Logical Methods in Computer Science
Volume 21, Issue 3, 2025, pp. 30:1–30:43
https://lmcs.episciences.org/

Submitted Mar. 21, 2024
Published Sep. 24, 2025

TSO GAMES - ON THE DECIDABILITY OF SAFETY GAMES UNDER THE TOTAL STORE ORDER SEMANTICS

STEPHAN SPENGLER [a] AND SANCHARI SIL [b]

[a] Uppsala University, Uppsala, Sweden
e-mail address: stephan.spengler@it.uu.se

[b] Chennai Mathematical Institute, Chennai, India
e-mail address: sanchari@cmi.ac.in

Abstract. We consider an extension of the classical Total Store Order (TSO) semantics by expanding it to turn-based 2-player safety games. During her turn, a player can select any of the communicating processes and perform its next transition. We consider different formulations of the safety game problem depending on whether one player or both of them transfer messages from the process buffers to the shared memory. We give the complete decidability picture for all the possible alternatives.

1. Introduction

Most modern architectures, such as Intel x86 [Int12], SPARC [spa94], IBM's POWER [IBM21], and ARM [ARM14], implement several relaxations and optimisations that reduce the latency of memory accesses. This has the effect of breaking the Sequential Consistency (SC) assumption [Lam79]. SC is the classical strong semantics for concurrent programs that interleaves the parallel executions of processes while maintaining the order in which instructions were issued. Programmers usually assume that the execution of programs follows the SC model. However, this is not true when we consider concurrent programs running on modern architectures. In fact, even simple programs such as mutual exclusion and producer-consumer protocols, that are correct under SC, may exhibit erroneous behaviours. This is mainly due to the relaxation of the execution order of the instructions. For instance, a standard relaxation is to allow the reordering of reads and writes of the same process if the reads have been issued after the writes and they concern different memory locations. This relaxation can be implemented using an unbounded perfect FIFO queue/buffer between each process and the memory. These buffers are used to store delayed writes. The corresponding model is called Total Store Ordering (TSO) and corresponds to the formalisation of SPARC and Intel x86 [OSS09, SSO$^+$10].

A process of a concurrent program is modelled as a finite-state transition system. Each transition is labelled with an instruction that describes how it interacts with the shared memory. For instance, the process can read a value from or write a value to a shared variable. The data domain of these instructions is assumed to be finite. More complex instructions like comparisons or arithmetic operations, and infinite data structures managed locally by the process are abstracted through the nondeterminism of the process. An example of a concurrent program is given in Figure 1, which shows the Dekker protocol for mutual

$$\begin{tikzcd}[row sep=small]
\mathsf{Proc}^1: & \mathsf{q}_1 \arrow[d, "\mathtt{wr}(\mathsf{x},1)"] & \mathsf{Proc}^2: & \mathsf{r}_1 \arrow[d, "\mathtt{wr}(\mathsf{y},1)"] \\
& \mathsf{q}_2 \arrow[d, "\mathtt{rd}(\mathsf{y},0)"] & & \mathsf{r}_2 \arrow[d, "\mathtt{rd}(\mathsf{x},0)"] \\
& \mathsf{q}_3 & & \mathsf{r}_3
\end{tikzcd}$$

Figure 1. A concurrent program $\mathcal{P} = \langle \mathsf{Proc}^1, \mathsf{Proc}^2 \rangle$ modelling (a simplified version of) the Dekker protocol.

exclusion. Under SC semantics, it is not possible for both processes to simultaneously reach the critical sections q_3 and r_3, respectively.

In TSO, an unbounded buffer is associated with each process. When a process executes a write operation, this write is appended to the end of the buffer of that process. A pending write operation on the variable x at the head of a buffer can be deleted in a non-deterministic manner. This updates the value of the shared variable x in the memory. To perform a read operation on a variable x, the process first checks its buffer for a pending write operation on the variable x. If such a write exists, then the process reads the value written by the newest pending write operation on x. Otherwise, the process fetches the value of the variable x from the memory.

Under TSO, both processes of the Dekker protocol in Figure 1 can reach the critical sections q_3 and r_3 simultaneously, thus violating mutual exclusion. This can happen since after executing $\mathtt{wr}(\mathsf{x}, 1)$, the operation only resides in the buffer of Proc^1 and is invisible to Proc^2. The converse holds for the write instruction of Proc^2. Thus, both processes are able to read the initial value 0 from the variable y or x, respectively.

In general, verification of programs running under TSO is challenging due to the unboundedness of the buffers. In fact, the induced state space of a program under TSO may be infinite even if the program itself is a finite-state system.

The reachability problem for programs under TSO checks whether a given program state is reachable during program execution. It is also called safety problem, in case the target state is considered to be a bad state. It has been shown decidable using different alternative semantics for TSO (e.g. [ABBM10, AAC$^+$12, AABN18]). These alternative semantics, such as the load-buffer semantics in contrast to the traditional store-buffer semantics, provide different formalisations of TSO. While they differ in how they model specific aspects of memory behaviour, they are equivalent with respect to the reachability of program states.

Furthermore, it has been shown in [ABBM10] that lossy channel systems (see e.g., [AJ96, FS01, AJ94, Sch02]) can be simulated by programs running under TSO. This entails that the reachability problem for programs under TSO is non-primitive recursive and that the repeated reachability problem is undecidable. This is an immediate consequence of the fact that the reachability problem for lossy channel is non-primitive recursive [Sch02] and that the repeated reachability problem is undecidable [AJ94]. The termination problem for programs running under TSO has been shown to be decidable in [Ati20] using the framework of well-structured transition systems [Fin87, FS01, AJ96].

The authors of [BDM13, BMM11] consider the robustness problem for programs running under TSO. This problem consists in checking whether, for any given TSO execution, there is an equivalent SC execution of the same program. Two executions are declared equivalent by the robustness criterion if they agree on (1) the order in which instructions are executed within the same process (i.e., program order), (2) the write instruction from which each read instruction fetches its value (i.e., read-from relation), and (3) the order in which write instruction on the same variable are committed to memory (i.e., store ordering). The problem of checking whether a program is robust has been shown to be PSpace-complete in [BDM13]. A variant of the robustness problem which is called persistence, declares that two runs are equivalent if (1) they have the same program order and (2) all write instructions reach the memory in the same order. Checking the persistency of a program under TSO has been shown to be PSpace-complete in [AAP15]. Observe that the persistency and robustness problems are stronger than the safety problem (i.e., if a program is safe under SC and robust/persistent, then it is also safe under TSO).

Due to the non-determinism of the buffer updates, the buffers associated with each process under TSO appear to exhibit a lossy behaviour. Previously, games on lossy channel systems (and more general on monotonic systems) were studied in [ABd08]. Unfortunately these results are not applicable / transferable to programs under TSO whose induced transition systems are not monotone [ABBM10].

In this paper, we consider a natural continuation of the works on both the study of the decidability/complexity of the formal verification of programs under TSO and the study of games on concurrent systems. This is further motivated by the fact that formal games provide a framework to reason about a system's behaviour, which can be leveraged in control model checking, for example in controller synthesis problems. In particular, games can model the adversarial nature of certain problems where one component tries to achieve a goal while another tries to prevent it. This is especially relevant in the context of concurrent programs running under relaxed memory models like TSO, where the non-deterministic behaviour of memory can be seen as an adversary to the correctness of the program.

One specific problem that can be modeled by our approach is the controller synthesis problem (e.g. [PR89, KV97, KV00]). In this context, the goal is to design a controller (player A) that ensures a system (player B) behaves correctly despite the uncertainties introduced by the memory model. The game framework allows us to formally define the objectives of the controller and the system, and to analyse whether a winning strategy exists for the controller. If such a strategy exists, it corresponds to a correct implementation of the controller that guarantees the desired behaviour of the system [BS04, AVW03].

Another problem that can be modeled by our approach is the verification of safety properties in concurrent programs. By framing the verification problem as a game, we can leverage game-theoretic techniques to determine whether a program can reach an undesirable state (a losing condition for player A). This approach provides a clear and formal method to analyse the safety of concurrent programs under TSO and other relaxed memory models.

Additionally, our approach lays the foundation for future research. For instance, by adding weights to the instructions/transitions, we can model fence insertion problems [AAP15]. This could be done by giving a player the choice to take a memory fence instruction or not, but we assign a penalty cost to each memory fence transition. An optimal, that is lowest-cost, winning strategy corresponds to the minimal amount of memory fences that need to be inserted into the program to ensure memory consistency. This is crucial for optimising the performance of concurrent programs under relaxed memory models.

In more detail, we consider (safety) games played on the transition systems induced by programs running under TSO. Given a program under TSO, we construct a game in which two players, A and B, take turns executing instructions. Player B aims to reach a given set of final configurations, while player A tries to prevent this, making it a reachability

		Player A:			
		always	before	after	never
Player B:	always	I (d)			
	before		II (d)		
	after			III (u)	
	never				IV (d)

Figure 2. Groups of TSO games, where players A and B are allowed to update the buffer: always, before their own move, after their own move, or never. The games in group I (light grey), II (white) and IV (dark grey) are decidable (d), the games in group III (medium grey) are undecidable (u).

game from the perspective of player B. The turn structure determines which player executes the next instruction, but it does not specify how memory updates occur. To address this, we allow players to update memory by removing pending writes from the buffer between instruction executions.

In some variants, only player A (considered to be the *good* player) has control over memory updates. These correspond to the problem of synthesizing an update controller that ensures correct execution despite an adversarial opponent. In other variants, only player B (the *bad* player) has control over updates, modelling the problem of synthesising a process or program under a non-cooperative update mechanism. In both cases, the precise timing of updates remains unclear — whether they occur before, after, or both before and after a player's turn. Additionally, there may be scenarios where it is beneficial to allow both players to update memory, depending on the problem setting.

Considering all these possibilities results in 16 different TSO game variants. While not all are equally relevant in practice, we analyse them for completeness, as they may provide insights for future research and yet unknown applications. We divide these 16 games into four different groups, depending on their decidability results.

- Group I (7 games) can be reduced to TSO games with 2-bounded buffers.
- Group II (1 game) can be reduced to TSO games with bounded buffers.
- Group III (7 games) can simulate perfect channel systems.
- Group IV (1 game) can be reduced to a finite game without buffers.

This classification is shown in Figure 2. Of these four groups, only Group III is undecidable, the others each reduce to a finite game and are thus decidable. Coincidentally, the undecidable group contains all of the most interesting variants, where exactly one of the two players can update memory.

Finally, we establish the exact computational complexity for the decidable games. In fact, we show that the problem is ExpTime-complete. We prove ExpTime-hardness by a reduction from the problem of acceptance of a word by a polynomially bounded alternating Turing machine [CS76, CKS81]. This result holds even in the case for games played on a single-process concurrent program following SC semantics. To prove ExpTime-membership, we show that it is possible to compute the winning regions for the players in exponential time. These results are surprising given the non-primitive recursive complexity of the reachability problem for programs under TSO and the undecidability of the repeated reachability problem.

This paper is an extended and revised version of the conference paper [SS23]. We present the formal proof of the lower complexity bound for the decidable games. These now also include the newly defined SC games, which are games played on programs following SC semantics. We utilise the framework of bisimulations to lift complexity results from SC games to TSO games, and to streamline the ExpTime-completeness proof of group IV. Furthermore, the appendix presents a formal proof of the undecidability result for the games in group III. We generally revised all sections of this paper. It contains clarifications, more elaborate explanations, and additional examples and figures.

Related Works. In addition to the related work mentioned in the introduction on the decidability / complexity of the verification problems of programs running under TSO, there have been some works on parameterized verification of programs running under TSO. The problem consists in verifying a concurrent program regardless of the number of involved processes (which are identical finite-state systems). The parameterised reachability problem of programs running under TSO has been shown to be decidable in [AABN16, AABN18]. While this problem for concurrent programs performing only read and writing operations (no atomic read-write instructions) is PSpace-complete [AAR20]. This result has been recently extended to processes manipulating abstract data types over infinite domains [AAF^{+}23]. Checking the robustness of a parameterised concurrent system is decidable and ExpSpace-hard [BDM13].

As far as we know this is the first work that considers the game problem for programs running under TSO. The proofs and techniques used in this paper are different from the ones used to prove decidability / complexity results for the verification of programs under TSO except the undecidability result which uses some ideas from the reduction from the reachability problem for lossy channel systems to its corresponding problem for programs under TSO [ABBM10]. However, our undecidability proof requires us to implement a protocol that detects lossiness of messages in order to turn the lossy channel system into a perfect one (which is the most intricate part of the proof).

2. Preliminaries

2.1. **Transition Systems.**

A *(labelled) transition system* is a triple $\langle \mathsf{C}, \mathsf{L}, \rightarrow \rangle$, where C is a set of *configurations*, L is a set of *labels*, and $\rightarrow \subseteq \mathsf{C} \times \mathsf{L} \times \mathsf{C}$ is a *transition relation*. We usually write $\mathsf{c}_1 \xrightarrow{\mathsf{label}} \mathsf{c}_2$ if $\langle \mathsf{c}_1, \mathsf{label}, \mathsf{c}_2 \rangle \in \rightarrow$. Furthermore, we write $\mathsf{c}_1 \rightarrow \mathsf{c}_2$ if there exists some label such that $\mathsf{c}_1 \xrightarrow{\mathsf{label}} \mathsf{c}_2$. A *run* π of $\mathcal{T}$ is a sequence of transitions $\mathsf{c}_0 \xrightarrow{\mathsf{label}_1} \mathsf{c}_1 \xrightarrow{\mathsf{label}_2} \mathsf{c}_2 \ldots \xrightarrow{\mathsf{label}_n} \mathsf{c}_n$. It is also written as $\mathsf{c}_0 \xrightarrow{\pi} \mathsf{c}_n$. A configuration c' is *reachable* from a configuration c, if there exists a run from c to c'.

For a configuration c, we define $\mathrm{Pre}(\mathsf{c}) := \{\mathsf{c}' \mid \mathsf{c}' \rightarrow \mathsf{c}\}$ and $\mathrm{Post}(\mathsf{c}) := \{\mathsf{c}' \mid \mathsf{c} \rightarrow \mathsf{c}'\}$. We extend these notions to sets of configurations C' with $\mathrm{Pre}(\mathsf{C}') := \bigcup_{\mathsf{c} \in \mathsf{C}'} \mathrm{Pre}(\mathsf{c})$ and $\mathrm{Post}(\mathsf{C}') := \bigcup_{\mathsf{c} \in \mathsf{C}'} \mathrm{Post}(\mathsf{c})$.

An *unlabelled transition system* is a transition system without labels. Formally, it is defined as a TS with a singleton label set. In this case, we omit the labels.

2.2. **Alternating Turing Machines.** Let $\mathcal{A} = \langle \Sigma, \mathsf{Q}, \mathsf{q}_0, \mathsf{q}_F, \mathsf{Q}_\exists, \mathsf{Q}_\forall, \delta \rangle$ be an alternating Turing Machine, where Σ is a finite alphabet, Q is a finite set of states partitioned into existential states $\mathsf{Q}_\exists$ and universal states $\mathsf{Q}_\forall$, $\mathsf{q}_0 \in \mathsf{Q}$ is the *initial state*, $\mathsf{q}_F \in \mathsf{Q}_\forall$ is the *accepting state*, and $\delta \subseteq \mathsf{Q} \times \Sigma \times \mathsf{Q} \times \Sigma \times \{\mathsf{L}, \mathsf{R}\}$ is a transition relation. The alphabet contains the blank symbol $\llcorner\!\lrcorner$. Let $\langle \mathsf{q}, \sigma, \mathsf{q}', \sigma', \mathsf{D} \rangle \in \delta$ be a transition. If machine $\mathcal{A}$ is in state q and its head reads letter $\sigma \in \Sigma$, then it replaces the contents of the current cell with the letter σ', moves the head in direction D ($\mathsf{D} = \mathsf{L}$ for left and $\mathsf{D} = \mathsf{R}$ for right) and changes control to state q'. We assume that q_F has no outgoing transitions.

A configuration of $\mathcal{A}$ is a triple $\mathsf{c} = \langle \mathsf{q}, \mathsf{i}, \omega \rangle$, where $\mathsf{q} \in \mathsf{Q}$, $\mathsf{i} \in \mathbb{Z}$ and $\omega \in \Sigma^{\mathbb{Z}}$. It represents the situation where $\mathcal{A}$ is in state q, the tape contains the (in both directions infinite) word ω and the head is at the i-th position of the tape. A *successor* of c is a configuration c' that can be obtained from c by executing a transition as described above.

Let $\mathsf{C}_0 := \{\langle \mathsf{q}_F, \mathsf{i}, \omega \rangle \mid \mathsf{i} \in \mathbb{Z}, \omega \in \Sigma^{\mathbb{Z}}\}$ and for all $k > 0$:

$$\mathsf{C}_k := \{\mathsf{c} = \langle \mathsf{q}, \mathsf{i}, \omega \rangle \mid (\mathsf{q} \in \mathsf{Q}_\exists \land \mathrm{Post}(\mathsf{c}) \cap \mathsf{C}_{k-1} \neq \emptyset) \lor (\mathsf{q} \in \mathsf{Q}_\forall \land \mathrm{Post}(\mathsf{c}) \subseteq \mathsf{C}_{k-1})\}$$

We call $\mathsf{C}_\infty := \bigcup_{k=0}^{\infty} \mathsf{C}_k$ the set of *accepting configurations* of $\mathcal{A}$. Intuitively, a configuration $\mathsf{c} = \langle \mathsf{q}, \mathsf{i}, \omega \rangle$ is accepting if and only if $\mathsf{q} \in \mathsf{Q}_\forall$ and all successors of c are accepting or $\mathsf{q} \in \mathsf{Q}_\exists$ and at least one successor of c is accepting.

We say that $\mathcal{A}$ accepts a word $\mathsf{w} \in \Sigma^n$, if $\langle \mathsf{q}_0, 1, \omega(\mathsf{w}) \rangle$ is an accepting configuration, where $\omega(\mathsf{w})$ contains w on positions 1 through n and the blank $\llcorner\!\lrcorner$ on all other positions. The **word acceptance problem** of ATM is, given an alternating Turing machine $\mathcal{A}$ and a word $\mathsf{w} \in \Sigma^n$, to decide whether $\mathcal{A}$ accepts w. The word acceptance problem for ATMs that only use space polynomial in the length of the input word is ExpTime-complete [CS76, CKS81].

2.3. **Perfect Channel Systems.** Given a finite set of messages M, define the set of channel operations $\mathsf{Op} := \{!\mathsf{m}, ?\mathsf{m} \mid \mathsf{m} \in \mathsf{M}\} \cup \{\mathtt{skip}\}$. A *perfect channel system* (PCS) is a triple $\mathcal{L} = \langle \mathsf{S}, \mathsf{M}, \delta \rangle$, where S is a finite set of states, M is the set of messages, and $\delta \subseteq \mathsf{S} \times \mathsf{Op} \times \mathsf{S}$ is a transition relation. We write $\mathsf{s}_1 \xrightarrow{\mathsf{op}} \mathsf{s}_2$ if $\langle \mathsf{s}_1, \mathsf{op}, \mathsf{s}_2 \rangle \in \delta$.

Intuitively, a PCS models a finite state automaton that is augmented by a *perfect* (i.e. non-lossy) FIFO buffer, called *channel.* During a *send operation* $!\mathsf{m}$, the channel system appends m to the tail of the channel. A transition $?\mathsf{m}$ is called *receive operation.* It is only enabled if the channel is not empty and m is its oldest message. When the channel system performs this operation, it removes m from the head of the channel. Lastly, a `skip` operation just changes the state, but does not modify the buffer.

The formal semantics of $\mathcal{L}$ are defined by a transition system $\mathcal{T}_\mathcal{L} = \langle \mathsf{C}_\mathcal{L}, \mathsf{L}_\mathcal{L}, \rightarrow_\mathcal{L} \rangle$, where $\mathsf{C}_\mathcal{L} := \mathsf{S} \times \mathsf{M}^*$, $\mathsf{L}_\mathcal{L} := \mathsf{Op}$ and the transition relation $\rightarrow_\mathcal{L}$ is the smallest relation given by:

- If $\mathsf{s}_1 \xrightarrow{!\mathsf{m}} \mathsf{s}_2$ and $\mathsf{w} \in \mathsf{M}^*$, then $\langle \mathsf{s}_1, \mathsf{w} \rangle \xrightarrow{!\mathsf{m}}_\mathcal{L} \langle \mathsf{s}_2, \mathsf{m} \cdot \mathsf{w} \rangle$.
- If $\mathsf{s}_1 \xrightarrow{?\mathsf{m}} \mathsf{s}_2$ and $\mathsf{w} \in \mathsf{M}^*$, then $\langle \mathsf{s}_1, \mathsf{w} \cdot \mathsf{m} \rangle \xrightarrow{?\mathsf{m}}_\mathcal{L} \langle \mathsf{s}_2, \mathsf{w} \rangle$.
- If $\mathsf{s}_1 \xrightarrow{\mathtt{skip}} \mathsf{s}_2$ and $\mathsf{w} \in \mathsf{M}^*$, then $\langle \mathsf{s}_1, \mathsf{w} \rangle \xrightarrow{\mathtt{skip}}_\mathcal{L} \langle \mathsf{s}_2, \mathsf{w} \rangle$.

A state $\mathsf{s}_F \in \mathsf{S}$ is *reachable* from a configuration $\mathsf{c}_0 \in \mathsf{C}_\mathcal{L}$, if there exists a configuration $\mathsf{c}_F = \langle \mathsf{s}_F, \mathsf{w}_F \rangle$ such that c_F is reachable from c_0 in $\mathcal{T}_\mathcal{L}$. The **state reachability problem** of PCS is, given a perfect channel system $\mathcal{L}$, an initial configuration $\mathsf{c}_0 \in \mathsf{C}_\mathcal{L}$ and a final state $\mathsf{s}_F \in \mathsf{S}$, to decide whether s_F is reachable from c_0 in $\mathcal{T}_\mathcal{L}$. It is undecidable [BZ83].

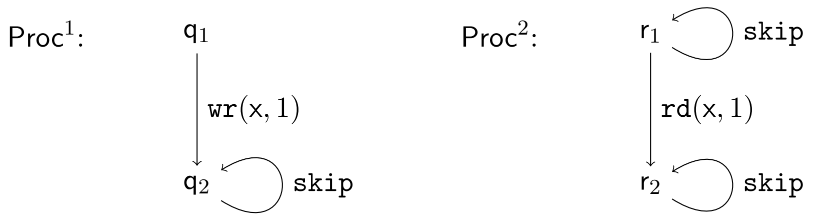

FIGURE 3. Concurrent program $\mathcal{P} = \langle \mathsf{Proc}^1, \mathsf{Proc}^2 \rangle$

3. Concurrent Programs

3.1. **Syntax.** Let Dom be a finite data domain and Vars be a finite set of shared variables over Dom. We define the *instruction set*

$$\begin{aligned}\mathsf{Instrs} := &\{\mathtt{rd}(\mathsf{x},\mathsf{d}), \mathtt{wr}(\mathsf{x},\mathsf{d}) \mid \mathsf{x} \in \mathsf{Vars}, \mathsf{d} \in \mathsf{Dom}\} \\ &\cup \{\mathtt{arw}(\mathsf{x},\mathsf{d},\mathsf{d}') \mid \mathsf{x} \in \mathsf{Vars}, \mathsf{d},\mathsf{d}' \in \mathsf{Dom}\} \\ &\cup \{\mathtt{skip}, \mathtt{mf}\}\end{aligned}$$

which are called *read*, *write*, *atomic read-write*, *skip* and *memory fence*, respectively. A process is represented by a finite state labelled transition system. It is given as the triple $\mathsf{Proc} = \langle \mathsf{Q}, \mathsf{Instrs}, \delta \rangle$, where Q is a finite set of *local states* and $\delta \subseteq \mathsf{Q} \times \mathsf{Instrs} \times \mathsf{Q}$ is the transition relation. As with transition systems, we write $\mathsf{q}_1 \xrightarrow{\mathsf{instr}} \mathsf{q}_2$ if $\langle \mathsf{q}_1, \mathsf{instr}, \mathsf{q}_2 \rangle \in \delta$ and $\mathsf{q}_1 \to \mathsf{q}_2$ if there exists some instr such that $\mathsf{q}_1 \xrightarrow{\mathsf{instr}} \mathsf{q}_2$.

