : rv])$ and for each $1 \leq j \leq v$, $q'_{m+u+j+2} = (p_{m+u+j+1}[i : i_m], d_{m+u+j+1}, u_{m+u+j+1}[i : call(m, g) \cdot l], r_{m+u+j+1})$. It is not hard to see that $path_2$ is a path of $[\![\mathcal{L}, n]\!]_{te}$ and is the path for t_2.

Assume $\beta' = call(i, m, g)$ and the path for t_3 is $path_3 = q_0 \xrightarrow{\alpha_1}_{te} q_1 \ldots \xrightarrow{\alpha_m}_{te} q_m \xrightarrow{\alpha_{m+1}}_{te} \ldots \xrightarrow{\alpha_{m+u}}_{te} q_{m+u} \xrightarrow{\alpha'}_{te} q_{m+u+1} \xrightarrow{\beta'}_{te} q_{m+u+2} \xrightarrow{\alpha_{m+u+1}}_{te} \ldots \xrightarrow{\alpha_{m+u+v}}_{te} q_{m+u+v+2} \xrightarrow{\alpha_{m+u+v+1}}_{te} \ldots \xrightarrow{\alpha_{m+u+v+k}}_{te} q_{m+u+v+k+2}$, where $q_0 = InitConf_{te}$, $\alpha_1 \cdot \ldots \cdot \alpha_m = l_1$, $\alpha_{m+1} \ldots \alpha_{m+u} = l_2$, $\alpha_{m+u+1} \ldots \alpha_{m+u+v} = l_3$ and $\alpha_{m+u+v+1} \ldots \alpha_{m+u+v+k} = l_4$. Let each configuration $q_j = (p_j, d_j, u_j, r_j)$. Assume $u_{m+u}(i) = l$ and $r_{m+u+1}(i) = rv$. Let us construct a new path $path_4 = q_0 \xrightarrow{\alpha_1}_{te} q_1 \ldots \xrightarrow{\alpha_m}_{te} q_m \xrightarrow{\alpha'}_{te} q'_{m+1} \xrightarrow{\alpha_{m+1}}_{te} \ldots \xrightarrow{\alpha_{m+u}}_{te} q'_{m+u+1} \xrightarrow{\alpha_{m+u+1}}_{te} \ldots \xrightarrow{\alpha_{m+u+v}}_{te} q'_{m+u+v+1} \xrightarrow{\beta'}_{te} q_{m+u+v+2} \xrightarrow{\alpha_{m+u+v+1}}_{te} q_{m+u+v+3} \ldots \xrightarrow{\alpha_{m+u+v+k}}_{te} q_{m+u+v+k+2}$, where $q'_{m+1} = (p_m[i : q'_c], d_m, u_m, r_m[i : rv])$, for each $1 \leq j \leq u$, $q'_{m+j+1} = (p_{m+j}[i : q'_c], d_{m+j}, u_{m+j}, r_{m+j}[i : rv])$ and for each $1 \leq j \leq v$, $q'_{m+u+j+1} = (p_{m+u+j+2}[i : q'_c], d_{m+u+j+2}, u_{m+u+j+2}[i : l], r_{m+u+j+2})$. It is not hard to see that $path_4$ is a path of $[\![\mathcal{L}, n]\!]_{te}$ and is the path for t_4. □

Similarly we can move return operation of process i and the next operation of process i and generate a new trace.

Lemma 9. *If $t_1 = l_1 \cdot \alpha \cdot l_2 \cdot \beta \cdot l_3 \cdot l_4 \in trace([\![\mathcal{L}, n]\!]_{te})$, $\alpha \in \Sigma_{ret} \cap \Sigma_{ei}$, $\beta \in \Sigma_{ei}$ is the next operation of process i, and there is no $(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret}) \cap \Sigma_{ei}$ operation in $l_2 \cdot l_3$, then $t_2 = l_1 \cdot l_2 \cdot l_3 \cdot \alpha \cdot \beta \cdot l_4 \in trace([\![\mathcal{L}, n]\!]_{te})$.*

If $t_3 = l_1 \cdot \alpha' \cdot l_2 \cdot l_3 \cdot \beta' \cdot l_4 \in trace([\![\mathcal{L}, n]\!]_{te})$, $\alpha' \in \Sigma_{ret} \cap \Sigma_{ei}$, $\beta' \in \Sigma_{ei}$ is the next operation of process i, and there is no $(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret}) \cap \Sigma_{ei}$ operation in $l_2 \cdot l_3$, then $t_4 = l_1 \cdot l_2 \cdot \alpha' \cdot \beta' \cdot l_3 \cdot l_4 \in trace([\![\mathcal{L}, n]\!]_{te})$.

If $t_5 = l_1 \cdot \alpha'' \cdot l_2 \cdot l_3 \in trace([\![\mathcal{L}, n]\!]_{te})$, $\alpha'' \in \Sigma_{ret} \cap \Sigma_{ei}$ and there is neither $(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret}) \cap \Sigma_{ei}$ operation nor operation that changes program position from q_c to q'_c of process i in l_2, then $t_6 = l_1 \cdot l_2 \cdot \alpha'' \cdot l_3 \in trace([\![\mathcal{L}, n]\!]_{te})$.

Similarly we can move flush call and flush return operations and generate a new trace.

Lemma 10. *If $t_1 = l_1 \cdot l_2 \cdot \alpha \cdot l_3 \in trace([\![\mathcal{L}, n]\!]_{te})$, $\alpha \in \Sigma_{fcal} \cap \Sigma_{ei}$ and there is no $(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret}) \cap \Sigma_{ei}$ operation in l_2, then $t_2 = l_1 \cdot \alpha \cdot l_2 \cdot l_3 \in trace([\![\mathcal{L}, n]\!]_{te})$.*

If $t_3 = l_1 \cdot \beta \cdot l_2 \cdot l_3 \in trace([\![\mathcal{L}, n]\!]_{te})$, $\beta \in \Sigma_{fret} \cap \Sigma_{ei}$ and there is no $(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret}) \cap \Sigma_{ei}$ operation in l_2, then $pt_4 = l_1 \cdot l_2 \cdot \beta \cdot l_3 \in trace([\![\mathcal{L}, n]\!]_{te})$.

