ISCAS-SKLCS-15-11 July, 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

Bounded TSO-to-SC Linearizability is Decidable

by

Chao Wang, Yi Lv, and Peng Wu

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

Bounded TSO-to-SC Linearizability is Decidable

Chao Wang, Yi Lv, and Peng Wu

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

Abstract. TSO-to-SC linearizability is a variant of linearizability for concurrent libraries on the Total Store Order (TSO) memory model. In this paper we propose the notion of k-bounded TSO-to-SC linearizability, a subclass of TSO-to-SC linearizability that concerns only bounded histories. This subclass is non-trivial in that it does not restrict the number of write, flush and *cas* (compare-and-swap) actions, nor the size of a store buffer, to be bounded. We prove that the decision problem of k-bounded TSO-to-SC linearizability is decidable for a bounded number of processes. We first reduce this decision problem to a marked violation problem of k-bounded TSO-to-SC linearizability, where specific *cas* actions are introduced to mark call and return actions. Then, we further reduce the marked violation problem to a control state reachability problem of a lossy channel machine, which is already known to be decidable. Moreover, we prove that the decision problem of k-bounded TSO-to-SC linearizability has non-primitive recursive complexity.

1 Introduction

Linearizability [9] has been accepted as a *de facto* correctness condition for a concurrent library with respect to its sequential specification on the sequential consistency (SC) memory model [10]. However, modern multiprocessors (e.g., x86 [12], POWER [13]) and programming languages (e.g., Java [11], C11/C++11 [3]) do not comply with the SC memory model. Instead, they provide *relaxed memory models* that allow non-SC behaviors due to hardware or compiler optimization. For instance, in a multiprocessor system implementing the TSO memory model [12], each processor is equipped with a FIFO store buffer. Any written action performed by a processor will append an item into its store buffer before the item is eventually flushed into the memory. The TSO memory model requires that all processes in a concurrent system observe the same order of write and *cas* actions, which is referred to as a total store order.

Accordingly, linearizability has been extended for relaxed memory models, e.g., *TSO-to-TSO linearizability* [7] and *TSO-to-SC linearizability* [8] for the TSO memory model and two variants of linearizability [3] for the C++ memory model. TSO-to-SC linearizability has been proposed for reasoning about the correctness of a concurrent library, which is native to the TSO memory model but is used with a concurrent program that needs to be protected from the relaxed semantics [8].

It is well known that the linearizability of a concurrent library on the SC memory model is decidable for a bounded number of processes [1], but undecidable for an unbounded number of processes [4]. However, to our knowledge, there are only a few decidability results about linearizability on relaxed memory models. We have recently proved that the decision problem of *TSO-to-TSO linearizability* is undecidable for a bounded number of processes [15,16]. But the decision problem of TSO-to-SC linearizability still remains open for a bounded number of processes.

We propose a decidable subclass of TSO-to-SC linearizability for a bounded number of processes, which is referred to as *k-bounded TSO-to-SC linearizability*. It concerns only *k-traces*, which are traces with at most k call and return actions, and hence it defined over k-bounded histories of TSO libraries. Note that k-traces may still contain arbitrarily many write, flush and *cas* actions, and store buffers may still contain arbitrarily many items along k-traces. Hence, the k-boundedness on the number of call and return actions does not necessarily restrict the behaviors of a concurrent program to be finite-state. As we prove in this paper, the decision problem of this non-trivial subclass of TSO-to-SC linearizability is decidable for a bounded number of processes.

As in [6,15,16], we first show that history inclusion is an equivalent characterization of k-bounded TSO-to-SC linearizability. Then, as inspired by [2], we consider to reduce the history inclusion problem to a control state reachability problem of a lossy channel machine. Thus, the decidability of k-bounded TSO-to-SC linearizability follows from the fact that a control state reachability problem of a lossy channel machine is decidable [2]. However, the reduction method in [2] does not directly apply to linearizability of concurrent libraries. This is because that the call and return actions concerned by linearizability are beyond the scope of the TSO memory model, while the reduction method in [2] ensures only the total store orders among write/*cas* actions.

We extend the reduction method in [2] to effectively handle call and return actions. Suppose a concurrent system that contains n client processes running independently and interacting with a shared library. We introduce a new process that keeps launching the specific *cas* actions nondeterministically. These specific *cas* actions are used to mark the possible occurrences of the call and return actions along a trace of the concurrent system. Then, a correctly marked trace of this new process replicates the history of the trace of the concurrent system with only specific *cas* actions. Correspondingly, a counterexample trace of TSO-to-SC linearizability in the original concurrent system (of n processes) can be witnessed by a marked trace of the extended concurrent system (of *n+1* processes) with the call and return actions bypassed. This marked trace is called a marked violation of TSO-to-SC linearizability. In this way, the complement problem of TSO-to-SC linearizability on the original concurrent system can be characterized by checking whether there exists a marked violation of TSO-to-SC linearizability (a marked violation problem), to which the reduction method in [2] can be applied.

A lossy channel machine M_i^k $(1 \leq i \leq n{+}1)$ is then constructed such that its traces contain at most k call and return actions and can simulate the k-bounded behaviors of the extended concurrent system from the perspective of each process P_i. Each M_i^k contains only one channel to store the pending written items according to the total store orders under the original concurrent system. Thus, a marked violation problem of k-bounded TSO-to-SC linearizability can be reduced to a control state reachability problem between a pair of specific configurations of the product of $M_1^{k\text{-}w}, \ldots, M_{n+1}^{k\text{-}w}$. Each $M_i^{k\text{-}w}$ is resulted from M_i^k by replacing its all but write and *cas* transitions with internal transitions. The reduction is achieved by requiring that each written item in a channel contains a run-time snapshot of the memory, while always keeping bounded the amount of information that needs to be stored as in a perfect channel. With these specialized lossy channels, missing some intermediate channel contents would not break the reachability between control states under perfect channels.

Furthermore, we can show that the decision problem of k-bounded TSO-to-SC linearizability has non-primitive recursive complexity. This can be proved by a reduction from a reachability problem of a lossy single-channel machine, which is known to have non-primitive recursive complexity [14]. Besides, the decision problem of TSO-to-SC linearizability can be reduced to a control state reachability problem of a perfect channel machine in a similar way. This opens a potential way towards determining the decidability of TSO-to-SC linearizability itself.

Related work Efforts have been devoted on verification of linearizability on the SC memory model [1,4,5,6]. A similar reduction method was applied to verify the linearizability of certain concurrent data structures for an unbounded number of processes on the SC memory model [5]. However, relaxed memory models remain a great challenge for linearizability verification. Our previous work [15,16] revealed the first undecidability result on TSO-to-TSO linearizability for a bounded number of processes. In [15,16], the trace inclusion problem of a classic-lossy single-channel system, which has been known to be undecidable, was reduced to the TSO-to-TSO linearizability problem. The closest work to ours is [2] by Atig *et al.*, where a state reachability problem of a concurrent system is reduced to a control state reachability problem of a lossy channel machine.

2 Concurrent Systems

In this section, we first present the notations of libraries, client programs, most general clients and concurrent systems. We then 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 $|l|$ and $l(i)$ denote the length and the i-th element of l, respectively, i.e., $|l| = k$ and $l(i) = \alpha_i$ for $1 \leq i \leq k$. Let $l \uparrow_{\Sigma}$ denote the projection of l to Σ. Given a function f, let $f[x : y]$ be the function that is the same as f everywhere, except for x, where it has the value y. Let $\_$ denote an item, of which the value is irrelevant, and ϵ the empty word.

A *labelled transition system* (LTS) is a tuple $\mathcal{A} = (Q, \Sigma, \rightarrow, q_0)$, where Q is a set of states (a.k.a. configurations), Σ is an alphabet of transition labels, $\rightarrow \subseteq Q \times \Sigma \times Q$ is a transition relation and q_0 is the initial state. A path of $\mathcal{A}$ is a finite transition sequence $q_0 \xrightarrow{\beta_1} q_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_k} q_k$ with $k \geq 0$. A trace of $\mathcal{A}$ is a finite sequence $t = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_k$ with $k \geq 0$ if there exists a path $q_0 \xrightarrow{\beta_1} q_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_k} q_k$ of $\mathcal{A}$.

2.2 Libraries and Client Programs

A library implementing a concurrent data structure provides a number of methods for accessing the data structure. A client program is a program that interacts with libraries. Libraries and client programs may contain private memory locations for their own uses. For simplicity of notations, we assume that a method has just one argument and one return value (if it returns).

Given a finite set $\mathcal{X}$ of memory locations, a finite set $\mathcal{M}$ of method names and a finite data domain $\mathcal{D}$, the set *PCom* of primitive commands has the forms below:

$$\mathit{PCom} ::= \tau \mid \mathit{read}(x,a) \mid \mathit{write}(x,a) \mid \mathit{cas\_suc}(x,a,b) \mid \mathit{cas\_fail}(x,a,b) \mid \mathit{call}(m,a)$$

where $a, b \in \mathcal{D}, x \in \mathcal{X}$ and $m \in \mathcal{M}$. Herein, τ is the internal command. A *cas* (compare-and-swap) command compresses a read and a write commands into a single one, which is meant to be executed atomically. A successful *cas* command $\mathit{cas\_suc}(x, a, b)$ changes the value of x from a to b, while a failed *cas* command $\mathit{cas\_fail}(x, a, b)$ does nothing and happens only when the value of x is not a.

A library $\mathcal{L}$ can then be defined as a tuple $\mathcal{L} = (\mathcal{X}_\mathcal{L}, \mathcal{M}_\mathcal{L}, \mathcal{D}_\mathcal{L}, Q_\mathcal{L}, \rightarrow_\mathcal{L})$, where $\mathcal{X}_\mathcal{L}$, $\mathcal{M}_\mathcal{L}$ and $\mathcal{D}_\mathcal{L}$ are a finite memory location set, a finite method name set and a finite data domain of $\mathcal{L}$ respectively; $Q_\mathcal{L} = \bigcup_{m\in\mathcal{M}_\mathcal{L}} Q_m$ is a finite set of program positions, and it is the union of disjoint sets Q_m of program positions of each method $m \in \mathcal{M}_\mathcal{L}$; $\rightarrow_\mathcal{L} = \bigcup_{m\in\mathcal{M}_\mathcal{L}} \rightarrow_m$ is the union of disjoint transition relations of each method $m \in \mathcal{M}_\mathcal{L}$. Let $\mathit{PCom}_\mathcal{L}$ be the set of primitive commands (except call commands) upon $\mathcal{X}_\mathcal{L}$, $\mathcal{M}_\mathcal{L}$ and $\mathcal{D}_\mathcal{L}$. Then, for each $m \in \mathcal{M}_\mathcal{L}$, $\rightarrow_m \subseteq Q_m \times \mathit{PCom}_\mathcal{L} \times Q_m$; while for each $a \in \mathcal{D}_\mathcal{L}$ there exists an initial state $\mathit{is}_{(m,a)}$ and a final state $\mathit{fs}_{(m,a)}$ in Q_m such that there are neither incoming transitions to $\mathit{is}_{(m,a)}$ nor outgoing transitions from $\mathit{fs}_{(m,a)}$ in $\rightarrow_m$. Similarly, a client program $\mathcal{C}$ can then be defined as a tuple $\mathcal{C} = (\mathcal{X}_\mathcal{C}, \mathcal{M}_\mathcal{C}, \mathcal{D}_\mathcal{C}, Q_\mathcal{C}, \rightarrow_\mathcal{C})$ where $\mathcal{X}_\mathcal{C}$, $\mathcal{M}_\mathcal{C}$, $\mathcal{D}_\mathcal{C}$ and $Q_\mathcal{C}$ are a finite memory location set, a finite method name set and a final data domain of $\mathcal{C}$ and a finite program position set, respectively. Let $\mathit{PCom}_\mathcal{C}$ be the set of primitive commands upon $\mathcal{X}_\mathcal{C}$, $\mathcal{M}_\mathcal{C}$ and $\mathcal{D}_\mathcal{C}$. Then, $\rightarrow_\mathcal{C} \subseteq Q_\mathcal{C} \times \mathit{PCom}_\mathcal{C} \times Q_\mathcal{C}$ is a transition relation of $\mathcal{C}$.

A most general client is a special client program that is designed to exhibit all the possible behaviors of a library. A most general client $\mathcal{MGC}$ can be formally defined as a client $(\mathcal{X}_\mathcal{C}, \mathcal{M}_\mathcal{C}, \mathcal{D}_\mathcal{C}, \{q_c\}, \rightarrow_{\mathit{mgc}})$, where q_c is a program position and $\rightarrow_{\mathit{mgc}} = \{(q_c, \mathit{call}(m,a), q_c) | m \in \mathcal{M}_\mathcal{C}, a \in \mathcal{D}_\mathcal{C}\}$ is a transition relation. Intuitively, a most general client simply repeatedly calls an arbitrary method with an arbitrary argument for arbitrarily many times. It does not access any memory location in $\mathcal{X}_\mathcal{C}$, so $\mathcal{X}_\mathcal{C}$ does not influence the behavior of a most general client.

2.3 Operational Semantics

Suppose a concurrent system $C(\mathcal{L})$ that consists of n processes, each of which runs a client program $\mathcal{C}_i = (\mathcal{X}_\mathcal{C}, \mathcal{M}, \mathcal{D}_\mathcal{C}, Q_{\mathcal{C}_i}, \rightarrow_{\mathcal{C}_i})$ on a separate processor for $1 \leq i \leq n$, and all the client programs interact with the same library $\mathcal{L} = (\mathcal{X}_\mathcal{L}, \mathcal{M}, \mathcal{D}_\mathcal{L}, Q_\mathcal{L}, \rightarrow_\mathcal{L})$. The library and client programs have disjoint memory locations, i.e., $\mathcal{X}_\mathcal{L} \cap \mathcal{X}_\mathcal{C} = \emptyset$. The operational semantics of the concurrent system $C(\mathcal{L})$ on the TSO memory model is defined as an LTS $[\![C(\mathcal{L}),n]\!]_{tso}$= $(Conf_{tso}, \Sigma_{tso}, \rightarrow_{tso}, InitConf_{tso})$, where $Conf_{tso}, \Sigma_{tso}, \rightarrow_{tso}$, $InitConf_{tso}$ are defined as follows.

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

- $p : \{1, \ldots, n\} \rightarrow Q_{\mathcal{C}i} \cup (Q_{\mathcal{L}} \times Q_{\mathcal{C}i})$ represents control states of each process. $p(i) = q_c \in Q_{\mathcal{C}i}$ represents that process i is executing client position q_c, while $p(i) = (q_l, q_c)$ represents that process i is executing library position q_l and after this method returns it will turn to execute client position q_c;
- $d : (\mathcal{X}_{\mathcal{L}} \rightarrow \mathcal{D}_{\mathcal{L}}) \cup (\mathcal{X}_{\mathcal{C}} \rightarrow \mathcal{D}_{\mathcal{C}})$ is the valuation of library and client memory locations;
- u represents contents of store buffers for each process. It takes a process id $i \in \{1, \ldots, n\}$ and returns a sequence in $\{(x,a)|(x \in \mathcal{X}_{\mathcal{L}} \wedge a \in \mathcal{D}_{\mathcal{L}}) \vee (x \in \mathcal{X}_{\mathcal{C}} \wedge a \in \mathcal{D}_{\mathcal{C}})\}^*$.

Σ_{tso} is a set of actions in the following forms:

$$\Sigma_{tso} ::= \tau(i) \mid read(i,x,a) \mid write(i,x,a) \mid cas(i,x,a,b) \mid \\ flush(i,x,a) \mid call(i,m,a) \mid return(i,m,a)$$

where $1 \leq i \leq n, m \in \mathcal{M}$ and either $x \in \mathcal{X}_{\mathcal{L}}$ and $a, b \in \mathcal{D}_{\mathcal{L}}$, or $x \in \mathcal{X}_{\mathcal{C}}$ and $a, b \in \mathcal{D}_{\mathcal{C}}$.

The relation T is used to define the transitions occur from library or client programs and is defined as $T = \{((q_{l1}, q_{cl}), \alpha, (q_{l2}, q_{cl}))|q_{l1} \xrightarrow{\alpha}_{\mathcal{L}} q_{l2}\} \cup \{(q_{c1}, \alpha, q_{c2})|\exists 1 \leq i \leq n$, $q_{c1} \xrightarrow{\alpha}_{\mathcal{C}i} q_{c2}\}$. The transition relation $\rightarrow_{tso}$ is the least relation satisfying the transition rules shown in Fig. 1 for each $1 \leq i \leq n$.