A *concurrent program* is a tuple of processes $\mathcal{P} = \langle \mathsf{Proc}^\iota \rangle_{\iota \in \mathcal{I}}$, where $\mathcal{I}$ is a finite set of process identifiers. For each $\iota \in \mathcal{I}$ we have $\mathsf{Proc}^\iota = \langle \mathsf{Q}^\iota, \mathsf{Instrs}, \delta^\iota \rangle$. A *global* state of $\mathcal{P}$ is a function $\mathcal{S} : \mathcal{I} \to \bigcup_{\iota \in \mathcal{I}} \mathsf{Q}^\iota$ that maps each process to its local state, i.e $\mathcal{S}(\iota) \in \mathsf{Q}^\iota$. Figure 3 shows a simple example of a concurrent program $\mathcal{P}$ consisting of two processes Proc^1 and Proc^2.

3.2. **SC Semantics.** Under SC (Sequential Consistency) semantics, all processes of a concurrent program interact with the shared memory directly, and all operations appear to be executed in some sequential order that is consistent with the program order of each individual process.

Formally, an SC *configuration* is a tuple $\mathsf{c} = \langle \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{S} : \mathcal{I} \to \bigcup_{\iota \in \mathcal{I}} \mathsf{Q}^\iota$ is a global state of $\mathcal{P}$ and $\mathcal{M} : \mathsf{Vars} \to \mathsf{Dom}$ represents the memory state of each shared variable. Given a configuration c, we write $\mathcal{S}(\mathsf{c})$ and $\mathcal{M}(\mathsf{c})$ for the global program state and memory state of c.

The semantics of a concurrent program running under SC is defined by a transition system $\mathcal{T}^{\mathsf{SC}}_{\mathcal{P}} = \langle \mathsf{C}^{\mathsf{SC}}_{\mathcal{P}}, \mathsf{L}^{\mathsf{SC}}_{\mathcal{P}}, \to^{\mathsf{SC}}_{\mathcal{P}} \rangle$, where $\mathsf{C}^{\mathsf{SC}}_{\mathcal{P}}$ is the set of all SC configurations of $\mathcal{P}$ and $\mathsf{L}^{\mathsf{SC}}_{\mathcal{P}} := \{\mathsf{instr}_\iota \mid \mathsf{instr} \in \mathsf{Instrs}, \iota \in \mathcal{I}\}$ is the set of labels. The transition relation $\to$ (we usually drop the indices $^{\mathsf{SC}}_{\mathcal{P}}$ whenever they are clear from context) is given by the following rules (see Figure 4):

$$\textbf{read}\quad \frac{\mathsf{q} \xrightarrow{\texttt{rd(x,d)}} \mathsf{q}' \qquad \mathcal{S}(\iota){=}\mathsf{q} \qquad \mathcal{M}(\mathsf{x}){=}\mathsf{d}}{\langle \mathcal{S},\mathcal{M}\rangle \xrightarrow{\texttt{rd(x,d)}_\iota} \langle \mathcal{S}[\iota{\leftarrow}\mathsf{q}'],\mathcal{M}\rangle}$$

$$\textbf{write}\quad \frac{\mathsf{q} \xrightarrow{\texttt{wr(x,d)}} \mathsf{q}' \qquad \mathcal{S}(\iota){=}\mathsf{q}}{\langle \mathcal{S},\mathcal{M}\rangle \xrightarrow{\texttt{wr(x,d)}_\iota} \langle \mathcal{S}[\iota{\leftarrow}\mathsf{q}'],\mathcal{M}[\mathsf{x}{\leftarrow}\mathsf{d}]\rangle}$$

$$\textbf{arw}\quad \frac{\mathsf{q} \xrightarrow{\texttt{arw(x,d,d}'\texttt{)}} \mathsf{q}' \qquad \mathcal{S}(\iota){=}\mathsf{q} \qquad \mathcal{M}(\mathsf{x}){=}\mathsf{d}}{\langle \mathcal{S},\mathcal{M}\rangle \xrightarrow{\texttt{arw(x,d,d}'\texttt{)}_\iota} \langle \mathcal{S}[\iota{\leftarrow}\mathsf{q}'],\mathcal{M}[\mathsf{x}{\leftarrow}\mathsf{d}']\rangle}$$

$$\textbf{skip}\quad \frac{\mathsf{q} \xrightarrow{\texttt{skip}} \mathsf{q}' \qquad \mathcal{S}(\iota){=}\mathsf{q}}{\langle \mathcal{S},\mathcal{M}\rangle \xrightarrow{\texttt{skip}_\iota} \langle \mathcal{S}[\iota{\leftarrow}\mathsf{q}'],\mathcal{M}\rangle}$$

Figure 4. SC semantics

- **Read:** If $\mathcal{S}(\iota) = \mathsf{q}$, $\mathcal{M}(\mathsf{x}) = \mathsf{d}$ and $\mathsf{q} \xrightarrow{\texttt{rd(x,d)}} \mathsf{q}'$, then $\langle \mathcal{S}, \mathcal{M}\rangle \xrightarrow{\texttt{rd(x,d)}_\iota} \langle \mathcal{S}', \mathcal{M}\rangle$, where $\mathcal{S}'(\iota) = \mathsf{q}'$ and $\mathcal{S}'(\iota') = \mathcal{S}(\iota')$ for all $\iota' \neq \iota$.
- **Write:** If $\mathcal{S}(\iota) = \mathsf{q}$ and $\mathsf{q} \xrightarrow{\texttt{wr(x,d)}} \mathsf{q}'$, then $\langle \mathcal{S}, \mathcal{M}\rangle \xrightarrow{\texttt{wr(x,d)}_\iota} \langle \mathcal{S}', \mathcal{M}'\rangle$, where $\mathcal{S}'(\iota) = \mathsf{q}'$, $\mathcal{S}'(\iota') = \mathcal{S}(\iota')$ for all $\iota' \neq \iota$, $\mathcal{M}'(\mathsf{x}) = \mathsf{d}$ and $\mathcal{M}'(\mathsf{x}') = \mathcal{M}(\mathsf{x}')$ for all $\mathsf{x}' \neq \mathsf{x}$.
- **ARW:** If $\mathcal{S}(\iota) = \mathsf{q}$, $\mathcal{M}(\mathsf{x}) = \mathsf{d}$ and $\mathsf{q} \xrightarrow{\texttt{arw(x,d,d}'\texttt{)}} \mathsf{q}'$, then $\langle \mathcal{S}, \mathcal{M}\rangle \xrightarrow{\texttt{arw(x,d,d}'\texttt{)}_\iota} \langle \mathcal{S}', \mathcal{M}'\rangle$, where $\mathcal{S}'(\iota) = \mathsf{q}'$, $\mathcal{S}'(\iota') = \mathcal{S}(\iota')$ for all $\iota' \neq \iota$, $\mathcal{M}'(\mathsf{x}) = \mathsf{d}$ and $\mathcal{M}'(\mathsf{x}') = \mathcal{M}(\mathsf{x}')$ for all $\mathsf{x}' \neq \mathsf{x}$.
- **Skip:** If $\mathcal{S}(\iota) = \mathsf{q}$ and $\mathsf{q} \xrightarrow{\texttt{skip}} \mathsf{q}'$, then $\langle \mathcal{S}, \mathcal{M}\rangle \xrightarrow{\texttt{skip}_\iota} \langle \mathcal{S}', \mathcal{M}\rangle$, where $\mathcal{S}'(\iota) = \mathsf{q}'$ and $\mathcal{S}'(\iota') = \mathcal{S}(\iota')$ for all $\iota' \neq \iota$.

The memory fence operation `mf` is not used in programs running under SC semantics. If needed, it can be formally defined as following the same semantics as `skip`.

A global state $\mathcal{S}_F$ is *reachable* from an initial configuration c_0, if there is a configuration c_F with $\mathcal{S}(\mathsf{c}_F) = \mathcal{S}_F$ such that c_F is reachable from c_0 in $\mathcal{T}_\mathcal{P}^{\mathsf{SC}}$. The **state reachability problem** of SC is, given a program $\mathcal{P}$, an initial configuration c_0 and a final global state $\mathcal{S}_F$, to decide whether $\mathcal{S}_F$ is reachable from c_0 in $\mathcal{T}_\mathcal{P}^{\mathsf{SC}}$.

3.3. **TSO Semantics.** Under TSO semantics, the processes of a concurrent program do not interact with the shared memory directly, but indirectly through FIFO *store buffers* instead. When performing a *write* instruction $\texttt{wr}(\mathsf{x}, \mathsf{d})$, the process adds a new message $\langle \mathsf{x}, \mathsf{d}\rangle$ to the tail of its own store buffer. A *read* instruction $\texttt{rd}(\mathsf{x}, \mathsf{d})$ works differently depending on the current buffer content of the process. If its buffer contains a write message on variable x, the value d must correspond to the value of the most recent such message. Otherwise, the value is read directly from memory. The *atomic read-write* instruction $\texttt{arw}(\mathsf{x}, \mathsf{d}, \mathsf{d}')$ is only enabled if the buffer of the process is empty and the value of x in the memory is d. When executed, this instruction directly changes the value of x in the memory to d'. A *skip* instruction only changes the local state of the process. The *memory fence* instruction is disabled, i.e. it cannot be executed, unless the buffer of the process is empty. Additionally, at any point during the execution, the process can *update* the write message at the head of its buffer to the memory. For example, if the oldest message in the buffer is $\langle \mathsf{x}, \mathsf{d}\rangle$, it will be removed from the buffer and the memory value of variable x will be updated to contain the value d. This happens in a non-deterministic manner.

read-own-write $$\frac{\mathsf{q} \xrightarrow{\mathtt{rd(x,d)}} \mathsf{q}' \qquad \mathcal{S}(\iota)=\mathsf{q} \qquad \mathcal{B}(\iota)|_{\{\mathsf{x}\}\times\mathsf{Dom}}=\langle\mathsf{x},\mathsf{d}\rangle\cdot\mathsf{w}}{\langle\mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\mathtt{rd(x,d)}_\iota} \langle\mathcal{S}[\iota\leftarrow\mathsf{q}'],\mathcal{B},\mathcal{M}\rangle}$$

read-from-memory $$\frac{\mathsf{q} \xrightarrow{\mathtt{rd(x,d)}} \mathsf{q}' \qquad \mathcal{S}(\iota)=\mathsf{q} \qquad \mathcal{B}(\iota)|_{\{\mathsf{x}\}\times\mathsf{Dom}}=\varepsilon \qquad \mathcal{M}(\mathsf{x})=\mathsf{d}}{\langle\mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\mathtt{rd(x,d)}_\iota} \langle\mathcal{S}[\iota\leftarrow\mathsf{q}'],\mathcal{B},\mathcal{M}\rangle}$$

write $$\frac{\mathsf{q} \xrightarrow{\mathtt{wr(x,d)}} \mathsf{q}' \qquad \mathcal{S}(\iota)=\mathsf{q}}{\langle\mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\mathtt{wr(x,d)}_\iota} \langle\mathcal{S}[\iota\leftarrow\mathsf{q}'],\mathcal{B}[\iota\leftarrow\langle\mathsf{x},\mathsf{d}\rangle\cdot\mathcal{B}(\iota)],\mathcal{M}\rangle}$$

arw $$\frac{\mathsf{q} \xrightarrow{\mathtt{arw(x,d,d')}} \mathsf{q}' \qquad \mathcal{S}(\iota)=\mathsf{q} \qquad \mathcal{B}(\iota)=\varepsilon \qquad \mathcal{M}(\mathsf{x})=\mathsf{d}}{\langle\mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\mathtt{arw(x,d,d')}_\iota} \langle\mathcal{S}[\iota\leftarrow\mathsf{q}'],\mathcal{B},\mathcal{M}[\mathsf{x}\leftarrow\mathsf{d}']\rangle}$$

skip $$\frac{\mathsf{q} \xrightarrow{\mathtt{skip}} \mathsf{q}' \qquad \mathcal{S}(\iota)=\mathsf{q}}{\langle\mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\mathtt{skip}_\iota} \langle\mathcal{S}[\iota\leftarrow\mathsf{q}'],\mathcal{B},\mathcal{M}\rangle}$$

memory-fence $$\frac{\mathsf{q} \xrightarrow{\mathtt{mf}} \mathsf{q}' \qquad \mathcal{S}(\iota)=\mathsf{q} \qquad \mathcal{B}(\iota)=\varepsilon}{\langle\mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\mathtt{mf}_\iota} \langle\mathcal{S}[\iota\leftarrow\mathsf{q}'],\mathcal{B},\mathcal{M}\rangle}$$

update $$\frac{\mathcal{B}(\iota)=\mathsf{w}\cdot\langle\mathsf{x},\mathsf{d}\rangle}{\langle\mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\mathtt{up}_\iota} \langle\mathcal{S},\mathcal{B}[\iota\leftarrow\mathsf{w}],\mathcal{M}[\mathsf{x}\leftarrow\mathsf{d}]\rangle}$$

FIGURE 5. TSO semantics

Formally, we introduce a TSO *configuration* as a tuple $\mathsf{c} = \langle\mathcal{S},\mathcal{B},\mathcal{M}\rangle$, where:

- $\mathcal{S} : \mathcal{I} \to \bigcup_{\iota\in\mathcal{I}} \mathsf{Q}^\iota$ is a global state of $\mathcal{P}$.
- $\mathcal{B} : \mathcal{I} \to (\mathsf{Vars} \times \mathsf{Dom})^*$ represents the buffer state of each process.
- $\mathcal{M} : \mathsf{Vars} \to \mathsf{Dom}$ represents the memory state of each shared variable.

Given a configuration c, we write $\mathcal{S}(\mathsf{c})$, $\mathcal{B}(\mathsf{c})$ and $\mathcal{M}(\mathsf{c})$ for the global program state, buffer state and memory state of c. The semantics of a concurrent program running under TSO is defined by a transition system $\mathcal{T}_\mathcal{P}^{\mathsf{TSO}} = \langle\mathsf{C}_\mathcal{P}^{\mathsf{TSO}}, \mathsf{L}_\mathcal{P}^{\mathsf{TSO}}, \to_\mathcal{P}^{\mathsf{TSO}}\rangle$, where $\mathsf{C}_\mathcal{P}^{\mathsf{TSO}}$ is the set of all possible TSO configurations and $\mathsf{L}_\mathcal{P}^{\mathsf{TSO}} := \{\mathsf{instr}_\iota \mid \mathsf{instr} \in \mathsf{Instrs}, \iota \in \mathcal{I}\} \cup \{\mathsf{up}_\iota \mid \iota \in \mathcal{I}\}$ is the set of labels. The transition relation $\to$ (i.e. $\to_\mathcal{P}^{\mathsf{TSO}}$) is given by the rules in Figure 5, where we use $\mathcal{B}(\iota)|_{\{\mathsf{x}\}\times\mathsf{Dom}}$ to denote the restriction of $\mathcal{B}(\iota)$ to write messages on the variable x.

A global state $\mathcal{S}_F$ is *reachable* from an initial configuration c_0, if there is a configuration c_F with $\mathcal{S}(\mathsf{c}_F) = \mathcal{S}_F$ such that c_F is reachable from c_0 in $\mathcal{T}_\mathcal{P}$. The **state reachability problem** of TSO is, given a program $\mathcal{P}$, an initial configuration c_0 and a final global state $\mathcal{S}_F$, to decide whether $\mathcal{S}_F$ is reachable from c_0 in $\mathcal{T}_\mathcal{P}$.

We define up^* to be the transitive closure of $\{\mathsf{up}_\iota \mid \iota \in \mathcal{I}\}$, i.e. $\mathsf{c}_1 \xrightarrow{\mathsf{up}^*}_\mathcal{P} \mathsf{c}_2$ if and only if c_2 can be obtained from c_1 by some amount of buffer updates.

4. GAMES

4.1. **Definitions.** A *(safety) game* is an unlabelled transition sytem, in which two players A and B take turns making a *move* in the transition system, i.e. changing the state of the game from one configuration to an adjacent one. The set of configurations is partitioned into two sets C_A and C_B representing the configurations of player A and B, respectively. Whenever the game is in a configuration that belongs to player A, it is her turn to decide which move to make. This continues until the game reaches a configuration that belongs to player B, at which point the control is passed to her instead. The goal of player B is to reach a given set of final configurations, while player A tries to avoid this. Thus, it can also be seen as a *reachability* game with respect to player B.

Formally, a game is defined as a tuple $\mathcal{G} = \langle \mathsf{C}, \mathsf{C}_A, \mathsf{C}_B, \rightarrow, \mathsf{C}_F \rangle$, where C is the set of configurations, C_A and C_B form a partition of C, the transition relation is $\rightarrow \subseteq \mathsf{C} \times \mathsf{C}$, and $\mathsf{C}_F \subseteq \mathsf{C}_A$ is a set of *final configurations*. Furthermore, we require that $\mathcal{G}$ is deadlock-free, i.e. $\mathrm{Post}(\mathsf{c}) \neq \emptyset$ for all $\mathsf{c} \in \mathsf{C}$.

A *play* P of $\mathcal{G}$ is an infinite sequence $\mathsf{c}_0, \mathsf{c}_1, \ldots$ such that $\mathsf{c}_i \rightarrow \mathsf{c}_{i+1}$ for all $i \in \mathbb{N}$. In the context of safety games, P is *winning* for player B if there is $i \in \mathbb{N}$ such that $\mathsf{c}_i \in \mathsf{C}_F$. Otherwise, it is *winning* for player A. This means that player B tries to force the play into C_F, while player A tries to avoid this.

A *strategy* of player A is a partial function $\sigma_A : \mathsf{C}^* \rightharpoonup \mathsf{C}_B$, such that $\sigma_A(\mathsf{c}_0, \ldots, \mathsf{c}_n)$ is defined if and only if $\mathsf{c}_0, \ldots, \mathsf{c}_n$ is a prefix of a play, $\mathsf{c}_n \in \mathsf{C}_A$ and $\sigma_A(\mathsf{c}_0, \ldots, \mathsf{c}_n) \in \mathrm{Post}(\mathsf{c}_n)$. A strategy σ_A is called *positional*, if it only depends on c_n, i.e. if $\sigma_A(\mathsf{c}_0, \ldots, \mathsf{c}_n) = \sigma_A(\mathsf{c}_n)$ for all $(\mathsf{c}_0, \ldots, \mathsf{c}_n)$ on which σ_A is defined. Thus, a positional strategy is usually given as a total function $\sigma_A : \mathsf{C}_A \rightarrow \mathsf{C}$. Given two games $\mathcal{G}$ and $\mathcal{G}'$ and a strategy σ_A for $\mathcal{G}$, an *extension* of σ_A to $\mathcal{G}'$ is a strategy σ'_A of $\mathcal{G}'$ that is also an extension of σ_A to the configuration set of $\mathcal{G}'$ in the mathematical sense, i.e. $\sigma_A(\mathsf{c}_0, \ldots, \mathsf{c}_n) = \sigma'_A(\mathsf{c}_0, \ldots, \mathsf{c}_n)$ for all $(\mathsf{c}_0, \ldots, \mathsf{c}_n)$ on which σ_A is defined. Conversely, σ_A is called the *restriction* of σ'_A to $\mathcal{G}$. For player B, strategies are defined accordingly.

Two strategies σ_A and σ_B together with an initial configuration c_0 induce a play $\mathsf{P}(\mathsf{c}_0, \sigma_A, \sigma_B) = \mathsf{c}_0, \mathsf{c}_1, \ldots$ such that $\mathsf{c}_{i+1} = \sigma_A(\mathsf{c}_0, \ldots, \mathsf{c}_i)$ for all $\mathsf{c}_i \in \mathsf{C}_A$ and $\mathsf{c}_{i+1} = \sigma_B(\mathsf{c}_0, \ldots, \mathsf{c}_i)$ for all $\mathsf{c}_i \in \mathsf{C}_B$. A strategy σ_A is *winning* from a configuration c, if for *all* strategies σ_B it holds that $\mathsf{P}(\sigma_A, \sigma_B, \mathsf{c})$ is a winning play for player A. A configuration c is *winning* for player A if she has a strategy that is winning from c. Equivalent notions exist for player B. The **safety problem** for a game $\mathcal{G}$ and a configuration c is to decide whether c is winning for player A.

Lemma 4.1 (Proposition 2.21 in [Maz01])**.** *In safety games, every configuration is winning for exactly one player. A player with a winning strategy also has a positional winning strategy.*

Since we only consider safety games in this paper, strategies will be considered to be positional unless explicitly stated otherwise. Furthermore, Lemma 4.1 implies the following:

- $\mathsf{c}_A \in \mathsf{C}_A$ is winning for player A $\iff$ there is $\mathsf{c} \in \mathrm{Post}(\mathsf{c}_A)$ that is winning for player A.
- $\mathsf{c}_B \in \mathsf{C}_B$ is winning for player A $\iff$ all $\mathsf{c} \in \mathrm{Post}(\mathsf{c}_B)$ are winning for player A.

A *finite game* is a game with a finite set of configurations. It is rather intuitive that the safety problem is decidable for finite games, e.g. by applying a backward induction algorithm. In particular, the winning configurations for each player are computable in linear time:

Lemma 4.2 (Chapter 2 in [GTW02])**.** *Computing the set of winning configurations for a finite game with n configurations and m transitions is in $\mathcal{O}(n + m)$.*

$$\begin{tikzcd}
\mathsf{c}_1 \arrow[r] \arrow[d, dash, "\approx" description] & \mathsf{c}_3 \arrow[d, dash, "\approx" description] \\
\mathsf{c}_2 \arrow[r] & \exists\ \mathsf{c}_4
\end{tikzcd}$$

(A) zig property

$$\begin{tikzcd}
\mathsf{c}_1 \arrow[r] \arrow[d, dash, "\approx" description] & \exists\ \mathsf{c}_3 \arrow[d, dash, "\approx" description] \\
\mathsf{c}_2 \arrow[r] & \mathsf{c}_4
\end{tikzcd}$$

(B) zag property

FIGURE 6. Configurations in a bisimulation

$$\begin{tikzcd}
\mathsf{c}^{\mathcal{G}}_{2n} \arrow[r, "\sigma^{\mathcal{G}}_A"] \arrow[d, dash, "\approx" description] & \mathsf{c}^{\mathcal{G}}_{2n+1} \arrow[r, "\sigma^{\mathcal{G}}_B"] \arrow[d, dash, "\approx" description] & \exists\ \mathsf{c}^{\mathcal{G}}_{2n+2} \arrow[d, dash, "\approx" description] \\
\mathsf{c}^{\mathcal{H}}_{2n} \arrow[r, "\sigma^{\mathcal{H}}_A"] & \exists\ \mathsf{c}^{\mathcal{H}}_{2n+1} \arrow[r, "\sigma^{\mathcal{H}}_B"] & \mathsf{c}^{\mathcal{H}}_{2n+2}
\end{tikzcd}$$

FIGURE 7. Configurations and strategies in Lemma 4.3

Bisimulations. A *bisimulation* (also called *zig-zag relation*) between two games $\mathcal{G} = \langle \mathsf{C}^{\mathcal{G}}, \mathsf{C}^{\mathcal{G}}_A, \mathsf{C}^{\mathcal{G}}_B, \rightarrow, \mathsf{C}^{\mathcal{G}}_F \rangle$ and $\mathcal{H} = \langle \mathsf{C}^{\mathcal{H}}, \mathsf{C}^{\mathcal{H}}_A, \mathsf{C}^{\mathcal{H}}_B, \rightarrow, \mathsf{C}^{\mathcal{H}}_F \rangle$ is a relation $\mathsf{Z} \subseteq \mathsf{C}^{\mathcal{G}} \times \mathsf{C}^{\mathcal{H}}$ such that for all pairs of related configurations $(\mathsf{c}_1, \mathsf{c}_2) \in \mathsf{Z}$ it holds that (cf. Figure 6):

- (*zig*) for each transition $\mathsf{c}_1 \rightarrow \mathsf{c}_3$ there is a transition $\mathsf{c}_2 \rightarrow \mathsf{c}_4$ such that $(\mathsf{c}_3, \mathsf{c}_4) \in \mathsf{Z}$.
- (*zag*) for each transition $\mathsf{c}_2 \rightarrow \mathsf{c}_4$ there is a transition $\mathsf{c}_1 \rightarrow \mathsf{c}_3$ such that $(\mathsf{c}_3, \mathsf{c}_4) \in \mathsf{Z}$.
- c_1 and c_2 are owned by the same player: $\mathsf{c}_1 \in \mathsf{C}^{\mathcal{G}}_A \iff \mathsf{c}_2 \in \mathsf{C}^{\mathcal{H}}_A$
- c_1 and c_2 agree on being a final configuration: $\mathsf{c}_1 \in \mathsf{C}^{\mathcal{G}}_F \iff \mathsf{c}_2 \in \mathsf{C}^{\mathcal{H}}_F$

We say that two related configurations c_1 and c_2 are *bisimilar* and write $\mathsf{c}_1 \approx \mathsf{c}_2$. We call $\mathcal{G}$ and $\mathcal{H}$ *bisimilar* if there is a bisimulation between them. It is common knowledge in game theory that winning strategies are preserved under bisimulations:

Lemma 4.3. *Given two bisimilar configurations $\mathsf{c}^{\mathcal{G}}_0 \in \mathsf{C}^{\mathcal{G}}$ and $\mathsf{c}^{\mathcal{H}}_0 \in \mathsf{C}^{\mathcal{H}}$, it holds that $\mathsf{c}^{\mathcal{G}}_0$ is winning for player A if and only if $\mathsf{c}^{\mathcal{H}}_0$ is winning for player A.*

Proof. Suppose that $\mathsf{c}^{\mathcal{G}}_0$ is winning for player A with (positional) strategy $\sigma^{\mathcal{G}}_A$ and consider the case $\mathsf{c}^{\mathcal{G}}_0 \in \mathsf{C}^{\mathcal{G}}_A$. Let $\sigma^{\mathcal{H}}_B$ be an arbitrary strategy for player A in $\mathcal{H}$. In the following, we will describe two (non-positional) winning strategies $\sigma^{\mathcal{H}}_A$ and $\sigma^{\mathcal{G}}_B$.