Proof. Assume $\alpha = flushCall(i, m, g)$ and the path for t_1 is $path_1 = q_0 \xrightarrow{\alpha_1}_{te} q_1 \ldots \xrightarrow{\alpha_m}_{te} q_m \xrightarrow{\alpha_{m+1}}_{te} \ldots \xrightarrow{\alpha_{m+u}}_{te} q_{m+u} \xrightarrow{\alpha}_{te} q_{m+u+1} \xrightarrow{\alpha_{m+u+1}}_{te} \ldots \xrightarrow{\alpha_{m+u+v}}_{te} q_{m+u+v+1}$, where $q_0 = InitConf_{te}$, $\alpha_1 \cdot \ldots \cdot \alpha_m = l_1$, $\alpha_{m+1} \ldots \alpha_{m+u} = l_2$ and $\alpha_{m+u+1} \ldots \alpha_{m+u+v} = l_3$. Let each configuration $q_j = (p_j, d_j, u_j, r_j)$. Because there is no $\Sigma_{cal} \cap \Sigma_{ei}$ operation in l_2, the corresponding call operation for α is in l_1. Assume $u_{m+u+1}(i) = l$. Let us construct a new path $path_2 = q_0 \xrightarrow{\alpha_1}_{te} q_1 \ldots \xrightarrow{\alpha_m}_{te} q_m \xrightarrow{\alpha}_{te} q'_{m+1} \xrightarrow{\alpha_{m+1}}_{te} \ldots \xrightarrow{\alpha_{m+u}}_{te} q'_{m+u+1} \xrightarrow{\alpha_{m+u+1}}_{te} q_{m+u+2} \ldots \xrightarrow{\alpha_{m+u+v}}_{te} q_{m+u+v+1}$, where $q'_{m+1} = (p_m, d_m, u_m[i : l], r_m)$ and for each $1 \leq j \leq u$, $q'_{m+j+1} = (p_{m+j}, d_{m+j}, u_{m+j}[i : l], r_{m+j})$. It is not hard to see that $path_2$ is a path of $[\![\mathcal{L}, n]\!]_{te}$ and is the path for t_2.

Assume $\beta = flushReturn(i, m, g)$ and the path for t_3 is $path_3 = q_0 \xrightarrow{\alpha_1}_{te} q_1 \ldots \xrightarrow{\alpha_m}_{te} q_m \xrightarrow{\beta}_{te} q_{m+1} \xrightarrow{\alpha_{m+1}}_{te} \ldots \xrightarrow{\alpha_{m+u}}_{te} q_{m+u+1} \xrightarrow{\alpha_{m+u+1}}_{te} \ldots \xrightarrow{\alpha_{m+u+v}}_{te} q_{m+u+v+1}$, where $q_0 = InitConf_{te}$, $\alpha_1 \cdot \ldots \cdot \alpha_m = l_1$, $\alpha_{m+1} \ldots \alpha_{m+u} = l_2$ and $\alpha_{m+u+1} \ldots \alpha_{m+u+v} = l_3$. Let each configuration $q_j = (p_j, d_j, u_j, r_j)$. The corresponding return operation for β is in l_1.

Assume $u_{m+1}(i) = l$. Let us construct a new path $path_4 = q_0 \xrightarrow{\alpha_1}_{te} q_1 \ldots \xrightarrow{\alpha_m}_{te} q_m \xrightarrow{\alpha_{m+1}}_{te} \ldots \xrightarrow{\alpha_{m+u}}_{te} q'_{m+u} \xrightarrow{\beta}_{te} q_{m+u+1} \xrightarrow{\alpha_{m+u+1}}_{te} q_{m+u+2} \ldots \xrightarrow{\alpha_{m+u+v}}_{te} q_{m+u+v+1}$, where for each $1 \leq j \leq u$, $q'_{m+j} = (p_{m+j+1}, d_{m+j+1}, u_{m+j+1}[i : l \cdot \mathit{return}(m, g)], r_{m+j+1})$. It is not hard to see that $path_4$ is a path of $[\![\mathcal{L}, n]\!]_{te}$ and is the path for t_4. □

With above lemmas we can prove the following lemma.

Lemma 11. *If $eh_1 \Rightarrow_{ER} eh_2 \wedge eh_2 \in ehistory([\![\mathcal{L}, n]\!]_{te})$, then $eh_1 \in ehistory([\![\mathcal{L}, n]\!]_{te})$.*

Proof. There exists a trace $t_2 \in trace([\![\mathcal{L}, n]\!]_{te})$ such that $t_2 \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret})} = eh_2$. We will construct a trace t_1 and prove t_1 have the following properties: $t_1 \in trace([\![\mathcal{L}, n]\!]_{te})$ and $t_1 \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret})} = eh_1$.

Assume $t_2 = l_1 \cdot \beta \cdot l_2 \cdot \alpha \cdot l_3$, β is the operation of process i, and $eh_1 \Rightarrow_{ER} eh_2$ by swapping α and β. By definition of $\Rightarrow_{ER}$, there is no $\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret}$ operation in l_2, and it is not the case that $(\alpha \in \Sigma_{ret} \cup \Sigma_{fret}) \wedge (\beta \in \Sigma_{cal} \cup \Sigma_{fcal})$.