- *Tau* rule: A τ transition only influences control state of one process.
- *Read* rule: A function $lookup(u,d,i,x)$ is used to search for the latest value of x from its processor-local store buffer or the main memory, i.e.,

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

  where $\Sigma_x = \{(x,a)|x \in \mathcal{X}_{\mathcal{L}} \wedge a \in \mathcal{D}_{\mathcal{L}}) \vee (x \in \mathcal{X}_{\mathcal{C}} \wedge a \in \mathcal{D}_{\mathcal{C}}\}$ is the set of pending write actions for x.

  Read action 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 action will insert a pair of location and value to the tail of its processor-local store buffer.
- *Cas-Suc* 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 while a failed *cas* command does not change memory.
- *Flush* rule: The memory system may decide to flush the entry at the head of processor-local store buffer to memory at any time.
- *Call* and *Return* rules: To deal with *call*$(\_, m, a)$ command, current process starts to execute the initial position of method m and parameter a. When the process comes to final position of method m and parameter a, it can launch a *return*$(\_, m, a)$ action and start to execute the most general client.

$$\frac{T(p(i), c, q'_i,), c = (\tau)}{(p, d, u) \xrightarrow{\tau(i)}_{tso} (p[i : q'_i], d, u)} \textit{Tau}$$

$$\frac{T(p(i), c, q'_i,), c = (\textit{read}(x, a)), \textit{lookup}(u, d, i, x) = a}{(p, d, u) \xrightarrow{\textit{read}(i,x,a)}_{tso} (p[i : q'_i], d, u])} \textit{Read}$$

$$\frac{T(p(i), c, q'_i,), c = (\textit{write}(x, a)), u(i) = l}{(p, d, u) \xrightarrow{\textit{write}(i,x,a)}_{tso} (p[i : q'_i], d, u[i : (x, a) \cdot l])} \textit{Write}$$

$$\frac{T(p(i), c, q'_i,), c = (\textit{cas\_suc}(x, a, b)), d(x) = a, u(i) = \epsilon}{(p, d, u) \xrightarrow{\textit{cas}(i,x,a,b)}_{tso} (p[i : q'_i], d[x : b], u)} \textit{Cas-Suc}$$

$$\frac{T(p(i), c, q'_i,), c = (\textit{cas\_fail}(x, a, b)), d(x) \neq a, u(i) = \epsilon}{(p, d, u) \xrightarrow{\textit{cas}(i,x,a,b)}_{tso} (p[i : q'_i], d, u)} \textit{Cas-Fail}$$

$$\frac{u(i) = l \cdot (x, a), (x \in \mathcal{X}_{\mathcal{L}} \wedge a \in \mathcal{D}_{\mathcal{L}}) \vee (x \in \mathcal{X}_{\mathcal{C}} \wedge a \in \mathcal{D}_{\mathcal{C}})}{(p, d, u) \xrightarrow{\textit{flush}(i,x,a)}_{tso} (p, d[x : a], u[i : l])} \textit{Flush}$$

$$\frac{p(i) = q_{c1}, q_{c1} \xrightarrow{\textit{call}(m,a)}_{\mathcal{C}i} q_{c2}}{(p, d, u) \xrightarrow{\textit{call}(i,m,a)}_{tso} (p[i : (\textit{is}_{(m,a)}, q_{c2})], d, u)} \textit{Call}$$

$$\frac{p(i) = (\textit{fs}_{(m,a)}, q_{cl})}{(p, d, u) \xrightarrow{\textit{return}(i,m,a)}_{tso} (p[i : q_{cl}], d, u)} \textit{Return}$$

Fig. 1. Transition Relation $\rightarrow_{tso}$

The initial configuration $\textit{InitConf}_{tso} \in \textit{Conf}_{tso}$ is a tuple $(p_{init}, d_{init}, \epsilon^n)$, where ϵ^n initializes each process with an empty buffer. If each client program C_i is a most general client, $[\![C(\mathcal{L}), n]\!]_{tso}$ can be abbreviated as $[\![\mathcal{L}, n]\!]_{tso}$.

According to [8], to give the semantics on SC, we do not need to define another abstract machine; instead, we identify the SC executions of a concurrent system with those of the TSO operational semantics that flush all write actions immediately. Formally, the operational semantics of the concurrent system $C(\mathcal{L})$ for n processes on SC memory model is defined as an LTS $[\![C(\mathcal{L}), n]\!]_{sc} = (\textit{Conf}_{sc}, \Sigma_{sc}, \rightarrow_{sc}, \textit{InitConf}_{sc})$, where $\textit{InitConf}_{sc} = \textit{InitConf}_{tso}$ and $\textit{Conf}_{sc}$, Σ_{sc} and $\rightarrow_{sc}$ are defined as follows.

- $\textit{Conf}_{sc}$ contains all the configurations of $\textit{Conf}_{tso}$ that has a empty buffer for each process.
- Σ_{sc} is generated from Σ_{tso} by discarding the flush actions.
- $\rightarrow_{sc}$ is generated from $\rightarrow_{tso}$ by discarding the *Flush* rule and changing the *Write* rule to *Write-SC* rule as follows:

$$\frac{T(p(i), c, q'_i), c = \textit{write}(x, a)}{(p, d, u) \xrightarrow{\textit{write}(i,x,a)}_{sc} (p[i : q'_i], d[x : a], u)} \textit{Write-SC}$$

When C maps each process id to a most general client, $[\![C(\mathcal{L}), n]\!]_{sc}$ can be shortened as $[\![\mathcal{L}, n]\!]_{sc}$.

3 Correctness Conditions and Equivalent Characterization

The behavior of a library is typically represented by histories of interactions between the library and the client programs calling it (through call and return actions). Let Σ_{cal} and Σ_{ret} represent the sets of all call and return actions, respectively. A finite sequence $h \in (\Sigma_{cal} \cup \Sigma_{ret})^*$ is a history of an LTS $\mathcal{A}$ if there exists a trace t of $\mathcal{A}$ such that $t \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})} = h$. Let *history*$(t)$ be the history along trace t, i.e., *history*$(t) = t \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}$, and *history*$(\mathcal{A})$ the set of all histories of $\mathcal{A}$. Moreover, let $h|_i$ denote the projection of history h to the call and return actions of process P_i.

TSO-to-SC linearizability is a variant of linearizability on the TSO memory model. It is used to reason about the interoperability between a high-level data race free client and a low-level library native to the TSO memory model. Hence, it concerns only call and return actions.

Definition 1 **(*TSO-to-SC linearizability* [8]).** *For histories $h_1, h_2 \in (\Sigma_{cal} \cup \Sigma_{ret})^*$, h_1 is linearizable to h_2, if*

- *for each process P_i, $h_1|_i = h_2|_i$.*
- *there is a bijection $\pi : \{1, \ldots, |h_1|\} \to \{1, \ldots, |h_2|\}$ such that for any $1 \le i \le |h_1|$, $h_1(i) = h_2(\pi(i))$ and for any $1 \le i < j \le |h_1|$, if $h_1(i) \in \Sigma_{ret} \wedge h_1(j) \in \Sigma_{cal}$, then $\pi(i) < \pi(j)$.*

For two libraries $\mathcal{L}$ and $\mathcal{L}'$, $\mathcal{L}'$ TSO-to-SC linearizes $\mathcal{L}$ for n processes, if for any history $h_1 \in$ history$([\![\mathcal{L}, n]\!]_{tso})$, there exists history $h_2 \in$ history$([\![\mathcal{L}', n]\!]_{sc})$, such that h_1 is linearizable to h_2.

The following lemma shows that history inclusion is an equivalent characterization of TSO-to-SC linearizability.

Lemma 1. *Library $\mathcal{L}'$ TSO-to-SC linearizes library $\mathcal{L}$ for n processes if and only if history$([\![\mathcal{L}, n]\!]_{tso}) \subseteq$ history$([\![\mathcal{L}', n]\!]_{sc})$.*

For an LTS $\mathcal{A}$, a k-trace $t \in$ *trace*$(\mathcal{A})$ is a trace that contains at most k call and return actions. Let *k-trace*$(\mathcal{A})$ denote all the k-traces of $\mathcal{A}$.

Definition 2 **(*k-bounded TSO-to-SC linearizability*).** *Library $\mathcal{L}'$ k-bounded TSO-to-SC linearizes library $\mathcal{L}$ for n processes, if for each k-trace $t \in k$-trace$([\![\mathcal{L}, n]\!]_{tso})$, there exists a history $h \in$ history$([\![\mathcal{L}', n]\!]_{sc})$, such that history$(t)$ is linearizable to h.*

For two libraries $\mathcal{L}$, $\mathcal{L}'$ and $n, k \ge 1$, the decision problem of (k-bounded) TSO-to-SC linearizability is to determine whether $\mathcal{L}'$ (k-bounded) TSO-to-SC linearizes $\mathcal{L}$ for n processes.

4 Perfect/Lossy Channel Machines

A classical channel machine is a finite control machine equipped with channels of unbounded sizes. It can perform send and receive operations on its channels. A lossy channel machine is a channel machine where arbitrary many items in its channels may be lost nondeterministically at any time without any notification. In this section we sketch our definition of (S, K)-channel machines, which slightly differs from the definition of channel machines in [2].

The channel machines defined in [2] extend classical channel machines in the following aspects:

- Each transition is guarded by a condition about whether the content of a channel is in a regular language.
- A substitution to the content of a channel may be performed before a send operation on the channel.
- A set of specific symbols, called "strong symbols", are introduced that are not allowed to be lost, but the number of strong symbols in a channel is always bounded.

In this paper, we extend the channel machines defined in [2] with multiple sets of strong symbols, while the number of strong symbols in a channel from the same strong symbol set is separately bounded.

Let $\mathcal{CH}$ be the finite set of channel names and $\Sigma_{\mathcal{CH}}$ be a finite alphabet of channel contents. The content of a channel is a finite sequence over $\Sigma_{\mathcal{CH}}$. For a given channel $c \in \mathcal{CH}$, a regular guard on channel c is a constraint of the form $c \in L$, where $L \subseteq \Sigma_{\mathcal{CH}}^*$ is a regular set of sequences. For a sequence $u \in \Sigma_{\mathcal{CH}}^*$ we write $u \models c \in L$ if $l \in L$. For notational convenience, we write $a \in c$ instead of $c \in \Sigma_{\mathcal{CH}}^* \cdot a \cdot \Sigma_{\mathcal{CH}}^*$, $c = \epsilon$ instead of $c \in \{\epsilon\}$ and $c : \Sigma'$ instead of $c \in \Sigma'^*$ for any subset Σ' of $\Sigma_{\mathcal{CH}}$. A regular guard over $\mathcal{CH}$ associates a regular guard for each channel of $\mathcal{CH}$. Let *Guard*$(\mathcal{CH})$ be the set of regular guards over $\mathcal{CH}$. The definition of $\models$ can be extended as follows: for $g \in$ *Guard*$(\mathcal{CH})$ and $u \in \mathcal{CH} \to \Sigma_{\mathcal{CH}}^*$, we write $u \models g$, if $u(c) \models g(c)$ for each $c \in \mathcal{CH}$.

Given a channel $c \in \mathcal{CH}$, a channel operation on c is either a *nop* (no operation), or an $c?a$ operation for some $a \in \Sigma_{\mathcal{CH}}$ (receive operation), or an $c[\sigma]!a$ operation (send operation) where σ is a substitution over $\Sigma_{\mathcal{CH}}$ and a is a element of $\Sigma_{\mathcal{CH}}$. We write $c!a$ instead of $c[\sigma]!a$ when σ is the identity substitution. For every $u, u' \in \Sigma_{\mathcal{CH}}^*$, we have $[\![nop]\!](u, u')$ if $u = u'$, $[\![c[\sigma]!a]\!](u, u')$ if $u' = a \cdot u[\sigma]$, $[\![c?a]\!](u, u')$ if $u = u' \cdot a$. A channel operation over $\mathcal{CH}$ is a mapping that associates with each channel c a channel operation on c. Let *Op*$(\mathcal{CH})$ be the set of channel operations over $\mathcal{CH}$. The definition of $[\![op]\!]$ can be extended as follows: for $op \in$ *Op*$(\mathcal{CH})$ and $u, u' \in \mathcal{CH} \to \Sigma_{\mathcal{CH}}^*$, we have $[\![op]\!](u, u')$, if $[\![op(c)]\!](u(c), u'(c))$ holds for each $c \in \mathcal{CH}$.

A *channel machine* is formally defined as a tuple $M = (Q, \mathcal{CH}, \Sigma_{\mathcal{CH}}, \Lambda, \Delta)$, where (1) Q is a finite set of states, (2) $\mathcal{CH}$ is a finite set of channel names, (3) $\Sigma_{\mathcal{CH}}$ is an alphabet for channel contents, (4) Λ is a finite set of transition labels, and (5) $\Delta \subseteq Q \times (\Lambda \cup \{\epsilon\}) \times$ *Guard*$(\mathcal{CH}) \times$ *Op*$(\mathcal{CH}) \times Q$ is a finite set of transitions.

We say a sequence $l_1 = a_1 \cdot \ldots \cdot a_u$ is a subword of another sequence $l_2 = b_1 \cdot \ldots \cdot b_v$, if there exists $i_1 < \ldots < i_u$, such that $a_j = b_{i_j}$ for each j. Let $S = \langle s_1, \ldots, s_m \rangle$ be a vector of sets with $s_i \subseteq \Sigma_{\mathcal{CH}}$ for $1 \le i \le m$, and $K = \langle k_1, \ldots, k_m \rangle$ be a vector of nature numbers or ∞. S is the sets of strong symbols that must be kept in transition, and K is the bounds for each set of strong symbols in S. For sequences $u, v \in \Sigma^*_{\mathcal{CH}}$, $u \preceq^K_S v$ holds if (1) u is a subword of v, (2) for each i, $u \uparrow_{s_i} = v \uparrow_{s_i}$ and (3) for each j, $|u \uparrow s_j| \le k_j$. This relation can be extended as follows: For every $u, v \in \mathcal{CH} \to \Sigma^*_{\mathcal{CH}}$, $u \preceq^K_S v$ holds, if $u(c) \preceq^K_S v(c)$ holds for each $c \in \mathcal{CH}$.

A *(S,K)-channel machine* (abbreviated as *(S,K)-CM*) is a channel machine $M = (Q, \mathcal{CH}, \Sigma_{\mathcal{CH}}, \Lambda, \Delta)$ with the strong symbol restriction (S, K). Its semantics is defined as an *LTS* $(\mathit{Conf}_M, \Lambda, \to_M, \mathit{initConf}_M)$. A configuration of Conf_M is a pair (q, u) where $q \in Q$, $u : \mathcal{CH} \to \Sigma^*_{\mathcal{CH}}$, and it satisfies the strong symbol restriction(S, K), i.e., for each c and i, $|u(c) \uparrow s_i| \le k_i$. The transition relation $\to_M$ is defined as follows: given $q, q' \in Q$ and $u, u' \in \mathcal{CH} \to \Sigma^*_{\mathcal{CH}}$, $(q, u) \xrightarrow{\alpha}_M (q', u')$, if there exists g and op, such that $(q, \alpha, g, op, q') \in \Delta$, $u \models g$ and $[\![op]\!](u, u')$. Similarly, a *(S,K)-lossy channel machine* (abbreviated as *(S,K)-LCM*) is a channel machine M with lossy channels and the strong symbol restriction (S, K). Its semantics is defined as an *LTS* $(\mathit{Conf}_M, \Lambda, \to_{(M,S,K)}, \mathit{initConf}_M)$. The transition relation $\to_{(M,S,K)}$ is defined as follows: $(q, u) \xrightarrow{\alpha}_{(M,S,K)} (q', u')$, if there exists $v, v' \in \mathcal{CH} \to \Sigma^*_{\mathcal{CH}}$, such that $v \preceq^K_S u$, $(q, v) \xrightarrow{\alpha}_M (q', v')$ and $u' \preceq^K_S v'$. Let $\to^*_M$ and $\to^*_{(M,S,K)}$ be the transition closure of $\to_M$ and $\to_{(M,S,K)}$.