For $n \in \mathbb{N}$, define recursively (see Figure 7):

- $\mathsf{c}^{\mathcal{G}}_{2n+1} := \sigma^{\mathcal{G}}_A(\mathsf{c}^{\mathcal{G}}_{2n})$
- $\mathsf{c}^{\mathcal{H}}_{2n+1}$ such that $\mathsf{c}^{\mathcal{H}}_{2n} \rightarrow \mathsf{c}^{\mathcal{H}}_{2n+1}$ and $\mathsf{c}^{\mathcal{G}}_{2n+1} \approx \mathsf{c}^{\mathcal{H}}_{2n+1}$, which exists by the zig property of Z.
- $\sigma^{\mathcal{H}}_A(\mathsf{c}^{\mathcal{H}}_0, \ldots, \mathsf{c}^{\mathcal{H}}_{2n}) := \mathsf{c}^{\mathcal{H}}_{2n+1}$
- $\mathsf{c}^{\mathcal{H}}_{2n+2} := \sigma^{\mathcal{H}}_B(\mathsf{c}^{\mathcal{H}}_{2n+1})$
- $\mathsf{c}^{\mathcal{G}}_{2n+2}$ such that $\mathsf{c}^{\mathcal{G}}_{2n+1} \rightarrow \mathsf{c}^{\mathcal{G}}_{2n+2}$ and $\mathsf{c}^{\mathcal{G}}_{2n+2} \approx \mathsf{c}^{\mathcal{H}}_{2n+2}$, which exists by the zag property of Z.
- $\sigma^{\mathcal{G}}_B(\mathsf{c}^{\mathcal{G}}_0, \ldots, \mathsf{c}^{\mathcal{G}}_{2n+1}) := \mathsf{c}^{\mathcal{G}}_{2n+2}$

Since $\sigma^{\mathcal{G}}_A$ is a winning strategy for player A in $\mathcal{G}$, the sequence $\mathsf{c}^{\mathcal{G}}_0, \mathsf{c}^{\mathcal{G}}_1, \ldots$ is a winning play for player A, i.e. it does not visit the set of final configurations $\mathsf{C}^{\mathcal{G}}_F$. Thus, $\mathsf{c}^{\mathcal{H}}_0, \mathsf{c}^{\mathcal{H}}_1, \ldots$

is also a winning play for player A, because for all $n \in \mathbb{N}$, $\mathsf{c}_n^{\mathcal{G}} \approx \mathsf{c}_n^{\mathcal{H}}$ and $\mathsf{c}_n^{\mathcal{G}} \notin \mathsf{C}_F^{\mathcal{G}}$ implies $\mathsf{c}_n^{\mathcal{H}} \notin \mathsf{C}_F^{\mathcal{H}}$. Since $\sigma_B^{\mathcal{H}}$ was chosen arbitrarily, this shows that $\sigma_A^{\mathcal{H}}$ is a winning strategy for player A in $\mathcal{H}$. Note that by Lemma 4.1, player A could also choose a positional strategy instead.

The case where $\mathsf{c}_0^{\mathcal{G}} \in \mathsf{C}_B^{\mathcal{G}}$ is similar, but with the recursive definition shifted by one, i.e. $\mathsf{c}_{2n+1}^{\mathcal{G}} := \sigma_B^{\mathcal{G}}(\mathsf{c}_{2n}^{\mathcal{G}})$ and so on. □

4.2. **SC Games.** We want to define an *SC game* as a safety game where the underlying transition system follows Sequential Consistency semantics. While the set of game configurations, the transition relation of the game and the set of its final configurations could be taken directly from the transition system $\mathcal{T}_{\mathcal{P}}^{\mathsf{SC}}$, it is not straightforward to decide how the configuration set should be partitioned into C_A and C_B. One idea that comes to mind would be to partition the *local states* of each process. But whose turn is it if Proc^1 is in a state belonging to player A but Proc^2 is in a state belonging to player B? Which player is allowed to move first? The standard approach to solve this problem is to consider the product of the concurrent processes instead, i.e. to partition the set of *global states.* Another common approach is to switch to turn-based games, where the players alternate in making a move in the game.

For this present work, we chose the latter approach, which means that the players take turns executing exactly one instruction from an arbitrary process. This is because we can argue that turn-based games do not pose a significant restriction compared to games with state ownership, but instead can emulate most of their behaviour (see below).

In the following formal definition of an SC game, we will create two copies of each SC configuration, annotated with A and B, respectively. The transition relation of the game will be restricted to transitions between configurations of different players. The use of two identical sets of configurations, one for each player, may seem unnecessary at first glance. However, this distinction is crucial for the definition of TSO games in the next section, where the available instructions to execute differ between the two players A and B. Furthermore, it allows us to describe turn-based games using the framework of games with configuration ownership, as introduced in the beginning of this section.

A program $\mathcal{P} = \langle \mathsf{Proc}^{\iota} \rangle_{\iota \in \mathcal{I}}$ and a set of final local states $\mathsf{Q}_F^{\mathcal{P}} \subseteq \mathsf{Q}^{\mathcal{P}}$ induce a safety game $\mathcal{G}^{\mathsf{SC}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}}) = \langle \mathsf{C}, \mathsf{C}_A, \mathsf{C}_B, \rightarrow, \mathsf{C}_F \rangle$ as follows. The sets C_A and C_B are copies of the set $\mathsf{C}^{\mathcal{P}}$ of TSO configurations, annotated by A and B, respectively:

$$\mathsf{C}_A := \{\mathsf{c}_A \mid \mathsf{c} \in \mathsf{C}_{\mathcal{P}}^{\mathsf{SC}}\} \qquad \text{and} \qquad \mathsf{C}_B := \{\mathsf{c}_B \mid \mathsf{c} \in \mathsf{C}_{\mathcal{P}}^{\mathsf{SC}}\}$$

The set of final configurations is defined as:

$$\mathsf{C}_F := \{\langle \mathcal{S}, \mathcal{B}, \mathcal{M} \rangle_A \in \mathsf{C}_A \mid \exists\, \iota \in \mathcal{I} : \mathcal{S}(\iota) \in \mathsf{Q}_F^{\mathcal{P}}\}$$

That is, it is the set of all configurations where at least one process is in a final state. As described previously, the transition relation is defined by the program's instructions, in a way that the two players take turns alternatingly to execute these instructions: For each transition $\mathsf{c} \xrightarrow{\mathsf{instr}_{\iota}} \mathsf{c}'$ where $\mathsf{c}, \mathsf{c}' \in \mathsf{C}_{\mathcal{P}}^{\mathsf{SC}}$, $\iota \in \mathcal{I}$ and $\mathsf{instr} \in \mathsf{Instrs}$, it holds that $\mathsf{c}_A \rightarrow \mathsf{c}'_B$ and $\mathsf{c}_B \rightarrow \mathsf{c}'_A$. Figure 8 shows the transitions relation of the SC game induced by the program from Figure 3.

Note that in the analysis of SC games (and later of TSO games), we often talk about the program instruction that gives rise to a transition of $\mathcal{G}^{\mathsf{SC}}$ instead of talking about the

$$\begin{tikzcd}
\langle(\mathsf{q}_1,\mathsf{r}_1),\{x\mapsto 0\}\rangle_A \arrow[d, "\mathtt{wr}(\mathsf{x},1)"] \arrow[r, shift right, "\mathtt{skip}"'] & \langle(\mathsf{q}_1,\mathsf{r}_1),\{x\mapsto 0\}\rangle_B \arrow[l, shift right, "\mathtt{skip}"'] \arrow[d, "\mathtt{wr}(\mathsf{x},1)"] \\
\langle(\mathsf{q}_2,\mathsf{r}_1),\{x\mapsto 1\}\rangle_B \arrow[d, "\mathtt{rd}(\mathsf{x},1)"] \arrow[r, shift right, "\mathtt{skip}"'] & \langle(\mathsf{q}_2,\mathsf{r}_1),\{x\mapsto 1\}\rangle_A \arrow[l, shift right, "\mathtt{skip}"'] \arrow[d, "\mathtt{rd}(\mathsf{x},1)"] \\
\langle(\mathsf{q}_2,\mathsf{r}_2),\{x\mapsto 1\}\rangle_A \arrow[r, shift right, "\mathtt{skip}"'] & \langle(\mathsf{q}_2,\mathsf{r}_2),\{x\mapsto 1\}\rangle_B \arrow[l, shift right, "\mathtt{skip}"']
\end{tikzcd}$$

FIGURE 8. The transition relation of the SC game $\mathcal{G}^{\mathsf{SC}}(\mathcal{P},\{\mathsf{r}_2\})$ induced by the program $\mathcal{P}$ from Figure 3. Note that only configurations reachable from $\langle(\mathsf{q}_1,\mathsf{r}_1),\{x\mapsto 0\}\rangle_A$ are shown. The labels in blue are not formally part of the game definition, but are included to indicate which instruction of $\mathcal{P}$ gives rise to the transition. The configuration in green is the final state induced by the set of final local states $\mathsf{Q}_F^{\mathcal{P}} := \{\mathsf{r}_2\}$.

transition explicitly. For example, in a situation where the game is in some configuration c_A with $\mathcal{S}(\mathsf{c})(\iota) = \mathsf{q}$, we might say that player A executes instruction $\mathsf{q} \xrightarrow{\mathsf{instr}} \mathsf{q}'$ and mean that player A moves from c_A to c'_B, where c' is the unique configuration obtained from executing instr at c. Similarly, for better readability we might drop the index A or B from a game configuration if it is clear from the context which player's turn it is.

State ownership. As mentioned above, turn-based games can emulate games with state ownership. The following example illustrates this informal idea. Consider an instruction $\mathsf{q}_1 \xrightarrow{\mathsf{instr}} \mathsf{q}_2$ in some process, where q_1 is supposed to belong to player B. We modify the process by adding some new states and transitions, as shown in Figure 9. Note that through a suitable definition of the set of final configurations C_F we can ensure that reaching q_F means that player B wins the game. Suppose that the game is in a configuration where the process is in q_1, it is player B's turn and she wants to execute the instruction instr. She can do so by taking the transition from q_1 to q'_2. Afterwards, it is the turn of player A. She is forced to respond by taking the $\mathtt{skip}$ instruction from q'_2 to q_2. Otherwise, player B could move to q_F in her turn and win immediately. Now consider the same situation as before but it is the turn of player A instead. If she would execute the instruction instr and move to q'_2, then player B could respond with moving to q_F and again win immediately. Therefore, player A is prevented from executing any instruction starting in q_1.

Complexity. In the following, we consider the safety problem for SC games. Unexpectedly, this problem is EXPTIME-hard even for games played on a single process. We prove this

$$\begin{tikzcd}
 & & \mathsf{q}_2 \\
\mathsf{q}_1 \arrow[r, "\mathsf{instr}"] & \mathsf{q}_2' \arrow[ur, "\mathtt{skip}"] \arrow[dr, "\mathtt{skip}"'] & \\
 & & \mathsf{q}_F
\end{tikzcd}$$

Figure 9. The gadget for a transition $\mathsf{q}_1 \xrightarrow{\mathsf{instr}} \mathsf{q}_2$

$$\text{if } \mathsf{q} \in \mathsf{Q}_\exists\text{: } \underbrace{\text{for all } \langle \mathsf{q}, \mathsf{b}, \mathsf{q}', \mathsf{b}', \mathsf{D}\rangle \in \delta}$$

$$\begin{tikzcd}
 & & (\mathsf{q}', \mathsf{i}, \mathsf{b}', \mathsf{D})^A \arrow[dr, "\mathtt{skip}"] & & \\
(\mathsf{q}, \mathsf{i}) \arrow[r, "\mathtt{rd}(\mathsf{i}, \mathsf{b})"] & (\mathsf{q}, \mathsf{i}, \mathsf{b})^B \arrow[ur, "\mathtt{skip}"] \arrow[dr, "\mathtt{skip}"'] & & (\mathsf{q}', \mathsf{i}, \mathsf{b}', \mathsf{D})^B \arrow[r, "\mathtt{wr}(\mathsf{i}, \mathsf{b}')"] & (\mathsf{q}', \mathsf{i} + \mathsf{D}) \\
 & & (\mathsf{q}, \mathsf{i}, \mathsf{b})^A \arrow[ur, "\mathtt{skip}"'] & &
\end{tikzcd}$$

$$\text{if } \mathsf{q} \in \mathsf{Q}_\forall\text{: } \overbrace{\text{for all } \langle \mathsf{q}, \mathsf{b}, \mathsf{q}', \mathsf{b}', \mathsf{D}\rangle \in \delta}$$

Figure 10. Construction of a program $\mathcal{P}$ that simulates an ATM $\mathcal{A}$. The states and transitions shown are added to Proc for all $\mathsf{q} \in \mathsf{Q}$ and $-p(n) \leq \mathsf{i} \leq p(n)$. We set $\mathsf{L} := -1$ and $\mathsf{R} := 1$ in the expression $\mathsf{i} + \mathsf{D}$.

by reducing the *word acceptance problem* of polynomial-space bounded *alternating Turing machines* (ATM) to the safety problem of a single-process SC game.

Consider an ATM $\mathcal{A} = \langle \Sigma, \mathsf{Q}, \mathsf{q}_0, \mathsf{q}_F, \mathsf{Q}_\exists, \mathsf{Q}_\forall, \delta\rangle$ that is space-bounded by some polynomial $p(n)$. Given a word $\mathsf{w} \in \Sigma^n$, we construct a concurrent program $\mathcal{P}$ with a single process that simulates $\mathcal{A}$. The key idea is to store the state and head position of the ATM in the local states of the process, and use a set of variables to save the word on the working tape. Based on the alternations of the Turing machine, either player A or player B decides which transition the program will simulate. More precisely, we let player B simulate $\mathcal{A}$ at the existential states while player A has control over the simulation at the universal states. Player B will win the SC game induced by $\mathcal{P}$ if and only if $\mathcal{A}$ accepts w.

$\mathcal{P} = \langle \mathsf{Proc}\rangle$ uses the variables $\mathsf{Vars} := \{\mathsf{i} \mid -p(n) \leq \mathsf{i} \leq p(n)\} \subset \mathbb{Z}$ over the domain $\mathsf{Dom} := \Sigma$ to store the content of the tape. The control state of $\mathcal{A}$, the head position i and all necessary intermediate values are stored in the local states of Proc. The construction of Proc is shown in Figure 10. Note that e.g. $(\mathsf{q}', \mathsf{i}, b', \mathsf{D})^A$ is a *state* of Proc and not a *configuration* of $\mathcal{P}$ or even $\mathcal{G}^{\mathsf{SC}}$.

Define the set of local final states $\mathsf{Q}_F^{\mathcal{P}} := \{(\mathsf{q}_F, \mathsf{i}) \mid -p(n) \leq \mathsf{i} \leq p(n)\}$ and the initial memory state $\mathcal{M}_0$, where $\mathcal{M}_0(\mathsf{i}) = \mathsf{w}(\mathsf{i})$ for $1 \leq \mathsf{i} \leq n$ and $\mathcal{M}_0(\mathsf{i}) = \textvisiblespace$ otherwise.

Theorem 4.4. *$\mathcal{A}$ accepts $\mathsf{w} \in \Sigma^n$ if and only if player B wins the game $\mathcal{G}^{\mathsf{SC}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}})$ from the initial configuration $\langle(\mathsf{q}_0, 1), \mathcal{M}_0\rangle_A$.*

Proof. Recall the definitions of C_∞ and C_k from subsection 2.2 and suppose that $\mathcal{A}$ accepts w. We argue that player B can force the simulation into q_F and describe a strategy for her to do so. The main idea is that if the simulation is in a configuration $\mathsf{c} \in \mathsf{C}_k$ $(k > 0)$, then player B can force the simulation into another configuration $\mathsf{c}' \in \mathsf{C}_{k-1}$.

Let the game be in configuration $\langle(\mathsf{q}, \mathsf{i}), \omega\rangle_A$ which simulates the $\mathcal{A}$-configuration $\langle \mathsf{q}, \mathsf{i}, \omega\rangle$. Note that (q, i) is the local state of Proc and $\omega : \mathsf{Vars} \to \Sigma$ is its memory state. (Formally, $\omega : \mathbb{Z} \to \Sigma$, but since $\mathsf{Vars} \subset \mathbb{Z}$, this is only a slight abuse of notation. Since $\omega(\mathsf{i}) = \text{␣}$ for all $\mathsf{i} \notin \mathsf{Vars}$, $\omega : \mathsf{Vars} \to \Sigma$ captures the same information as $\omega : \mathbb{Z} \to \Sigma$.) Let $\mathsf{b} = \omega(\mathsf{i})$ be the symbol under the head of the Turing machine and $\langle \mathsf{q}, \mathsf{i}, \omega\rangle \in \mathsf{C}_k$ for some $k > 0$. Player A has to take the only enabled transition, which is $\mathtt{rd}(\mathsf{i}, \mathsf{b})$ leading to the local state $(\mathsf{q}, \mathsf{i}, \mathsf{b})^B$. Now, the possible choices for player B depend on whether q is an existential or universal state.

If $\mathsf{q} \in \mathsf{Q}_\exists$, then player B chooses an $\mathcal{A}$-transition $\langle \mathsf{q}, \mathsf{b}, \mathsf{q}', \mathsf{b}', \mathsf{D}\rangle \in \delta$ with $\langle \mathsf{q}', \mathsf{i} + \mathsf{D}, \mathsf{w}[\mathsf{i} \leftarrow \mathsf{b}']\rangle \in \mathsf{C}_{k-1}$, which exists by definition of C_k. Player B then moves to the local state $(\mathsf{q}', \mathsf{i}, \mathsf{b}', \mathsf{D})^A$ and player A has to respond with $(\mathsf{q}', \mathsf{i}, \mathsf{b}', \mathsf{D})^B$.

Otherwise, if $\mathsf{q} \in \mathsf{Q}_\forall$, then player B must take the transition to the state $(\mathsf{q}, \mathsf{i}, \mathsf{b})^A$. From there, player A moves to some state $(\mathsf{q}', \mathsf{i}, \mathsf{b}', \mathsf{D})^B$. This implies that there is an $\mathcal{A}$-transition $\langle \mathsf{q}, \mathsf{b}, \mathsf{q}', \mathsf{b}', \mathsf{D}\rangle \in \delta$. From the construction of C_k, it follows that $\langle \mathsf{q}', \mathsf{i} + \mathsf{D}, \omega[\mathsf{i} \leftarrow \mathsf{b}']\rangle \in \mathsf{C}_{k-1}$.

In both cases, the game is now in some configuration $\langle(\mathsf{q}', \mathsf{i}, \mathsf{b}', \mathsf{D})^B, \omega\rangle_B$ with $\langle \mathsf{q}', \mathsf{i} + \mathsf{D}, \omega[\mathsf{i} \leftarrow \mathsf{b}']\rangle_A \in \mathsf{C}_{k-1}$. Player B takes the transition $\mathtt{wr}(\mathsf{i}, \mathsf{b}')$ and the game is in configuration $\langle(\mathsf{q}', \mathsf{i} + \mathsf{D}), \omega[\mathsf{i} \leftarrow \mathsf{b}']\rangle_A$. It follows by simple induction over k that player B can force the game into simulating an $\mathcal{A}$-configuration of C_0.