- If $\beta \in \Sigma_{cal} \cup \Sigma_{fcal} \cup \Sigma_{ref} \cup \Sigma_{fret}$ and $\alpha \in \Sigma_{cal}$, let γ be the operation that executes *havoc* command to set r_f for α, and either $t_2 = l_1 \cdot \beta \cdot l'_2 \cdot \gamma \cdot l''_2 \cdot \alpha \cdot l_3$ or $t_2 = l'_1 \cdot \gamma \cdot l''_1 \cdot \beta \cdot l_2 \cdot \alpha \cdot l_3$.
  If $t_2 = l_1 \cdot \beta \cdot l'_2 \cdot \gamma \cdot l''_2 \cdot \alpha \cdot l_3$, by Lemma 8, $t_1 = l_1 \cdot \gamma \cdot \alpha \cdot \beta \cdot l'_2 \cdot l''_2 \cdot l_3$ holds as required. If $t_2 = l'_1 \cdot \gamma \cdot l''_1 \cdot \beta \cdot l_2 \cdot \alpha \cdot l_3$, by Lemma 8, $t_1 = l'_1 \cdot l''_1 \cdot \gamma \cdot \alpha \cdot \beta \cdot l_2 \cdot l_3$ holds as required.
- If $\beta \in \Sigma_{cal} \cup \Sigma_{fcal} \cup \Sigma_{ret} \cup \Sigma_{fret}$ and $\alpha \in \Sigma_{fcal}$, by Lemma 10, $t_1 = l_1 \cdot \alpha \cdot \beta \cdot l_2 \cdot l_3$ holds as required.
- If $\beta \in \Sigma_{ret}$ and $\alpha \in \Sigma_{ret} \cup \Sigma_{fret}$, then the sequence of t_2 is of the following three forms: (1)$t_2 = l_1 \cdot \beta \cdot l'_2 \cdot \gamma \cdot l''_2 \cdot \alpha \cdot l_3$ where γ is the next operation of β on process i, (2) $t_2 = l_1 \cdot \beta \cdot l_2 \cdot \alpha \cdot l'_3 \cdot \gamma \cdot l''_3$, and (3) $t_2 = l_1 \cdot \beta \cdot l_2 \cdot \alpha \cdot l_3$ and such γ does not exists.
  (1) If $t_2 = l_1 \cdot \beta \cdot l'_2 \cdot \gamma \cdot l''_2 \cdot \alpha \cdot l_3$, by Lemma 9, $t_1 = l_1 \cdot l'_2 \cdot l''_2 \cdot \alpha \cdot \beta \cdot \gamma \cdot l_3$ holds as required. (2) If $t_2 = l_1 \cdot \beta \cdot l_2 \cdot \alpha \cdot l'_3 \cdot \gamma \cdot l''_3$, by Lemma 9, $t_1 = l_1 \cdot l_2 \cdot \alpha \cdot \beta \cdot \gamma \cdot l'_3 \cdot l''_3$ holds as required. (3) If $t_2 = l_1 \cdot \beta \cdot l_2 \cdot \alpha \cdot l_3$ and such γ does not exists, by Lemma 9, $t_1 = l_1 \cdot l_2 \cdot \alpha \cdot \beta \cdot l_3$ holds as required.
- If $\beta \in \Sigma_{fret}$ and $\alpha \in \Sigma_{ret} \cup \Sigma_{fret}$, then by Lemma 10, $t_1 = l_1 \cdot l_2 \cdot \alpha \cdot \beta \cdot l_3$ holds as required.

Therefore, trace t_1 can be constructed as required for all situations, this completes the proof. □

Now we can prove that extended history inclusion is an equivalent characterization of TSO-to-TSO linearizabilty.

Lemma 1. *For any two libraries $\mathcal{L}_1$ and $\mathcal{L}_2$, $\mathcal{L}_2$ TSO-to-TSO linearizes $\mathcal{L}_1$ if and only if $ehistory([\![\mathcal{L}_1, n]\!]_{te}) \subseteq ehistory([\![\mathcal{L}_2, n]\!]_{te})$.*

Proof. The *if* direction is obvious, the *only if* direction can be easily proved by Lemma 7 and Lemma 11. □

A.2 Proof of Proposition 1

Similarly as Appendix A.1 we can prove that history inclusion is an equivalent characterization ofstandard linearizability on TSO and SC memory models. Let relation R = $(\Sigma_{ret} \times \Sigma_{cal}) \cup (\Sigma_{e1} \times \Sigma_{e1}) \cup \ldots \cup (\Sigma_{en} \times \Sigma_{en})$. A transformation $\Rightarrow_R$ is a relation between two histories and it is defined as follows: $h_1 \Rightarrow_R h_2$, if h_1 = $l_1 \cdot \alpha \cdot \beta \cdot l_2$, h_2 = $l_1 \cdot \beta \cdot \alpha \cdot l_2$ and $(\alpha, \beta) \notin R$. We write $\Rightarrow_R^*$ to denote the transition closure of $\Rightarrow_R$. Given two equivalent histories h_1 and h_2, we say that they correspond by π, if π is a bijection between $\{1, \ldots, |h_1|\}$ and $\{1, \ldots, |h_2|\}$, moreover for all i, j, k, if $j < k \wedge h_1(j), h_1(k) \in \Sigma_{ei}$, then $\pi(i) < \pi(j)$. It is not hard to see that given two equivalent histories h_1 and h_2, there exists only one such bijection. Given two equivalent histories h_1 and h_2 and assume h_1 and h_2 correspond by π, predicate *wit*(h_1, h_2, i_1, i_2) holds if $i_1 < i_2$ and $\pi^{-1}(i_1) > \pi^{-1}(i_2)$. Given two equivalent histories h_1 and h_2, *witSum*(h_1, h_2) = $|\{(m, n)|$*wit*$(h_1, h_2, m, n)\ holds\}|$ is a distance function which is used to measure difference between h_1 and h_2.

The following lemmas are similar to those in Appendix A.1. They can be similarly proved and we omit their proofs here.

Lemma 12. *Assume two equivalent histories h_1 and h_2 correspond by π. If witSum*$(h_1, h_2) > 0$*, then there exists positive number $m \in \{1, \ldots, |eh_2|\}, \pi^{-1}(m) > \pi^{-1}(m+1)$.*

Lemma 13. *If a history h_1 is linearizable to a history h_2 and $h_1 \neq h_2$, then there exists a history h_3 such that: (1) $h_3 \Rightarrow_R h_2$, (2) h_1 is linearizable to h_3, (3) h_3 is linearizable to h_2, (4) witSum$(h_1, h_2) >$ witSum(h_1, h_3).*

Lemma 14. *If h_1 is linearizable to h_2, then $h_1 \Rightarrow_{ER}^* h_2$.*

Lemma 15. *Given $x \in \{te, sc\}$. If $h_1 = l_1 \cdot l_2 \cdot \alpha \cdot l_3 \in$ history$(\llbracket \mathcal{L}, n \rrbracket_x)$, $\alpha \in \Sigma_{cal} \cap \Sigma_{ei}$ and there is no $(\Sigma_{cal} \cup \Sigma_{ret}) \cap \Sigma_{ei}$ operation in l_2, then $h_2 = l_1 \cdot \alpha \cdot l_2 \cdot l_3 \in$ history$(\llbracket \mathcal{L}, n \rrbracket_x)$. If $h_3 = l_1 \cdot \beta \cdot l_2 \cdot l_3 \in$ history$(\llbracket \mathcal{L}, n \rrbracket_x)$, $\beta \in \Sigma_{ret} \cap \Sigma_{ei}$ and there is no $(\Sigma_{cal} \cup \Sigma_{ret}) \cap \Sigma_{ei}$ operation in l_2, then $h_4 = l_1 \cdot l_2 \cdot \beta \cdot l_3 \in$ history$(\llbracket \mathcal{L}, n \rrbracket_x)$.*