Given a channel machine M, we say that $(q_0, u_0) \cdot \alpha_1 \cdot (q_1, u_1) \cdot \ldots \cdot \alpha_w \cdot (q_w, u_w)$ is a finite run of M from (q, u) to (q', u'), if (1) $(q_0, u_0) = (q, u)$, (2) $(q_i, u_i) \xrightarrow{\alpha_{i+1}}_M (q_{i+1}, u_{i+1})$ for each i and (3) $(q_w, u_w) = (q', u')$. We say that l is a trace of a finite run ρ if $l = \rho \uparrow_\Lambda$. Given $q, q' \in Q$, let $T^{S,K}_{q,q'}(M)$ denote the set of traces of all finite runs of a (S, K)-*CM* M from the configuration $(q, \epsilon^{|n|})$ to the configuration $(q', \epsilon^{|n|})$. For $(S, K) - \mathit{LCM}\ M$, the notations of finite run and its trace are defined as in the non-lossy case by replacing $\to_M$ with $\to_{(M,S,K)}$. Let $\mathit{LT}^{S,K}_{q,q'}(M)$ denote the set of traces of all finite runs of (S, K)-*LCM* M from the configuration $(q, \epsilon^{|n|})$ to the configuration $(q', \epsilon^{|n|})$.

For channel machines $M_1 = (Q_1, \mathcal{CH}_1, \Sigma_{\mathcal{CH}}, \Lambda, \Delta_1)$ and $M_2 = (Q_2, \mathcal{CH}_2, \Sigma_{\mathcal{CH}}, \Lambda, \Delta_2)$ such that $\mathcal{CH}_1 \cap \mathcal{CH}_2 = \emptyset$, the product of M_1 and M_2 is also a channel machine $M_1 \otimes M_2 = (Q_1 \times Q_2, \mathcal{CH}_1 \cup \mathcal{CH}_2, \Sigma_{\mathcal{CH}}, \Lambda, \Delta_{12})$, where Δ_{12} is defined by synchronizing transitions sharing the same label in Λ under the conjunction of their guards, and letting other transitions asynchronous. The following lemma holds as in [2].

Lemma 2. *Given channel machines* $M_1 = (Q_1, \mathcal{CH}_1, \Sigma_{\mathcal{CH}}, \Lambda, \Delta_1)$ *and* $M_2 = (Q_2, \mathcal{CH}_2, \Sigma_{\mathcal{CH}}, \Lambda, \Delta_2)$*, let* $q_1, q_1' \in Q_1$*,* $q_2, q_2' \in Q_2$*,* $q = (q_1, q_2)$*,* $q' = (q_1', q_2')$*, then* $\mathit{LT}^{S,K}_{q,q'}(M_1 \otimes M_2) = \mathit{LT}^{S,K}_{q_1,q_1'}(M_1) \cap \mathit{LT}^{S,K}_{q_2,q_2'}(M_2)$ *and* $T^{S,K}_{q,q'}(M_1 \otimes M_2) = T^{S,K}_{q_1,q_1'}(M_1) \cap T^{S,K}_{q_2,q_2'}(M_2)$*.*

Given a (S, K)-*CM* (respectively, (S, K)-*LCM*) M and two states $q, q' \in Q$, a control state reachability problem of M is to determine whether $T^{S,K}_{q,q'}(M) \ne \emptyset$ (respectively, $\mathit{LT}^{S,K}_{q,q'}(M) \ne \emptyset$). As in [2], it can be shown that the control state reachability problem is decidable for (S, K)-*LCM*.

5 Verification of k-Bounded TSO-to-SC Linearizability

In this section we show the proof idea about the decidability of k-bounded TSO-to-SC linearizability for a bounded number of processes. The main theme is to reduce its complement problem to a control state reachability problem of a (S, K)-lossy channel machine. In the same way, we can reduce the decision problem of TSO-to-SC linearizability to a control state reachability problem of a (S, K)-channel machine.

5.1 Marked Violation Problem of (k-Bounded) TSO-to-SC Linearizability

Recall that call and return actions cannot be handled directly by the reduction method in [2]. We introduce a fresh new process to captures the call and return actions, which occur along the traces (or k-traces) of $[\![\mathcal{L}, n]\!]_{tso}$ by the specific *cas* actions. In this way, the behaviors of a concurrent system $[\![\mathcal{L}, n]\!]_{tso}$ can be characterized exactly by the extended concurrent system $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$ (defined below), with the benefit that the call and return actions need not be involved for reduction later.

Let $markedVal(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n) = \{call(i,m,a), return(i,m,a) | 1 \leq i \leq n, m \in \mathcal{M}, a \in \mathcal{D}_{\mathcal{L}}\}$ denote the set of values that are used by the specific *cas* actions to mark the call and return actions in $[\![\mathcal{L}, n]\!]_{tso}$. Then, the concurrent system $Clt(\mathcal{L})$ consists of $n{+}1$ processes P_i ($1 \leq i \leq n{+}1$). For each $1 \leq i \leq n$, process P_i runs the most general client program $(\{x_{wit}\}, \mathcal{M}, \mathcal{D}_{\mathcal{L}}, \{q_c\}, \rightarrow_{mgc})$. The process P_{n+1} runs the client program $C_{marked} = (\{x_{wit}\}, \mathcal{M}, markedVal(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n), \{q_{wit}\}, \rightarrow_{wit})$, where $x_{wit} \notin \mathcal{X}_{\mathcal{L}}$ is the memory location used by the specific *cas* actions; $\rightarrow_{wit} = \{(q_{wit}, cas\_suc(x_{wit}, \_, a), q_{wit}) |\ a \in markedVal(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n)\}$ is the transition relation of C_{marked}.

A marked violation is a trace of $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$ that can witness the violation of TSO-to-SC linearizability. It correctly captures the corresponding call and return actions, stops immediately when a non-linearizable action takes place and flushes all the stored items so far. Formally, a trace $t \in trace([\![Clt(\mathcal{L}), n{+}1]\!]_{tso})$ is a *marked violation* of TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes, if

- The specific *cas* actions mark correctly the call and return actions, i.e., for each $1 \leq i \leq |t|-1$, $m \in \mathcal{M}$ and $a \in \mathcal{D}_{\mathcal{L}}$, the following conditions hold:
  1. $t(i) = cas(n{+}1, x_{wit}, call(i, m, a))$ if and only if $t(i{+}1) = call(i, m, a)$.
  2. $t(i) = cas(n{+}1, x_{wit}, return(i, m, a))$ if and only if $t(i{+}1) = return(i, m, a)$.
- $history(t) \notin history([\![\mathcal{L}', n]\!]_{sc})$, and for each prefix t' of t such that $history(t) \neq history(t')$, $history(t') \in history([\![\mathcal{L}', n]\!]_{sc})$.
- $t = t_1 \cdot t_2$ such that t_1 ends with a call or return action, and t_2 is a sequence of flush actions. Moreover, all the write actions in t have been flushed.

Furthermore, the trace t is a *marked violation* of k-bounded TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes, if t is a k-trace. For two libraries $\mathcal{L}$, $\mathcal{L}'$, and $n, k \geq 1$, a (k-bounded) TSO-to-SC marked violation problem is to determine whether there is a marked violation of (k-bounded) TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes. The following lemma relates a (k-bounded) TSO-to-SC marked violation problem with the complement problem of (k-bounded) TSO-to-SC linearizability.

Lemma 3. *$\mathcal{L}'$ does not (k-bounded) TSO-to-SC linearizes $\mathcal{L}$ for n processes, if and only if there is a marked violation of (k-bounded) TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes.*

The specific *cas* actions are launched nondeterministically in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$ and hence may result in many incorrectly guessed traces that do not occur in $[\![\mathcal{L}, n]\!]_{tso}$. However, the channel machines M_i^k we constructed can guarantee that the incorrectly guessed traces will be safely excluded during the verification procedure.

5.2 Simulating $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$ with A Channel Machine

In the rest of this section, we show that for libraries $\mathcal{L}$ and $\mathcal{L}'$, how the k-bounded behaviors of the concurrent system $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$ can be further characterized by a (S, K)-channel machine. As in [2], this amounts to construct a channel machines M_i^k corresponding to each process P_i in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$.

Each M_i^k ($1 \leq i \leq n{+}1$) launches actions of process P_i according to the control state of this process, and nondeterministically guesses the write, call or return actions of the other processes. It contains only one channel c_i that is used to store the pending written items according to the total store orders in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$. Each item sent to each channel c_i contains the current valuation of all the memory locations, i.e., the current snapshot of the memory.

We first use the example shown in Fig. 2 to illustrate the main idea of our construction method. Fig. 2 (a) presents a k-trace t of a concurrent system $[\![Clt(\mathcal{L}), 3]\!]_{tso}$ with $k = 4$, while Fig. 2 (b),(c),(d) present the corresponding traces of M_1^k, M_2^k, M_3^k, respectively. Each pair of a call and its accompanying return action is associated with a (dashed) line interval. In Fig. 2, *r(x)0* is an action that reads 0 from x; *w(x)1* is an action that writes 1 to x; *f(x)1* is a flush action that changes the value of x to 1; *c(y)1* is a *cas* action that changes the value of y to 1 successfully; $c_1, \ldots, c_4$ are the specific *cas* actions for marking the corresponding call and return actions; *g(x)1* and *f(x)1* are the guessed write action and its accompanying flush action for *w(x)1*; *g(y)1* and *f(y)1* are the guessed write action and its accompanying flush actions for *c(y)1*; g_i and f_i are the guessed write action and its accompanying flush actions for the action c_i ($1 \leq i \leq 4$); Noted that the actions in Fig. 2 (a) contain only values, while the actions in Fig. 2 (b),(c),(d) contain the snapshots of the memory.

In this example, M_1^k first guesses a marked write action g_1, performs the accompanying flush action f_1 and the call action of process P_1 and then reads 0 from x. Before M_1^k performs the *w(x)1* action, it need to guess the write and *cas* actions of processes P_2 and P_3. These actions need to occur later than *w(x)1* but their accompanying flush actions need to occur earlier than *f(x)1* in t. Therefore, it guesses g_2, g_3 and *g(y)1* accordingly. Then, M_1^k flushes g_2 (with f_2), guesses the call action of process P_2, flushes g_3 (with f_3), performs the return action of process P_1, and flushes *g(y)1* (with *f(y)1*). At last, M_1^k flushes *w(x)1* (with *f(x)1*), guesses the marked write action g_4, performs the accompanying flush action f_4 and guesses the return action of process P_2.

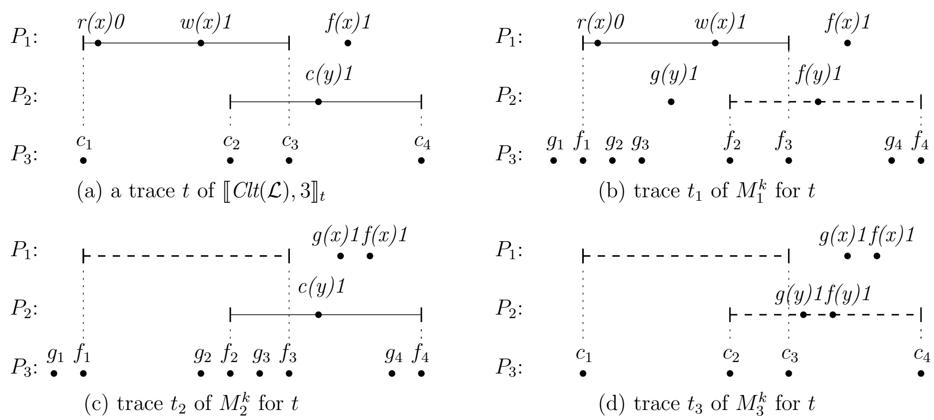

Fig. 2. traces of M_1^k, M_2^k and M_3^k for a trace t of $[\![Clt(\mathcal{L}), 3]\!]_{tso}$

5.3 Construction of M_i^k and M_i^{ts}

Note that *history*$([\![\mathcal{L}', n]\!]_{sc})$ is a regular language, because the LTS $[\![\mathcal{L}', n]\!]_{sc}$ contains a finite number of states. Let $\mathcal{A}_{Spec} = (Q_s, \Sigma_s, \rightarrow_s, q_{is})$ be a deterministic finite state automaton that accepts *history*$([\![\mathcal{L}', n]\!]_{sc})$, where Q_s is a set of states, Σ_s is a set of transition labels, $\rightarrow_s \subseteq Q_s \times \Sigma_s \times Q_s$ is a transition relation and q_{is} is the initial state. It can be seen that each state in Q_s can be assumed as a final state because *history*$([\![\mathcal{L}', n]\!]_{sc})$ is prefix-closed. Let $q_{error} \notin Q_s$ be a trap state. A new transition relation $\rightarrow_{s'}$ can be derived from $\rightarrow_s$ such that $q_1 \xrightarrow{\alpha}_{s'} q_2$ if either $q_1 \xrightarrow{\alpha}_s q_2$, or $q_1 \in Q_s$, $q_2 = q_{error}$ and there is no outgoing transitions from q_1 in $\xrightarrow{\alpha}_s$.

Let *Val* be the set of valuation functions that map a memory location in $\mathcal{X}_{\mathcal{L}}$ to a value in $\mathcal{D}_{\mathcal{L}}$ and x_{wit} to a value in *markedVal*$(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n)$. Channel machine M_i^k $(1 \leq i \leq n)$ is a tuple $(Q_i^k, \{c_i\}, \Sigma, \Lambda, \Delta_i^k)$, where c_i is name of the single channel of M_i^k. Q_i, c_i, Σ, Λ and Δ_i^k are defined as follows:

$Q_i^k = (\{q_c\} \cup (Q_{\mathcal{L}} \times \{q_c\})) \times Val \times Val \times (Q_s \cup \{q_{error}\}) \times (markedVal(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n) \cup \{\epsilon\}) \times \{0, \ldots, k\}$ is the state set. A configuration $(q, d_c, d_g, q_s, mak, cnt) \in Q_1$ consists of a control state q, a valuation d_c of the memory, a valuation d_g of the memory with all the stored items in c_i applied, a state q_s for monitoring the violation of the linearizability condition, a marker *mak* indicating that each marked *cas* action is immediately followed by a corresponding call or return action, and the number *cnt* of the call and return actions already occurred in the whole trace.

$\Sigma = \Sigma_{s1} \cup \Sigma_{s2} \cup \Sigma_{s3}$ is the alphabet of channel contents with $\Sigma_{s1} = \{(n{+}1, x_{wit}, d) | d \in Val\}$, $\Sigma_{s2} = \{((i, x, d), \sharp) | 1 \leq i \leq n, x \in \mathcal{X}_{\mathcal{L}}, d \in Val\}$ and $\Sigma_{s3} = \{a | (a, \sharp) \in \Sigma_{s2}\}$. Σ_{s1} contains the items appended by guessing the marked *cas* actions. Σ_{s2} are the newest item in c_i or the newest one for a memory location. Σ_{s3} is similar to Σ_{s2} except the symbols $\sharp$ are removed. In case that M_i^k is interpreted with a lossy channel, Σ_{s1} and Σ_{s2} are the sets of strong symbols of M_i^k.

Λ is the set of transition labels and is union of the following sets:

- $\{write(i, x, d), cas(i, x, d) | (1 \leq i \leq n \wedge x \in \mathcal{X}_{\mathcal{L}}) \vee (i = n{+}1 \wedge x = x_{wit}), d \in Val\}$.

- $\{\mathit{flush}(i,x,d), \mathit{flush}(n{+}1,x_r,d) | 1 \leq i \leq n, \mathcal{X}_{\mathcal{L}}, d \in \mathit{Val}\}$.
- $\{\mathit{call}(i,m,a), \mathit{return}(i,m,a) | 1 \leq i \leq n, m \in \mathcal{M}, a \in \mathcal{D}_{\mathcal{L}}\}$.

Λ does not contain read or τ actions, which are seen as ϵ transition in M_i^k.