For the other direction, assume that $\mathcal{A}$ does not accept w. This time we describe a strategy for player A to win the game. Similar to the previous argumentation, we can show by induction that player A can simulate a sequence of configurations that are *not* accepting. The difference is that if $\mathsf{q} \in \mathsf{Q}_\exists$, then player B can never choose any transition that leads to an accepting configuration, while in the case of $\mathsf{q} \in \mathsf{Q}_\forall$, player A can always avoid accepting configurations. This follows from the fact that the simulated $\mathcal{A}$-configurations $\langle \mathsf{q}, \mathsf{i}, \omega\rangle$ are not elements of any of the C_k. □

Theorem 4.5. *The safety problem for SC games is* ExpTime*-complete.*

Proof. The word acceptance problem of a linearly bounded ATM $\mathcal{A}$ is ExpTime-hard [CS76, CKS81]. In Theorem 4.4, it is reduced to the safety problem for the SC game induced by a concurrent program $\mathcal{P}$. This program has $\mathcal{O}(|\mathsf{Q}| \cdot |\Sigma| \cdot p(n))$ local states, which is polynomial in the size of $\mathcal{A}$. It follows that the safety problem for TSO games is ExpTime-hard. Membership follows directly from Lemma 4.2 and the fact that for any $\mathcal{P}$, the game $\mathcal{G}^{\mathsf{SC}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}})$ has $\mathcal{O}(\Pi_{\iota \in \mathcal{I}} |\mathsf{Q}^\iota| \cdot |\mathsf{Dom}|^{|\mathsf{Vars}|})$ many configurations. □

4.3. TSO Games.

Given a $\mathcal{P} = \langle \mathsf{Proc}^\iota \rangle_{\iota \in \mathcal{I}}$ and a set of final local states $\mathsf{Q}_F^{\mathcal{P}} \subseteq \mathsf{Q}^{\mathcal{P}}$, the TSO game $\mathcal{G}^{\mathsf{TSO}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}})$ is defined similar to the SC game $\mathcal{G}^{\mathsf{SC}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}})$. The only addition is that the game needs to be able to handle buffer updates. Therefore, we allow one or both of the players to perform buffer updates, either before or after their turn. Which player is allowed to do so and when exactly she is allowed to do so depends on the concrete instantiation of the TSO game. The complete transition rules of the TSO game are:

$$\begin{tikzcd}
\langle(\mathsf{q}_1,\mathsf{r}_1),(\varepsilon,\varepsilon),\{x\mapsto 0\}\rangle_A \arrow[r, shift right, "\texttt{skip}"'] \arrow[d, "\texttt{wr}(\mathsf{x},1)"] \arrow[dd, bend right, "\texttt{wr}(\mathsf{x},1);\texttt{up}"'] & \langle(\mathsf{q}_1,\mathsf{r}_1),(\varepsilon,\varepsilon),\{x\mapsto 0\}\rangle_B \arrow[l, shift right, "\texttt{skip}"'] \arrow[d, "\texttt{wr}(\mathsf{x},1)"] \\
\langle(\mathsf{q}_2,\mathsf{r}_1),(\langle\mathsf{x},1\rangle,\varepsilon),\{x\mapsto 0\}\rangle_B \arrow[r, shift right, "\texttt{skip}"'] & \langle(\mathsf{q}_2,\mathsf{r}_1),(\langle\mathsf{x},1\rangle,\varepsilon),\{x\mapsto 0\}\rangle_A \arrow[l, shift right, "\texttt{skip}"'] \arrow[dl, "\texttt{skip};\texttt{up} \text{ or } \texttt{up};\texttt{skip}"] \arrow[dd, bend left, "\texttt{up};\texttt{rd}(\mathsf{x},1)"] \\
\langle(\mathsf{q}_2,\mathsf{r}_1),(\varepsilon,\varepsilon),\{x\mapsto 1\}\rangle_B \arrow[r, shift right, "\texttt{skip}"'] \arrow[d, "\texttt{rd}(\mathsf{x},1)"] & \langle(\mathsf{q}_2,\mathsf{r}_1),(\varepsilon,\varepsilon),\{x\mapsto 1\}\rangle_A \arrow[l, shift right, "\texttt{skip}"'] \arrow[d, "\texttt{rd}(\mathsf{x},1)"] \\
\langle(\mathsf{q}_2,\mathsf{r}_2),(\varepsilon,\varepsilon),\{x\mapsto 1\}\rangle_A \arrow[r, shift right, "\texttt{skip}"'] & \langle(\mathsf{q}_2,\mathsf{r}_2),(\varepsilon,\varepsilon),\{x\mapsto 1\}\rangle_B \arrow[l, shift right, "\texttt{skip}"']
\end{tikzcd}$$

Figure 11. The transition relation of the TSO game $\mathcal{G}^{\mathsf{TSO}}(\mathcal{P},\{\mathsf{r}_2\})$ induced by the program $\mathcal{P}$ from Figure 3, in the case where player A is allowed to update before and after her turn, but player B is not allowed to update buffer messages. Note that only configurations reachable from $\langle(\mathsf{q}_1,\mathsf{r}_1),(\varepsilon,\varepsilon),\{x\mapsto 0\}\rangle_A$ are shown. The labels in blue are not formally part of the game definition, but are included to indicate which instruction of $\mathcal{P}$ gives rise to the transition. The transitions in red are due to updates; they correspond to both an instruction and an update operation. The configuration in green is the final state induced by the set of final local states $\mathsf{Q}_F^{\mathcal{P}} := \{\mathsf{r}_2\}$.

- For each transition $\mathsf{c}\xrightarrow{\mathsf{instr}_\iota}\mathsf{c}'$ where $\mathsf{c},\mathsf{c}'\in\mathsf{C}^{\mathcal{P}}$, $\iota\in\mathcal{I}$ and $\mathsf{instr}\in\mathsf{Instrs}$, it holds that $\mathsf{c}_A\rightarrow\mathsf{c}'_B$ and $\mathsf{c}_B\rightarrow\mathsf{c}'_A$. This is the same as for SC and means that each player can execute any TSO instruction, but they take turns alternatingly.
- *If player A can update before her own turn:* For each transition $\mathsf{c}_A\rightarrow\mathsf{c}'_B$ introduced by any of the previous rules, it holds that $\tilde{\mathsf{c}}_A\rightarrow\mathsf{c}'_B$ for all $\tilde{\mathsf{c}}$ with $\tilde{\mathsf{c}}\xrightarrow{\mathsf{up}^*}\mathsf{c}$.
- *If player A can update after her own turn:* For each transition $\mathsf{c}_A\rightarrow\mathsf{c}'_B$ introduced by any of the previous rules, it holds that $\mathsf{c}_A\rightarrow\tilde{\mathsf{c}}'_B$ for all $\tilde{\mathsf{c}}'$ with $\mathsf{c}'\xrightarrow{\mathsf{up}^*}\tilde{\mathsf{c}}'$.
- The update rules for player B are defined in a similar manner.

From this definition, we obtain 16 different variants of TSO games, which differ in whether each of the players can update *never*, *before* her turn, *after* her turn, or *always* (before and after her turn). Figure 11 shows the transition relation of the TSO game induced by the program from Figure 3, in the case where player A is allowed to update buffer messages both before and after her turn, but player B is not allowed to do so.

We group games with similar decidability and complexity results together. An overview of these four groups is presented in Figure 2. Each group is described in detail in the following sections.

But first, we use the complexity results for SC games to obtain a lower bound for all groups of TSO games. Interestingly, we can argue that the exact type of TSO game is irrelevant. Note that the program $\mathcal{P}$ from Theorem 4.4 uses only one process and no memory fences or atomic read-writes. Consider the view the single process Proc has on a variable of the concurrent system. Independently of the buffer updates, the process will always read the last value that it has written: Either the last write is still in the buffer and will be read from there, or the process reads directly from the memory, which must contain the last value written by the process. Thus, any sequence of instructions executed in the SC game is also enabled in the TSO game. This is formalised through a bimsimulation between the games $\mathcal{G}^{\mathsf{TSO}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}}) = \langle \mathsf{C}^{\mathsf{TSO}}, \mathsf{C}_A^{\mathsf{TSO}}, \mathsf{C}_B^{\mathsf{TSO}}, \rightarrow, \mathsf{C}_F^{\mathsf{TSO}} \rangle$ and $\mathcal{G}^{\mathsf{SC}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}}) = \langle \mathsf{C}^{\mathsf{SC}}, \mathsf{C}_A^{\mathsf{SC}}, \mathsf{C}_B^{\mathsf{SC}}, \rightarrow, \mathsf{C}_F^{\mathsf{SC}} \rangle$.

For a TSO configuration c, let $\bar{\mathsf{c}}$ be the configuration obtained from c by updating all buffer messages to the memory, i.e. $\mathsf{c} \xrightarrow{\mathsf{up}^*} \bar{\mathsf{c}}$ and $\mathcal{B}(\bar{\mathsf{c}}) = \langle \varepsilon \rangle$. Note that $\bar{\mathsf{c}}$ is unique since $\mathcal{P}$ has only one process and thus there is only one way to update all messages. We extend this notation to game configurations in the obvious way.

Lemma 4.6. *The relation*

$$\mathsf{Z} := \{(\mathsf{c}, \langle \mathcal{S}(\mathsf{c}), \mathcal{M}(\bar{\mathsf{c}}) \rangle) \mid \mathsf{c} \in \mathsf{C}^{\mathsf{TSO}}\} \subset (\mathsf{C}^{\mathsf{TSO}} \times \mathsf{C}^{\mathsf{SC}})$$

is a bisimulation between $\mathcal{G}^{\mathsf{TSO}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}})$ *and* $\mathcal{G}^{\mathsf{SC}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}})$.

Proof. We need to show that for all $(\mathsf{c}_1, \mathsf{c}_2) \in \mathsf{Z}$:

- For all $\mathsf{c}_1 \rightarrow \mathsf{c}_3$ in $\mathcal{G}^{\mathsf{TSO}}$, there is $\mathsf{c}_2 \rightarrow \mathsf{c}_4$ in $\mathcal{G}^{\mathsf{SC}}$ with $\mathsf{c}_3 \approx \mathsf{c}_4$.
- For all $\mathsf{c}_2 \rightarrow \mathsf{c}_4$ in $\mathcal{G}^{\mathsf{SC}}$, there is $\mathsf{c}_1 \rightarrow \mathsf{c}_3$ in $\mathcal{G}^{\mathsf{TSO}}$ with $\mathsf{c}_3 \approx \mathsf{c}_4$.
- $\mathsf{c}_1 \in \mathsf{C}_A^{\mathsf{TSO}}$ if and only if $\mathsf{c}_2 \in \mathsf{C}_A^{\mathsf{SC}}$.
- $\mathsf{c}_1 \in \mathsf{C}_F^{\mathsf{TSO}}$ if and only if $\mathsf{c}_2 \in \mathsf{C}_F^{\mathsf{SC}}$.

For the first property, consider a transition $\mathsf{c}_1 \rightarrow \mathsf{c}_3$ in $\mathcal{G}^{\mathsf{TSO}}$. It is due to some instruction $\mathcal{S}(\mathsf{c}_1) \xrightarrow{\mathsf{instr}} \mathcal{S}(\mathsf{c}_3)$ in Proc of $\mathcal{P}$. If $\mathsf{instr} = \mathtt{rd}(\mathsf{x}, \mathsf{d})$ for some variable x and value d, then d must be the value of the last x-message in the buffer of c_1, or the buffer does not contain such a message and d is the value of x in the memory. In both cases, $\mathcal{M}(\bar{\mathsf{c}}_1)(\mathsf{x}) = \mathsf{d}$. Thus, $\mathtt{rd}(\mathsf{x}, \mathsf{d})$ is enabled at c_2, since $\mathcal{S}(\mathsf{c}_1) = \mathcal{S}(\mathsf{c}_2)$ and $\mathcal{M}(\bar{\mathsf{c}}_1) = \mathcal{M}(\mathsf{c}_2)$. Otherwise, if instr is an instruction other than $\mathtt{rd}$, it is always enabled at $\mathcal{S}(\mathsf{c}_2)$. Note that $\mathsf{instr} \neq \mathtt{mf}$ and $\mathsf{instr} \neq \mathtt{arw}(\mathsf{x}, \mathsf{d}, \mathsf{d}')$ since $\mathcal{P}$ does not use memory fences.

Let c_4 be the unique SC configuration obtained from c_2 after executing the program instruction $\mathcal{S}(\mathsf{c}_1) \xrightarrow{\mathsf{instr}} \mathcal{S}(\mathsf{c}_3)$. If $\mathsf{instr} = \mathtt{wr}(\mathsf{x}, \mathsf{d})$ for some variable x and value d, then $\mathcal{M}(\bar{\mathsf{c}}_3) = \mathcal{M}(\bar{\mathsf{c}}_1)[\mathsf{x} \leftarrow \mathsf{d}]$ and $\mathcal{M}(\mathsf{c}_4) = \mathcal{M}(\mathsf{c}_2)[\mathsf{x} \leftarrow \mathsf{d}]$. This holds even if $\mathsf{c}_1 \rightarrow \mathsf{c}_3$ includes buffer message updates of any kind. Otherwise, if instr is an instruction other than $\mathtt{wr}$, it simply holds that $\mathcal{M}(\bar{\mathsf{c}}_3) = \mathcal{M}(\bar{\mathsf{c}}_1)$ and $\mathcal{M}(\mathsf{c}_4) = \mathcal{M}(\mathsf{c}_2)$. Since $\mathcal{M}(\bar{\mathsf{c}}_1) = \mathcal{M}(\mathsf{c}_2)$, we have $\mathcal{M}(\bar{\mathsf{c}}_3) = \mathcal{M}(\mathsf{c}_4)$. Using $\mathcal{S}(\mathsf{c}_3) = \mathcal{S}(\mathsf{c}_4)$ we conclude that $\mathsf{c}_3 \approx \mathsf{c}_4$.

The second property is proven analogously. The third and fourth properties are trivially fulfilled by the definition of Z. □

$$\begin{tikzcd}
\mathsf{c} \arrow[r, "\sigma_X"] \arrow[dr, "\bar\sigma_X"'] & \mathsf{c}' \arrow[r, "\bar\sigma_Y"] \arrow[d, "\mathsf{up}^*"] & \mathsf{c}'' \\
& \bar{\mathsf{c}}' \arrow[ur, "\sigma_Y"'] &
\end{tikzcd}$$

Figure 12. Commutative diagram of strategies in games of group I.

Corollary 4.7. *The reachability problem for TSO games is* ExpTime*-hard.*

Proof. This follows directly from Lemma 4.3, Theorem 4.4, Lemma 4.6 and the fact that the word acceptance problem of linearly bounded ATMs is ExpTime-hard. Note that Lemma 4.3 is applicable since $\mathsf{c} \approx \mathsf{c}'$ implies $\mathcal{S}(\mathsf{c}) \in \mathsf{Q}_F^{\mathcal{P}} \iff \mathcal{S}(\mathsf{c}') \in \mathsf{Q}_F^{\mathcal{P}}$. ☐

5. Group I

All TSO games in this group have the following in common: There is one player that can update messages *after* her turn, and the other player can update messages *before* her turn. Both players might be allowed to do more than that, but fortunately we do not need to differentiate between those cases. In the following, we call the player that updates after her turn *player X*, and the other one *player Y*. Although the definition of safety games seems to be of asymmetric nature (player B tries to *reach* a final configuration, while player A tries to *avoid* them), the proof does not rely on the exact identity of player X and Y.

In this section, given a configuration c, we write $\bar{\mathsf{c}}$ to denote a configuration obtained from c after updating all messages to the memory. More formally, $\mathsf{c} \xrightarrow{\mathsf{up}^*} \bar{\mathsf{c}}$ and all buffers of $\bar{\mathsf{c}}$ are empty. Note that if the buffers of multiple processes contain messages at configuration c, then $\bar{\mathsf{c}}$ is not unique: Although the global state $\mathcal{S}(\bar{\mathsf{c}})$ and the buffer content $\mathcal{B}(\bar{\mathsf{c}})$ are uniquely defined, the memory $\mathcal{M}(\bar{\mathsf{c}})$ may depend on the order in which the buffer messages have been updated to the memory.

Let $\mathcal{G} = \langle \mathsf{C}, \mathsf{C}_A, \mathsf{C}_B, \to, \mathsf{C}_F \rangle$ be a TSO game as described above, currently in some configuration $\mathsf{c}_0 \in \mathsf{C}$. We first consider the situation where player X has a winning strategy σ_X from c_0. Let σ_Y be an arbitrary strategy for player Y and define two more strategies $\bar\sigma_X : \mathsf{c} \mapsto \overline{\sigma_X(\mathsf{c})}$ and $\bar\sigma_Y : \mathsf{c} \mapsto \sigma_Y(\bar{\mathsf{c}})$. That is, they act like σ_X and σ_Y, respectively, with the addition that $\bar\sigma_X$ empties the buffer *after* each turn and $\bar\sigma_Y$ empties the buffer *before* each turn. From the definitions it follows directly that $\bar\sigma_Y(\sigma_X(\mathsf{c})) = \sigma_Y(\bar\sigma_X(\mathsf{c}))$ for all $\mathsf{c} \in \mathsf{C}_X$. An example can be seen in Figure 12.

We argue that $\bar\sigma_X$ is a winning strategy for player X. The intuition behind this is as follows: Using the notation of Figure 12, if a configuration c'' is reachable from $\bar{\mathsf{c}}'$, then it is also reachable from c', since player Y can empty all buffers at the start of her turn and then proceed as if she started in $\bar{\mathsf{c}}'$. On the other hand, there might be configurations reachable from c' but not $\bar{\mathsf{c}}'$, for example a read transition might get disabled by one of the buffer updates. Thus, player X never gets a disadvantage by emptying the buffers.

Claim 5.1. $\bar\sigma_X$ is a winning strategy from c_0.

Proof. <u>**Case** $\mathsf{c}_0 \in \mathsf{C}_X$:</u> Since $\bar{\sigma}_Y(\sigma_X(\mathsf{c})) = \sigma_Y(\bar{\sigma}_X(\mathsf{c}))$ for all $\mathsf{c} \in \mathsf{C}_X$, the plays $\mathsf{P}_1 := \mathsf{P}(\mathsf{c}_0, \sigma_X, \bar{\sigma}_Y)$ and $\mathsf{P}_2 := \mathsf{P}(\mathsf{c}_0, \bar{\sigma}_X, \sigma_Y)$ agree on every second configuration, i.e. the configurations in C_X. Moreover, the configurations in between (after an odd number of steps) at least share the same global state, i.e. $\mathcal{S}(\sigma_X(\mathsf{c})) = \mathcal{S}(\bar{\sigma}_X(\mathsf{c}))$. In particular, the sequence of visited global TSO states is the same in both plays. Since σ_X is a winning strategy from c_0, it means that P_1 is winning for player X. This means that P_2 is also winning, because for both players, a winning play is clearly determined by the sequence of visited global TSO states. Because we chose σ_Y arbitrarily, it follows that $\bar{\sigma}_X$ is a winning strategy.

<u>**Case** $\mathsf{c}_0 \in \mathsf{C}_Y$:</u> For the other case, we consider the configurations in $\mathrm{Post}(\mathsf{c}_0)$ instead. We observe that σ_X must be a winning strategy for all $\mathsf{c} \in \mathrm{Post}(\mathsf{c}_0)$. We apply the first case of this proof to each of these configurations and obtain that $\bar{\sigma}_X$ is a winning strategy for all of them. It follows that $\bar{\sigma}_X$ is a winning strategy for c_0. □

Suppose that player X plays her modified strategy as described above. We observe that after at most two steps, every play induced by her strategy and an arbitrary strategy of the opposing player only visits configurations with at most one message in the buffers: Player X will empty all buffers at the end of each of her turns and player Y can only add at most one message to the buffers in between. Hence, they can play on a finite set of configurations instead.

To show this, we construct a finite game $\mathcal{G}' = \langle \mathsf{C}', \mathsf{C}'_A, \mathsf{C}'_B, \rightarrow', \mathsf{C}'_F \rangle$ as follows. C'_Y contains all configurations of C_Y that have at most one buffer message, i.e.:

$$\mathsf{C}'_Y := \{\langle \mathcal{S}, \mathcal{B}, \mathcal{M} \rangle_Y \in \mathsf{C}_Y \mid \sum_{\iota \in \mathcal{I}} |\mathcal{B}(\iota)| \leq 1\}$$

If $\mathsf{c}_0 \in \mathsf{C}_Y$, we also add it to C'_Y, otherwise to C'_X. Lastly, we add $\mathrm{Post}(\mathsf{C}'_Y)$ to C'_X, where Post is with respect to $\mathcal{G}$. $\rightarrow'$ is defined as the restriction of $\rightarrow$ to configurations of $\mathcal{G}'$, and $\mathsf{C}'_F = \mathsf{C}_F \cap \mathsf{C}'_A$. Note that C'_X also contains configurations with two messages. This is needed to account for the case that player Y has a winning strategy, which is handled later in this proof. Now, let $\bar{\sigma}'_X$ be the restriction of $\bar{\sigma}_X$ to C'_X (in the mathematical sense, i.e $\bar{\sigma}'_X : \mathsf{C}'_X \rightarrow \mathsf{C}_Y$ and $\bar{\sigma}_X(\mathsf{c}) = \bar{\sigma}'_X(\mathsf{c})$ for all $\mathsf{c} \in \mathsf{C}'_X$).

Claim 5.2. $\bar{\sigma}'_X$ is a winning strategy for c_0 in $\mathcal{G}'$.

Proof. Looking at the definitions, we confirm that $\bar{\sigma}'_X$ actually is a valid strategy for $\mathcal{G}'$, i.e. $\bar{\sigma}'_X(\mathsf{c}) \in \mathsf{C}'_Y$, for all $\mathsf{c} \in \mathsf{C}'_X$, since $\bar{\sigma}'_X(\mathsf{c})$ has empty buffers. (This makes $\bar{\sigma}'_X$ the restriction of $\bar{\sigma}_X$ to $\mathcal{G}'$.) Consider a strategy σ'_Y for player Y in $\mathcal{G}'$ and an arbitrary extension σ_Y to $\mathcal{G}$. Because $\bar{\sigma}'_X$ and $\bar{\sigma}_X$ agree on C'_X and $\bar{\sigma}'_Y$ and $\bar{\sigma}_Y$ agree on C'_Y, $\mathsf{P} := \mathsf{P}(\mathsf{c}_0, \bar{\sigma}'_X, \bar{\sigma}_Y)$ and $\mathsf{P}' := \mathsf{P}(\mathsf{c}_0, \bar{\sigma}'_X, \bar{\sigma}_Y)$ are in fact the exact same play. Since $\bar{\sigma}_X$ is a winning strategy, P is a winning play, and thus also P'. Here, note that $\mathcal{G}$ and $\mathcal{G}'$ agree on the final configurations within C'. Since σ'_Y was arbitrary, it follows that $\bar{\sigma}'_X$ is a winning strategy from c_0 in $\mathcal{G}'$. □

What is left to show is that a winning strategy for $\mathcal{G}'$ induces a winning strategy for $\mathcal{G}$. Suppose σ'_X is a winning strategy for player X in game $\mathcal{G}'$ for the configuration c_0. Let σ_X be an arbitrary extension of σ'_X to $\mathcal{G}$.

Claim 5.3. σ_X is a winning strategy for c_0 in $\mathcal{G}$.

Proof. Let σ_Y be a strategy of player Y in $\mathcal{G}$ and σ'_Y the restriction of σ_Y to C'_Y (again, in the mathematical sense). Since the outgoing transitions of every $\mathsf{c} \in \mathsf{C}'_Y$ are the same in both $\mathcal{G}$ and $\mathcal{G}'$, σ'_Y is a strategy for $\mathcal{G}'$ (and the restriction of σ_Y to $\mathcal{G}'$). Furthermore, starting from c_0, we see that σ_X and σ_Y induce the exact same play in $\mathcal{G}$ as σ'_X and σ'_Y in $\mathcal{G}'$. Since the former play is winning, so must be the latter one. □

Now, we quickly cover the situation where it is player Y that has a winning strategy. We follow the same arguments as previously, with minor changes. This time, assume σ_Y to be a winning strategy and let σ_X be arbitrary. Define $\bar{\sigma}_X$ and $\bar{\sigma}_Y$ as above. Following the beginning of the proof of Claim 5.1, we can conclude that the sequence of visited global TSO states is the same in both play P_1 and P_2. For the remainder of the proof, we swap the roles of X and Y and obtain that $\bar{\sigma}_Y$ is a winning strategy.

Let $\bar{\sigma}'_Y$ be the restriction of $\bar{\sigma}_Y$ to C'_Y. Since $\bar{\sigma}'_Y(\mathsf{C}'_Y) = \bar{\sigma}_Y(\mathsf{C}'_Y) \subseteq \mathrm{Post}(\mathsf{C}'_Y) \subseteq \mathsf{C}'_X$, it follows that $\bar{\sigma}'_Y$ is a strategy of $\mathcal{G}'$ (Post is again with respect to $\mathcal{G}$). Consider a strategy σ'_X for player X in $\mathcal{G}'$ and an arbitrary extension σ_X to $\mathcal{G}$. Similar as in Claim 5.2, we see that $\mathsf{P}(\mathsf{c}_0, \bar{\sigma}'_X, \bar{\sigma}_Y) = \mathsf{P}(\mathsf{c}_0, \bar{\sigma}'_X, \bar{\sigma}_Y)$ and conclude that $\bar{\sigma}'_Y$ is a winning strategy.

The other direction follows from the proof of Claim 5.3, with the roles of X and Y swapped.

Theorem 5.4. *The safety problem for games of group I is* ExpTime*-complete.*

Proof. By Claim 5.1 and Claim 5.2, if a configuration c_0 is winning for player X in $\mathcal{G}$, then it is also winning in $\mathcal{G}'$. The reverse holds by Claim 5.3. The equivalent statement for player Y follows from results outlined above. Thus, the safety problem for $\mathcal{G}$ is equivalent to the safety problem for $\mathcal{G}'$. $\mathcal{G}'$ is finite and has exponentially many configurations. ExpTime-completeness follows immediately from Lemma 4.2 (membership) and Corollary 4.7 (hardness). □

Remark 5.5. In the game where both players are allowed to update the buffer at any time, we can show an interesting conclusion. By Claim 5.1 and the equivalent statement for the second player, we can restrict both players to strategies that empty the buffer after each turn. Thus, the game is played only on configurations with empty buffer, except for the initial configuration which might contain some buffer messages. This implies that the TSO program that is described by the game implicitly follows SC semantics.