Lemma 16. *Given $x \in \{te, sc\}$. $h_1 \Rightarrow_R h_2 \wedge h_2 \in$ history$(\llbracket \mathcal{L}, n \rrbracket_x)$, then $h_1 \in$ history$(\llbracket \mathcal{L}, n \rrbracket_x)$.*

Proposition 1. *For any two libraries $\mathcal{L}_1$ and $\mathcal{L}_2$, $\mathcal{L}_1$ is linearizable with respect to $\mathcal{L}_2$ on TSO (SC) if and only if history$(\llbracket \mathcal{L}_1, n \rrbracket_{te}) \subseteq$ history$(\llbracket \mathcal{L}_2, n \rrbracket_{te})$ (history$(\llbracket \mathcal{L}_1, n \rrbracket_{sc}) \subseteq$ history$(\llbracket \mathcal{L}_2, n \rrbracket_{sc})$).*

Proof. The *if* direction is obvious, the *only if* direction can be easily proved by Lemma 14 and Lemma 16. □

B Proofs In Section 4

Notation. Given a path $p = q_1 \xrightarrow{\alpha_1} q_2 \ldots \xrightarrow{\alpha_k} q_{k+1}$, we use *pathtoTrace*$(p)$ to denote the sequence $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_k$.

B.1 Proof of Lemma 2

To prove Lemma 2, we present the following two lemmas and prove them in Appendix B.2 and B.3 respectively.

Lemma 17. *Given a classic-lossy single-channel system $\mathcal{S}$ and two configurations (q_1, W_1), $(q_1, W_2) \in$ Conf$_{cs}$, if history$(\llbracket \mathcal{L}_{\mathcal{S},q_1,W_1}, 3 \rrbracket_{te}) \subseteq$ history$(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_{te})$, then trace$(\mathcal{CL}(\mathcal{S}), (q_1, W_1)) \subseteq$ trace$(\mathcal{CL}(\mathcal{S}), (q_2, W_2))$.*

Lemma 18. *Given a classic-lossy single-channel system $\mathcal{S}$ and two configurations (q_1, W_1), $(q_1, W_2) \in$ Conf$_{cs}$, if trace$(\mathcal{CL}(\mathcal{S}), (q_1, W_1)) \subseteq$ trace$(\mathcal{CL}(\mathcal{S}), (q_2, W_2))$, then history$(\llbracket \mathcal{L}_{\mathcal{S},q_1,W_1}, 3 \rrbracket_{te}) \subseteq$ history$(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_{te})$.*

Now we can prove that history inclusion is undecidable for a bounded number of processes on TSO.

Lemma 2. *For any two libraries $\mathcal{L}_1$ and $\mathcal{L}_2$, it is undecidable whether history$(\llbracket \mathcal{L}_1, 3 \rrbracket_{te}) \subseteq$ history$(\llbracket \mathcal{L}_2, 3 \rrbracket_{te})$.*

Proof. Given a classic-lossy single-channel system $\mathcal{S}$ and two configurations (q_1, W_1), $(q_1, W_2) \in$ Conf$_{cs}$, by Lemma 17 and Lemma 18 the trace inclusion problem between (q_1, W_1) and (q_2, W_2) can be reduced to whether history$(\llbracket \mathcal{L}_{\mathcal{S},q_1,W_1}, 3 \rrbracket_{te}) \subseteq$ history$(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_{te})$. According to [16] trace inclusion between configurations of $\mathcal{CL}(\mathcal{S})$ is undecidable. □

B.2 Proof of Lemma 17

The definition of effective trace is recalled here. We say a trace $t \in$ trace$(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$ is *effective*, if t contains at least one return$(_, M_3, _)$ operation. Otherwise, we say t is *ineffective*. Given an effective trace $t \in$ trace$(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$, we say that $\langle p_1, p_2, p_3 \rangle$ is the *process indexes* of t, if $\exists i_1, i_2, i_3$, such that (1) $t(i_1) =$ call$(p_1, M_1, _)$, (2) $t(i_2) =$ call$(p_2, M_2, _)$, (3) $t(i_3) =$ return$(p_3, M_3, _)$, and (4) $i_1 < i_3 \wedge i_2 < i_3$. Intuitively $\langle p_1, p_2, p_3 \rangle$ is the process indexes of t, if the working M_1 and M_2 run on process p_1 and process p_2 respectively, and M_3 repeatedly runs on process p_3. We say that an effective trace $t \in$ trace$(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$ has k rounds, if there are k return$(_, M_3, _)$ operations on t.

Given an effective trace $t \in$ trace$(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$ with process indexes $\langle p_1, p_2, p_3 \rangle$ and k rounds, we say that *j-th* ($j \leq k$) round of M_2 starts at $t(i_1)$ and ends at $t(i_2)$, if the operation of $t(i_1)$ starts to execute Line 1 or Line 4 in M_2 of process p_2 for the *j-th* time, and the operation of $t(i_2)$ ends the execution of Line 2 or Line 13 in M_2 of process p_2 for the *j-th* time.

The following lemma states that "the sets of ineffective histories of each $\mathcal{L}_{\mathcal{S},q,W}$ are the same".

Lemma 19. *Given a classic-lossy single-channel system $\mathcal{S}$ and two configurations $(q_1, W_1), (q_2, W_2) \in$ Conf$_{cs}$, $\{h | h \in$ history$(\llbracket \mathcal{L}_{\mathcal{S},q_1,W_1}, 3 \rrbracket_{te}), h$ is ineffective$\}$ is equal to $\{h | h \in$ history$(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_{te}), h$ is ineffective$\}$.*

Proof. Recall that M_1 and M_2 never return, and an ineffective history does not contain any return operation of M_3. Therefore, an ineffective history h contains at most three operations and all operations in it are call operations. The number of candidate for ineffective histories is finite, and for each configuration $(q, W) \in \mathit{Conf}_{cs}$ it is not hard to see that each candidate ineffective history belongs to $\mathit{history}(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$. □

We reproduce the definition of conservative here. Given a path $p = (q_1, W_1) \xrightarrow{\alpha_1}_{cs} (q_2, W_2) \ldots \xrightarrow{\alpha_k}_{cs} (q_{k+1}, W_{k+1}) \in \mathit{path}(\mathcal{CL}(\mathcal{S}), (q_1, W_1))$, we say that p is *conservative*, if the following conditions hold: (1) it contains at least one transition, (2) assume *i-th* transition uses rule $r_i = (q_i, U_i, \alpha_i, q_{i+1}, V_i)$, then for each i, there exists $W_i', W_i'' \in \Sigma_{cs}^*$, such that $U_i \cdot W_i' \sqsubseteq W_i$, $W_i'' \sqsubseteq W_i'$ and $W_{i+1} = W_i'' \cdot V_i$. Intuitively, each step of a conservative path does not lose any element in V_i.