Δ_i^k is the transition relation of M_i^k, it is the smallest set of transitions such that $\forall q \in \{q_c\} \cup (Q_{\mathcal{L}} \times \{q_c\})$, $q_1, q_2 \in Q_{\mathcal{L}}$, $d_c, d_g \in \mathit{Val}$, $q_s \in Q_s$ and $\mathit{cnt} < k$,

- Nop: if $q_1 \xrightarrow{\tau}_{\mathcal{L}} q_2$, then
  $((q_1,q_c), d_c, d_g, q_s, \epsilon, \mathit{cnt}) \xrightarrow{\epsilon, c_i:\Sigma, nop}_{\Delta_i^k} ((q_2,q_c), d_c, d_g, q_s, \epsilon, \mathit{cnt})$.
- Library write: if $q_1 \xrightarrow{\mathit{write}(x,a)}_{\mathcal{L}} q_2$, then for each $d_1, d_2 \in \mathit{Ass}$

$$((q_1,q_c), d_c, d_g, q_s, \epsilon, \mathit{cnt}) \xrightarrow{op,(\beta_1,\sharp)\in c_i \wedge (\beta_2,\sharp)\in c_i, c_i[\beta_1/(\beta_1,\sharp),\beta_2/(\beta_2,\sharp)]!\beta_3}_{\Delta_i^k} ((q_2,q_c), d_c, d'_g, q_s, \epsilon, \mathit{cnt})$$

$$((q_1,q_c), d_c, d_g, q_s, \epsilon, \mathit{cnt}) \xrightarrow{op,(\beta_1,\sharp)\in c_i \wedge c_i:\Theta_2, c_i[\beta_1/(\beta_1,\sharp)]!\beta_3}_{\Delta_i^k} ((q_2,q_c), d_c, d'_g, q_s, \epsilon, \mathit{cnt})$$

$$((q_1,q_c), d_c, d_g, q_s, \epsilon, \mathit{cnt}) \xrightarrow{op,c_i:\Theta_1 \wedge (\beta_2,\sharp), c_i[\beta_2/(\beta_2,\sharp)]!\beta_3}_{\Delta_i^k} ((q_2,q_c), d_c, d'_g, q_s, \epsilon, \mathit{cnt})$$

$$((q_1,q_c), d_c, d_g, q_s, \epsilon, \mathit{cnt}) \xrightarrow{op,c_i:\Theta_1 \wedge c_i:\Theta_2, c_i!\beta_3}_{\Delta_i^k} ((q_2,q_c), d_c, d'_g, q_s, \epsilon, \mathit{cnt})$$

  where $\beta_1 = (i,x,d_1)$, $\Theta_1 = \Sigma \backslash \{((i,x,d'),\sharp) | d' \in \mathit{Val}\}$, $\beta_2 = (j,x,d_2)$ with $1 \leq j \leq n \wedge j \neq i$, $\Theta_2 = \Sigma \backslash \{((j,\_,d'),\sharp) | j \neq i, d' \in \mathit{Val}\}$, $d'_g = d_g[x:a]$, $\beta_3 = ((i,x,d'_g),\sharp)$ and $op = \mathit{write}(i,x,d'_g)$.
- Guess write: if $1 \leq j \leq n \wedge j \neq i \wedge x \in \mathcal{X}_{\mathcal{L}} \wedge a \in \mathcal{D}_{\mathcal{L}}$, then

$$(q, d_c, d_g, q_s, \epsilon, \mathit{cnt}) \xrightarrow{op,(\beta,\sharp)\in c_i, c_i[\beta/(\beta,\sharp)]!\beta'}_{\Delta_i^k} (q, d_c, d'_g, q_s, \epsilon, \mathit{cnt})$$

$$(q, d_c, d_g, q_s, \epsilon, \mathit{cnt}) \xrightarrow{op,c_i:\Theta, c_i!\beta'}_{\Delta_i^k} (q, d_c, d'_g, q_s, \epsilon, \mathit{cnt})$$

  where $\beta = (j',\_,\_)$ with $j' \neq i$, $d'_g = d_g[x:a]$, $\beta' = ((j,x,d'_g),\sharp)$, $\Theta = \Sigma \backslash \{((j_1,\_,\_),\sharp) | j_1 \neq i\}$ and $op = \mathit{write}(j,x,d'_g)$.
  If $j = n{+}1 \wedge x = x_{\mathit{wit}} \wedge a \in \mathit{markedVal}(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n)$, then

$$(q, d_c, d_g, q_s, \epsilon, \mathit{cnt}) \xrightarrow{op,(\beta,\sharp)\in c_i, c_i[\beta/(\beta,\sharp)]!\beta'}_{\Delta_i^k} (q, d_c, d'_g, q_s, \epsilon, \mathit{cnt})$$

$$(q, d_c, d_g, q_s, \epsilon, \mathit{cnt}) \xrightarrow{op,c_i:\Theta, c_i!\beta'}_{\Delta_i^k} (q, d_c, d'_g, q_s, \epsilon, \mathit{cnt})$$

  where $\beta = (j',\_,\_)$ with $j' \neq i$, $d'_g = d_g[x_{\mathit{wit}}:a]$, $\beta' = (n{+}1, x_{\mathit{wit}}, d'_g)$, $\Theta = \Sigma \backslash \{((j',\_,\_),\sharp) | j' \neq i\}$ and $op = \mathit{write}(n{+}1, x_{\mathit{wit}}, d'_g)$.
- Library read: if $q_1 \xrightarrow{\mathit{read}(x,a)}_{\mathcal{L}} q_2$, then for each $d \in \mathit{Val}$ with $d(x) = a$,

$$((q_1,q_c), d_c, d_g, q_s, \epsilon, \mathit{cnt}) \xrightarrow{\epsilon,(\beta,\sharp)\in c_i, nop}_{\Delta_i^k} ((q_2,q_c), d_c, d_g, q_s, \epsilon, \mathit{cnt})$$

$$((q_1,q_c), d, d_g, q_s, \epsilon, \mathit{cnt}) \xrightarrow{\epsilon, c_i:\Theta, nop}_{\Delta_i^k} ((q_2,q_c), d, d_g, q_s, \epsilon, \mathit{cnt})$$

  where $\beta = (i,x,d)$ and $\Theta = \Sigma \backslash \{((i,x,d'),\sharp) | d' \in \mathit{Ass}\}$.

- Library *cas*: if $q_1 \xrightarrow{cas\_suc(x,a,b)}_{\mathcal{L}} q_2$, then for each $d \in Val$ with $d(x) = a$,

$$((q_1, q_c), d, d, q_s, \epsilon, cnt) \xrightarrow{cas(i,x,d[x:b]),c_i=\epsilon,nop}_{\Delta_i^k} ((q_2, q_c), d[x : b], d[x : b], q_s, \epsilon, cnt)$$

If $q_1 \xrightarrow{cas\_fail(x,a,b)}_{\mathcal{L}} q_2$, then for each $d \in Val$ with $d(x) \neq a$,

$$((q_3, q_c), d, d, q_s, \epsilon, cnt) \xrightarrow{cas(i,x,d),c_i=\epsilon,nop}_{\Delta_i^k} ((q_4, q_c), d, d, q_s, \epsilon, cnt)$$

- Flush items of process 1 to n: if $1 \leq j \leq n$, then for each $x \in \mathcal{D}_{\mathcal{L}}$, $d \in Val$,

$$(q, d_c, d_g, q'_s, \epsilon, cnt') \xrightarrow{op,c_i:\Sigma,c_i?(j,x,d)}_{\Delta_i^k} (q, d, d_g, q'_s, \epsilon, cnt')$$

$$(q, d_c, d_g, q'_s, \epsilon, cnt') \xrightarrow{op,c_i:\Sigma,c_i?((j,x,d),\sharp)}_{\Delta_i^k} (q, d, d_g, q'_s, \epsilon, cnt')$$

where $cnt' \leq k$, $q'_s \in Q_s \cup \{q_{error}\}$ and $op = flush(j, x, d)$.

- Flush marked item of call:

$$(q, d_c, d_g, q_s, \epsilon, cnt) \xrightarrow{op,c_i:\Sigma,c_i?(n+1,x_{wit},d)}_{\Delta_i^k} (q, d, d_g, q_s, call(j, m, c), cnt)$$

where $d(x_{wit}) = call(j, m, c)$ and $op = flush(n+1, x_{wit}, d)$.

- Flush marked item of return:

$$(q, d_c, d_g, q_s, \epsilon, cnt) \xrightarrow{op,c_i:\Sigma,c_i?(n+1,x_{wit},d)}_{\Delta_i^k} (q, d, d_g, q_s, return(j, m, c), cnt)$$

where $d(x_{wit}) = return(j, m, c)$ and $op = flush(n+1, x_{wit}, d)$.

- Call: if $q_s \xrightarrow{call(i,m,a)}_{s'} q'_s$, then

$$(q_c, d_c, d_g, q_s, call(i, m, a), cnt) \xrightarrow{call(i,m,a),c_i:\Sigma,nop}_{\Delta_i^k} ((is_{(m,a)}, q_c), d_c, d_g, q'_s, \epsilon, cnt+1)$$

- Guess call: if $q_s \xrightarrow{call(j,m,a)}_{s'} q'_s$, $1 \leq j \leq n$ and $j \neq i$, then

$$(q, d_c, d_g, q_s, call(j, m, a), cnt) \xrightarrow{call(j,m,a),c_i:\Sigma,nop}_{\Delta_i^k} (q, d_c, d_g, q'_s, \epsilon, cnt+1)$$

- Return: if $q_s \xrightarrow{return(i,m,a)}_{s'} q'_s$, then

$$((fs_{(m,a)}, q_c), d_c, d_g, q_s, return(i, m, a), cnt) \xrightarrow{return(i,m,a),c_i:\Sigma,nop}_{\Delta_i^k} (q_c, d_c, d_g, q'_s, \epsilon, cnt+1)$$

- Guess return: if $q_s \xrightarrow{return(j,m,a)}_{s'} q'_s$, $1 \leq j \leq n$ and $j \neq i$, then

$$(q, d_c, d_g, q_s, return(j, m, a), cnt) \xrightarrow{return(j,m,a),c_i:\Sigma,nop}_{\Delta_i^k} (q, d_c, d_g, q'_s, \epsilon, cnt+1)$$

Channel machine M_i^{ts} is a tuple $(Q_i^{ts}, \{c_i\}, \Sigma, \Lambda, \Delta_i^{ts})$. $Q_i^{ts} = (\{q_c\} \cup (Q_{\mathcal{L}} \times \{q_c\})) \times Val \times Val \times (Q_s \cup \{q_{error}\}) \times (markedVal(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n) \cup \{\epsilon\}))$ is the state set of M_i^{ts}. Each configuration (q, d_c, d_g, q_s, mak) of M_i^{ts} does not contain counters. Δ_i^{ts} is generated from Δ_i^k by ignoring the counter element, i.e., $(q, d_c, d_g, q_s, mak) \xrightarrow{l,g,op}_{\Delta_i^{ts}} (q', d'_c, d'_g, q'_s, mak')$ holds, if there exists cnt, cnt', such that $(q, d_c, d_g, q_s, mak, cnt) \xrightarrow{l,g,op}_{\Delta_i^k} (q', d'_c, d'_g, q'_s, mak', cnt')$.

5.4 Construction of M^k_{n+1} and M^{ts}_{n+1}

Channel machine M^k_{n+1} is a tuple $(Q^k_{n+1}, \{c_{n+1}\}, \Sigma, \Lambda, \Delta^k_{n+1})$, where Q_{n+1}, c_{n+1} and Δ^k_{n+1} are defined as follows:

$Q^k_{n+1} = \{q_r\} \times Val \times Val \times (Q_s \cup \{q_{error}\}) \times (markedVal(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n) \cup \{\epsilon\}) \times \{1, \dots, k\text{-}1\})$ is the state set of M^k_{n+1}.

c_{n+1} is name of the single channel of M^k_{n+1}.

Δ^k_{n+1} is the transition relation of M^k_{n+1}, it is the smallest set of transitions such that $\forall d_c, d_g \in Val$, $q_s \in Q_s$ and $cnt < k$,

- Client *cas*: if $b \in markedVal(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n)$ and $d \in Val$, then

$$(q_{wit}, d, d, q_s, \epsilon, cnt) \xrightarrow{cas(i,x,d[x_{wit}:b]), c_i=\epsilon, nop}_{\Delta^{ts}_i} (q_{wit}, d[x_{wit} : b], d[x_{wit} : b], q_s, b, cnt)$$

- Guess write: if $1 \leq j \leq n \wedge x \in \mathcal{X}_{\mathcal{L}} \wedge a \in \mathcal{D}_{\mathcal{L}}$, then

$$(q_r, d_c, d_g, q_s, \epsilon, cnt) \xrightarrow{op, (\beta,\sharp) \in c_i, c_i[\beta/(\beta,\sharp)]!\beta'}_{\Delta^{ts}_i} (q_r, d_c, d'_g, q_s, \epsilon, cnt)$$

$$(q_r, d_c, d_g, q_s, \epsilon, cnt) \xrightarrow{op, c_i:\Theta, c_i!\beta'}_{\Delta^{ts}_i} (q_r, d_c, d'_g, q_s, \epsilon, cnt)$$

where $\beta = (j', \_, \_)$, $d'_g = d_g[x : a]$, $\beta' = ((j, x, d'_g), \sharp)$, $\Theta = \Sigma \backslash \{((j_1, \_, \_), \sharp) | 1 \leq j_1 \leq n\}$ and $op = write(j, x, d'_g)$.
- Flush items of process 1 to n: if $1 \leq j \leq n$, then for each $x \in \mathcal{D}_{\mathcal{L}}$, $d \in Val$,

$$(q_r, d_c, d_g, q'_s, \epsilon, cnt') \xrightarrow{op, c_i:\Sigma, c_i?(j,x,d)}_{\Delta^{ts}_i} (q_r, d, d_g, q'_s, \epsilon, cnt')$$

$$(q_r, d_c, d_g, q'_s, \epsilon, cnt') \xrightarrow{op, c_i:\Sigma, c_i?((j,x,d),\sharp)}_{\Delta^{ts}_i} (q_r, d, d_g, q'_s, \epsilon, cnt')$$

where $cnt' \leq k$, $q'_s \in Q_s \cup \{q_{error}\}$ and $op = flush(j, x, d)$.
- Guess call: if $q_s \xrightarrow{call(j,m,a)}_{s'} q'_s$ and $1 \leq j \leq n$, then

$$(q_{wit}, d_c, d_g, q_s, call(j, m, a), cnt) \xrightarrow{call(j,m,a), c_i:\Sigma, nop}_{\Delta^{ts}_i} (q_{wit}, d_c, d_g, q'_s, \epsilon, cnt{+}1)$$

- Guess return: if $q_s \xrightarrow{return(j,m,a)}_{s'} q'_s$ and $1 \leq j \leq n$, then

$$(q_{wit}, d_c, d_g, q_s, return(j, m, a), cnt) \xrightarrow{return(j,m,a), c_i:\Sigma, nop}_{\Delta^{ts}_i} (q_{wit}, d_c, d_g, q'_s, \epsilon, cnt{+}1)$$

Channel machine M^{ts}_{n+1} is a tuple $(Q^{ts}_{n+1}, \{c_{n+1}\}, \Sigma, \Lambda, \Delta^{ts}_{n+1})$. $Q^{ts}_{n+1} = \{q_{wit}\} \times Val \times Val \times (Q_s \cup \{q_{error}\}) \times (markedVal(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n) \cup \{\epsilon\}))$ is the state set of M^{ts}_{n+1}. Δ^{ts}_{n+1} is generated from Δ^k_{n+1} by ignoring the counter element, i.e., $(q, d_c, d_g, q_s, mak) \xrightarrow{l,g,op}_{\Delta^{ts}_{n+1}} (q', d'_c, d'_g, q'_s, mak')$ holds, if there exists cnt, cnt', such that $(q, d_c, d_g, q_s, mak, cnt) \xrightarrow{l,g,op}_{\Delta^k_{n+1}} (q', d'_c, d'_g, q'_s, mak', cnt')$.

5.5 Reducing to A Control State Reachability Problem

Let $M_i^{\textit{k-w}}$ ($M_i^{\textit{k-f}}$) be a channel machine that is resulted from M_i^k by replacing its all but write (flush) and *cas* transitions with internal transitions and the remaining *cas* actions can be regarded as write (flush) actions. Let $M_i^{\textit{k-(f,c,r)}}$ be a channel machine that is resulted from M_i^k by replacing its all but flush, *cas*, call and return transitions with internal transitions and the remaining *cas* actions can be regarded as flush actions. Channel machines $M_i^{\textit{ts-w}}$, $M_i^{\textit{ts-f}}$ and $M_i^{\textit{ts-(f,c,r)}}$ are similarly built from M_i^{ts}.

Since a k-trace contains at most k call and return actions, and the first marked item can be guessed and flushed as in t_1 of Fig. 2 (b) without influence subsequent executions, the number of marked items in a k-trace can be always less than k at any time. Let $S = \langle \Sigma_{\mathit{s1}}, \Sigma_{\mathit{s2}} \rangle$, $K_1 = \langle \textit{k-1}, |\mathcal{X}_{\mathcal{L}}|+1 \rangle$, the following lemma states that a control state reachability problem of a (S, K_1)-channel machine is enough to capture the complement problem of k-bounded TSO-to-SC linearizability.