6. Group II

This group contains TSO games where both players are allowed to update the buffer *only* before their own move. Let player X be the player that has a winning strategy and player Y her opponent. Note that this differs from the previous section, in which the players X and Y were defined based on their updating capabilities.

Similar to the argumentation for Group I, we want to show that player X also has a winning strategy where she empties the buffer in each move. But, in contrast to before, this time there is an exception: Since the player has to update the buffer *before* her move, by updating a memory variable she might disable a read transition that she intended to execute. Thus, we do not require her to empty the buffer in that case.

Formally, let $\mathcal{G} = \langle \mathsf{C}, \mathsf{C}_A, \mathsf{C}_B, \to, \mathsf{C}_F \rangle$ be a TSO game where both players are allowed to perform buffer updates exactly before their own moves. Suppose σ_X is a winning strategy for player X and some configuration c_0. We construct another strategy $\bar{\sigma}_X$ for player X. Let $\mathsf{c} \in \mathsf{C}_X$, $\mathsf{c}' = \sigma_X(\mathsf{c})$ and $\bar{\mathsf{c}}$ as in the previous section, i.e. a (non-unique) configuration such that $\mathsf{c} \xrightarrow{\mathsf{up}^*} \bar{\mathsf{c}}$ and the buffers of $\bar{\mathsf{c}}$ are empty. Suppose that $\mathsf{c} \xrightarrow{\mathsf{instr}_\iota} \mathsf{c}'$, where instr_ι is not a

$$\begin{tikzcd}
\mathsf{c} \arrow[r, "\sigma_X", "\mathsf{instr}_\iota"'] \arrow[d, "\mathsf{up}^*"'] \arrow[dr, "\bar{\sigma}_X"] & \mathsf{c}' \arrow[r, "\bar{\sigma}_Y"] \arrow[d, "\mathsf{up}^*"'] & \mathsf{c}'' \\
\bar{\mathsf{c}} \arrow[r, "\mathsf{instr}_\iota"'] & \tilde{\mathsf{c}}' \arrow[ur, "\sigma_Y"'] &
\end{tikzcd}$$

FIGURE 13. Commutative diagram of strategies in games of group II, in the case where $\mathsf{instr}_\iota \neq \mathtt{rd}(\mathsf{x}, \mathsf{d})$ and $\mathsf{instr}_\iota \neq \mathtt{arw}(\mathsf{x}, \mathsf{d}, \mathsf{d}')$.

read or atomic read-write instruction. Then, starting from c, updating all buffer messages does not change that the transition from $\mathcal{S}(\mathsf{c})(\iota)$ to $\mathcal{S}(\mathsf{c}')(\iota)$ is enabled. Thus, instr_ι can also be executed from $\bar{\mathsf{c}}$. We call the resulting configuration $\tilde{\mathsf{c}}'$ and observe that $\bar{\mathsf{c}} \rightarrow \tilde{\mathsf{c}}'$ and $\mathsf{c}' \xrightarrow{\mathtt{up}^*} \tilde{\mathsf{c}}'$. We define $\bar{\sigma}_X(\mathsf{c}) := \tilde{\mathsf{c}}'$. This can be seen in Figure 13. Note that $\tilde{\mathsf{c}}'$ may have at most one message in its buffers. In the other case, where there is no transition from c to c' other than read or atomic read-write instructions, we define $\bar{\sigma}_X(\mathsf{c}) := \sigma_X(\mathsf{c}) = \mathsf{c}'$ and observe that c' cannot have more buffer messages than c.

Claim 6.1. $\bar{\sigma}_X$ is a winning strategy for c_0.

Proof. First, suppose that $\mathsf{c}_0 \in \mathsf{C}_X$ and let σ_Y be an arbitrary strategy of player Y. We define another (non-positional) strategy $\bar{\sigma}_Y$, that depends on the last two configurations, by $\bar{\sigma}_Y(\mathsf{c}, \mathsf{c}') := \sigma_Y(\bar{\sigma}_X(\mathsf{c}))$. We observe that for all $\mathsf{c} \in \mathsf{C}_X$, it holds that $\bar{\sigma}_Y(\mathsf{c}, \sigma_X(\mathsf{c})) = \sigma_Y(\bar{\sigma}_X(\mathsf{c}))$. It follows that the play P_1 induced by σ_X and $\bar{\sigma}_Y$ and the play P_2 induced by $\bar{\sigma}_X$ and σ_Y agree on every second configuration, i.e. the configurations in C_X. In particular, the sequence of visited global TSO configurations is the same in both plays. Since σ_X is winning, it means that P_1 is winning for player X and thus also P_2 is winning. Because we chose σ_Y arbitrarily, it follows that $\bar{\sigma}_X$ is a winning strategy.

Otherwise, if $\mathsf{c}_0 \in \mathsf{C}_Y$, we consider the successors of c_0 instead. We note that $\bar{\sigma}_X$ must also be a winning strategy for each $\mathsf{c} \in \mathrm{Post}(\mathsf{c}_0)$. But then, we can apply the previous arguments to each of those configurations and conclude that $\bar{\sigma}_X$ is a winning strategy for all of them. Thus, it is also a winning strategy for c_0. □

We conclude that if player X has a winning strategy σ_X, then she also has a winning strategy $\bar{\sigma}_X$ where she empties the buffers before every turn in which she does not perform a read operation. By symmetry, the same holds true for player Y. Thus, we can limit our analysis to this type of strategies. We see that the number of messages in the buffers is bounded: Suppose that the game is in configuration $\mathsf{c} \in \mathsf{C}_X$. Then, $\bar{\sigma}_X$ either empties the buffer and adds at most one new message, or it performs a transition due to a read instruction, which does not increase the size of the buffers. The analogous argumentation holds for player Y. Hence, we can reduce the game to a game on bounded buffers, which is finite state and thus decidable.

Given the configuration c_0 as above, we construct a finite game $\mathcal{G}' = \langle \mathsf{C}', \mathsf{C}'_X, \mathsf{C}'_Y, \rightarrow', \mathsf{C}'_F \rangle$ as follows. The set C'_X contains all configurations from C_X which have at most as many buffer messages than c_0 (or at most one message, if c_0 has empty buffers):

$$\mathsf{C}'_X := \{\mathsf{c} \in \mathsf{C}_X \mid |\mathcal{B}(\mathsf{c})| \leq \max\{1, |\mathcal{B}(\mathsf{c}_0)|\}\} \qquad \text{where} \qquad |\mathcal{B}| = \sum_{\iota \in \mathcal{I}} |\mathcal{B}(\iota)|$$

The set C'_Y is defined accordingly. Note that both sets are finite. Lastly, $\rightarrow'$ is defined as the restriction of $\rightarrow$ to configurations of $\mathcal{G}'$, and $\mathsf{C}'_F := \mathsf{C}_F \cap \mathsf{C}'_A$. We define $\bar{\sigma}'_X$ to be the restriction of $\bar{\sigma}_X$ to C'_X. Since $\bar{\sigma}'_X(\mathsf{c}) \in \mathsf{C}'_Y$ for all $\mathsf{c} \in \mathsf{C}'_X$, $\bar{\sigma}'_X$ is indeed a valid strategy for $\mathcal{G}'$. In particular, it is the restriction of $\bar{\sigma}_X$ to $\mathcal{G}'$.

Claim 6.2. $\bar{\sigma}'_X$ is a winning strategy for c_0 in $\mathcal{G}'$.

Proof. First, consider the case where $\mathsf{c}_0 \in \mathsf{C}_X$. Let σ'_Y be a strategy for player Y in $\mathcal{G}'$ and let σ_Y be an arbitrary extension of σ'_Y to $\mathcal{G}$. The play P induced by $\bar{\sigma}_X$ and σ_Y in $\mathcal{G}$ is the same as the play P' induced by $\bar{\sigma}'_X$ and σ'_Y in $\mathcal{G}'$. Since $\bar{\sigma}_X$ is a winning strategy, P is a winning play. It follows that P' must also be a winning strategy. Since σ'_Y was arbitrary, it follows that $\bar{\sigma}'_X$ is a winning strategy and c_0 is winning in $\mathcal{G}'$. □

Theorem 6.3. *The safety problem for games of group II is* ExpTime*-complete.*

Proof. By Claim 6.1 and Claim 6.2, if a configuration c_0 is winning for player A in game $\mathcal{G}$, then it is also winning in $\mathcal{G}'$. The same holds true for player B. Thus, the safety problem for $\mathcal{G}$ is equivalent to the safety problem for $\mathcal{G}'$. Similar to the games of group I, $\mathcal{G}'$ is finite and has exponentially many configurations. By Lemma 4.2 and Corollary 4.7, we can again conclude that the safety problem is ExpTime-complete. □

7. Group III

This group consists of all games where exactly one player has control over the buffer updates, and additionally the game where both players are allowed to update buffer messages *after* their own move. Intuitively, all of them have in common that the TSO program can attribute a buffer update to one specific player. If only one player can update messages, this is clear. In the other game, the first player who observes that a buffer message has reached the memory is not the one who has performed the buffer update. Thus, the program is able to punish misbehaviour, i.e. not following protocols or losing messages.

We will show that the safety problem is undecidable for this group of games. To accomplish that, we reduce the state reachability problem of PCS to the safety problem of each game. Since the former problem is undecidable, so is the latter.

The case where player A is allowed to perform buffer updates at any time is called the *A-TSO game*. It is explained in detail in the following. The other cases work similar, but require slightly different program constructions. They are presented in the appendix.

Consider the A-TSO game, i.e. the case where player A can update messages at any time, but player B can never do so. Given a PCS $\mathcal{L} = \langle \mathsf{S}, \mathsf{M}, \delta \rangle$ and a final state $\mathsf{s}_F \in \mathsf{S}$, we construct a TSO program $\mathcal{P}$ that simulates $\mathcal{L}$. We design the program such that s_F is reachable in $\mathcal{L}$ if and only if player B wins the safety game induced by $\mathcal{P}$. Thus, the construction gives her the initiative to decide which transitions of $\mathcal{L}$ will be simulated. Meanwhile, the task of player A is to take care of the buffer updates.

$\mathcal{P}$ consists of three processes Proc^1, Proc^2 and Proc^3, that operate on the variables $\{\mathsf{x}_{\mathtt{wr}}, \mathsf{x}_{\mathtt{rd}}, \mathsf{y}\}$ over the domain $\mathsf{M} \uplus \{0, 1, \perp\}$. The first process simulates the control flow and the message channel of the PCS $\mathcal{L}$. The second process provides a mean to read from the channel. The only task of the third process is to prevent deadlocks, or rather to make any deadlocked player lose. Proc^3 achieves this with four states: the initial state, an intermediate state, and one winning state for each player, respectively. If one of the players cannot move in both Proc^1 and Proc^2, they have to take a transition in Proc^3. From the initial state of this process, there exists only one outgoing transition, which is to the intermediate state. From there, the other player can move to her respective winning state and the process will only self-loop from then on. For player A, her state is winning because she can refuse to update any messages, which will ensure that player B keeps being deadlocked in Proc^1 and Proc^2. For player B, her state simply is contained in $\mathsf{Q}_F^{\mathcal{P}}$. In the following, we will mostly omit Proc^3 from the analysis and just assume that both players avoid reaching a configuration where they cannot take any transition in either Proc^1 or Proc^2.

As mentioned above, we will construct Proc^1 and Proc^2 to simulate the perfect channel system in a way that gives player B the control about which channel operation will be simulated. To achieve this, each channel operation will need an even number of transitions to be simulated in $\mathcal{P}$. Since player B starts the game, this means that after every fully completed simulation step, it is again her turn and she can initiate another simulation step as she pleases. Furthermore, during the simulation of a skip or send operation, we want to prevent player A from executing Proc^2, since this process is only needed for the receive operation. Suppose that we want to block player A from taking a transition $\mathsf{q} \xrightarrow{\mathsf{instr}} \mathsf{q}'$. We add a new transition $\mathsf{q}' \xrightarrow{\mathtt{skip}} \mathsf{q}_F$, where $\mathsf{q}_F \in \mathcal{S}_F^{\mathcal{P}}$. Hence, reaching q' is immediately losing for player A, since player B can respond by moving to q_F.

Next, we will describe how Proc^1 and Proc^2 simulate the perfect channel system $\mathcal{L}$. For each transition in $\mathcal{L}$, we construct a sequence of transitions in Proc^1 that simulates both the state change and the channel behaviour of the $\mathcal{L}$-transition. To achieve this, Proc^1 uses its buffer to store the messages of the PCS's channel. In particular, to simulate a send operation $!\mathsf{m}$, Proc^1 adds the message $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m}\rangle$ to its buffer. For receive operations, Proc^1 cannot read its own oldest buffer message, since it is overshadowed by the more recent messages. Thus, the program uses Proc^2 to read the message from memory and copies it to the variable $\mathsf{x}_{\mathtt{rd}}$, where it can be read by Proc^1. We call the combination of reading a message m from $\mathsf{x}_{\mathtt{wr}}$ and writing it to $\mathsf{x}_{\mathtt{rd}}$ the *rotation* of m.

While this is sufficient to simulate all behaviours of the PCS, it also allows for additional behaviour that is not captured by $\mathcal{L}$. More precisely, we need to ensure that each channel message is received *once and only once*. Equivalently, we need to prevent the *loss* and *duplication* of messages. This can happen due to multiple reasons.

The first phenomenon that allows the loss of messages is the seeming lossiness of the TSO buffer. Although it is not strictly lossy, it can appear so: Consider an execution of $\mathcal{P}$ that simulates two send operations $!\mathsf{m}_1$ and $!\mathsf{m}_2$, i.e. Proc^1 adds $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m}_1\rangle$ and $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m}_2\rangle$ to its buffer. Assume that player A decides to update both messages to the memory, without Proc^2 performing a message rotation in between. The first message m_1 is overwritten by the second message m_2 and is lost beyond recovery.

To prevent this, we extend the construction of Proc^1 such that it inserts an auxiliary message $\langle \mathsf{y}, 1\rangle$ into its buffer after the simulation of each send operation. After a message rotation, that is, after Proc^2 copied a message from $\mathsf{x}_{\mathtt{wr}}$ to $\mathsf{x}_{\mathtt{rd}}$, the process then resets the value of $\mathsf{x}_{\mathtt{wr}}$ to its initial value $\perp$. Next, the process checks that y contains the value 0, which indicates that only one message was updated to the memory. Now, player A is allowed to update exactly one $\langle \mathsf{y}, 1\rangle$ buffer message, after which Proc^2 resets y to 0. To ensure that player A has actually updated only one message in this step, Proc^2 then checks that $\mathsf{x}_{\mathtt{wr}}$ is still empty. Since player A is exclusively responsible for buffer updates, Proc^2 deadlocks her whenever one of these checks fails.

In the next scenario, we discover a different way of message loss. Consider again an execution of $\mathcal{P}$ that simulates two send operations $!\mathsf{m}_1$ and $!\mathsf{m}_2$. Assume Player A updates m_1 to the memory and Proc^2 performs a message rotation. Immediately afterwards, the same happens to m_2, without Proc^1 simulating a receive operation in between. Again, m_1 is overwritten by m_2 before being received, thus it is lost.

Player A is prevented from losing a message in this way by disallowing her to perform a complete message rotation (including the update of one $\langle \mathsf{y}, 1\rangle$-message and the reset of the variables) entirely on her own. More precisely, we add a winning transition for player B to Proc^2 that she can take if and only if player A is the one initiating the update of $\langle \mathsf{y}, 1\rangle$. On the other hand, player A can prevent player B from performing two rotations right after each other by refusing to update the next buffer message until Proc^1 initiates the simulation of a receive operation.

Lastly, we investigate message duplication. This occurs if Proc^1 simulates two receive operations without Proc^2 performing a message rotation in between. In this case, the most recently rotated message is received twice.

The program prevents this by blocking Proc^1 from progressing after a receive operation until Proc^2 has finished a full rotation. In detail, at the very end of the message rotation and $\langle \mathsf{y}, 1\rangle$-update, Proc^2 reset the value of $\mathsf{x}_{\mathtt{rd}}$ to its initial value $\perp$. After simulating a receive operation, Proc^1 is blocked until it can read this value from memory.

This concludes the mechanisms implemented to ensure that each channel message is received *once and only once*. Thus, we have constructed an A-TSO game that simulates a perfect channel system. We summarise our results in the following theorem. The formal proof can be found in Appendix A.

Theorem 7.1. *The safety problem for the A-TSO game is undecidable.*

8. Group IV

In TSO games where no player is allowed to perform any buffer updates, there is no communication between the processes at all. A read operation of a process Proc^ι on a variable x either reads the initial value from the shared memory, or the value of the last write of Proc^ι on x from the buffer, if such a write operation has happened.

Thus, we are only interested in the transitions that are enabled for each process, but we do not need to care about the actual buffer content. In particular, the information that we need to capture from the buffers and the memory is the values that each process can read from the variables, and whether a process can execute memory fence and atomic read-write instructions or not. Together with the global state of the current configuration, this completely determines the enabled transitions in the system.

We call this concept the *view* of the processes on the concurrent system and define it formally as a tuple $\mathsf{v} = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}\rangle$, where:

- $\mathcal{S} : \mathcal{I} \rightarrow \bigcup_{\iota \in \mathcal{I}} \mathsf{Q}^\iota$ is a global state of $\mathcal{P}$.
- $\mathcal{V} : \mathcal{I} \times \mathsf{Vars} \rightarrow \mathsf{Dom}$ defines which value each process reads from a variable.
- $\mathcal{F} : \mathcal{I} \rightarrow \{\mathsf{true}, \mathsf{false}\}$ represents the possibility to perform a memory fence instruction.

Given a view $\mathsf{v} = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}\rangle$, we write $\mathcal{S}(\mathsf{v})$, $\mathcal{V}(\mathsf{v})$ and $\mathcal{F}(\mathsf{v})$ for the global program state $\mathcal{S}$, the value state $\mathcal{V}$ and the fence state $\mathcal{F}$ of v.

The view of a configuration c is denoted by $\mathsf{v}(\mathsf{c})$ and defined in the following way. First, $\mathcal{S}(\mathsf{v}(\mathsf{c})) = \mathcal{S}(\mathsf{c})$. For all $\iota \in \mathcal{I}$ and $\mathsf{x} \in \mathsf{Vars}$, if $\mathcal{B}(\mathsf{c})(\iota)|_{\{\mathsf{x}\}\times\mathsf{Dom}} = \langle \mathsf{x}, \mathsf{d}\rangle \cdot \mathsf{w}$, then

$\mathcal{V}(\mathsf{v}(\mathsf{c}))(\iota, \mathsf{x}) = \mathsf{d}$. Otherwise, $\mathcal{V}(\mathsf{v}(\mathsf{c}))(\iota, \mathsf{x}) = \mathcal{M}(\mathsf{c})(\mathsf{x})$. Lastly, $\mathcal{F}(\mathsf{v}(\mathsf{c}))(\iota) = \mathsf{true}$ if and only if $\mathcal{B}(\mathsf{c})(\iota) = \varepsilon$. We extend the notation to sets of configurations in the usual way, i.e. $\mathsf{v}(\mathsf{C}') := \{\mathsf{v}(\mathsf{c}) \mid \mathsf{c} \in \mathsf{C}'\}$, and to game configurations by $\mathsf{v}(\mathsf{c}_A) := \mathsf{v}(\mathsf{c})_A$ and $\mathsf{v}(\mathsf{c}_B) := \mathsf{v}(\mathsf{c})_B$

If $\mathsf{v}(\mathsf{c}) = \mathsf{v}(\mathsf{c}')$ for some $\mathsf{c}, \mathsf{c}' \in \mathsf{C}_{\mathcal{P}}$, a local process of $\mathcal{P}$ cannot differentiate between c and c' in the sense that the enabled transitions in both configurations are the same. We formalise this idea by defining a bisimulation between $\mathcal{G}$ and a game $\mathcal{H} = \langle \mathsf{V}, \mathsf{V}_A, \mathsf{V}_B, \rightarrow, \mathsf{V}_F \rangle$ played on the views of $\mathcal{G}$. Define:

$$\mathsf{V}_A := \{\mathsf{v}(\mathsf{c})_A \mid \mathsf{c}_A \in \mathsf{C}_A\} \qquad \mathsf{V}_B := \{\mathsf{v}(\mathsf{c})_B \mid \mathsf{c}_B \in \mathsf{C}_B\} \qquad \mathsf{V}_F := \{\mathsf{v}(\mathsf{c})_A \mid \mathsf{c}_A \in \mathsf{C}_F\}$$

Lastly, we have $\mathsf{v}(\mathsf{c}) \rightarrow \mathsf{v}(\mathsf{c}')$ whenever $\mathsf{c} \rightarrow \mathsf{c}'$.

Lemma 8.1. *The relation $\mathsf{Z} := \{(\mathsf{c}, \mathsf{v}(\mathsf{c})) \mid \mathsf{c} \in \mathsf{C}\}$ is a bisimulation between $\mathcal{G}$ and $\mathcal{H}$.*

Proof. We need to show that for all $(\mathsf{c}_1, \mathsf{v}_2 := \mathsf{v}(\mathsf{c}_1)) \in \mathsf{Z}$:

- For all $\mathsf{c}_1 \rightarrow \mathsf{c}_3$ in $\mathcal{G}$, there is $\mathsf{v}_2 \rightarrow \mathsf{v}_4$ in $\mathcal{H}$ with $\mathsf{c}_3 \approx \mathsf{v}_4$.
- For all $\mathsf{v}_2 \rightarrow \mathsf{v}_4$ in $\mathcal{G}$, there is $\mathsf{c}_1 \rightarrow \mathsf{c}_3$ in $\mathcal{H}$ with $\mathsf{c}_3 \approx \mathsf{v}_4$.
- $\mathsf{c}_1 \in \mathsf{C}_A^{\mathcal{G}}$ if and only if $\mathsf{v}_2 \in \mathsf{C}_A^{\mathcal{H}}$.
- $\mathsf{c}_1 \in \mathsf{C}_F^{\mathcal{G}}$ if and only if $\mathsf{v}_2 \in \mathsf{C}_F^{\mathcal{H}}$.

The first property is trivially fulfilled since $\mathsf{v}(\mathsf{c}_1) \rightarrow \mathsf{v}(\mathsf{c}_3)$ for any $\mathsf{c}_1 \rightarrow \mathsf{c}_3$ by definition of $\mathcal{H}$.

For the second property, suppose that $\mathsf{v}_2 \rightarrow \mathsf{v}_4$ is due to some transition $\mathsf{c}_2 \xrightarrow{\mathsf{instr}_\iota} \mathsf{c}_4$. We first show that instr_ι is enabled at c_1. Since $\mathsf{v}(\mathsf{c}_1) = \mathsf{v}(\mathsf{c}_2)$, it holds that $\mathcal{S}(\mathsf{c}_1) = \mathcal{S}(\mathsf{c}_2)$. Furthermore, if $\mathsf{instr}_\iota = \mathtt{rd}(\mathsf{x}, \mathsf{d})_\iota$, then $\mathcal{V}(\mathsf{v}(\mathsf{c}_1))(\iota, \mathsf{x}) = \mathcal{V}(\mathsf{v}(\mathsf{c}_2))(\iota, \mathsf{x}) = \mathsf{d}$. Also, if $\mathsf{instr}_\iota = \mathtt{mf}_\iota$, then $\mathcal{B}(\mathsf{c}_2)(\iota) = \varepsilon$ and since $\mathcal{F}(\mathsf{v}(\mathsf{c}_1))(\iota) = \mathcal{F}(\mathsf{v}(\mathsf{c}_2))(\iota) = \mathsf{true}$ it follows that $\mathcal{B}(\mathsf{c}_1)(\iota) = \varepsilon$. Similarly, if $\mathsf{instr}_\iota = \mathtt{arw}(\mathsf{x}, \mathsf{d}, \mathsf{d}')_\iota$, then $\mathcal{V}(\mathsf{v}(\mathsf{c}_1))(\iota, \mathsf{x}) = \mathcal{V}(\mathsf{v}(\mathsf{c}_2))(\iota, \mathsf{x}) = \mathsf{d}$ and $\mathcal{B}(\mathsf{c}_1)(\iota) = \mathcal{B}(\mathsf{c}_2)(\iota) = \varepsilon$. From these considerations and the definition of the TSO semantics (see Figure 5), it follows that instr_ι is indeed enabled at c_1.