The following lemma states that the concept conservative does not restrict traces of $\mathcal{CL}(\mathcal{S})$ substantially.

Lemma 20. *$\forall p \in \mathit{path}(\mathcal{CL}(\mathcal{S}), (q_1, W_1))$ with at least one transition, there exists a conservative path $p' \in \mathit{path}(\mathcal{CL}(\mathcal{S}), (q_1, W_1))$, such that $\mathit{pathtoTrace}(p) = \mathit{pathtoTrace}(p')$.*

Proof. We generate p' from p step by step. Intuitively, assume the *i-th* transition of p uses rule $r_i = (q_i, U_i, \alpha_i, q_{i+1}, V_i)$ and some element in V_i is lost in *i-th* transition of p, then such element is kept in *i-th* transition of p' and will be lost in the *i+1-th* transition of p'.

Assume path $p = (q_1, W_1) \xrightarrow{\alpha_1}_{cs} (q_2, W_2) \ldots \xrightarrow{\alpha_k}_{cs} (q_{k+1}, W_{k+1})$, for each $1 \leq i \leq k$, *i-th* transition uses rule $r_i = (q_i, U_i, \alpha_i, q_{i+1}, V_i)$, $\exists W_i'$, such that $U_i \cdot W_i' \sqsubseteq W_i$ and $W_{i+1} \sqsubseteq W_i' \cdot V_i$. By definition of $\rightarrow_{cs}$, for each $1 \leq i \leq k$, there exist W_i'' and V_i', such that $W_{i+1} = W_i'' \cdot V_i'$, $W_i'' \sqsubseteq W_i'$ and $V_i' \sqsubseteq V_i$.

Let path $p' = (q_1, NW_1) \xrightarrow{\alpha_1}_{cs} (q_2, NW_2) \ldots \xrightarrow{\alpha_k}_{cs} (q_{k+1}, NW_{k+1})$, where $NW_1 = W_1$, and for each $1 < i \leq k$, $NW_{i+1} = W_i'' \cdot V_i$.

It is not hard to see that for each $1 \leq i \leq k$, $W_{i+1} = W_i'' \cdot V_i' \sqsubseteq NW_{i+1}$. So for each $1 \leq i \leq k$, we can prove that $U_i \cdot W_i' \sqsubseteq W_i \sqsubseteq NW_i$ and $NW_{i+1} = W_i'' \cdot V_i \sqsubseteq W_i' \cdot V_i$. By definition of $\rightarrow_{cs}$, $p' \in \mathit{path}(\mathcal{CL}(\mathcal{S}), (q_1, W_1))$, and $\mathit{pathtoTrace}(p) = \mathit{pathtoTrace}(p')$ holds trivially. □

Given an effective trace $t \in \mathit{trace}(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$ and a conservative path $p \in \mathit{path}(\mathcal{CL}(\mathcal{S}), (q, W))$, we say t and p correspond, if the sequence of return values of M_3 in t is the same as the sequence of transition labels of p.

The following lemma states the claim: for each conservative path of $\mathcal{CL}(\mathcal{S})$ that starts from (q, W), there exists an effective trace of $\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te}$ that corresponds to the path. This lemma is proved by constructing a trace in $\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te}$ step by step.

Lemma 21. *Given a conservative path $p_{\mathcal{S}} \in \mathit{path}(\mathcal{CL}(\mathcal{S}), (q, W))$, there exists an effective trace $t_{\mathcal{L}} \in \mathit{trace}(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$ such that t and p correspond.*

Proof. Assume $p_{\mathcal{S}} = (q_1, W_1) \xrightarrow{\alpha_1}_{cs} (q_2, W_2) \xrightarrow{\alpha_2}_{cs} \ldots \xrightarrow{\alpha_k}_{cs} (q_{k+1}, W_{k+1})$, where (1) $(q_1, W_1) = (q, W)$, (2) for each i, the *i-th* transition uses rule $r_i = (q_i, U_i, \alpha_i, q_{i+1}, V_i)$, (3) for each i, $\exists W_i', W_i''$, such that $U_i \cdot W_i' \sqsubseteq W_i$, $W_i'' \sqsubseteq W_i'$ and $W_{i+1} = W_i'' \cdot V_i$.

Let us sketch how to construct a path $p_{\mathcal{L}} \in \mathit{path}([\![\mathcal{L}_{\mathcal{S},q,W}, 3]\!]_{te})$ that starts at $\mathit{InitConf}_{te}$ and simulates k transitions on $p_{\mathcal{S}}$. Thus $\mathit{pathtoTrace}(p_{\mathcal{L}})$ and $p_{\mathcal{S}}$ correspond.

From $\mathit{InitConf}_{te}$, we need to perform the following operations in sequence:

- Call M_1 in process 1 and call M_2 in process 2.
- Run M_2 until M_2 finishes its first round. M_2 write $r_1 \cdot \mathit{start} \cdot W_1 \cdot \mathit{end}$ to x and no flush operation of process 2 happens during this period.

To simulate the *i-th* ($1 \leq i \leq k$) transition of $p_{\mathcal{S}}$, we need to perform the following operations in sequence:

- Call M_3 in process 3. No flush operation happens during this period.
- Run M_2 until M_2 finishes its *i+1-th* round. M_2 reads $r_i \cdot \mathit{start} \cdot U_i \cdot W_i'' \cdot \mathit{end}$ from y and writes $r_{i+1} \cdot \mathit{start} \cdot W_i'' \cdot V_i \cdot \mathit{end}$ to x (in the case of $i = k$, M_2 write $\mathit{rule}_f \cdot \mathit{start} \cdot W_k'' \cdot V_k \cdot \mathit{end}$ instead). Then M_2 transmits transition label α_i to M_3 and M_3 returns α_i. Since W_{i+1} is equal to $W_i'' \cdot V_i$, M_2 writes W_{i+1} to x while it simulates the *i-th* transition of $p_{\mathcal{S}}$.

It is obvious that $p_{\mathcal{L}}$ holds as required. This completes the proof of Lemma 21. □

The following lemma states that for each effective trace t of $[\![\mathcal{L}_{\mathcal{S},q,W}, 3]\!]_{te}$ there exists a conservative path p of $\mathcal{CL}(\mathcal{S})$ such that t and p correspond.