Lemma 4. *There exists a marked violation t of k-bounded TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes from $(p_{\mathit{init}}, d_{\mathit{init}}, \epsilon^n)$ to (p_w, d_w, ϵ^n) in $[\![\mathit{Clt}(\mathcal{L}), \mathit{n+1}]\!]_{\mathit{tso}}$, if and only if $\bigcap_{i=1}^{\mathit{n+1}} T_{(q_i, q_i')}^{(S,K_1)} M_i^{\textit{k-w}} \neq \emptyset$, where for each $1 \leq i \leq \mathit{n+1}$, $q_i = (p_{\mathit{init}}(i), d_{\mathit{init}}, d_{\mathit{init}}, q_{\mathit{is}}, \epsilon, 0)$, $q_i' = (p_w(i), d_w, d_w, q_{\mathit{error}}, \epsilon, |t \uparrow_{(\Sigma_{\mathit{cal}} \cup \Sigma_{\mathit{ret}})}|)$.*

Proof. (Sketch)
This lemma is a direct consequence of the following three claims:

- The first claim states that we can reduce the complement problem of k-bounded TSO-to-SC linearizability to a control state reachability problem of a channel machine which is the production of $M_1^{\textit{k-(f,c,r)}}$ to $M_{\mathit{n+1}}^{\textit{k-(f,c,r)}}$. The *if* direction of this claim is proved by constructing a weak simulation relation between $M_1^{\textit{k-(f,c,r)}} \otimes \ldots \otimes M_{\mathit{n+1}}^{\textit{k-(f,c,r)}}$ and $[\![\mathit{Clt}(\mathcal{L}), \mathit{n+1}]\!]_{\mathit{tso}}$. To prove the *only if* direction, an new *LTS* $[\![\mathit{Clt}(\mathcal{L}), \mathit{n+1}]\!]_{\mathit{tso}}^g$ is generated from $[\![\mathit{Clt}(\mathcal{L}), \mathit{n+1}]\!]_{\mathit{tso}}$. Configurations of $[\![\mathit{Clt}(\mathcal{L}), \mathit{n+1}]\!]_{\mathit{tso}}^g$ extend configuration of $[\![\mathit{Clt}(\mathcal{L}), \mathit{n+1}]\!]_{\mathit{tso}}$ by additionally containing the information about the total store order of the trace. We prove that for each trace t_1 of $[\![\mathit{Clt}(\mathcal{L}), \mathit{n+1}]\!]_{\mathit{tso}}$, we can generate a trace t_2 of $[\![\mathit{Clt}(\mathcal{L}), \mathit{n+1}]\!]_{\mathit{tso}}^g$, and from t_2 we can generate a trace of $M_1^{\textit{k-(f,c,r)}} \otimes \ldots \otimes M_{\mathit{n+1}}^{\textit{k-(f,c,r)}}$ from $(p_{\mathit{init}}, d_{\mathit{init}}, \epsilon^n)$ to (p_w, d_w, ϵ^n).
- The second claim shows that there is a trace t_1 of $M_i^{\textit{k-(f,c,r)}}$ from (q_i, ϵ^n) to (q_i', ϵ^n), if and only if there is a trace t_2 of $M_i^{\textit{k-f}}$ from (q_i, ϵ^n) to (q_i', ϵ^n), where the projection of t_1 to flush actions is equivalent to t_2.
- The third claim shows that there is a trace t_1 of $M_i^{\textit{k-f}}$ from (q_i, ϵ^n) to (q_i', ϵ^n), if and only if there is a trace t_2 of $M_i^{\textit{k-w}}$ from (q_i, ϵ^n) to (q_i', ϵ^n), where t_1 can be generated from t_2 by substitution each write action *write*(i, x, d) to a corresponding flush action *flush*(i, x, d).

The detailed proof of this lemma can be found in Appendix A. □

The following lemma shows that, the complement problem of k-bounded TSO-to-SC linearizability can be further reduced to the control state reachability problem of a (S, K_1)-lossy channel machine, which is the production of $M_1^{\textit{k-w}}$ to $M_{\mathit{n+1}}^{\textit{k-w}}$ (interpreted with lossy channel).

Lemma 5. *There exists a marked violation t of k-bounded TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes from $(p_{init}, d_{init}, \epsilon^n)$ to (p_w, d_w, ϵ^n) in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, if and only if $\bigcap_{i=1}^{n+1} LT^{(S,K_1)}_{(q_i,q'_i)} M^{k\text{-}w}_i \neq \emptyset$, where for each $1 \leq i \leq n{+}1$, $q_i = (p_{init}(i), d_{init}, d_{init}, q_{is}, \epsilon, 0)$, $q'_i = (p_w(i), d_w, d_w, q_{error}, \epsilon, |t \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}|)$.*

Proof. (Sketch)

This lemma follows directly from Lemma 4 and a claim: $T^{(S',K')}_{(q_i,q'_i)} M^{k\text{-}w}_i = LT^{(S',K')}_{(q_i,q'_i)} M^{k\text{-}w}_i$. The $\subseteq$ direction of this claim is obvious, the $\supseteq$ direction is proved by constructing a weak simulation between configurations of lossy channel machine $M^{k\text{-}w}_i$ and lossy channel machine $M^{k\text{-}w}_i$.

The detailed proof of this proposition can be found in Appendix B. □

Since there is only one $(p_{init}, d_{init}, \epsilon^n)$ and a finite number of (p_w, d_w, ϵ^n) in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, thus to decide k-bounded TSO-to-SC linearizability we only need to apply Lemma 5 for a finite number of times and only a finite number of (p_w, d_w, ϵ^n) configurations are concerned. By Lemma 3 and Lemma 5, it is obvious that the k-bounded TSO-to-SC linearizability problem is decidable.

Theorem 1. *The decision problem of k-bounded TSO-to-SC linearizability is decidable.*

The following proposition shows that the k-bounded TSO-to-SC linearizability problem has non-primitive recursive complexity.

Proposition 1. *The decision problem of k-bound TSO-to-SC linearizability has non-primitive recursive complexity.*

Proof. (sketch)

According to [14], it is obvious that the reachability problem of a lossy simple channel system (a subclass of channel machine which has only one channel, uses only ϵ transitions and empty guards, and does not uses substitution before send operation) has non-primitive recursive complexity. The reachability problem of a lossy simple channel system M and configurations s_1, s_2 is to decide whether s_2 is reachable from s_1 in lossy semantics of M.

To prove this proposition, we reduce the reachability problem of a lossy simple channel system to a 3-bounded TSO-to-SC linearizability problem for 2 processes.

The implementation library is presented as a library template that can be instantiated as a specific library for a begin and a end configuration of a lossy simple channel machine. This library has two methods: M_1 and M_2. Given a lossy simple channel machine M and configurations s_1, s_2, the implementation library $\mathcal{L}^M_{(s_1,s_2)}$ uses two processes P_1 and P_2, calling methods M_1 and M_2, respectively, to simulate the behavior of M starting from s_1. If the behavior under simulation reaches s_2, M_1 will stop the simulation and return. Otherwise, M_1 and M_2 will not return.

The abstract library $\mathcal{L}_{pend}$ is a library where all its methods (M_1 and M_2) are pending in any case.

Similarly to [15,16], we can prove that s_2 is reachable from s_1 in lossy semantics of M, if and only if there exists a history $h \in [\![\mathcal{L}^M_{(s_1,s_2)}, 2]\!]_{tso}$ which has three call and return actions, and one of them is a return action. It is easy to see that each history of the abstract library $\mathcal{L}_{pend}$ contains at most two call actions and can not contain return action. Therefore, the existence of such history h represents that $\mathcal{L}_{pend}$ does not 3-bound TSO-to-SC linearize $\mathcal{L}^{M}_{(s_1,s_2)}$ for 2 processes.

The detailed definition of the libraries and the detailed proof of this proposition can be found in Appendix C. $\square$

Let $K_2 = \langle \infty, |\mathcal{X}_{\mathcal{L}}|+1 \rangle$. Similar to Lemma 4, the complement problem of TSO-to-SC linearizability can be reduced to a finite number of control state reachability problems of a channel machine where the amount of marked items in a channel is unbounded, or specifically, a (S, K_2)-channel machine that is the product of $M_1^{\textit{ts-w}}, \ldots,$ $M_{\textit{n+1}}^{\textit{ts-w}}$. Since the number of strong symbol is unbounded, we still do not know whether this problem is decidable or undecidable.

Theorem 2. *The TSO-to-SC standard violation problem can be reduced to a control state reachability problem of a* (S, K_2)*-lossy channel machine, where the number of strong symbol is unbounded.*

6 Conclusion and Future Work

We have shown in this paper that the decision problem of k-bounded TSO-to-SC linearizability is decidable for a concurrent system with $n \geq 1$ processes. The proof method is essentially by a reduction to a control state reachability problem of a lossy channel machine, which is already known to be decidable. To facilitate the reduction, a new process is introduced to use the specific *cas* actions to capture the call and return actions of the original concurrent system. In this way, the complement problem of TSO-to-SC linearizability on the n processes can be transformed to a marked violation problem on the *n+1* processes. Then, a channel machine M_i^k ($1 \leq i \leq$ *n+1*) is constructed to simulate the k-bounded behaviors of the extended concurrent system from the perspective of each process P_i. We then demonstrate that the product of $M_1^{\textit{k-w}}, \ldots, M_{\textit{n+1}}^{\textit{k-w}}$, when interpreted with lossy channels, can characterize the TSO behaviors of the original concurrent system. Furthermore, we show that the k-bounded TSO-to-SC linearizability problem has non-primitive recursive complexity.

Since the notion of k-bounded TSO-to-SC linearizability does not require the size of a store buffer or the length of a trace of a concurrent system to be bounded, it still allows infinite-state behaviors. Hence, our decidability result is non-trivial. It sheds light on developing algorithms for automatically verifying concurrent libraries on relaxed memory models.

We have successfully reduced the decision problem of TSO-to-SC linearizability to a control state reachability problem of a lossy-channel machine with unbounded number of strong symbols. However, the decidability of this problem still remains open. As future work, we would like to pursue this problem further with other possible heuristics. Also we would like to continue investigating the decidability of other correctness conditions of concurrent libraries and programs.

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. et al. (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. Bouajjani, A., Emmi, M., Enea, C., Hamza, J.: Verifying concurrent programs against sequential specifications. In: Felleisen, M., Gardner, P. (eds.) ESOP 2013, pp. 290–309. Springer (2013)
5. Bouajjani, A., Emmi, M., Enea, C., Hamza, J.: On reducing linearizability to state reachability. In: Halldórsson, M.M. et al. (eds.) ICALP 2015, Part II, pp. 95–107. Springer (2015)
6. Bouajjani, A., Emmi, M., Enea, C., Hamza, J.: Tractable Refinement Checking for Concurrent Objects. In: Rajamani, S. K. et al. (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, pp. 87–107. Springer (2012)
8. Gotsman, A., Musuvathi, M., Yang, H.: Show no weakness: Sequentially consistent specifications of TSO libraries. In: Aguilera, M. K. (eds.) DISC 2012, pp. 31–45. Springer (2012)
9. Herlihy, M.P., Wing, J.M.: Linearizability: A correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. **12**(3), 463–492 (1990)
10. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess program. IEEE Transactions on Computers **28**(9), 690–691 (1979)
11. Manson, J., Pugh, W., Adve, S.V.: The Java memory model. In: Palsberg, J., Abadi, M. (eds.) POPL 2005, pp. 378–391. ACM (2005)
12. 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, pp. 391–407. Springer (2009)
13. 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)
14. Schnoebelen, P.: Verifying lossy channel systems has nonprimitive recursive complexity. Information Processing Letters **83**(5), 251–261 (2002)
15. Wang, C., Lv, Y., Wu, P.: TSO-to-TSO Linearizability is Undecidable. Technical Report ISCAS-SKLCS-15-03, State Key Laboratory of Computer Science, ISCAS, CAS (2015), `http://lcs.ios.ac.cn/~lvyi/files/ISCAS-SKLCS-15-03.pdf`
16. Wang, C., Lv, Y., Wu, P.: TSO-to-TSO linearizability is undecidable. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) ATVA 2015. Springer (2015) (to appear)

A Proof of Lemma 4

A.1 Proof Sketch of Lemma 4

Given a finite sequence $l = \alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_k$, we say that the element α_i is left (right) to element α_j, if $i < j$ ($i > j$). We say that α_i is left most element in l if $i = 1$, and α_i is right most element in l if $i = |l|$.

Given a finite sequence l of flush and *cas* actions, let $R_{f \to w}(l)$ be a finite sequence that is generated from l by transforming each *flush*(i, x, d) action to *write*(i, x, d) action.

To prove Lemma 4, we present the following four lemmas. The proof of Lemma 6 and Lemma 7 are given in Appendix A.3 and A.4 respectively.

Lemma 6. *If there exists a marked violation t of k-bounded TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes from $(p_{init}, d_{init}, \epsilon^n)$ to (p_w, d_w, ϵ^n) in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, then $\cap_{i=1}^{n+1} T_{(q_i,q_i')}^{(S,K_1)} M_i^{k\text{-}(f,c,r)} \neq \emptyset$, where for each $1 \leq i \leq n{+}1$, $q_i = (p_{init}(i), d_{init}, d_{init}, q_{is}, \epsilon, 0)$, $q_i' = (p_w(i), d_w, d_w, q_{error}, \epsilon, |t \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}|)$.*

Lemma 7. *If $\cap_{i=1}^{n+1} T_{(q_i,q_i')}^{(S,K_1)} M_i^{k\text{-}w} \neq \emptyset$, where for each $1 \leq i \leq n{+}1$, $q_i = (p_{init}(i), d_{init}, d_{init}, q_{is}, \epsilon, 0)$, $q_i' = (p_w(i), d_w, d_w, q_{error}, \epsilon, a)$, then there exists a marked violation t of k-bounded TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes from $(p_{init}, d_{init}, \epsilon^n)$ to (p_w, d_w, ϵ^n) in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, and $|t \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}| = a$.*

Lemma 8. *$l \in T_{(q,q')}^{(S,K_1)} M_i^{k\text{-}(f,c,r)}$ if and only if $l \uparrow_{\Sigma_f} \in T_{(q,q')}^{(S,K_1)} M_i^{k\text{-}f}$, where $(q = (q_c, d_{init}, d_{init}, q_{is}, \epsilon, 0) \wedge 1 \leq i \leq n) \vee (q = (q_{wit}, d_{init}, d_{init}, q_{is}, \epsilon, 0) \wedge i = n{+}1)$ and $q' = (\_, d, d, q_{error}, \epsilon, |l \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}|)$ for some $d \in Val$.*

Proof. This lemma holds because in M_i^k, each marked *cas* actions coincides with a call or return action. So it is safe to ignore call and return actions. □

Lemma 9. *$l \in T_{(q,q')}^{(S,K_1)} M_i^{k\text{-}f}$ if and only if $R_{f \to w}(l) \in T_{(q,q')}^{(S,K_1)} M_i^{k\text{-}w}$, where $(q = (q_c, d_{init}, d_{init}, q_{is}, \epsilon, 0) \wedge 1 \leq i \leq n) \vee (q = (q_{wit}, d_{init}, d_{init}, q_{is}, \epsilon, 0) \wedge i = n{+}1)$ and $q' = (\_, d, d, q_{error}, \epsilon, a)$ for some $d \in Val$, and a is the number of internal actions derived from call or return transition in l.*

Proof. This lemma holds because for a perfect channel machine, the sequences of input (write actions) is always equal to the sequences of output (flush actions). □

With these lemmas, we can now prove Lemma 4.

Lemma 4. *There exists a marked violation t of k-bounded TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes from $(p_{init}, d_{init}, \epsilon^n)$ to (p_w, d_w, ϵ^n) in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, if and only if $\bigcap_{i=1}^{n+1} T_{(q_i,q_i')}^{(S,K_1)} M_i^{k\text{-}w} \neq \emptyset$, where for each $1 \leq i \leq n{+}1$, $q_i = (p_{init}(i), d_{init}, d_{init}, q_{is}, \epsilon, 0)$, $q_i' = (p_w(i), d_w, d_w, q_{error}, \epsilon, |t \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}|)$.*

Proof. Lemma 4 is a direct consequence of Lemma 6, Lemma 7, Lemma 8 and Lemma 9. □

A.2 Construction of $[\![Clt(\mathcal{L}), n{+}1]\!]^g_{tso}$

It is quite hard to build a weak simulation relation between configurations of $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$ and configurations of $\cap_{i=1}^{n+1} M_i^{k\text{-}(f,c,r)}$. This is because that for a configuration (p, d, u) of $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, more than one process may be possible to do a flush action. Therefore, the total store orders of traces from a configuration is not fixed in this case. While for a configuration of $\cap_{i=1}^{n+1} M_i^{k\text{-}(f,c,r)}$, the process id of the next flush action is nearly fixed because the channel already contains items which reflect total store order, and such items must be flush in a fixed FIFO order.