Let c_3 be the configuration obtained after performing instr_ι, i.e. $\mathsf{c}_1 \xrightarrow{\mathsf{instr}_\iota} \mathsf{c}_3$. It holds that $\mathcal{S}(\mathsf{c}_3) = \mathcal{S}(\mathsf{c}_4) = \mathcal{S}(\mathsf{c}_1)[\iota \leftarrow \mathcal{S}(\mathsf{c}_4)(\iota)]$. If $\mathsf{instr}_\iota = \mathtt{wr}(\mathsf{x}, \mathsf{d})_\iota$, then $\mathcal{V}(\mathsf{v}(\mathsf{c}_3)) = \mathcal{V}(\mathsf{v}(\mathsf{c}_4)) = \mathcal{V}(\mathsf{v}(\mathsf{c}_1))[(\iota, \mathsf{x}) \leftarrow \mathsf{d}]$ and $\mathcal{F}(\mathsf{v}(\mathsf{c}_3)) = \mathcal{F}(\mathsf{v}(\mathsf{c}_4)) = \mathcal{F}(\mathsf{v}(\mathsf{c}_1))[\iota \leftarrow \mathsf{false}]$. Similarly, if $\mathsf{instr}_\iota = \mathtt{arw}(\mathsf{x}, \mathsf{d}, \mathsf{d}')_\iota$, then $\mathcal{V}(\mathsf{v}(\mathsf{c}_3)) = \mathcal{V}(\mathsf{v}(\mathsf{c}_4)) = \mathcal{V}(\mathsf{v}(\mathsf{c}_1))[(\iota, \mathsf{x}) \leftarrow \mathsf{d}']$ but $\mathcal{F}(\mathsf{v}(\mathsf{c}_3)) = \mathcal{F}(\mathsf{v}(\mathsf{c}_4)) = \mathcal{F}(\mathsf{v}(\mathsf{c}_1))$. Otherwise, $\mathcal{V}(\mathsf{v}(\mathsf{c}_3)) = \mathcal{V}(\mathsf{v}(\mathsf{c}_4)) = \mathcal{V}(\mathsf{v}(\mathsf{c}_1))$ and $\mathcal{F}(\mathsf{v}(\mathsf{c}_3)) = \mathcal{F}(\mathsf{v}(\mathsf{c}_4)) = \mathcal{F}(\mathsf{v}(\mathsf{c}_1))$. In all cases it follows that $\mathsf{v}(\mathsf{c}_3) = \mathsf{v}(\mathsf{c}_4) = \mathsf{v}_4$.

The third and fourth properties are trivially fulfilled by the definition of Z and $\mathcal{H}$. □

Theorem 8.2. *The safety problem for games in group IV is* ExpTime*-complete.*

Proof. Apply Lemma 4.3 to Z and obtain that the safety problem for $\mathcal{G}$ is equivalent to the safety problem of $\mathcal{H}$. Since there exist only exponentially many views, ExpTime-completeness follows from Lemma 4.2 and Corollary 4.7. □

9. Conclusion and Future Work

In this work we have addressed for the first time the game problem for programs running under weak memory models in general and TSO in particular. Surprisingly, our results show that depending on when the updates take place, the problem can turn out to be undecidable or decidable. In fact, there is a subtle difference between the decidable (group I, II and IV) and undecidable (group III) TSO games. For the former games, when a player is taking a turn, the system does not know who was responsible for the last update. But for the latter games, the last update can be attributed to a specific player. Another surprising finding is the complexity of the game problem for the groups I, II and IV which is ExpTime-complete in contrast with the non-primitive recursive complexity of the reachability problem for programs running under TSO and the undecidability of the repeated reachability problem.

In future work, the games where exactly one player has control over the buffer seem to be the most natural ones to expand on. In particular, the A-TSO game (where player A can update before and after her move) and the B-TSO game (same, but for player B). On the other hand, the games of groups I, II and IV seem to be degenerate cases and therefore rather uninteresting. In particular, we have shown that they are not more powerful than games on programs that follow SC semantics.

Another direction for future work is considering other memory models, such as the partial store ordering semantics, the release-acquire semantics, and the ARM semantics. It is also interesting to define stochastic games for programs running under TSO as extension of the probabilistic TSO semantics [AAA+22].

References

[AAA+22] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Raj Aryan Agarwal, Adwait Godbole, and Shankara Narayanan Krishna. Probabilistic total store ordering. In Ilya Sergey, editor, *Programming Languages and Systems - 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*, volume 13240 of *Lecture Notes in Computer Science*, pages 317–345. Springer, 2022. doi:10.1007/978-3-030-99336-8_12.

[AABN16] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmed Bouajjani, and Tuan Phong Ngo. The benefits of duality in verifying concurrent programs under TSO. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 5:1–5:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.5.

[AABN18] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmed Bouajjani, and Tuan Phong Ngo. A load-buffer semantics for total store ordering. *Log. Methods Comput. Sci.*, 14(1), 2018. doi:10.23638/LMCS-14(1:9)2018.

[AAC+12] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Carl Leonardsson, and Ahmed Rezine. Counter-example guided fence insertion under TSO. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7214 of *Lecture Notes in Computer Science*, pages 204–219. Springer, 2012. doi:10.1007/978-3-642-28756-5_15.

[AAF+23] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Florian Furbach, Adwait Amit Godbole, Yacoub G. Hendi, Shankara Narayanan Krishna, and Stephan Spengler. Parameterized verification under TSO with data types. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I*, volume 13993 of *Lecture Notes in Computer Science*, pages 588–606. Springer, 2023. doi:10.1007/978-3-031-30823-9_30.

[AAP15] Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Ngo Tuan Phong. The best of both worlds: Trading efficiency and optimality in fence insertion for TSO. In Jan Vitek, editor, *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9032 of *Lecture Notes in Computer Science*, pages 308–332. Springer, 2015. doi:10.1007/978-3-662-46669-8_13.

[AAR20] Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Rojin Rezvan. Parameterized verification under TSO is pspace-complete. *Proc. ACM Program. Lang.*, 4(POPL):26:1–26:29, 2020. doi:10.1145/3371094.

[ABBM10] Mohamed Faouzi Atig, Ahmed Bouajjani, Sebastian Burckhardt, and Madanlal Musuvathi. On the verification problem for weak memory models. In Manuel V. Hermenegildo and Jens Palsberg, editors, *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*, pages 7–18. ACM, 2010. doi:10.1145/1706299.1706303.

[ABd08] Parosh Aziz Abdulla, Ahmed Bouajjani, and Julien d'Orso. Monotonic and downward closed games. *J. Log. Comput.*, 18(1):153–169, 2008. doi:10.1093/logcom/exm062.

[AJ94] Parosh Aziz Abdulla and Bengt Jonsson. Undecidable verification problems for programs with unreliable channels. In Serge Abiteboul and Eli Shamir, editors, *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*, volume 820 of *Lecture Notes in Computer Science*, pages 316–327. Springer, 1994. doi:10.1007/3-540-58201-0_78.

[AJ96] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Inf. Comput.*, 127(2):91–101, 1996. doi:10.1006/inco.1996.0053.

[ARM14] ARM. *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition*, 2014. URL: https://developer.arm.com/documentation/ddi0406/latest/.

[Ati20] Mohamed Faouzi Atig. What is decidable under the TSO memory model? *ACM SIGLOG News*, 7(4):4–19, 2020. doi:10.1145/3458593.3458595.

[AVW03] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003. Logic and Complexity in Computer Science. URL: https://www.sciencedirect.com/science/article/pii/S0304397502004425, doi:10.1016/S0304-3975(02)00442-5.

[BDM13] Ahmed Bouajjani, Egor Derevenetc, and Roland Meyer. Checking and enforcing robustness against TSO. In Matthias Felleisen and Philippa Gardner, editors, *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7792 of *Lecture Notes in Computer Science*, pages 533–553. Springer, 2013. doi:10.1007/978-3-642-37036-6_29.

[BMM11] Ahmed Bouajjani, Roland Meyer, and Eike Möhlmann. Deciding robustness against total store ordering. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 428–440. Springer, 2011. doi:10.1007/978-3-642-22012-8_34.

[BS04] R.-J. Back and C.C. Seceleanu. Contracts and games in controller synthesis for discrete systems. In *Proceedings. 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2004.*, pages 307–314, 2004. doi:10.1109/ECBS.2004.1316713.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983. doi:10.1145/322374.322380.

[CKS81] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

[CS76] Ashok K. Chandra and Larry J. Stockmeyer. Alternation. In *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*, pages 98–108. IEEE Computer Society, 1976. doi:10.1109/SFCS.1976.4.

[Fin87] Alain Finkel. A generalization of the procedure of karp and miller to well structured transition systems. In Thomas Ottmann, editor, *Automata, Languages and Programming, 14th International Colloquium, ICALP87, Karlsruhe, Germany, July 13-17, 1987, Proceedings*, volume 267 of *Lecture Notes in Computer Science*, pages 499–508. Springer, 1987. doi:10.1007/3-540-18088-5_43.

[FS01] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

[IBM21] IBM. *Power ISA, Version 3.1b*, 2021. URL: https://files.openpower.foundation/s/dAYSdGzTfW4j2r2/download/OPF_PowerISA_v3.1B.pdf.

[Int12] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developers Manual*, 2012. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.

[KV97] Oded Kupferman and Mosab Y. Vardi. Synthesis with incomplete information. In *Proceedings of the 2nd International Conference on Temporal Logic*, volume 171 of *Lecture Notes in Computer Science*, pages 91–106. Springer, 1997. doi:10.1007/978-94-015-9586-5_6.

[KV00] Orna Kupferman and Moshe Y. Vardi. μ-calculus synthesis. In Mogens Nielsen and Branislav Rovan, editors, *Mathematical Foundations of Computer Science 2000, 25th International Symposium, MFCS 2000, Bratislava, Slovakia, August 28 - September 1, 2000, Proceedings*, volume 1893 of *Lecture Notes in Computer Science*, pages 497–507. Springer, 2000. doi:10.1007/3-540-44612-5_45.

[Lam79] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Computers*, 28(9):690–691, 1979. doi:10.1109/TC.1979.1675439.

[Maz01] René Mazala. Infinite games. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*, pages 23–42. Springer, 2001. doi:10.1007/3-540-36387-4_2.

[OSS09] Scott Owens, Susmit Sarkar, and Peter Sewell. A better x86 memory model: x86-tso. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *Lecture Notes in Computer Science*, pages 391–407. Springer, 2009. doi:10.1007/978-3-642-03359-9_27.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989. doi:10.1145/75277.75293.

[Sch02] Philippe Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Inf. Process. Lett.*, 83(5):251–261, 2002. doi:10.1016/S0020-0190(01)00337-4.

[spa94] SPARC International, Inc. *SPARC Architecture Manual Version 9*, 1994. URL: https://sparc.org/wp-content/uploads/2014/01/SPARCV9.pdf.gz.

[SS23] Stephan Spengler and Sanchari Sil. TSO games - on the decidability of safety games under the total store order semantics. In Antonis Achilleos and Dario Della Monica, editors, *Proceedings of the Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2023, Udine, Italy, 18-20th September 2023*, volume 390 of *EPTCS*, pages 82–98, 2023. doi:10.4204/EPTCS.390.6.

[SSO+10] Peter Sewell, Susmit Sarkar, Scott Owens, Francesco Zappa Nardelli, and Magnus O. Myreen. x86-tso: a rigorous and usable programmer's model for x86 multiprocessors. *Commun. ACM*, 53(7):89–97, 2010. doi:10.1145/1785414.1785443.

Appendix A. Undecidability of the A-TSO game

In this section, we will formally prove the correctness of the reduction presented in section 7. We begin with the construction of the TSO program $\mathcal{P} = \langle \mathsf{Proc}^1, \mathsf{Proc}^2, \mathsf{Proc}^3 \rangle$. For Proc^1, we start by adding the states of S to Q^1. Then, for each transition of $\mathcal{L}$, we add some auxiliary states $\mathsf{h}_1, \mathsf{h}_2, \ldots$ and transitions between them to simulate the PCS behaviour. Figure 14 shows this construction for each of the transitions $\mathsf{s} \xrightarrow{\mathsf{skip}} \mathsf{s}'$, $\mathsf{s} \xrightarrow{!\mathsf{m}} \mathsf{s}'$ and $\mathsf{s} \xrightarrow{?\mathsf{m}} \mathsf{s}'$. Note that all the auxiliary states of Proc^1 (labelled as h_i) are supposed to be distinct for each transition.

Additionally, Figure 15 and Figure 16 show the process definitions of Proc^2 and Proc^3, respectively. The set of final states is $\mathsf{Q}_F^{\mathcal{P}} := \mathsf{S}_F \cup \{\mathsf{q}_F, \mathsf{r}_F\}$, where $\mathsf{S}_F \subset \mathsf{Q}^1$, $\mathsf{q}_F \in \mathsf{Q}^2$ and $\mathsf{r}_F \in \mathsf{Q}^3$.

$$\begin{tikzcd}
\mathsf{s} \arrow[d, "\mathtt{skip}"] \\
\mathsf{h}_1 \arrow[d, "\mathtt{skip}"] \\
\mathsf{s}'
\end{tikzcd}$$

(A) skip operation $\mathsf{s} \xrightarrow{\mathtt{skip}} \mathsf{s}'$

$$\begin{tikzcd}
\mathsf{s} \arrow[d, "\mathtt{wr}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"] \\
\mathsf{h}_1 \arrow[d, "\mathtt{wr}(\mathsf{y}, 1)"] \\
\mathsf{s}'
\end{tikzcd}$$

(B) send operation $\mathsf{s} \xrightarrow{!\mathsf{m}} \mathsf{s}'$

$$\begin{tikzcd}
\mathsf{s} \arrow[d, "\mathtt{skip}"] & \\
\mathsf{h}_1 \arrow[r, "\mathtt{mf}"] \arrow[d, "\mathtt{rd}(\mathsf{x}_{\mathtt{rd}}, \mathsf{m})"] & \mathsf{h}_4 \arrow[d, "\mathtt{skip}"] \\
\mathsf{h}_2 \arrow[d, "\mathtt{rd}(\mathsf{x}_{\mathtt{rd}}, \perp)"] & \mathsf{h}_5 \arrow[d, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \perp)"] \\
\mathsf{h}_3 \arrow[d, "\mathtt{skip}"] & \mathsf{h}_6 \\
\mathsf{s}' &
\end{tikzcd}$$

(C) receive operation $\mathsf{s} \xrightarrow{?\mathsf{m}} \mathsf{s}'$

FIGURE 14. Proc^1 of the A-TSO reduction from PCS

Theorem A.1. *Consider the A-TSO game, where player A can update the buffer before and after her turn, but player B can never update. The set of final states* S_F *of* $\mathcal{L}$ *is reachable from* $\mathsf{s}_0 \in \mathsf{S}$ *if and only if player B wins the game* $\mathcal{G}^{\mathsf{TSO}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}})$ *starting from the configuration* $\mathsf{c}_0 := \langle\langle \mathsf{s}_0, \mathsf{q}_1, \mathsf{r}_1\rangle, \langle \varepsilon, \varepsilon, \varepsilon\rangle, \{\mathsf{x}_{\mathtt{wr}} \mapsto \perp, \mathsf{x}_{\mathtt{rd}} \mapsto \perp, \mathsf{y} \mapsto 0\}\rangle_B \in \mathsf{C}_B$.

Note that the definition of Proc^3 secures deadlock-freedom of $\mathcal{P}$. Furthermore, if one of the players has no enabled transitions in either Proc^1 or Proc^2 in her turn, she will lose the game: If player A has no enabled transition, she needs to move from r_1 to r_2 in Proc^3. Player B can then respond with $\mathsf{r}_2 \to \mathsf{r}_3 \in \mathsf{Q}_F^{\mathcal{P}}$, which player A cannot escape from. Similarly, if player B needs to move to r_2, player A can respond with r_4. Since player A controls the buffer updates, she can decide to not perform any buffer updates, which maintains the situation that player B can only move in Proc^3. Thus, the game will loop in $\mathsf{r}_4 \to \mathsf{r}_4$ forever, which means that player A wins.

In the following, we say that a player is *deadlocked*, if it is her turn and there is no enabled outgoing transition in either Proc^1 or Proc^2. Naturally, both players avoid being deadlocked, since it means that they lose the game, respectively.

A.1. From $\mathcal{L}$-Reachability to a Winning Strategy in A-TSO.

Suppose S_F is reachable from s_0. Let $\mathsf{c}_0^{\mathcal{L}} \xrightarrow{\mathsf{op}_1} \mathsf{c}_1^{\mathcal{L}} \xrightarrow{\mathsf{op}_2} \dots \xrightarrow{\mathsf{op}_n} \mathsf{c}_n^{\mathcal{L}}$ be a run with $\mathsf{c}_k^{\mathcal{L}} = \langle \mathsf{s}_k, \mathsf{w}_k\rangle$ for all $k = 1, \dots, n$ and $s_n \in \mathsf{S}_F$. Here, w_k is a word over M consisting of the letters $\mathsf{w}_k[1], \dots, \mathsf{w}_k[|\mathsf{w}_k|]$. Without loss of generality we can assume that this run does not contain any duplicate configurations.

For all $k = 0, \dots, n$, we define a game configuration $\mathsf{c}_k = \langle \mathcal{S}_k, \mathcal{B}_k, \mathcal{M}_k\rangle_B \in \mathsf{C}_B$:

$$\begin{tikzcd}
 & q_1 \arrow[r, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"] \arrow[d, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"] & q_F \\
\text{for all } \mathsf{m} \in \mathsf{M} \left\{ \vphantom{\begin{matrix} a \\ b \\ c \end{matrix}} \right. & q_{\mathsf{m}} \arrow[d, "\mathtt{wr}(\mathsf{x}_{\mathtt{rd}}, \mathsf{m})"] & \\
 & q_3 \arrow[r, "\mathtt{rd}(\mathsf{y}, 1)"] \arrow[d, "\mathtt{wr}(\mathsf{x}_{\mathtt{wr}}, \bot)"] & q_F \\
 & q_4 \arrow[r, "\mathtt{skip}"] \arrow[d, "\mathtt{mf}"] & q_F \\
 & q_5 \arrow[r, "\mathtt{rd}(\mathsf{y}, 1)"] \arrow[d, "\mathtt{skip}"] & q_F \\
 & q_6 \arrow[d, "\mathtt{rd}(\mathsf{y}, 1)"] & \\
 & q_7 \arrow[d, "\mathtt{wr}(\mathsf{y}, 0)"] & \\
 & q_8 \arrow[d, "\mathtt{mf}"] & \\
 & q_9 \arrow[r, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"] \arrow[d, "\mathtt{wr}(\mathsf{x}_{\mathtt{rd}}, \bot)"] & q_F \\
 & q_{10} \arrow[d, "\mathtt{mf}"] & \\
 & q_1 &
\end{tikzcd}$$

FIGURE 15. Proc^2 of the A-TSO reduction from PCS

$$\begin{tikzcd}
 & r_1 \arrow[d, "\mathtt{skip}"] & \\
 & r_2 \arrow[dl, "\mathtt{skip}"'] \arrow[dr, "\mathtt{skip}"] & \\
r_3 \arrow[loop below, "\mathtt{skip}"] & & r_F \arrow[loop below, "\mathtt{skip}"]
\end{tikzcd}$$

FIGURE 16. Proc^3 of the A-TSO reduction from PCS

(1) $\mathcal{S}_k := \langle \mathsf{s}_k, \mathsf{q}_1, \mathsf{r}_1 \rangle$
(2) $\mathcal{B}_k := \langle \langle \langle \mathsf{y}, 1 \rangle, \langle \mathsf{x}_{\mathtt{wr}}, \mathsf{w}_k[1] \rangle, \ldots, \langle \mathsf{y}, 1 \rangle, \langle \mathsf{x}_{\mathtt{wr}}, \mathsf{w}_k[|\mathsf{w}_k|] \rangle \rangle, \varepsilon, \varepsilon \rangle$
(3) $\mathcal{M}_k := \{ \mathsf{x}_{\mathtt{wr}} \mapsto \bot, \mathsf{x}_{\mathtt{rd}} \mapsto \bot, \mathsf{y} \mapsto 0 \}$

The basic idea is to develop a strategy for player B that visits all configurations $\mathsf{c}_1, \ldots, \mathsf{c}_n$. Inspecting the construction of Proc^2, we observe that there is no way to prevent player A from performing the buffer update and rotation of the oldest channel message whenever she decides to do so. We account for that by introducing additional configurations $c'_k = \langle \mathcal{S}'_k, \mathcal{B}'_k, \mathcal{M}'_k \rangle_B$:

(1) $\mathcal{S}'_k := \langle \mathsf{s}_k, \mathsf{q}_3, \mathsf{r}_1 \rangle$

(2) $\mathcal{B}'_k := \langle\langle\langle \mathsf{y}, 1\rangle, \langle \mathsf{x_{wr}}, \mathsf{w}_k[1]\rangle, \ldots, \langle \mathsf{y}, 1\rangle, \langle \mathsf{x_{wr}}, \mathsf{w}_k[|\mathsf{w}_k| - 1]\rangle, \langle \mathsf{y}, 1\rangle\rangle, \varepsilon, \varepsilon\rangle$
(3) $\mathcal{M}_k^{(3)} := \{\mathsf{x_{wr}} \mapsto \mathsf{w}_k[|\mathsf{w}_k|], \mathsf{x_{rd}} \mapsto \mathsf{w}_k[|\mathsf{w}_k|], \mathsf{y} \mapsto 0\}$

The configurations c'_k refer to the case where player A has updated the oldest message to the memory and also performed the message rotation. Note that if $\mathsf{w}_k = \varepsilon$, then c'_k is not defined.

As a first step, consider a situation where it is player A's turn and Proc^2 is in state q_1. If player A decides to initiate the message rotation by following the transition $\mathsf{q}_1 \xrightarrow{\mathtt{rd}(\mathsf{x_{wr}},\mathsf{m})} \mathsf{q_m}$ for some $\mathsf{m} \in \mathsf{M}$, we always let player B immediately respond with $\mathsf{q_m} \xrightarrow{\mathtt{wr}(\mathsf{x_{rd}},\mathsf{m})} \mathsf{q}_3$. On the other hand, if Proc^2 is already in state q_3, player A cannot take any further transitions in this process, since after $\mathsf{q}_3 \xrightarrow{\mathtt{wr}(\mathsf{x_{wr}},\perp)} \mathsf{q}_4$, player B can take the transition $\mathsf{q}_4 \xrightarrow{\mathtt{skip}} \mathsf{q}_F$ and wins. We will use these observations in the following argumentation.

We show by induction that starting from c_0, Player B can force a play that either visits one of c_k, c'_k or C_F. For $k = 0$, this clearly holds true. So, suppose that the claim holds for some arbitrary but fixed $0 \leq k < n$. By the induction hypothesis, we can assume that the game is either in state c_k or c'_k.

If $\mathsf{op}_{k+1} = \mathtt{skip}$, then player B takes the transition $\mathsf{s}_k \xrightarrow{\mathtt{skip}} \mathsf{h}_1$ in Proc^1. After a possible message rotation in Proc^2 (i.e. one move by player A and B, respectively), player A is left with only one non-losing transition, which is $\mathsf{h}_1 \xrightarrow{\mathtt{skip}} \mathsf{s}_{k+1}$ in Proc^1. Assuming that player A can prevent an immediate defeat, we now need to show that the game is either in configuration c_{k+1} or c'_{k+1}.