Lemma 22. *For each effective trace $t_{\mathcal{L}} \in \mathit{trace}([\![\mathcal{L}_{\mathcal{S},q,W}, 3]\!]_{te})$, there exists a conservative path $p_{\mathcal{S}} \in \mathit{path}(\mathcal{CL}(\mathcal{S}), (q, W))$ such that $t_{\mathcal{L}}$ and $p_{\mathcal{S}}$ correspond.*

Proof. There exists a path $p_{\mathcal{L}} \in \mathit{path}([\![\mathcal{L}_{\mathcal{S},q,W}, 3]\!]_{te})$ such that $t_{\mathcal{L}} = \mathit{pathtoTrace}(p_{\mathcal{L}})$. It is easy to see that the sequence of values of x which is read by M_1 in its *i-th* round is a subword of the the sequence of values of x which is written by M_2 in its *i-th* round. And the sequence of values of y which is read by M_2 in its *i+1-th* round is a subword of the the sequence of values of y which is written by M_1 in its *i-th* round.

Assume $t_{\mathcal{L}}$ has k rounds, and for each $1 \leq i \leq k$, M_2 guesses rule $r_i = (q_i, U_i, \alpha_i, q_{i+1}, V_i)$ in its *i-th* round. M_2 writes $r_1 \cdot \mathit{start} \cdot W \cdot \mathit{end}$ to x during its first round. Assume for each $1 < i \leq k$, M_2 reads $r_i \cdot \mathit{start} \cdot U_i \cdot L_i' \cdot \mathit{end}$ from y during its *i+1-th* round and writes $r_i \cdot \mathit{start} \cdot L_i' \cdot V_i \cdot \mathit{end}$ to x during its *i+1-th* round.

Let path $p_{\mathcal{S}} = (q, W) \xrightarrow{\alpha_1}_{cs} (q_2, L_1' \cdot V_1) \xrightarrow{\alpha_2}_{cs} \ldots \xrightarrow{\alpha_k}_{cs} (q_{k+1}, L_k' \cdot V_k)$. It is not difficult to see that $p_{\mathcal{S}} \in \mathit{path}(\mathcal{CL}(\mathcal{S}), (q, W))$ and $p_{\mathcal{S}}$ is conservative. Therefore, $t_{\mathcal{L}}$ and $p_{\mathcal{S}}$ correspond. □

With Lemma 20, Lemma 21 and Lemma 22, we can now prove Lemma 17 as follows.

Lemma 17. *Given a classic-lossy single-channel system $\mathcal{S}$ and two configurations (q_1, W_1), $(q_1, W_2) \in \mathit{Conf}_{cs}$, if $\mathit{history}([\![\mathcal{L}_{\mathcal{S},q_1,W_1}, 3]\!]_{te}) \subseteq \mathit{history}([\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te})$, then $\mathit{trace}(\mathcal{CL}(\mathcal{S}), (q_1, W_1)) \subseteq \mathit{trace}(\mathcal{CL}(\mathcal{S}), (q_2, W_2))$.*

Proof. By contradiction. Assume *history*($[\![\mathcal{L}_{\mathcal{S},q_1,W_1}, 3]\!]_{te}$) $\subseteq$ *history*($[\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te}$) but *trace*($\mathcal{CL}(\mathcal{S}), (q_1, W_1)$) is not a subset of *trace*($\mathcal{CL}(\mathcal{S}), (q_2, W_2)$). Thus there must exists a trace $t_{\mathcal{S}1}$, such that $t_{\mathcal{S}1} \in$ *trace*($\mathcal{CL}(\mathcal{S}), (q_1, W_1)$) and $t_{\mathcal{S}1} \notin$ *trace*($\mathcal{CL}(\mathcal{S}), (q_2, W_2)$). It is clear that $t_{\mathcal{S}1} \neq \epsilon$.

Let $p_{\mathcal{S}1} \in$ *path*($\mathcal{CL}(\mathcal{S}), (q_1, W_1)$) such that *pathtoTrace*($p_{\mathcal{S}1}$) $= t_{\mathcal{S}1}$. According to Lemma 20 we can safely assume that $p_{\mathcal{S}1}$ is conservative. According to Lemma 21 there exists an effective trace $t_{\mathcal{L}1} \in$ *trace*($[\![\mathcal{L}_{\mathcal{S},q_1,W_1}, 3]\!]_{te}$), such that $t_{\mathcal{L}1}$ and $p_{\mathcal{S}1}$ correspond.

Let history $h = t_{\mathcal{L}1} \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}$. It is obvious that $h \in$ *history*($[\![\mathcal{L}_{\mathcal{S},q_1,W_1}, 3]\!]_{te}$) and by assumption $h \in$ *history*($[\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te}$).

There exists a trace $t_{\mathcal{L}2} \in$ *trace*($[\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te}$) such that $h = t_{\mathcal{L}2} \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}$. It is obvious that $t_{\mathcal{L}2}$ is effective. According to Lemma 22, there exists a conservative path $p_{\mathcal{S}2} \in$ *path*($\mathcal{CL}(\mathcal{S}), (q_2, W_2)$) such that $t_{\mathcal{L}2}$ and $p_{\mathcal{S}2}$ correspond. Let trace $t_{\mathcal{S}2} =$ *pathtoTrace*($p_{\mathcal{S}2}$). Thus $t_{\mathcal{S}2} \in$ *trace*($\mathcal{CL}(\mathcal{S}), (q_2, W_2)$) by its definition. Because the sequence of return values of M_3 in $t_{\mathcal{L}1}$ is same to that in $t_{\mathcal{L}2}$, $t_{\mathcal{L}1}$ and $p_{\mathcal{S}1}$ correspond, and $t_{\mathcal{L}2}$ and $p_{\mathcal{S}2}$ correspond, we can obtain that $t_{\mathcal{S}1} = t_{\mathcal{S}2}$ and $t_{\mathcal{S}1} \in$ *trace*($\mathcal{CL}(\mathcal{S}), (q_2, W_2)$), which contradicts our assumption. □

B.3 Proof of Lemma 18

Given a operation $\alpha \in \Sigma_{te}$, we say another operation $\beta \in \Sigma_{te}$ is generated from α by changing process id to j, if:

- If $\alpha = \tau(i)$, then $\beta = \tau(j)$.
- For each x, a, if $\alpha =$ *read*(i, x, a), then $\beta =$ *read*(j, x, a).
- For each x, a, if $\alpha =$ *write*(i, x, a), then $\beta =$ *write*(j, x, a).
- For each x, a, b, if $\alpha =$ *cas*(i, x, a, b), then $\beta =$ *cas*(j, x, a, b).
- For each x, a, if $\alpha =$ *flush*(i, x, a), then $\beta =$ *flush*(j, x, a).
- For each m, a, if $\alpha =$ *call*(i, m, a), then $\beta =$ *call*(j, m, a).
- For each m, a, if $\alpha =$ *return*(i, m, a), then $\beta =$ *return*(j, m, a).
- For each m, a, if $\alpha =$ *flushCall*(i, m, a), then $\beta =$ *flushCall*(j, m, a).
- For each m, a, if $\alpha =$ *flushReturn*(i, m, a), then $\beta =$ *flushReturn*(j, m, a).