To deal with this problem, a intermediate transition system is introduced, whose configuration extends configurations of $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$ and contains the total store order of one trace. Formally, given

- library $\mathcal{L} = (\mathcal{X}_{\mathcal{L}}, \mathcal{M}, \mathcal{D}_{\mathcal{L}}, Q_{\mathcal{L}}, \rightarrow_{\mathcal{L}})$, positive integer n,
- a deterministic finite state automaton $\mathcal{A}_{Spec} = (Q_s, \Sigma_s, \rightarrow_s, q_{is})$ that accepts *history* $([\![\mathcal{L}', n]\!]_{sc})$ and transition relation $\rightarrow'_s$ as in section 5.3.

The extended semantics of $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$ is defined as an *LTS* $[\![Clt(\mathcal{L}), n{+}1]\!]^g_{tso} = (Conf_e, \Sigma_e, \rightarrow_e, InitConf_e)$, where $\Sigma_e = \Sigma_{tso}$, and $Conf_e, \rightarrow_e, InitConf_e$ are defined as follows.

Each configuration of $Conf_e$ is a tuple $(p, d, u, q_s, mak, flag, g)$, where,

- (p, d, u) is a configuration of $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$,
- $q_s \in Q_s$, $mak \in markedVal(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n)$,
- $g \in (\Sigma_{e0} \cup \Sigma_{e1} \cup \Sigma_{e2} \cup \Sigma_{e3})^*$, where Σ_{e0}, Σ_{e1}, Σ_{e2} and Σ_{e3} are defined below. g should satisfies some requirements shown below. $flag \in \{T, F\}$ is used to denote whether g has been initialized.

The four alphabets of $\Sigma_{e0}, \Sigma_{e1}, \Sigma_{e2}, \Sigma_{e3}$ is defined as follows:

- $\Sigma_{e0} = \{(i, x, d) | 1 \le i \le n{+}1, x \in \mathcal{X}_{\mathcal{L}} \cup x_{wit}, d \in Val\}$ represents the items in total store order of a trace that are not used now and will be flushed later than any item in current buffer.
- $\Sigma_{e1} = \{(i, x, d)' | (i, x, d) \in \Sigma_{e0}\}$ represents items in the total store order of a trace that are not used now and will be flushed earlier than some item in buffer.
- $\Sigma_{e2} = \{(i, x, d)'' | (i, x, d) \in \Sigma_{e0}\}$ represents items in the total store order of a trace that are already inserted into buffer and not flushed yet.
- $\Sigma_{e3} = \{(i, x, d)''' | (i, x, d) \in \Sigma_{e0}\}$ represents items in the total store order of a trace that have already been flushed out from buffer.

g stores the total store order of a trace. It is a concatenation of sequences l_{g1}, l_{g2} and l_{g3}. $l_{g1} \in \Sigma_{e0}^*$ represents the sequences of items that have not been used and will be flushed later than any item in current buffer. $l_{g2} \in (\Sigma_{e1} \cup \Sigma_{e2})^*$ represents the sequences of items that either in concurrent buffer, or items not in current buffer but will be flushed earlier than some item in concurrent buffer. $l_{g3} \in \Sigma_{e3}^*$ represents the sequences of items that have already been flushed.

Moreover, let Σ_i be the items of process i, $\Sigma_{(i,x)}$ be the items of process i and memory location x,

- If $l_{g2} \neq \epsilon$, then $l_{g2}(1) \in \Sigma_{e2}$. For each i, $l_{g2} \uparrow_{\Sigma_i} \in \Sigma_{e1}^* \cdot \Sigma_{e2}^*$.
- For each i, j, if $g \uparrow_{(\Sigma_i \cup \Sigma_{e2})} (j) = (i, x, d)''$ with $d(x) = a$, then $u(i)(j) = (i, x, a)$, and vice versa.
- Let g' be the sequence generated from g by discarding all the $'$ symbols of each item in g. If $g'(|g'|) = (i_1, x_1, d_1)$, then $d_1 = d[x_1 : d_1(x_1)]$. For each i, if $g'(i) = (i_2, x_2, d_2)$, $g'(i{+}1) = (i_3, x_3, d_3)$, then $d_2 = d_3[x_2 : d_2(x_2)]$.

The initial configuration $InitConf_e$ is a tuple $(p_{init}, d_{init}, \epsilon^{n+1}, q_{is}, \epsilon, F, \epsilon)$.
The transition relation $\rightarrow_e$ is defined as follows:

- Initial transition: the first transition from $InitConf_e$ is to guess the tuple g: $(p_{init}, d_{init}, \epsilon^{n+1}, q_{is}, \epsilon, F, \epsilon) \xrightarrow{\epsilon}_e (p_{init}, d_{init}, \epsilon^{n+1}, q_{is}, \epsilon, T, g)$ for g being a sequence defined above.
- τ and read transitions: $(p, d, u, q_s, \epsilon, T, g) \xrightarrow{\alpha}_e (p', d', u', q_s, \epsilon, T, g)$ with τ or read action α, if $(p, d, u) \xrightarrow{\alpha}_{tso} (p', d', u') \wedge q_s \neq q_{error}$.
- Write transitions: $(p, d, u, q_s, \epsilon, T, g) \xrightarrow{write(i,x,a)}_e (p', d', u', q_s, \epsilon, T, g')$, if $(p, d, u) \xrightarrow{write(i,x,a)}_{tso} (p', d', u')$, $q_s \neq q_{error}$, and one of the following conditions holds: (1) $l_{g2} \uparrow_{\Sigma_{(i,x)}}$ contains at least one item in Σ_{e1}, and g' is generated from g by transforming the right most item of $l_{g2} \uparrow_{(\Sigma_{e1} \cap \Sigma_{(i,x)})}$, $(i, x, d_1)'$ for some d_1 with $d_1(x) = a$, to $(i, x, d_1)''$, (2) $l_{g2} \uparrow \Sigma_{(i,x)}$ does not contain any item of Σ_{e1}, and g' is generated from g by translate the right most item of $l_{g1} \uparrow_{(\Sigma_{e0} \cap \Sigma_{(i,x)})}$, (i, x, d_1) for some d_1 with $d_1(x) = a$, to $(i, x, d_1)''$, and mark all the items which are right to this item in l_{g1} with $'$ symbol.
- *Cas* transitions: $(p, d, u, q_s, \epsilon, T, g) \xrightarrow{cas(i,x,a,b)}_e (p', d', u', q_s, \epsilon, T, g')$, if $(p, d, u) \xrightarrow{cas(i,x,a,b)}_{tso} (p', d', u')$, $q_s \neq q_{error}$, and one of the following conditions holds: (1) $l_{g2} \neq \epsilon$, l_{g2} ends with $(i, x, d')'$, and g' is generated from g by changing this $(i, x, d')'$ item to $(i, x, d')'''$, (2) $l_{g2} = \epsilon$, l_{g1} ends with (i, x, d'), and g' is generated from g by changing this (i, x, d') item to $(i, x, d')'''$.
- Flush transitions: $(p, d, u, q_s, \epsilon, T, g) \xrightarrow{flush(i,x,a)}_e (p', d', u', q_s, mak, T, g')$, if l_{g2} ends with $(i, x, d')''$, and g' is generated from g by transforming this $(i, x, d')''$ item to $(i, x, d')'''$. Moreover, if $i = n{+}1 \wedge x = x_{wit} \wedge d'(x_{wit}) = \alpha \in markedVal(\mathcal{M}, \mathcal{D}_{\mathcal{L}}, n)$, then $mak' = \alpha$. Otherwise, if $1 \leq i \leq n$, then $mak' = \epsilon$.
- Call and return transitions: $(p, d, u, q_s, call(i, m, a), T, g) \xrightarrow{call(i,m,a)}_e (p', d', u', q_s', \epsilon, T, g)$, if $(p, d, u) \xrightarrow{call(i,m,a)}_{tso} (p', d', u')$ and $q_s \xrightarrow{call(i,m,a)}_{s'} q_s'$. Similarly, $(p, d, u, q_s, return(i, m, a), T, g) \xrightarrow{return(i,m,a)}_e (p', d', u', q_s', \epsilon, T, g)$, if $(p, d, u) \xrightarrow{return(i,m,a)}_{tso} (p', d', u')$ and $q_s \xrightarrow{return(i,m,a)}_{s'} q_s'$.

A.3 Proof of Lemma 6

Given a trace t and a sequence g which satisfies requirement in Appendix A.2, we say that g contains the total store order of t, if: let t' be the projection of t to write and *cas* actions. If $t'(1) = write(j, x, b)$ or $cas(j, x, a, b)$, then $g(|t'|) = (j, x, d)'''$, where $d = d_{init}[x : b]$. For each $i > 1$, if $t'(i) = write(j, x, b)$ or $cas(j, x, a, b)$, and $g(|t'| - i + 2) = (j', y, d')'''$, then $g'(|t'| - i + 1) = (i, x, d)'''$, where $d = d'[x : b]$.

The following lemma states that if $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$ contains a witness violation of k-bounded TSO-to-SC linearizability, then *LTS* $[\![Clt(\mathcal{L}), n{+}1]\!]^g_{tso}$ also contains a witness violation of k-bounded TSO-to-SC linearizability.

Lemma 10. *If trace t is a witness violation of k-bounded TSO-to-SC linearizability from $(p_{init}, d_{init}, \epsilon^{n+1})$ to $(p_w, d_w, \epsilon^{n+1})$ in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, then t is also a witness violation of k-bounded TSO-to-SC linearizability from $(p_{init}, d_{init}, \epsilon^{n+1}, q_{is}, \epsilon, F, \epsilon)$ to $(p_w, d_w, \epsilon^{n+1}, q_{error}, \epsilon, T, g)$ in $[\![Clt(\mathcal{L}), n{+}1]\!]^g_{tso}$, where g contains the total store order of t.*

Proof. The *if* direction is obvious, since it is obvious that $trace([\![Clt(\mathcal{L}), n{+}1]\!]^g_{tso}) \subseteq trace([\![Clt(\mathcal{L}), n{+}1]\!]_{tso})$.

To prove the *only if* direction, for each path of $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, we generate a path of $[\![Clt(\mathcal{L}), n{+}1]\!]^g_{tso}$ step by step.

Assume $(p_{init}, d_{init}, \epsilon^{n+1}) \xrightarrow{\alpha_1}_t (p_1, d_1, u_1) \ldots \xrightarrow{\alpha_w}_t (p_w, d_w, u_w)$ is the path of standard violation t in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, where $u_w = \epsilon^{n+1}$. For each configuration (p_i, d_i, u_i) we construct another configuration $(p_i, d_i, u_i, q^i_s, mak_i, T, g_i)$, where

- q^i_s is generated from q_{is} by call and return actions in $\alpha_1 \cdot \ldots \cdot \alpha_i$.
- mak_i is β if $\alpha_{i+1} = \beta$ and β is a call or return action, otherwise, it is ϵ.
- Let g'_i be generated from g_i by discarding $'$ symbols of each item of g. Then $g'_1 = \ldots = g'_w$. Let $g_0 = g'_1$.
- l_{gi3} contains all the items that has been flushed when reaching (p_i, d_i, u_i). Recall that for each j_1, j_2, if $g_i \uparrow_{(\Sigma_{j_1} \cup \Sigma_{e2})} (j_2) = (j_1, x, d)''$ with $d(x) = a$, then $u(j_1)(j_2) = (j_1, x, a)$, and vice versa. Let ind_1 be the minimal index of Σ_{e2} item in g_i and ind_2 be the minimal index of Σ_{e3} item in g_i. Each not mentioned item $g_i(j)$, where $ind_1 < j < ind_2$, belongs to Σ_{e1}. The remaining items of g_i belong to l_{gi1}.

It is not hard to prove that $(p_{init}, d_{init}, \epsilon^{n+1}, q_{is}, \epsilon, F, \epsilon) \xrightarrow{\epsilon}_e (p_{init}, d_{init}, \epsilon^{n+1}, q_{is}, \epsilon, T, g_0)$, and for each i, $(p_i, d_i, u_i, q^i_s, mak_i, T, g_i) \xrightarrow{\alpha_{i+1}}_e (p_{i+1}, d_{i+1}, u_{i+1}, q^{i+1}_s, mak_{i+1}, T, g_{i+1})$. Therefore, $\alpha_1 \cdot \ldots \cdot \alpha_w$ is also a trace of $[\![Clt(\mathcal{L}), n{+}1]\!]^g_{tso}$. $\square$

Given a sequence l of flush, call and return actions and a sequence g as defined in Appendix A.2, we say that l is consistent with g, if: let l_f be the projection of l to flush actions and g' be generated from g by discarding $'$ symbol of each item in g, then for each i, $l_f(i) = \mathit{flush}(i, x, d)$, if and only if $g(|g| - i + 1) = (i, x, d)$.

The following lemma states that *LTS* $[\![Clt(\mathcal{L}), n{+}1]\!]^g_{tso}$ has a witness violation of k-bounded TSO-to-SC linearizability implies a control state reachability problem of (S, K_1)-channel machine $M^{k\text{-}(f,c,r)}_1 \otimes \ldots \otimes M^{k\text{-}(f,c,r)}_{n+1}$.

Lemma 11. *If trace t is a witness violation of k-bounded TSO-to-SC linearizability from $(p_{init}, d_{init}, \epsilon^{n+1}, q_{is}, \epsilon, F, \epsilon)$ to $(p_w, d_w, \epsilon^{n+1}, q_{error}, \epsilon, T, g)$ in $[\![Clt(\mathcal{L}), n{+}1]\!]^g_{tso}$ with sequence g which contains the total store order of t, then there exists a sequence l, such that for each process id $1 \leq i \leq n{+}1$, $l \in T^{(S,K_1)}_{(q_i, q'_i)} M^{k\text{-}(f,c,r)}_i$, where $q_i = (p_{init}(i), d_{init}, d_{init}, q_{is}, \epsilon)$, $q'_i = (p_w(i), d_w, d_w, q_{error}, \epsilon)$, and l is consistent with g.*

Proof. This lemma is proved by constructing a weak simulation between configurations of $[\![Clt(\mathcal{L}), n{+}1]\!]^g_{iso}$ in t and configurations of (S, K_1)-channel machine $M_1^{k\text{-}(f,c,r)} \otimes \ldots \otimes M_{n+1}^{k\text{-}(f,c,r)}$.