First, consider the case where Proc^2 is in state q_1, which implies that the sub-run started at c_k rather than c'_k. If player A had updated at least one message from the buffer to the memory, $\mathsf{x_{wr}}$ would now contain some channel message m. Thus, player B could immediately win by taking the transition $\mathsf{q}_1 \xrightarrow{\mathtt{rd}(\mathsf{x_{wr}},\mathsf{m})} \mathsf{q}_F$. We conclude that the game must be in configuration c_{k+1}.

In the other case, Proc^2 is in state q_3. It follows that either the sub-run started in c'_k or player A has updated a message from the buffer and performed the message rotation. Suppose she had additionally updated at least one more message, in particular a message $\langle \mathsf{y}, 1\rangle$. Then, player B is again able to win immediately by following $\mathsf{q}_3 \xrightarrow{\mathtt{rd}(\mathsf{y},1)} \mathsf{q}_F$. Thus, the game must be in configuration c'_{k+1}.

Next, we examine the situation where $\mathsf{op}_{k+1} = !\mathsf{m}$, which is very similar to the previous one. Player B takes the transition $\mathsf{s}_k \xrightarrow{\mathtt{wr}(\mathsf{x_{wr}},\mathsf{m})} \mathsf{h}_1$ and player A eventually responds with $\mathsf{h}_1 \xrightarrow{\mathtt{wr}(\mathsf{y},1)} \mathsf{s}_{k+1}$. Using the same argumentation as above, either the game is now in configuration c_{k+1} or c'_{k+1}, or player B can force an immediate win.

Lastly, if $\mathsf{op}_{k+1} = ?\mathsf{m}$, player B starts by taking the transition $\mathsf{s}_k \xrightarrow{\mathtt{skip}} \mathsf{h}_1$ in Proc^1. For player A, $\mathsf{h}_1 \xrightarrow{\mathtt{mf}} \mathsf{h}_4$ is losing: Since $\mathsf{op}_{k+1} = ?\mathsf{m}$, we know that $\mathsf{w}_k \neq \varepsilon$ and after flushing the buffer, the last character of w_k is written to $\mathsf{x_{wr}}$. Thus, after player B responds with $\mathsf{h}_4 \xrightarrow{\mathtt{skip}} \mathsf{h}_5$, player A is effectively deadlocked, since the outgoing transition in Proc^1 is disabled and all actions in Proc^2 will eventually lead to player B reaching q_F.

We conclude that player A updates and rotates the oldest message if not already done so, and then proceeds with $\mathsf{h}_1 \xrightarrow{\mathtt{rd}(\mathsf{x_{rd}},\mathsf{m})} \mathsf{h}_2$ in Proc^1. The only possible continuation for both players is now in Proc^2:

$$\mathsf{q}_3 \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}},\bot)} \mathsf{q}_4 \xrightarrow{\mathtt{mf}} \mathsf{q}_5 \xrightarrow{\mathtt{skip}} \mathsf{q}_6 \xrightarrow{\mathtt{rd}(\mathsf{y},1)} \mathsf{q}_7 \xrightarrow{\mathtt{wr}(\mathsf{y},0)} \mathsf{q}_8 \xrightarrow{\mathtt{mf}} \mathsf{q}_9 \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{rd}},\bot)} \mathsf{q}_{10}$$

Because of $\mathsf{q}_5 \xrightarrow{\mathtt{rd}(\mathsf{y},1)} \mathsf{q}_F$, player A cannot update the pending $\langle \mathsf{y}, 1\rangle$-message before reaching state q_6, but then definitely has to do so to enable the next transition. In q_9, y is reset to 0 again. Furthermore, the transition $\mathsf{q}_9 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}},\mathsf{m})} \mathsf{q}_F$ ensures that no other message was updated. When q_{10} is reached, player A needs to flush the buffer of Proc^2 again, to enable both $\mathsf{h}_2 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{rd}},\bot)} \mathsf{h}_3$ in Proc^1 and $\mathsf{q}_9 \xrightarrow{\mathtt{mf}} \mathsf{q}_{10}$ in Proc^2. She then takes one of those two transitions and player B takes the other one. Eventually (after maybe the update and rotation of the next message), player A will take the transition $\mathsf{h}_3 \xrightarrow{\mathtt{skip}} \mathsf{s}_{k+1}$ in Proc^1.

At this point, Proc^1 moved from s_k to s_{k+1} and Proc^2 is back in q_1 or q_3. Exactly one $\langle \mathsf{y}, 1\rangle$-message was removed from the buffer and the memory values of y and $\mathsf{x}_{\mathtt{rd}}$ were reset to 0 and $\bot$, respectively. Using the same argumentation as previously, we conclude that the game is now either in configuration c_{k+1} or c'_{k+1}, or player B can force an immediate win.

This concludes the proof by induction. In summary, starting from c_0, player B can force a play that reaches C_B or one of the configurations c_n or c'_n. In both of these configurations, Proc^1 is in the local state s_n, which is a final state of $\mathcal{L}$. Thus, $\mathsf{c}_n, \mathsf{c}'_n \in \mathsf{C}_B$ and player B wins in any case.

A.2. From a Winning Strategy in A-TSO to $\mathcal{L}$-Reachability.

For the other direction, suppose that player B has a winning strategy. Consider a strategy of player A that avoids reaching both $\mathsf{q}_F \in \mathsf{Q}^2 \cap \mathsf{Q}_F^{\mathcal{P}}$ and $\mathsf{r}_F \in \mathsf{Q}^3 \cap \mathsf{Q}_F^{\mathcal{P}}$ whenever possible. Furthermore, assume that player A prefers to not update any messages and to move in Proc^2, as long as it does not contradict the previous statement.

Let $\mathsf{s}_0, \ldots, \mathsf{s}_n$ be the sequence of channel states that are visited by Proc^1 during the run induced by these two strategies. Since player B uses a winning strategy, this sequence is finite. We will show by induction that for each $k = 0, \ldots, n$, the run contains a game configuration $\mathsf{c}_k = \langle \mathcal{S}_k, \mathcal{B}_k, \mathcal{M}_k\rangle_B \in \mathsf{C}_B$ such that:

(1) $\mathcal{S}_k := \langle \mathsf{s}_k, \mathsf{q}_1, \mathsf{r}_1\rangle$
(2) $\mathcal{B}_k := \langle\langle\langle \mathsf{y}, 1\rangle, \langle \mathsf{x}_{\mathtt{wr}}, \mathsf{w}_k[1]\rangle, \ldots, \langle \mathsf{y}, 1\rangle, \langle \mathsf{x}_{\mathtt{wr}}, \mathsf{w}_k[|\mathsf{w}_k|]\rangle\rangle, \varepsilon, \varepsilon\rangle$, where $\mathsf{w}_k \in \mathsf{M}^*$
(3) $\mathcal{M}_k := \{\mathsf{x}_{\mathtt{wr}} \mapsto \bot, \mathsf{x}_{\mathtt{rd}} \mapsto \bot, \mathsf{y} \mapsto 0\}$
(4) If $k > 0$, there is a label op_k such that $\langle \mathsf{s}_{k-1}, \mathsf{w}_{k-1}\rangle \xrightarrow{\mathsf{op}_k} \langle \mathsf{s}_k, \mathsf{w}_k\rangle$.

For $k = 0$, this clearly holds true. So, suppose that the claim holds for some arbitrary but fixed $0 \leq k < n$.

Starting in c_k, player B cannot move in Proc^2, since $\mathcal{M}(\mathsf{x}_{\mathtt{wr}}) = \bot$, $\mathcal{B}(2) = \varepsilon$ and the only outgoing transitions from $\mathcal{S}(2) = \mathsf{q}_1$ are of the form $\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})$ for $\mathsf{m} \in \mathsf{M}$. Thus, player B must first move in Proc^1 instead. Looking at the construction of Proc^1, we see that the sub-run from c_k to (the very next occurence of) s_{k+1} takes a sequence of transitions that were added to $\mathcal{P}$ due to some unique transition $\mathsf{s}_k \xrightarrow{\mathsf{op}_{k+1}} \mathsf{s}_{k+1}$ of $\mathcal{L}$.

If $\mathsf{op}_{k+1} = \mathtt{skip}$, we conclude that player B takes the transition $\mathsf{s}_k \xrightarrow{\mathtt{skip}}_{\mathcal{P}} \mathsf{h}_1$. Player A has to respond with the only enabled transition in the resulting configuration, which is $\mathsf{h}_1 \xrightarrow{\mathtt{skip}}_{\mathcal{P}} \mathsf{s}_{k+1}$. The game is now in a configuration with states Q_k, as defined above. Furthermore, we conclude that the buffer must be $\mathcal{B}_{k+1}$, where $\mathsf{w}_{k+1} := \mathsf{w}_k$, and the memory is $\mathcal{M}_{k+1}$. Finally, we have that $\langle \mathsf{s}_k, \mathsf{w}_k\rangle \xrightarrow{\texttt{skip}} \langle \mathsf{s}_{k+1}, \mathsf{w}_{k+1}\rangle$. In other words, the game is now in position c_k.

If $\mathsf{op}_{k+1} = !\mathsf{m}$, we argue in the very same way as in the previous case that the run must have followed the transitions $\mathsf{s}_k \xrightarrow{\texttt{wr}(\mathsf{x}_{\texttt{wr}},\mathsf{m})}_{\mathcal{P}} \mathsf{h}_1 \xrightarrow{\texttt{wr}(\mathsf{y},1)}_{\mathcal{P}} \mathsf{s}_{k+11}$ in Proc^1. Similarly, the game is now in configuration $\mathsf{c}_{k+1} = \langle \mathcal{S}_{k+1}, \mathcal{B}_{k+1}, \mathcal{M}_{k+1}\rangle_B$ with $\mathsf{w}_{k+1} := \mathsf{m} \bullet \mathsf{w}_k$.

Lastly, if $\mathsf{op}_{k+1} = ?\mathsf{m}$, then player B takes $\mathsf{s}_k \xrightarrow{\texttt{skip}}_{\mathcal{P}} \mathsf{h}_1$. Assume that $\mathsf{w}_k = \varepsilon$, i.e. the buffer of Proc^1 is empty. Then, player A cannot update any message to the memory, which means that the transition $\mathsf{h}_1 \xrightarrow{\texttt{mf}}_{\mathcal{P}} \mathsf{h}_4$ is the only one that is enabled. This leads to the path $\mathsf{h}_1 \xrightarrow{\texttt{mf}}_{\mathcal{P}} \mathsf{h}_4 \xrightarrow{\texttt{skip}}_{\mathcal{P}} \mathsf{h}_5 \xrightarrow{\texttt{rd}(\mathsf{x}_{\texttt{wr}},\perp)}_{\mathcal{P}} \mathsf{h}_6$, which ends in a deadlocked configuration for player B. Since we already know that player B is winning, we conclude that the assumption was wrong and there is at least one pair of messages in the buffer of Proc^1.

Thus, player A instead performs one buffer update of the oldest message $\langle \mathsf{x}_{\texttt{wr}}, \mathsf{m}\rangle$, where m is the last character of w_k. This enables $\mathsf{q}_1 \xrightarrow{\texttt{rd}(\mathsf{x}_{\texttt{wr}},\mathsf{m})} \mathsf{q}_{\mathsf{m}}$ in Proc^2, which player A executes. Player B responds with $\mathsf{q}_{\mathsf{m}} \xrightarrow{\texttt{wr}(\mathsf{x}_{\texttt{rd}},\mathsf{m})} \mathsf{q}_3$ in Proc^2. Player A immediately updates this message to the memory, which enables $\mathsf{h}_1 \xrightarrow{\texttt{rd}(\mathsf{x}_{\texttt{rd}},\mathsf{m})} \mathsf{h}_2$ in Proc^1, which she takes. Note that this is her only choice since $\mathsf{q}_3 \xrightarrow{\texttt{wr}(\mathsf{x}_{\texttt{wr}},\perp)} \mathsf{q}_4$ opens up the possibility for player B to end the game by moving to q_F.

Now, Proc^1 is blocked again and the run continues in Proc^2. After

$$\mathsf{q}_3 \xrightarrow{\texttt{wr}(\mathsf{x}_{\texttt{wr}},\perp)} \mathsf{q}_4 \xrightarrow{\texttt{mf}} \mathsf{q}_5 \xrightarrow{\texttt{skip}} \mathsf{q}_6\ ,$$

player A updates the oldest buffer message of Proc^1, which is $\langle \mathsf{y}, 1\rangle$. This is needed to enable

$$\mathsf{q}_6 \xrightarrow{\texttt{rd}(\mathsf{y},1)} \mathsf{q}_7 \xrightarrow{\texttt{wr}(\mathsf{y},0)} \mathsf{q}_8 \xrightarrow{\texttt{mf}} \mathsf{q}_9 \xrightarrow{\texttt{wr}(\mathsf{x}_{\texttt{rd}},\perp)} \mathsf{q}_{10}\ .$$

Now, player A empties the buffer of Proc^2 to execute the transition $\mathsf{q}_{10} \xrightarrow{\texttt{mf}} \mathsf{q}_1$. This is followed by $\mathsf{h}_2 \xrightarrow{\texttt{rd}(\mathsf{x}_{\texttt{rd}},\perp)} \mathsf{h}_3 \xrightarrow{\texttt{skip}} \mathsf{s}_{k+1}$ in Proc^1.

We see that Proc^1 is now in state s_{k+1} and Proc^2 is back in state q_1, i.e. the process state is $\mathcal{S}_{k+1}$. Since all memory values have also been reset to their default values, we conclude that the buffer state of the current game configuration is $\mathcal{B}_{k+1}$, where $\mathsf{w}_{k+1} \bullet \mathsf{m} := \mathsf{w}_k$, and the memory is $\mathcal{M}_{k+1}$. Furthermore, the existence of the transition $\mathsf{s}_k \xrightarrow{?\mathsf{m}} \mathsf{s}_{k+1}$ follows from the construction of $\mathcal{P}$, and it is enabled at the configuration $\langle \mathsf{s}_k, \mathsf{w}_k\rangle$ of $\mathcal{L}$, since m is the last character in w_k.

This concludes the proof by induction. In summary, we showed that $\mathcal{L}$ can reach the state s_n starting from s_0. What is left to show is that $\mathsf{s}_n \in \mathsf{S}_F$. Consider the continuation of the run after c_n. If player B starts to simulate a skip or send operation, the run will inadvertedly visit another state of S after two moves, which is a contradiction, since we know that s_n is the last such state that is visited. Otherwise, if player B starts to simulate a receive operation, there are again two cases to consider. If the buffer of Proc^1 is empty, we have already seen that player A wins, which we know is not the case. Otherwise, player A can just proceed as usually and continue to simulate the receive operation in Proc^1 and Proc^2. In any case, player B has no opportunity to force the play into q_F or into a deadlock of player A. In particular, player A can avoid to take $\mathsf{q}_3 \xrightarrow{\texttt{skip}} \mathsf{q}_4$, since there are exactly four (i.e. an even number of) moves to take before.

We conclude that the only way that player B wins the game is that $\mathsf{s}_n \in \mathsf{Q}_F^{\mathcal{P}}$, or $\mathsf{s}_n \in \mathsf{S}_F$, equivalently. Since $\mathsf{s}_k \rightarrow \mathsf{s}_{k+1}$ for all $k < n$, it follows that S_F is reachable from s_0.

A.3. **Proof of Theorem 7.1.** This follows directly from Theorem A.1 and the undecidability of the state reachability problem for perfect channel systems [BZ83].

A.4. **Variants of A-TSO.** In the game where player A is restricted to perform updates only before her turn, one can check that the construction still works. However, if player A is only allowed to update after her turn, we need to adjust the construction. This is due to the fact that in the A-TSO game, there exist transitions where she is expected to update before her move.

We can easily fix this problem by adding two auxilliary states just before each of the affected transitions, and connect them with `skip` instructions. This gives player A time to update the buffer after taking the auxiliary transition.

Furthermore, we add a transition from the second auxiliary state to a sink state, that is, a state with just a self-loop. This prevents player B from taking the first transition, since player A can then go to the sink state, which effectively disables the process. It also ensures that whenever player A takes the first auxiliary transition, player B has to respond by taking the second one.

We conclude that the modified construction gives rise to the same behaviour than the construction for A-TSO. The formal proof is along the lines of Theorem A.1 or Theorem B.1.

Appendix B. Undecidability of the B-TSO Game

In this section we prove the undecidability of the B-TSO game, which is the TSO game where player B has full control over the buffer updates. Similar to what was presented in the A-TSO section, we construct a program consisting of two processes $\mathcal{P} = \langle \mathsf{Proc}^1, \mathsf{Proc}^2 \rangle$. As previously, Proc^1 starts out with all states of S and is then augmented by auxiliary states and transitions to simulate the PCS behaviour. This is sketched in Figure 17. The construction of Proc^2 is shown in Figure 18. Note that each state of Proc^1 labelled as h is supposed to be distinct for each transition. However, all other states with the same label, i.e. h_L, h_F, q_L, q_F and q_1, are actually representing the same state, respectively. The visual separation was done just for clarity. Note that Proc^2 is deadlock-free, which makes a dedicated Proc^3 superfluous. The set of final states is defined as $\mathsf{Q}_F^{\mathcal{P}} := \{\mathsf{q}_F\}$.

This construction, which will be described in more detail in the following, also works out for the variants of B-TSO where player B is allowed to update either only before or only after her move. In the former case, this is clear, since the construction never requires her to perform an update between her move and the next one of player A. In the latter case, this is due to the fact that the construction gives her enough opportunities to update during auxiliary `skip` transitions, which would not be needed in the B-TSO game with full buffer control.

In particular, consider the transition $\mathsf{q}_1 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}},\mathsf{m})} \mathsf{q}_\mathsf{m}$ in Proc^2 as an example. If player B wants to take this transition while the $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m} \rangle$ message is still in the buffer, she can instead move to h_1 and perform an update after her move. Player A then immediately needs to respond with going back to q_1, since q_F is winning for player B. For the same reason,

$$\begin{tikzcd}
\mathsf{s} \arrow[d, "\mathtt{skip}"] & & \\
\mathsf{h} \arrow[rr, "\mathtt{rd}(\mathsf{x}_{\mathtt{rd}}, \mathsf{m}')"] \arrow[d, "\mathtt{skip}"] & & \mathsf{h}_L \arrow[loop right, "\mathtt{skip}"] \\
\mathsf{s}' & &
\end{tikzcd}$$

(A) skip operation $\mathsf{s} \xrightarrow{\mathtt{skip}}_{\mathcal{L}} \mathsf{s}'$

$$\begin{tikzcd}
\mathsf{s} \arrow[d, "\mathtt{wr}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"] & & \\
\mathsf{h} \arrow[rr, "\mathtt{rd}(\mathsf{x}_{\mathtt{rd}}, \mathsf{m}')"] \arrow[d, "\mathtt{wr}(\mathsf{y}, 1)"] & & \mathsf{h}_L \arrow[loop right, "\mathtt{skip}"] \\
\mathsf{s}' & &
\end{tikzcd}$$

(B) send operation $\mathsf{s} \xrightarrow{!\mathsf{m}}_{\mathcal{L}} \mathsf{s}'$

$$\begin{tikzcd}
\mathsf{s} \arrow[d, "\mathtt{rd}(\mathsf{x}_{\mathtt{rd}}, \mathsf{m})"] \\
\mathsf{h} \arrow[d, "\mathtt{rd}(\mathsf{x}_{\mathtt{rd}}, \bot)"] \\
\mathsf{s}'
\end{tikzcd}$$

(C) receive operation $\mathsf{s} \xrightarrow{?\mathsf{m}}_{\mathcal{L}} \mathsf{s}'$

$$\begin{tikzcd}
\mathsf{s}_F \arrow[d, "\mathtt{wr}(\mathsf{x}_{\mathtt{wr}}, \top)"] \\
\mathsf{h}_F \arrow[loop below, "\mathtt{skip}"]
\end{tikzcd}$$

(D) construction for each $\mathsf{s}_F \in \mathsf{S}_F$

FIGURE 17. Proc^1 of the B-TSO reduction from PCS

player A cannot transition to h_1 herself, which prevents the potential of entering an infinite loop.

Furthermore, consider q_L in Proc^2. Note that it is immediately losing for player B, since the final state q_F is not reachable from there.

Theorem B.1. *Consider the B-TSO game or one of its variants. The set of final states* S_F *of* $\mathcal{L}$ *is reachable from* $\mathsf{s}_0 \in \mathsf{S}$ *if and only if player B wins the game* $\mathcal{G}^{\mathsf{TSO}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}})$ *starting from the configuration* $\mathsf{c}_0 := \langle\langle \mathsf{s}_0, \mathsf{q}_1\rangle, \langle\varepsilon, \varepsilon\rangle, \{\mathsf{x}_{\mathtt{wr}} \mapsto \bot, \mathsf{x}_{\mathtt{rd}} \mapsto \bot, \mathsf{y} \mapsto 0\}\rangle_B \in \mathsf{C}_B$.

B.1. From $\mathcal{L}$-Reachability to a Winning Strategy in B-TSO.

Proceeding as in the case of A-TSO games, suppose S_F is reachable from s_0. Given a run $\mathsf{c}_0^{\mathcal{L}}, \ldots, \mathsf{c}_n^{\mathcal{L}}$ in the PCS, recall the definitions of $\mathsf{c}_k^{\mathcal{L}}$ in Appendix A. For all $k = 0, \ldots, n$, we define a game configuration $\mathsf{c}_k = \langle \mathcal{S}_k, \mathcal{B}_k, \mathcal{M}_k\rangle_B$:

(1) $\mathcal{S}_k := \langle \mathsf{s}_k, \mathsf{q}_1\rangle$
(2) $\mathcal{B}_k := \langle\langle\langle \mathsf{y}, 1\rangle, \langle \mathsf{x}_{\mathtt{wr}}, \mathsf{w}_k[1]\rangle, \ldots, \langle \mathsf{y}, 1\rangle, \langle \mathsf{x}_{\mathtt{wr}}, \mathsf{w}_k[|\mathsf{w}_k|]\rangle, \langle \mathsf{y}, 1\rangle\rangle, \varepsilon\rangle$
(3) $\mathcal{M}_k := \{\mathsf{x}_{\mathtt{wr}} \mapsto \bot, \mathsf{x}_{\mathtt{rd}} \mapsto \bot, \mathsf{y} \mapsto 0\}$

We describe a strategy for player B that forces the play into visiting all configurations $\mathsf{c}_1, \ldots, \mathsf{c}_n$. Since $\mathsf{c}_n \in \mathsf{C}_F$, this means that the strategy is winning. To achieve this, we show by induction over k that starting from c_0, Player B can force a play that visits either c_k or C_F. For $k = 0$, this is true since c_0 is the initial configuration. Let the induction hypothesis

$$\begin{tikzcd}
\mathsf{q}_F \arrow[loop left, "\texttt{skip}"] & \mathsf{h}_1 \arrow[l, "\texttt{skip}"'] \arrow[r, bend right, "\texttt{skip}"'] & \mathsf{q}_1 \arrow[l, bend right] \arrow[r, "\texttt{rd}(\mathsf{x}_{\texttt{wr}}, \mathsf{m})"] \arrow[d, "\texttt{rd}(\mathsf{x}_{\texttt{wr}}, \mathsf{m})"] & \mathsf{q}_L \arrow[loop right, "\texttt{skip}"] \\
 & \text{for all } \mathsf{m} \in \mathsf{M} \{ & \mathsf{q}_{\mathsf{m}} \arrow[r, "\texttt{skip}"] \arrow[d, "\texttt{wr}(\mathsf{x}_{\texttt{rd}}, \mathsf{m})"] & \mathsf{q}_F \arrow[loop right, "\texttt{skip}"] \\
\mathsf{q}_F \arrow[loop left, "\texttt{skip}"] & \mathsf{h}_2 \arrow[l, "\texttt{skip}"'] \arrow[r, bend right, "\texttt{skip}"'] & \mathsf{q}_3 \arrow[l, bend right] \arrow[r, "\texttt{skip}"] \arrow[d, "\texttt{mf}"] & \mathsf{q}_L \arrow[loop right, "\texttt{skip}"] \\
 & & \mathsf{q}_4 \arrow[r, "\texttt{skip}"] \arrow[d, "\texttt{wr}(\mathsf{x}_{\texttt{wr}}, \bot)"] & \mathsf{q}_F \arrow[loop right, "\texttt{skip}"] \\
 & & \mathsf{q}_5 \arrow[d, "\texttt{skip}"] & \\
 & & \mathsf{q}_6 \arrow[r, "\texttt{skip}"] \arrow[d, "\texttt{mf}"] & \mathsf{q}_L \arrow[loop right, "\texttt{skip}"] \\
 & & \mathsf{q}_7 \arrow[r, "\texttt{rd}(\mathsf{y}, 1)"] \arrow[d, "\texttt{rd}(\mathsf{y}, 0)"] & \mathsf{q}_L \arrow[loop right, "\texttt{skip}"] \\
\mathsf{q}_F \arrow[loop left, "\texttt{skip}"] & \mathsf{h}_3 \arrow[l, "\texttt{skip}"'] \arrow[r, bend right, "\texttt{skip}"'] & \mathsf{q}_8 \arrow[l, bend right] \arrow[d, "\texttt{rd}(\mathsf{y}, 1)"] & \\
 & & \mathsf{q}_9 \arrow[d, "\texttt{wr}(\mathsf{y}, 0)"] & \\
\mathsf{q}_F \arrow[loop left, "\texttt{skip}"] & \mathsf{h}_4 \arrow[l, "\texttt{skip}"'] \arrow[r, bend right, "\texttt{skip}"'] & \mathsf{q}_{10} \arrow[l, bend right] \arrow[d, "\texttt{mf}"] & \\
 & & \mathsf{q}_{11} \arrow[r, "\texttt{rd}(\mathsf{x}_{\texttt{wr}}, \mathsf{m})"] \arrow[d, "\texttt{rd}(\mathsf{x}_{\texttt{wr}}, \bot)"] & \mathsf{q}_L \arrow[loop right, "\texttt{skip}"] \\
 & & \mathsf{q}_{12} \arrow[d, "\texttt{wr}(\mathsf{x}_{\texttt{rd}}, \bot)"] & \\
 & & \mathsf{q}_{13} \arrow[d, "\texttt{skip}"] & \\
\mathsf{q}_F \arrow[loop left, "\texttt{skip}"] & \mathsf{h}_5 \arrow[l, "\texttt{skip}"'] \arrow[r, bend right, "\texttt{skip}"'] & \mathsf{q}_{14} \arrow[l, bend right] \arrow[r, "\texttt{rd}(\mathsf{x}_{\texttt{wr}}, \mathsf{m})"] \arrow[d, "\texttt{mf}"] & \mathsf{q}_L \arrow[loop right, "\texttt{skip}"] \\
 & & \mathsf{q}_1 \arrow[r, "\texttt{rd}(\mathsf{x}_{\texttt{wr}}, \top)"] & \mathsf{q}_F \arrow[loop right, "\texttt{skip}"]
\end{tikzcd}$$

Figure 18. Proc^2 of the B-TSO reduction from PCS be that the claim holds true for some arbitrary but fixed $0 \leq k < n$. For the induction step, we can assume that the game is in configuration c_k.