Given a trace $t = \alpha_1 \dots \alpha_k$, we write $t[i/j]$ for another trace t' which satisfies the following conditions:

- $t' = \alpha'_1 \dots \alpha'_k$.
- $\forall 1 \leq ind \leq k$, if process id of α_{ind} is not j, then $\alpha'_{ind} = \alpha_{ind}$.
- $\forall 1 \leq ind \leq k$, if process id of α_{ind} is j, then α'_{ind} is generated from α_{ind} by changing process id to i.

Given a trace $t = \alpha_1 \dots \alpha_k$, we write $t[i_1/j_1, i_2/j_2, i_3/j_3]$ for another trace t' which satisfies the following conditions:

- $t' = \alpha'_1 \dots \alpha'_k$.
- $\forall 1 \leq ind \leq k$, if process id of α_{ind} is j_1, then α'_{ind} is generated from α_{ind} by changing process id to i_1.

- $\forall 1 \leq ind \leq k$, if process id of α_{ind} is j_2, then α'_{ind} is generated from α_{ind} by changing process id to i_2.
- $\forall 1 \leq ind \leq k$, if process id of α_{ind} is j_3, then α'_{ind} is generated from α_{ind} by changing process id to i_3.
- Otherwise, $\alpha'_{ind} = \alpha_{ind}$.

A *return*(i, m, a) operation matches a call operation *call*(j, n, b) if $i = j \wedge m = n$. Given a history h, a call operation $h(i)$ is a pending if $h(j)$ does not match $h(i)$ for each $j > i$.

Lemma 23. *Assume t_1 is an effective trace of $[\![\mathcal{L}_{\mathcal{S},q_1,W_1}, 3]\!]_{te}$ with process indexes $\langle i_1, i_2, i_3\rangle$, t_2 is an effective trace of $[\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te}$ with process indexes $\langle j_1, j_2, j_3\rangle$, and $(t_1 \uparrow_{(\Sigma_{ei_3} \cap \Sigma_{ret})})[j_3/i_3] = t_2 \uparrow_{(\Sigma_{ej_3} \cap \Sigma_{ret})}$. Thus there exists an effective trace t_3 of $[\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te}$, such that $t_3 \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})} = t_1 \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}$.*

Proof. Let history $h_2 = t_2 \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}$. An sequence h'_2 can be generated from h_2 as follows:

- Transform the process id of each operation from j_1 (j_2, j_3) to i_1 (i_2, i_3).
- Remove the pending call operation of process i_3 if it exists.
- If there is a pending call operation of process i_3 in t_1, then append this operation to the tail.

It is obvious that h'_2 is a history of $[\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te}$ and $h'_2 \uparrow_{\Sigma_{ei_k}} = t_1 \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret}) \cap \Sigma_{ei_k}}$ for $1 \leq k \leq 3$. The first three call and return operations of h'_2 must be *call*$(i_1, M_1, \_)$, *call*$(i_2, M_2, \_)$ and *call*$(i_3, M_3, \_)$ operations, and process i_1 and i_2 does not have return operations. Let trace t'_2 be the trace of $[\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te}$ where $h'_2 = t'_2 \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}$. By Lemma 8 we can move positions of these three call operations and generate a trace t_3 from t'_2, where t_3 is a trace of $[\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te}$ and $t_3 \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})} = t_1 \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}$.

□

With above lemmas we can prove Lemma 18.

Lemma 18. *Given a classic-lossy single-channel system $\mathcal{S}$ and two configurations (q_1, W_1), $(q_1, W_2) \in$ Conf$_{cs}$, if trace$(\mathcal{CL}(\mathcal{S}), (q_1, W_1)) \subseteq$ trace$(\mathcal{CL}(\mathcal{S}), (q_2, W_2))$, then history$([\![\mathcal{L}_{\mathcal{S},q_1,W_1}, 3]\!]_{te}) \subseteq$ history$([\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te})$.*

Proof. We prove it by contradiction. Assume *trace*$(\mathcal{CL}(\mathcal{S}), (q_1, W_1)) \subseteq$ *trace*$(\mathcal{CL}(\mathcal{S}), (q_2, W_2))$ but *history*$([\![\mathcal{L}_{\mathcal{S},q_1,W_1}, 3]\!]_{te})$ is not a subset of *history*$([\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te})$. Then there must exists a history $h_{\mathcal{L}1}$, such that $h_{\mathcal{L}1} \in$ *history*$(([\![\mathcal{L}_{\mathcal{S},q_1,W_1}, 3]\!]_{te})$ and $h_{\mathcal{L}1} \notin$ *history*$([\![\mathcal{L}_{\mathcal{S},q_2,W_2}, 3]\!]_{te})$.

According to Lemma 19 there exists an effective trace $t_{\mathcal{L}1} \in$ *trace*$([\![\mathcal{L}_{\mathcal{S},q_1,W_1}, 3]\!]_{te})$ such that $t_{\mathcal{L}1} \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})} = h_{\mathcal{L}1}$. According to Lemma 22, there is a conservative path $p_{\mathcal{S}1} \in$ *path*$(\mathcal{CL}(\mathcal{S}), (q_1, W_1))$ such that $t_{\mathcal{L}1}$ and $p_{\mathcal{S}1}$ correspond. We can obtain trace $t_{\mathcal{S}} =$ *pathtoTrace* $(p_{\mathcal{S}1})$. By assumption $t_{\mathcal{S}} \in$ *trace*$(\mathcal{CL}(\mathcal{S}), (q_2, W_2))$.