Assume $(p_{init}, d_{init}, \epsilon^{n+1}, q_{is}, \epsilon, F, \epsilon) \xrightarrow{\alpha_1}_e \ldots \xrightarrow{\alpha_w}_e (p_w, d_w, u_w, q_{error}, \epsilon, T, g)$ is the path of standard violation t in $[\![Clt(\mathcal{L}), n{+}1]\!]^g_{iso}$ and $u_w = \epsilon^{n+1}$. Let $(p, d, u, r, q_s, mak, T, g)$ be the *(v+1)-th* configuration of the path. Let $((cs_1, \ldots, cs_{n+1}), (c_1, \ldots, c_{n+1}))$ be a configuration of $M_1^{k\text{-}(f,c,r)} \otimes \ldots \otimes M_{n+1}^{k\text{-}(f,c,r)}$, and for each i, $cs_i = (q_i, d_{ci}, d_{gi}, q_s^i, mak_i, cnt_i)$. A relation $\sim$ is defined as follows: $(p, d, u, r, q_s, mak, T, g) \sim ((cs_1, \ldots, cs_{n+1}), (c_1, \ldots, c_{n+1}))$, if for each process id i,

- $p(i) = q_i$, $d = d_{ci}$, $q_s = q_s^i$, $mak = mak_i$, and d_{gi} is generated from d_{ci} by doing all the updates in $c_i(|c_i|), \ldots, c_i(1)$.
- $cnt_1 = \ldots = cnt_{n+1}$. And cnt_1 is the number of call and return actions in $\alpha_1 \cdot \ldots \cdot \alpha_v$.
- If $l_{g2} \uparrow_{(\Sigma_{e2} \cap \Sigma_i)} \neq \epsilon$, then let ind_1, ind_2 be the index of the leftmost $\Sigma_{e2} \cap \Sigma_i$ item on g and the rightmost item on l_{g2} respectively, let g' be generated from g by discarding $'$ symbols of each item in g. Assume $g' = (i_1, x_1, d_1) \cdot (i_2, x_2, d_2) \cdot \ldots$. Then c_i contains $ind_2 - ind_1 + 1$ items, and $\forall 1 \leq j \leq ind_2 - ind_1 + 1$, $c_i(ind_2 - ind_1 - j + 2) = (i_{(ind_2-j+1)}, x_{(ind_2-j+1)}, d_{(ind_2-j+1)})$ or $((i_{(ind_2-j+1)}, x_{(ind_2-j+1)}, d_{(ind_2-j+1)}), \sharp)$.
- If $l_{g2} \uparrow_{(\Sigma_{e2} \cap \Sigma_i)} = \epsilon$ and $l_{g2} \uparrow_{(\Sigma_{e1} \cap \Sigma_i)} \neq \epsilon$, then let ind_1, ind_2 be the index of the rightmost $\Sigma_{e1} \cap \Sigma_i$ item on g and the rightmost item on l_{g2} respectively, let g' be generated from g by discarding $'$ symbols of each item in g. Assume $g' = (i_1, x_1, d_1) \cdot (i_2, x_2, d_2) \cdot \ldots$. Then c_i contains $ind_2 - ind_1$ items, and $\forall 1 \leq j \leq ind_2 - ind_1$, $c_i(ind_2 - ind_1 - j + 1) = (i_{(ind_2-j+1)}, x_{(ind_2-j+1)}, d_{(ind_2-j+1)})$ or $((i_{(ind_2-j+1)}, x_{(ind_2-j+1)}, d_{(ind_2-j+1)}), \sharp)$.
- If $l_{g2} \uparrow_{(\Sigma_{e2} \cap \Sigma_i)} = \epsilon$ and $l_{g2} \uparrow_{(\Sigma_{e1} \cap \Sigma_i)} = \epsilon$, then let ind_1, ind_2 be the index of the leftmost item on l_{g2} and the rightmost item on l_{g2} respectively, let g' be generated from g by discarding $'$ symbols of each item in g. Assume $g' = (i_1, x_1, d_1) \cdot (i_2, x_2, d_2) \cdot \ldots$. Then c_i contains $ind_2 - ind_1 + 1$ items, and $\forall 1 \leq j \leq ind_2 - ind_1 + 1$, $c_i(ind_2 - ind_1 - j + 2) = (i_{(ind_2-j+1)}, x_{(ind_2-j+1)}, d_{(ind_2-j+1)})$ or $((i_{(ind_2-j+1)}, x_{(ind_2-j+1)}, d_{(ind_2-j+1)}), \sharp)$.

Given $(p, d, u, r, q_s, mak, T, g) \sim ((cs_1, \ldots, cs_{n+1}), (c_1, \ldots, c_{n+1}))$ defined as above, we say that a item $c_i(j)$ has index *ind* in g, if during above construction $c_i(j)$ is generated from $g'(ind)$.

It remains to prove that, if $(p, d, u, r, q_s, mak, T, g) \sim ((cs_1, \ldots, cs_{n+1}), (c_1, \ldots, c_{n+1}))$ holds, $(p, d, u, r, q_s, mak, T, g) \xrightarrow{\alpha_{v+1}}_e (p', d', u', r', q'_s, mak', T, g')$ and $(p', d', u', r', q'_s, mak', T, g')$ is the *v+2-th* configuration of the path of t, then there exists $cs'_1, \ldots, cs'_{n+1}, c'_1, \ldots, c'_{n+1}$ and β_{v+1}, such that $((cs_1, \ldots, cs_{n+1}), (c_1, \ldots, c_{n+1})) \xrightarrow{\beta_{v+1}}{}^{*}_{M_i^{k\text{-}(f,c,r)}} (cs'_1, \ldots, cs'_{n+1}), (c'_1, \ldots, c'_{n+1}))$, $(p', d', u', r', q'_s, mak', T, g') \sim ((cs'_1, \ldots, cs'_{n+1}), (c'_1, \ldots, c'_{n+1}))$ holds, and the flush, call and return actions in α_{v+1} is same to that in β_{v+1}. Assume for each i, $cs'_i = (q'_i, d'_{ci}, d'_{gi}, q_s^{i'}, mak'_i, cnt'_i)$.

- When α_{v+1} is a τ or *read* action, it is obvious to see that $\beta_{v+1} = \epsilon$ and this holds trivially.
- When α_{v+1} is a call or return action, it is obvious to see that $\beta_{v+1} = \alpha_{v+1}$ and this holds trivially.

- When α_{v+1} is a write actions of process i, $\beta_{v+1} = \epsilon$, and the channels are changed as follows:
  - If $l_{g2} \uparrow_{(\Sigma_{e1} \cap \Sigma_i)} \neq \epsilon$, then let ind_1 be the index of the right most $\Sigma_{e1} \cap \Sigma_i$ item on g, let ind_2 be the index of $c_i(1)$ in g if $c_i \neq \epsilon$, or otherwise the index of the left most item of l_{g3}. c_i' is generated from c_i by putting updates of $g(ind_2 + 1), \ldots, g(ind_1)$ into c_i. During this process a write and then several guess write actions happen. For channel $j \neq i$, $c_j' = c_j$.
  - If $l_{g2} \uparrow_{(\Sigma_{e1} \cap \Sigma_i)} = \epsilon$,
    - For channel i. Let ind_1 be the index of the right most Σ_i item of l_{g1} in g, let ind_2 be the index of right most item of l_{g1} in g. c_i' is generated from c_i by putting updates of $g(ind_2), \ldots, g(ind_1)$ into c_i. During this process several guess write and then a write actions happen.
    - For channel $j \neq i$. If $l_{g2} \uparrow_{\Sigma_j} = \epsilon$ holds. Let ind_1 be the index of the right most Σ_i item of l_{g1} in g. Let ind_2 be the index of right most item of l_{g1} in g. Let sequence $g' = g(ind_1) \cdot \ldots \cdot g(ind_2)$.
    If $g' \uparrow_{\Sigma_j} \neq \epsilon$, let ind_3 be the index of right most item of $g' \uparrow_{\Sigma_j}$ in g, and c_j' is generated from c_j by putting updates of $g(ind_2), \ldots, g(ind_3)$. During this process several guess write actions happen.
    Otherwise, if $g' \uparrow_{\Sigma_j} = \epsilon$, c_j' is generated from c_j by putting updates of $g(ind_2), \ldots, g(ind_1)$. During this process several guess write actions happen.
    - For channel $j \neq i$. If $l_{g2} \uparrow_{\Sigma_j} \neq \epsilon$ holds, $c_j' = c_j$.
- When α_{v+1} is a flush action of process i, then $\beta_{v+1} = \alpha_{v+1}$. The channels are changed as follows:
  - For process $j \neq i$, c_j' is generated from c_j by discarding the right most item of c_j. During this process a flush action happen.
  - For process i, the channel c_i' is changed as follows:
    - If $|l_{g2} \uparrow_{(\Sigma_{e2} \cap \Sigma_i)}| \geq 2$, then c_i' is generated from c_i by discarding the right most item. During this process a flush action happen.
    - Otherwise, $|l_{g2} \uparrow_{(\Sigma_{e2} \cap \Sigma_i)}| = 1$.
    If $l_{g2} \uparrow_{(\Sigma_{e1} \cap \Sigma_i)} \neq \epsilon$, let ind_1 be the index of the right most $\Sigma_{e1} \cap \Sigma_i$ item in g, otherwise, let ind_1 be the index of the right most item of l_{g1} in g. Let ind_2 be the index of the $\Sigma_{e2} \cap \Sigma_i$ item in g. c_i' is generated from c_i be putting updates of $g(ind_2 - 1), \ldots, g(ind + 1)$ and discarding the right most item of c_i. During this process several guess write actions and a flush action happen.
- When α_{v+1} is a $cas(i, x, val)$ action of process i, then $\beta_{v+1} = flush(i, x, val)$. The channels are changed as follows:
  - If $l_{g2} = \epsilon$. For process $j \neq i$, $c_j' = c_j = \epsilon$, and during transition the update (i, x, val) need to be inserted into c_j' by a guess write action and then flushed our of c_j' using a flush action. For process i, $c_i' = c_i = \epsilon$, and during this process a *cas* action happen.
  - If $l_{g2} \neq \epsilon$. For process $j \neq i$, c_j' is generated from c_j by discarding the right most item using a flush action. For process i, the channel c_i' is generated as follows:

- If $|l_{g2} \uparrow_{(\Sigma_{el} \cap \Sigma_i)}| \geq 2$, let ind_1 be the index of the second right most $\Sigma_{el} \cap \Sigma_i$ item in g, let ind_2 be the index of the right most $\Sigma_{el} \cap \Sigma_i$ item in g. c'_i is generated from c_i by putting the updates of $g(ind_2 - 1), \ldots, g(ind + 1)$ into c_i. During this process several guess write action happen.
- If $l_{g2} \uparrow_{(\Sigma_{el} \cap \Sigma_i)} = \epsilon$, let ind_1 be the index of the right most item of l_{g1} in g, let ind_2 be the index of the $\Sigma_{el} \cap \Sigma_i$ item in g. c'_i is generated from c_i by putting the updates of $g(ind_2 - 1), \ldots, g(ind + 1)$ into c_i. During this process several guess write actions happen.

It is not hard to prove that at each time, a item is in a channel c_i of some process i, if at least this item is in l_{g2}. Therefore, It can be seen that for each configuration $(p, d, u, r, q_s, mak, T, g)$ of t, g contains at most *k-1* marked items. Therefore, each c satisfies strong symbol restriction (S, K_1). □

With above two lemmas we can now prove Lemma 6.

Lemma 6. *If there exists a marked violation t of k-bounded TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes from $(p_{init}, d_{init}, \epsilon^n)$ to (p_w, d_w, ϵ^n) in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, then $\cap_{i=1}^{n+1} T_{(q_i, q'_i)}^{(S,K_1)} M_i^{k\text{-}(f,c,r)} \neq \emptyset$, where for each $1 \leq i \leq n{+}1$, $q_i = (p_{init}(i), d_{init}, d_{init}, q_{is}, \epsilon, 0)$, $q'_i = (p_w(i), d_w, d_w, q_{error}, \epsilon, |t \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}|)$.*

Proof. Lemma 6 is a direct consequence of Lemma 10 and Lemma 11. □

A.4 Proof of Lemma 7

Lemma 7. *If $\cap_{i=1}^{n+1} T_{(q_i, q'_i)}^{(S,K_1)} M_i^{k\text{-}w} \neq \emptyset$, where for each $1 \leq i \leq n{+}1$, $q_i = (p_{init}(i), d_{init}, d_{init}, q_{is}, \epsilon, 0)$, $q'_i = (p_w(i), d_w, d_w, q_{error}, \epsilon, a)$, then there exists a marked violation t of k-bounded TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes from $(p_{init}, d_{init}, \epsilon^n)$ to (p_w, d_w, ϵ^n) in $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, and $|t \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}| = a$.*

Proof. Since $\cap_{i=1}^{n+1} T_{(q_i, q'_i)}^{(S,K_1)} M_i^{k\text{-}(f,c,r)} \neq \emptyset$, there is a path $(cs_1^0, \ldots, cs_{n+1}^0), (c_1^0, \ldots, c_{n+1}^0)) \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_w} (cs_1^w, \ldots, cs_{n+1}^w), (c_1^w, \ldots, c_{n+1}^w))$ of $M_1^{k\text{-}(f,c,r)} \otimes \ldots \otimes M_{n+1}^{k\text{-}(f,c,r)}$, such that for each process id i, $cs_i^0 = (p_{init}(i), d_{init}, d_{init}, q_{is}, \epsilon)$, $c_i^0 = \epsilon$, $cs_i^w = (p_w(i), d_w, d_w, q_{error}, \epsilon)$ and $cs_i^w = \epsilon$. Let $cs_i^j = (q_i^j, d_{ci}^j, d_{gi}^j, q_{si}^j, mak_i^j, cnt_i^j)$ for each process id i.

We prove this lemma by constructing a weak simulation between configuration of $M_1^{k\text{-}(f,c,r)} \otimes \ldots \otimes M_{n+1}^{k\text{-}(f,c,r)}$ and configuration of $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$.

A relation $\sim$ is defined as follows: given configuration $((cs_1^v, \ldots, cs_{n+1}^v), (c_1^v, \ldots, c_{n+1}^v))$ for the *v+1-th* configuration of the path, and a configuration (p, d, u) of $[\![Clt(\mathcal{L}), n{+}1]\!]_{tso}$, $((cs_1^v, \ldots, cs_{n+1}^v), (c_1^v, \ldots, c_{n+1}^v)) \sim (p, d, u)$, if,

- For each process id i, $q_i^v = p(i)$, $d_{ci}^v = d$.
- For each process id i_1, i_2, $q_{si_1}^v = q_{si_2}^v$, $mak_{i_1}^v = mak_{i_1}^v$, $cnt_{i_1}^v = cnt_{i_2}^v$.
- Let $c_i^{v'}$ be generated from c_i^v by discarding items of all but process i. Then for each ind, $u(ind) = (x, a)$, if and only if $c_i^{v'}(ind) = (i, x, val)$ or $((i, x, val), \sharp)$ for some val where $val(x) = a$.

It remains to prove that if $((cs_1^v,\dots,cs_{n+1}^v),(c_1^v,\dots,c_{n+1}^v))\sim(p,d,u)$ and $((cs_1^v,\dots,cs_{n+1}^v),(c_1^v,\dots,c_{n+1}^v))\xrightarrow{\alpha_{v+1}}((cs_1^{v+1},\dots,cs_{n+1}^{v+1}),(c_1^{v+1},\dots,c_{n+1}^{v+1}))$, then one of the following two cases holds:

- Case 1: there exists configuration (p',d',u'), such that $((cs_1^{v+1},\dots,cs_{v+1}^{n+1}),(c_1^{v+1},\dots,c_{n+1}^{v+1}))\sim(p',d',u')$, $(p,d,u)\xrightarrow{\beta_{v+1}}_{tso}(p',d',u')$, the flush, call and return action in α_{v+1} are same to that in β_{v+1},
- Case 2: $((cs_1^{v+1},\dots,cs_{n+1}^{v+1}),(c_1^{v+1},\dots,c_{n+1}^{v+1}))\sim(p,d,u)$.

We prove this by considering all kinds of transition label α_{v+1},

- If α_{v+1} is a internal action derived from a τ or read action of some process, then $\beta_{v+1}=\epsilon$ and case 1 holds trivially.
- When α_{v+1} is a call or return action, case 1 holds trivially.
- If α_{v+1} is a internal action derived from a $\mathit{write}(i,x,\mathit{val})$ transition of $M_i^{k\text{-}(f,c,r)}$, then $\beta_{v+1}=\epsilon$, case 1 holds, (p',d',u') is generated from (p,d,u) by a write transition and $u'(i)=(x,\mathit{val}(x))\cdot u(i)$.
- If α_{v+1} is a internal action derived from a guessing write transition of $M_i^{k\text{-}(f,c,r)}$, then it is obvious that case 2 holds.
- When α_{v+1} is a flush action derived from a $\mathit{flush}(i,x,\mathit{val})$ transition of $M_i^{k\text{-}(f,c,r)}$, then $\beta_{v+1}=\alpha_{v+1}$, case 1 holds, (p',d',u') is generated from (p,d,u) by a flush transition and $u(i)=u'(i)\cdot(x,\mathit{val}(x))$.
- When α_{v+1} is a flush action from a $\mathit{cas}(i,x,\mathit{val})$ transition of $M_i^{k\text{-}(f,c,r)}$, then $\beta_{v+1}=\alpha_{v+1}$, case 1 holds and (p',d',u') is generated from (p,d,u) by a *cas* transition.

Moreover, the counter tuples (cnt_i^j) in cs_i^j guarantee that number of call and flush actions in this path is less or equal than k. Therefore, $\beta_1\cdot\ldots\cdot\beta_w$ is a marked violation of k-bounded TSO-to-SC linearizability. □

B Proof of Lemma 5

A configuration $((q,d_c,d_g,q_s,\mathit{mak},\mathit{cnt}),c)$ of $M_i^{k\text{-}w}$ is called standard, if either $c=\epsilon\wedge d_c=d_g$, or $c\neq\epsilon$, $c(1)$ is a strong symbol and $c(1)=(\_,\_,d_g)$ or $((\_,\_,d_g),\sharp)$. It is obvious if a path of (S,K_1)-(lossy) channel machine starts from a standard configuration, then each configuration on this path is standard.

The following lemma shows that there is a weak simulation between configurations of (S,K_1)-channel machine $M_i^{k\text{-}w}$ and configurations of (S,K_1)-lossy channel machine $M_i^{k\text{-}w}$.

Lemma 12. *Given standard configuration $((p_1,d_{c1},d_{g1},q_s^1,\mathit{mak}_1,\mathit{cnt}_1),c_1)$, if $c_1\preceq_S^{K_1}c_1'$ and $((p_1,d_{c1},d_{g1},q_s^1,\mathit{mak}_1,\mathit{cnt}_1),c_1)\xrightarrow{\alpha}_{(M_i^{k\text{-}w},S,K_1)}((p_2,d_{c2},d_{g2},q_s^2,\mathit{mak}_2,\mathit{cnt}_2),c_2)$, then there exists c_2' and β, such that $c_2\preceq_S^{K_1}c_2'$, $((p_1,d_{c1},d_{g1},q_s^1,\mathit{mak}_1,\mathit{cnt}_1),c_1')\xrightarrow{\beta}{}^{*}_{M_i^{k\text{-}w}}((p_2,d_{c2},d_{g2},q_s^2,\mathit{mak}_2,\mathit{cnt}_2),c_2')$, and the write actions in α equals that in β.*

Proof. This is proved by considering all kinds of transitions.

- If α is a internal action derived from a τ or read action, then $c'_2 = c'_1$ and this holds trivially.
- If α is a internal action derived from a call or return action, then $c'_2 = c'_1$ and this holds trivially.
- If α is a write action derived from a *cas* action, then $c'_2 = \epsilon$ and this holds trivially.
- If α is a write action derived from a $\mathit{write}(\mathit{ind}, x, d)$ action, then c'_2 is generated from c'_1 by a write action that puts an item of memory location x and valuation d, and this holds trivially.
- If α is a internal action derived from a flush action, assume $c_1 = \alpha_1 \cdot \ldots \cdot \alpha_l$, $c'_1 = \beta_1 \cdot \ldots \cdot \beta_w$, since $c_1 \preceq_S^{K_1} c'_1$, there exists $i_1, \ldots, i_l$, such that for each *ind*, $\alpha_j = \beta_{i_{ind}}$.
  Assume during the transition $((p_1, d_{c1}, \mathit{dg1}, q_s^1, \mathit{mak}_1, \mathit{cnt}_1), c_1) \xrightarrow{\alpha}_{(M_i^{k\text{-}w}, S, K_1)} ((p_2, d_{c2}, \mathit{dg2}, q_s^2, \mathit{mak}_2, \mathit{cnt}_2), c_2)$, the item which is flushed into memory is the *j-th* element in c_1. Then $\beta = \epsilon$, and $((p_2, d_{c2}, \mathit{dg2}, q_s^2, \mathit{mak}_2, \mathit{cnt}_2), c'_2)$ is generated from $((p_1, d_{c1}, \mathit{dg1}, q_s^1, \mathit{mak}_1, \mathit{cnt}_1), c'_1)$ by first flushing items $\beta_w, \ldots, \beta_{i_j+1}$, and then flush item β_{i_j}.

□

With Lemma 12 we can now prove Lemma 5.

Lemma 5. *There exists a marked violation t of k-bounded TSO-to-SC linearizability between libraries $\mathcal{L}$ and $\mathcal{L}'$ for n processes from $(p_{init}, d_{init}, \epsilon^n)$ to (p_w, d_w, ϵ^n) in $[\![\mathit{Clt}(\mathcal{L}), n{+}1]\!]_{tso}$, if and only if $\bigcap_{i=1}^{n+1} \mathit{LT}_{(q_i, q'_i)}^{(S,K_1)} M_i^{k\text{-}w} \neq \emptyset$, where for each $1 \leq i \leq n{+}1$, $q_i = (p_{init}(i), d_{init}, d_{init}, q_{is}, \epsilon, 0)$, $q'_i = (p_w(i), d_w, d_w, q_{error}, \epsilon, |t \uparrow_{(\Sigma_{cal} \cup \Sigma_{ret})}|)$.*

Proof. Lemma 5 is a direct consequence of Lemma 6 and Lemma 12. □

C Library For Simulating Single-Channel Machines

C.1 Simple channel machine

A simple channel machine is a channel machine that has only one channel and has a simpler definition than channel machine. Formally, a *simple channel machine* is a tuple $M = (Q, \mathcal{CH}, \Sigma_{\mathcal{CH}}, \Lambda, \Delta)$, where

- M is a channel machine,
- M bas only one channel,
- each transition of M uses a ϵ transition label,
- each transition of M uses an empty guard,
- each transition of M does not uses substitution,
- each item in channel is not a strong symbol,

For simplicity, a simple channel machine M can be redefined as $M = (Q, \{c\}, \Sigma_c, \Delta_M)$, where Q is a finite set of states; c is the name of the only channel of M; Σ_c is the alphabet for channel contents; $\Delta_M \subseteq Q \times (\Sigma_{\mathcal{CH}} \cup \{\epsilon\}) \times (\Sigma_{\mathcal{CH}} \cup \{\epsilon\}) \times Q$ is the transition relation. A rule (q_1, u, v, q_2) is in Δ_M, if one of the following cases holds:

- there exists $(q_1, \epsilon, \epsilon, c?a, q_2) \in \Delta$, $u = a$ and $v = \epsilon$,
- there exists $(q_1, \epsilon, \epsilon, c!a, q_2) \in \Delta$, $u = \epsilon$ and $v = a$,
- there exists $(q_1, \epsilon, \epsilon, \mathit{nop}, q_2) \in \Delta$, $u = \epsilon$ and $v = \epsilon$

Intuitively, a transition rule (q_1, u, v, q_2) represents a transition from q_1 to q_2, which gets u from channel c and puts v into channel c.

The semantics of a simple channel machine M is given by an *LTS* $(\mathit{Conf}_M, \emptyset, \rightarrow_M, \mathit{initConf}_M)$. A configuration of Conf_M is a pair (q, u) where $q \in Q$ and $u : \{c\} \rightarrow \Sigma_c^*$. The transition relation $\rightarrow_M$ is defined as follows: given $q, q' \in Q$ and $u, u' \in \{c\} \rightarrow \Sigma_c^*$, $(q, u) \xrightarrow{\alpha}_M (q', u')$, if there exists transition rule $(q, a, b, q') \in \Delta_M$, such that $b \cdot u = u' \cdot a$.

A lossy simple channel machine M is a simple channel machine M with lossy channel, and its semantics is given by an *LTS* $(\mathit{Conf}_M, \emptyset, \rightarrow_l, \mathit{initConf}_M)$. The transition relation $\rightarrow_l$ is defined as follows: given $q, q' \in Q$ and $u, u' \in \{c\} \rightarrow \Sigma_c^*$, $(q, u) \xrightarrow{\alpha}_l (q', u')$, if there exists transition rule $(q, a, b, q') \in \Delta_M$ and $v, v' \in \{c\} \rightarrow \Sigma_c^*$, such that $b \cdot v = v' \cdot a$, v is a subword of u and u' is a subword of v'.

Given a lossy simple channel machine M, we say that $(q_0, u_0) \cdot \alpha_1 (q_1, u_1) \cdot \ldots \cdot \alpha_w \cdot (q_w, u_w)$ is a finite run of M from (q, u) to (q', u'), if (1) $(q_0, u_0) = (q, u)$, (2) $(q_i, u_i) \xrightarrow{\alpha_{i+1}}_l (q_{i+1}, u_{i+1})$ for each i and (3) $(q_w, u_w) = (q', u')$. Given a lossy simple channel machine M and two configurations s_1, s_2 of M, the reachability problem of M is to determine whether there is a finite run from s_1 to s_2 in lossy semantics of M.

According to [14], it is obvious that the reachability problem of lossy simple channel machine has nonprimitive recursive complexity.

C.2 Definition of Implementation Library

On the TSO memory model flush operations are launched nondeterministically by the memory system. Therefore, between two consecutive read actions, more than one flush actions may happen. The next read action can only read the latest flush action to x, while missing the intermediate ones. These missing flush actions are similar to the missing messages that may happen in a lossy channel machine. This makes it possible to simulate a lossy simple channel machine with a concurrent program running on the TSO memory model. We implement such simulation through a most general client and a library $\mathcal{L}^M_{(s_1,s_2)}$ specifically constructed based on a lossy simple channel machine M and configurations s_1 and s_2.

For a simple channel machine $M = (Q, \{c\}, \Sigma_c, \Delta_M)$ and configurations $s_1 = (q_1, W_1), s_2 = (q_2, W_2)$, the finite data domain of the library is $\mathcal{D}_\mathcal{L} = Q \cup \Sigma_c \cup \{\mathit{start}, \mathit{end}, \sharp, \bot, 0, \ldots, |W_2|+1\}$. The library $\mathcal{L}^M_{(s_1,s_2)}$ is constructed with two methods M_1 and M_2, and the following memory locations:

- a memory location x that is used to transmit the channel contents from M_1 to M_2,
- a memory location y that is used to transmit the channel contents from M_2 to M_1,
- a memory location *cnt* that is used in M_1 to count that, in each round, whether $|W_2|$ items has been read,
- an array $W_2\mathit{Seq}$ which is of length $|W_2|$ and stores W_2 as initial value,

- an array *RecvSeq* which is of length $|W_2|$ and is used to store the first $|W_2|$ items read in each round,

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. $\perp$ is the initial value of elements in *RecvSeq* in each round.

We now present the three methods in the pseudo-code, shown in Methods 1 and 2. For the sake of brevity, the following macro notations are used:

- For sequence $l = a_1 \cdot \ldots \cdot a_m$, we use *writeSeq(x,l)* to represent the commands of writing $a_1, \sharp, \ldots, a_m, \sharp$ to x in sequence,
- We use $v := \textit{readOne}(x)$ to represent the commands of reading $e, \sharp$ from x in sequence for some $e \neq \sharp$ and then assigning e to v. We use $\textit{readOne}(x, v)$ to represent the commands of reading $a, \sharp$ from x in sequence where a is the value of v. If $\textit{readOne}(x)$ or $\textit{readOne}(x, v)$ fails to read the specified content, then the calling process will no long proceed.
- We use $\textit{writeOne}(x, v)$ to represent the commands of writing a, $\sharp$ to x in sequence where a is the current value of v.
- We use $\textit{initRecvSeq}()$ to represent the commands that assigns 1 to *cnt* and assigns $\perp$ to $\textit{RecvSeq}(1), \ldots, \textit{RecvSeq}(|W_2|)$.
- We use *det(tempQ,cnt,ele)* to represent the macro which will either nondeterministically return *false*, or update the *cnt-th* element of *RecvSeq* to *ele* and then determine whether contents of *RecvSeq* equals W_2. It works as follows:
  - It may nondeterministically decide to do noting and return *false*;
  - If $\textit{tmpQ} \neq q_2 \vee \textit{cnt} = 0 \vee \textit{cnt} > |W_2|$, then it assigns $\textit{min}\{|W_2| + 1, \textit{cnt} + 1\}$ to *cnt* and returns *false*.
  - Else, if $1 \leq \textit{cnt} < W_2|$, then it assigns *ele* to $\textit{RecvSeq}(\textit{cnt})$, assigns $\textit{min}\{|W_2| + 1, \textit{cnt} + 1\}$ to *cnt* and returns *false*,
  - Otherwise, $\textit{cnt} = |W_2|$ in this case, then it assigns *ele* to $\textit{RecvSeq}(|W_2|)$, assigns $\textit{min}\{|W_2| + 1, \textit{cnt} + 1\}$ to *cnt*, and checks whether contents of *RecvSeq* equals W_2. If it holds, returns *true*, else, returns *false*.

There are two kinds of losing in our implementation library $\mathcal{L}^M_{(s_1,s_2)}$. The first kind of losing comes from that between two consecutive read actions, more than one flush actions may happen and the intermediate flush may be lost. The second kind of losing comes from that *det*, which is designed to check whether (q_2, W_2) has been reached, may loses some information nondeterministically.

The pseudo-code of method M_1 is shown in Method 1. M_1 first puts $q_1 \cdot \textit{start} \cdot W_1 \cdot \textit{end}$ into the processor-local store buffer by writing them to x (Line 1). Then, it begins an infinite loop that never returns unless (q_2, W_2) is reached (Lines $2-24$). At each round of the loop, it reads the current state *tmpQ* (Line 3) and guesses a transition rule $\textit{rul} = (\textit{tmpQ}, u, v) \in \Delta_M$ (Line 4). M_1 initializes *RecvSeq* (Line 5), check whether it is the case that $\textit{tempQ} = q_2 \wedge W_2 = \epsilon$ (Lines $6-7$). If so, it returns as soon as possible. It not, it reads u from y (Lines 8) if $u \neq \epsilon$. 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 to x (Lines 13-22). In each round of the while loop of Lines $2-24$, when a item is read from y (Lines $11-12, 18-19$), or when write v to x (Lines $13-24$), it uses *det* to check whether (q_2, W_2) is reached. If so, M_1 return as soon as possible. It should be noted that *det* may nondeterministically loses items.

The pseudo-code of method M_2 is shown in Method 2. M_2 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.

Method 1: M_1

Input: an arbitrary argument
Output: an arbitrary argument

```
1  writeSeq(x, q_1 · start · W_1 · end);
2  while true do
3      tmpQ := readOne(y) for some state tmpQ ∈ Q;
4      guess a transition rule rul = (tmpQ, u, v) ∈ Δ_M;
5      initRecvSeq();
6      if tempQ = q_2 ∧ W_2 = ϵ then
7          return;
8      readOne(y, start);
9      if u ≠ ϵ then
10         readOne(y, u);
11         if det(tempQ, cnt, u) = true then
12             return;
13     while true do
14         tmp = readOne(y);
15         if temp = end then
16             break;
17         writeOne(x, tmp);
18         if det(tempQ, cnt, u) = true then
19             return;
20     if v ≠ ϵ then
21         writeOne(x, v);
22     writeOne(x, end);
23     if det(tempQ, cnt, u) = true then
24         return;
```

Method 2: M_2

Input: an arbitrary argument

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

C.3 Definition of Abstraction Library

The library $\mathcal{L}_{pend}$ is constructed with two methods M_1 and M_2 and it does contain private memory locations. M_1 and M_2 are pending in any cases. They only contains a *while*(*true*); loop. It is obvious that in each trace of $\mathcal{L}_{pend}$, no method can return.

C.4 Proof of Proposition 1

Similarly to [15,16], we can prove the following lemma, which states that a history of $[\![\mathcal{L}^M_{(s_1,s_2)}, 2]\!]_{tso}$ contains a return action, if and only if s_2 is reachable from s_1 in lossy semantics of M.

Lemma 13. *There exists a history $h \in$ history$([\![\mathcal{L}^M_{(s_1,s_2)}, 2]\!]_{tso})$ such that $h \uparrow_{\Sigma_{return}} \neq \epsilon$, if and only if s_2 is reachable from s_1 in lossy semantics of M.*

With above lemma, we can prove Proposition 1.

Proposition 1. *The decision problem of k-bound TSO-to-SC linearizability has non-primitive recursive complexity.*

Proof. From Lemma 13 and the following facts:

- Each history in *history*$([\![\mathcal{L}_{pend}, 2]\!]_{tso})$ does not contain return action.
- If a history h in *history*$([\![\mathcal{L}^M_{(s_1,s_2)}, 2]\!]_{tso})$ contains a return action, then it must contains a call action of M_1, its accompanying flush action, a call action of M_2. If h also contains one or two additional pending call actions, we can discarding these additional pending actions and generate a history h' of just three actions with a return action.

It is obvious that s_2 is reachable from s_1 in lossy semantics of M, if and only if $[\![\mathcal{L}_{pend}]\!]_{tso}$ does not 3-bound TSO-to-SC linearizes $[\![\mathcal{L}^M_{(s_1,s_2)}]\!]_{tso}$ for 2 processes. This proposition then holds because that the reachability problem of lossy simple channel machine has nonprimitive recursive complexity.