There exists a path $p_{\mathcal{S}2} \in$ *path*$(\mathcal{CL}(\mathcal{S}), (q_2, W_2))$ such that $t_{\mathcal{S}} =$ *pathtoTrace*$(p_{\mathcal{S}2})$. By Lemma 20 we can assume that $p_{\mathcal{S}2}$ is conservative. By Lemma 21, there exists an

effective trace $t_{\mathcal{L}2} \in \mathit{trace}(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_{te})$, such that $t_{\mathcal{L}2}$ and $p_{\mathcal{S}2}$ correspond. It is obvious that the return values of M_3 of $t_{\mathcal{L}1}$ and $t_{\mathcal{L}2}$ are equal. Therefore, by Lemma 23 there is a trace $t'_{\mathcal{L}2} \in \mathit{trace}(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_{te})$, such that $t_{\mathcal{L}1} \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})} = t'_{\mathcal{L}2} \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}$. Recall that $h_{\mathcal{L}1} = t_{\mathcal{L}1} \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}$, thus $h_{\mathcal{L}1} \in \mathit{history}(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_{te})$, which contradicts our assumption. □

B.4 Proof of Lemma 3

Lemma 3. *For a classic-lossy single-channel system $\mathcal{S}$ and two configurations (q_1, W_1), $(q_2, W_2) \in \mathit{Conf}_{cs}$, if $eh_1 \in \mathit{ehistory}(\llbracket \mathcal{L}_{\mathcal{S},q_1,W_1}, 3 \rrbracket_{te})$ is an effective extended history and $eh_1 \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})} \in \mathit{history}(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_{te})$, then $eh_1 \in \mathit{ehistory}(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_{te})$.*

Proof. Given an effective trace $eh_1 \in \mathit{ehistory}(\llbracket \mathcal{L}_{\mathcal{S},q_1,W_1}, 3 \rrbracket_{te})$ with processes indexes $\langle i_1, i_2, i_3 \rangle$, it is not hard to see the following observations:

- The first six operations of eh_1 must be $\mathit{call}(\_, M_1, \_)$, $\mathit{call}(\_, M_2, \_)$, $\mathit{call}(\_, M_3, \_)$, and three corresponding flush call operations. The can be in any arbitrary orders.
- Since M_1 and M_2 never return, after the first six operations, the remaining operations of eh_1 come from process i_3.
- Since M_3 always uses a *cas* operation before it returns, each flush call operation of M_3 can occur before return operation of M_3. Because M_3 does not insert any pending write operation into the process i_3's store buffer, each flush return operation may occur alternatively at two positions: the first position is after the return operation of M_3 and before the next round of a call operation of M_3, and the second one is after the next round of a call operation of M_3 and before the consequent flush call operation.

Given an effective trace $eh_1 \in \mathit{ehistory}(\llbracket \mathcal{L}_{\mathcal{S},q_1,W_1}, 3 \rrbracket_{te})$, there exists an effective trace $t_{\mathcal{L}1} \in \mathit{trace}(\llbracket \mathcal{L}_{\mathcal{S},q_1,W_1}, 3 \rrbracket_{te})$ such that $eh_1 = t_{\mathcal{L}1} \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal} \cup \Sigma_{fret})}$ and its process indexes is $\langle i_1, i_2, i_3 \rangle$. By Lemma 22 there exists a conservative path $p_{\mathcal{S}}$ such that $t_{\mathcal{L}1}$ and $p_{\mathcal{S}}$ correspond. Let $p_{\mathcal{S}} = (\hat{q}_1, \hat{W}_1) \xrightarrow{\alpha_1}_{cs} (\hat{q}_2, \hat{W}_2) \xrightarrow{\alpha_2}_{cs} \ldots \xrightarrow{\alpha_k}_{cs} (\hat{q}_{k+1}, \hat{W}_{k+1})$, where (1) $(\hat{q}_1, \hat{W}_1) = (q_1, W_1)$, (2) for each i, the *i-th* transition uses rule $r_i = (q_i, U_i, \alpha_i, q_{i+1}, V_i)$, and (3) for each t, there exist $\hat{W}'_i, \hat{W}''_i$, such that $U_i \cdot \hat{W}'_i \sqsubseteq \hat{W}_i$, $\hat{W}''_i \sqsubseteq \hat{W}'_i$ and $\hat{W}_{i+1} = \hat{W}''_i \cdot V_i$.

Let us sketch how to construct a path $p_{\mathcal{L}_2} \in \mathit{path}(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_t)$ such that $p_{\mathcal{L}_2}$ starts at $\mathit{InitConf}_{te}$, and $\mathit{pathtoTrace}(p_{\mathcal{L}_2}) \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal}) \cup \Sigma_{fret}} = eh_1$.

From $\mathit{InitConf}_{te}$, we perform the following operations in sequence:

- Perform the first six operations of eh_1 and make this order the same as in eh_1.
- Run M_2 until M_2 finishes its first round and puts $r_1 \cdot \mathit{start} \cdot WW_1 \cdot \mathit{end}$ into its buffer. No flush operation happens during this period.
- Then we mimic the k transitions similarly as Lemma 21. We additionally arrange the positions of operations after the sixth operation in eh_1 the same between eh_1 and $\mathit{pathtoTrace}(p_{\mathcal{L}_2}) \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret} \cup \Sigma_{fcal}) \cup \Sigma_{fret}}$.

It is not hard to see that $p_{\mathcal{L}_2}$ holds as required. This completes the proof. □

B.5 Proof of Lemma 24

The following lemma states that "the sets of ineffective extended histories of each $\mathcal{L}_{\mathcal{S},q,W}$ are the same".

Lemma 24. *Given a classic-lossy single-channel system $\mathcal{S}$, and two configurations $(q_1, W_1), (q_2, W_2) \in$ Conf$_{cs}$, $\{eh | eh \in$ ehistory$(\llbracket \mathcal{L}_{\mathcal{S},q_1,W_1}, 3 \rrbracket_{te})$, eh is ineffective $\}$ is equal to $\{eh | eh \in$ ehistory$(\llbracket \mathcal{L}_{\mathcal{S},q_2,W_2}, 3 \rrbracket_{te})$, eh is ineffective $\}$.*

Proof. Recall that M_1 and M_2 never return, and an ineffective extended history does not contain return operation of M_3. Therefore, an ineffective extended history *eh* contains at most six operations, and all of them are call or flush call operations. The number of candidate for ineffective extended histories is finite, and for each configuration $(q, W) \in$ Conf$_{cs}$ it is not hard to see that each candidate of extended history belongs to ehistory$(\llbracket \mathcal{L}_{\mathcal{S},q,W}, 3 \rrbracket_{te})$. □