In the following, we assume that player A avoids immediate defeats. This does not lose generality, since otherwise the game would be in C_F and the induction would still hold.

If $\mathsf{op}_{k+1} = \mathtt{skip}$, then player B first takes the transition $\mathsf{s}_k \xrightarrow{\mathtt{skip}} \mathsf{h}$ in Proc^1. Since all other outgoing transitions are disabled, player A has to follow up with the transition $\mathsf{h} \xrightarrow{\mathtt{skip}} \mathsf{s}_{k+1}$. As there is no change in the states of Proc^2, the buffers, or the memory, we can conclude that the current configuration of the game is c_{k+1}.

Next, consider the case where $\mathsf{op}_{k+1} = !\mathsf{m}$, which is similar to the previous one. Player B takes $\mathsf{s}_k \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}},\mathsf{m})} \mathsf{h}$ and the only option for player A is $\mathsf{h} \xrightarrow{\mathtt{wr}(\mathsf{y},1)} \mathsf{s}_{k+1}$. The buffer of Proc^1 has now two additional pending write operations, which correspond to the channel message m, i.e. the last letter of w_{k+1}. Thus, the game is in configuration c_{k+1}.

The last case is where $\mathsf{op}_{k+1} = ?\mathsf{m}$, which is more complicated. Player B starts by updating the message $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m}\rangle$ and taking the transition $\mathsf{q}_1 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}},\mathsf{m})} \mathsf{q}_\mathsf{m}$ in Proc^2 (recall the remark about the B-TSO variants above – she might need to take a detour over h_1). Player A has to respond with $\mathsf{q}_\mathsf{m} \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{rd}},\mathsf{m})} \mathsf{q}_3$, since otherwise player B could move to q_F and win. Now, player B is in a similar situation, where she needs to leave q_3, or else player A can move to q_L which prevents the game from ever reaching q_F. So, player B empties the buffer of Proc^2 (possibly using h_2) and takes the transition $\mathsf{q}_3 \xrightarrow{\mathtt{mf}} \mathsf{q}_4$. Player A has to respond with $\mathsf{q}_4 \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}},\perp)} \mathsf{q}_5$. At this point, player B cannot further proceed in Proc^2, since player A could respond with $\mathsf{q}_6 \xrightarrow{\mathtt{skip}} \mathsf{q}_L$ and win. Thus, she has to continue in Proc^1 instead, where the only enabled transition is $\mathsf{s}_k \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{rd}},\mathsf{m})} \mathsf{h}$. Now, Proc^1 is disabled and the play continues in Proc^2. Eventually, it will go through all states $\mathsf{q}_6, \ldots, \mathsf{q}_{14}$. Note that player B has to update the messages $\langle \mathsf{x}_{\mathtt{wr}}, \perp\rangle$, $\langle \mathsf{y}, 1\rangle$, $\langle \mathsf{y}, 0\rangle$ and $\langle \mathsf{x}_{\mathtt{rd}}, \perp\rangle$ along the way, and has enough opportunities to do so. Finally, player B takes the transition $\mathsf{q}_{14} \xrightarrow{\mathtt{mf}} \mathsf{q}_1$ in Proc^2 and player A is forced to respond with $\mathsf{h} \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{rd}},\perp)} \mathsf{s}_{k+1}$ in Proc^1. Note that the last transition is enabled since the $\langle \mathsf{x}_{\mathtt{rd}}, \perp\rangle$-message has certainly reached the memory.

To summarise the changes: Proc^1 transitioned from s_k to s_{k+1}, while Proc^2 is back in state q_1. The buffer of Proc^1 has updated its two oldest messages, which correspond to the last letter in w_k, and the buffer of Proc^2 has been emptied. All variables have their default values.

We conclude that the game is in configuration c_{k+1}, which completes the proof by induction. In summary, starting from c_0, the game either reaches the configuration $\mathsf{c}_n \in \mathsf{C}_F$, or another configuration in C_F. In the latter case, player B already won, but in the former case, she takes the transition $\mathsf{s}_n \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}},\top)} \mathsf{h}_F$, updates this message, and finally reaches $\mathsf{q}_1 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}},\top)} \mathsf{q}_F \in \mathsf{Q}_F^{\mathcal{P}}$ in her next turn.

B.2. **From a Winning Strategy in B-TSO to $\mathcal{L}$-Reachability.**

For the other direction, suppose that player B has a winning strategy. Consider a strategy of player A that avoids reaching $\mathsf{q}_F \in \mathsf{Q}^2 \cap \mathsf{Q}_F^{\mathcal{P}}$ whenever possible. Furthermore, to support our argumentation we can assume arbitrary behaviour of player A, since the strategy of player B must be winning in any case.

Let $\mathsf{s}_0, \dots, \mathsf{s}_n$ be the sequence of channel states that are visited by Proc^1 during the run induced by these two strategies. Since player B uses a winning strategy, this sequence is finite. We will show by induction that for each $k = 0, \dots, n$, the run contains the game configuration $\mathsf{c}_k = \langle \mathcal{S}_k, \mathcal{B}_k, \mathcal{M}_k \rangle_B \in \mathsf{C}_B$, where:

(1) $\mathcal{S}_k := \langle \mathsf{s}_k, \mathsf{q}_1 \rangle$
(2) $\mathcal{B}_k := \langle \langle \langle \mathsf{y}, 1 \rangle, \langle \mathsf{x}_{\mathtt{wr}}, \mathsf{w}_k[1] \rangle, \dots, \langle \mathsf{y}, 1 \rangle, \langle \mathsf{x}_{\mathtt{wr}}, \mathsf{w}_k[|\mathsf{w}_k|] \rangle \rangle, \varepsilon \rangle$, where $\mathsf{w}_k \in \mathsf{M}^*$
(3) $\mathcal{M}_k := \{ \mathsf{x}_{\mathtt{wr}} \mapsto \perp, \mathsf{x}_{\mathtt{rd}} \mapsto \perp, \mathsf{y} \mapsto 0 \}$
(4) If $k > 0$, there is a label op_k such that $\langle \mathsf{s}_{k-1}, \mathsf{w}_{k-1} \rangle \xrightarrow{\mathsf{op}_k} \langle \mathsf{s}_k, \mathsf{w}_k \rangle$.

The induction base case clearly holds true, since c_0 is the initial configuration of the game. So, suppose that the claim holds for some arbitrary but fixed $0 \leq k < n$. By the induction hypothesis, we can assume that the game is in configuration c_k. Consider the different moves player B can make.

First, she may take a transition $\mathsf{s}_k \xrightarrow{\mathtt{skip}} \mathsf{h}$. She surely does not update any buffer messages during that move, since this would enable player A to take $\mathsf{q}_1 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})} \mathsf{q}_L$ and win. So, we assume that player A moves along $\mathsf{h} \xrightarrow{\mathtt{skip}} \mathsf{s}'$. By the construction of Proc^1, the path $\mathsf{s}_k \xrightarrow{\mathtt{skip}} \mathsf{h} \xrightarrow{\mathtt{skip}} \mathsf{s}'$ implies the existence of a transition $\mathsf{s}_k \xrightarrow{\mathtt{skip}} \mathsf{s}'$. We define $\mathsf{s}_{k+1} := \mathsf{s}'$ and $\mathsf{w}_{k+1} := \mathsf{w}_k$ and conclude that the game is now in configuration c_{k+1}.

Another option for player A is to take the transition $\mathsf{s}_k \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})} \mathsf{h}$. The argumentation continues exactly as in the previous paragraph, the only difference is that $\mathsf{s}_k \xrightarrow{!\mathsf{m}} \mathsf{s}'$ and $\mathsf{w}_{k+1} := \mathsf{m} \bullet \mathsf{w}_k$.

Since all other transitions in Proc^1 are disabled, the last possibility for player B to move is in Proc^2. She first has to update a $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m} \rangle$-message to the memory (potentially using the loop through h_1), and then she is able to perform $\mathsf{q}_1 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})} \mathsf{q}_{\mathsf{m}}$. Assume that player A responds with $\mathsf{q}_{\mathsf{m}} \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{rd}}, \mathsf{m})} \mathsf{q}_3$. Since staying in q_3 is losing for player B, she needs to update the $\langle \mathsf{x}_{\mathtt{rd}}, \mathsf{m} \rangle$-message to the memory, which enables $\mathsf{q}_3 \xrightarrow{\mathtt{skip}} \mathsf{q}_4$ in Proc^2. Player A is now forced to proceed from there and takes $\mathsf{q}_4 \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}}, \perp)} \mathsf{q}_5$.

At this point, player B cannot continue in Proc^2, since moving to q_6 opens up the opportunity for player A to immediately force her win. However, also in Proc^1 she has limited possibilities. In case she starts to simulate a skip or send operation, player A can react with $\mathsf{h} \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{rd}}, \mathsf{m})} \mathsf{q}_L$ and win. Since player B employs a winning strategy, we conclude that at least one transition $\mathsf{s}_k \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{rd}}, \mathsf{m})} \mathsf{h}$ is enabled, which player B takes. Player A then has to proceed in Proc^2 and the play continues there until reaching q_{14}.

We note the following observations, based on the fact that player A cannot force a win. First, in q_7, the variable y apparently has value 0. This means that up to this point, no message from the buffer of Proc^1 has been updated (other than the single $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m} \rangle$ from before). Second, between q_8 and q_9, the value of y is 1. We conclude that at least one $\langle \mathsf{y}, 1 \rangle$-message has been updated. Third, in q_{11}, player A reads $\perp$ from $\mathsf{x}_{\mathtt{wr}}$. This shows that since the transition $\mathsf{q}_6 \xrightarrow{\mathtt{mf}} \mathsf{q}_7$, no further $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m}' \rangle$-message has been updated. In summary, we conclude that during the whole process, exactly the two messages $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m} \rangle$ and $\langle \mathsf{y}, 1 \rangle$ have been updated to the memory.

We continue with the execution from q_{14}. Player B empties the buffer of Proc^2 and takes either $\mathsf{h} \xrightarrow{\mathtt{rd}(\mathtt{x_{rd}},\bot)} \mathsf{s}'$ in Proc^1 or $\mathsf{q}_{14} \xrightarrow{\mathtt{mf}} \mathsf{q}_1$ in Proc^2. We assume that player A takes the other one, respectively.

We define $\mathsf{s}_{k+1} := \mathsf{s}'$ and w_{k+1} by $\mathsf{w}_{k+1}\bullet\mathsf{m} := \mathsf{w}_k$. The path $\mathsf{s}_k \xrightarrow{\mathtt{rd}(\mathtt{x_{rd}},\mathsf{m})} \mathsf{h} \xrightarrow{\mathtt{rd}(\mathtt{x_{rd}},\bot)} \mathsf{s}_{k+1}$ implies that $\langle \mathsf{s}_k, \mathsf{w}_k\rangle \xrightarrow{?\mathsf{m}} \langle \mathsf{s}_{k+1}, \mathsf{w}_{k+1}\rangle$. Furthermore, the observations above imply that the game is in configuration c_{k+1} as defined in the beginning. This concludes the proof by induction.

Now, we investigate how the run can continue after c_n. Player B cannot start to simulate a skip or send operation, since we can assume that player A will move to s', which contradicts the initial assumption that s_n is the last channel state visited in this run. The same holds true for a receive operation: Following the argumentation from above, we have already seen that player A can either force a win, or lead the game into completing the simulation of the receive operation. The only possibility left is that s_n is a final channel state and player B takes the transition $\mathsf{s}_n \xrightarrow{\mathtt{wr}(\mathtt{x_{wr}},\top)} \mathsf{h}_F$.

In summary, we constructed a path through $\mathcal{L}$ from $\langle \mathsf{s}_0, \varepsilon\rangle$ to $\langle \mathsf{s}_n, \mathsf{w}_n\rangle$, where $\mathsf{s}_n \in \mathsf{S}_F$. This shows that the set of final states of $\mathcal{L}$ is reachable.

B.3. Undecidability.

Theorem B.2. *The safety problem for the B-TSO game is undecidable.*

Proof. This follows directly from Theorem B.1 and the undecidability of the state reachability problem for perfect channel systems [BZ83]. □

Appendix C. Undecidability of the AB-TSO game

In this final section we prove undecidability of the AB-TSO game, which is the TSO game where the players share control over buffer updates in the way that both of them are allowed to update exactly after their own move. The construction will be very similar to the one for the B-TSO game. In particular, Proc^1 is exactly the same and is sketched in Figure 17. The construction for Proc^2 on the other hand is different and shown in Figure 19. Note that this process is deadlock-free by design. As previously, the set of final states is defined as $\mathsf{Q}_F^{\mathcal{P}} := \{\mathsf{q}_F\}$. Note that q_L in Proc^2 is immediately losing for player B, since the final state q_F is not reachable from there.

Theorem C.1. *Consider the AB-TSO game. The set of final states S_F of $\mathcal{L}$ is reachable from $\mathsf{s}_0 \in \mathsf{S}$ if and only if player B wins the game $\mathcal{G}^{\mathsf{TSO}}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}})$ starting from the configuration $\mathsf{c}_0 := \langle\langle \mathsf{s}_0, \mathsf{q}_1\rangle, \langle\varepsilon,\varepsilon\rangle, \{\mathtt{x_{wr}} \mapsto \bot, \mathtt{x_{rd}} \mapsto \bot, \mathsf{y} \mapsto 0\}\rangle_B \in \mathsf{C}_B$.*

C.1. From $\mathcal{L}$-Reachability to a Winning Strategy in AB-TSO.

Proceeding as in the case of the B-TSO games, suppose S_F is reachable from s_0. Given a run $\mathsf{c}_0^{\mathcal{L}}, \ldots, \mathsf{c}_n^{\mathcal{L}}$ in the PCS, recall the definitions of $\mathsf{c}_k^{\mathcal{L}}$ in Appendix A and of c_k in Appendix B.

We describe a strategy for player B that forces the play into visiting all configurations $\mathsf{c}_1, \ldots, \mathsf{c}_n$. Since $\mathsf{c}_n \in \mathsf{C}_F$, this means that the strategy is winning. To achieve this, we show by induction over k that starting from c_0, Player B can force a play that visits either c_k or C_F. For $k = 0$, this is true since c_0 is the initial configuration. Let the induction hypothesis

$$\begin{tikzcd}
\mathsf{q}_L \arrow[loop left, "\mathtt{skip}"] & \mathsf{q}_1 \arrow[l, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"'] \arrow[r, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"] \arrow[d, "\mathtt{skip}"] & \mathsf{q}_F \arrow[loop right, "\mathtt{skip}"] \\
\mathsf{q}_L \arrow[loop left, "\mathtt{skip}"] & \mathsf{q}_2 \arrow[l, "\mathtt{rd}(\mathsf{y}, 1)"', "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \bot)"] \arrow[r, "\mathtt{skip}"] \arrow[d, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"] & \mathsf{q}_F \arrow[loop right, "\mathtt{skip}"] \\
\mathsf{q}_L \arrow[loop left, "\mathtt{skip}"] & \mathsf{q}_{\mathsf{m}} \arrow[l, "\mathtt{skip}"'] \arrow[r, "\mathtt{rd}(\mathsf{y}, 1)"] \arrow[d, "\mathtt{wr}(\mathsf{x}_{\mathtt{rd}}, \mathsf{m})"] & \mathsf{q}_F \arrow[loop right, "\mathtt{skip}"] \\
\mathsf{q}_L \arrow[loop left, "\mathtt{skip}"] & \mathsf{q}_4 \arrow[l, "\mathtt{rd}(\mathsf{y}, 1)"'] \arrow[r, "\mathtt{skip}"] \arrow[d, "\mathtt{wr}(\mathsf{x}_{\mathtt{wr}}, \bot)"] & \mathsf{q}_F \arrow[loop right, "\mathtt{skip}"] \\
\mathsf{q}_L \arrow[loop left, "\mathtt{skip}"] & \mathsf{q}_5 \arrow[l, "\mathtt{rd}(\mathsf{y}, 1)"'] \arrow[r, "\mathtt{rd}(\mathsf{y}, 1)"] \arrow[d, "\mathtt{skip}"] & \mathsf{q}_F \arrow[loop right, "\mathtt{skip}"] \\
\mathsf{q}_L \arrow[loop left, "\mathtt{skip}"] & \mathsf{q}_6 \arrow[l, "\mathtt{skip}"'] \arrow[r, "\mathtt{rd}(\mathsf{y}, 1)"] \arrow[d, "\mathtt{mf}"] & \mathsf{q}_F \arrow[loop right, "\mathtt{skip}"] \\
\mathsf{q}_L \arrow[loop left, "\mathtt{skip}"] & \mathsf{q}_7 \arrow[l, "\mathtt{rd}(\mathsf{y}, 1)"'] \arrow[d, "\mathtt{skip}"] & \\
 & \mathsf{q}_8 \arrow[r, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})", "\mathtt{rd}(\mathsf{y}, 0)"'] \arrow[d, "\mathtt{wr}(\mathsf{y}, 0)"] & \mathsf{q}_F \arrow[loop right, "\mathtt{skip}"] \\
\mathsf{q}_L \arrow[loop left, "\mathtt{skip}"] & \mathsf{q}_9 \arrow[l, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"'] \arrow[d, "\mathtt{skip}"] & \\
\mathsf{q}_L \arrow[loop left, "\mathtt{skip}"] & \mathsf{q}_{10} \arrow[l, "\mathtt{skip}"'] \arrow[r, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"] \arrow[d, "\mathtt{mf}"] & \mathsf{q}_F \arrow[loop right, "\mathtt{skip}"] \\
\mathsf{q}_L \arrow[loop left, "\mathtt{skip}"] & \mathsf{q}_{11} \arrow[l, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"'] \arrow[d, "\mathtt{wr}(\mathsf{x}_{\mathtt{rd}}, \bot)"] & \\
 & \mathsf{q}_{12} \arrow[r, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \mathsf{m})"] \arrow[d, "\mathtt{skip}"] & \mathsf{q}_F \arrow[loop right, "\mathtt{skip}"] \\
 & \mathsf{q}_1 \arrow[r, "\mathtt{rd}(\mathsf{x}_{\mathtt{wr}}, \top)"] & \mathsf{q}_F \arrow[loop right, "\mathtt{skip}"]
\end{tikzcd}$$

Figure 19. Proc^2 of the AB-TSO reduction from PCS be that the claim holds true for some arbitrary but fixed $0 \leq k < n$. For the induction step, we can assume that the game is in configuration c_k.

In the following, we assume that player A avoids immediate defeats. This does not lose generality, since otherwise the game would be in C_F and the induction would still hold.

If $\mathsf{op}_{k+1} = \mathtt{skip}$, then player B first takes the transition $\mathsf{s}_k \xrightarrow{\mathtt{skip}} \mathsf{h}$ in Proc^1. For player A, there are only two enabled transitions. After $\mathsf{q}_1 \xrightarrow{\mathtt{skip}} \mathsf{q}_2$, player B could move to q_F and win. Thus, player A has to follow up with the transition $\mathsf{h} \xrightarrow{\mathtt{skip}} \mathsf{s}_{k+1}$ instead. Suppose she also updates one or more messages after her move. Then, player B could take $\mathsf{q}_1 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}},\mathsf{m}')} \mathsf{q}_F$ for some message m' and win. So, we continue assuming that player A does not update anything to the memory. As there is no change in the states of Proc^2, the buffers or the memory, we can conclude that the current configuration of the game is c_{k+1}.

Next, consider the case where $\mathsf{op}_{k+1} = !\mathsf{m}$, which is similar to the previous one. Player B takes $\mathsf{s}_k \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}},\mathsf{m})} \mathsf{h}$ and the only option for player A is $\mathsf{h} \xrightarrow{\mathtt{wr}(\mathsf{y},1)} \mathsf{s}_{k+1}$. As above, we argue that she does not perform any buffer updates. The buffer of Proc^1 has now two additional pending write operations, which correspond to the channel message m, i.e. the last letter of w_{k+1}. Thus, the game is in configuration c_{k+1}.

The last case is where $\mathsf{op}_{k+1} = ?\mathsf{m}$, which is more complicated. Player B starts by taking the transition $\mathsf{q}_1 \xrightarrow{\mathtt{skip}} \mathsf{q}_2$ in Proc^2 and then updates the message $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m} \rangle$ to the memory. Now, player A is forced to leave q_2, since otherwise player B could move to q_F in her next turn. The only enabled transition from there is $\mathsf{q}_2 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}},\mathsf{m})} \mathsf{q}_3$. Again, we see that she cannot update any messages without losing in the very next turn. Player B responds with $\mathsf{q}_3 \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{rd}},\mathsf{m})} \mathsf{q}_4$, which she immediately updates to the memory. As before, player A must leave this state, the only option for this is to follow $\mathsf{q}_4 \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}},\perp)} \mathsf{q}_5$. She does so without buffer updates. Now, player B takes the transition $\mathsf{s}_k \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{rd}},\mathsf{m})} \mathsf{h}$ in Proc^1. She also empties the buffer of Proc^2. Player A has to continue in Proc^2, since all outgoing transitions in Proc^1 are disabled. The only option is to go to q_6, as usual without performing any updates. Next, player B can perform $\mathsf{q}_6 \xrightarrow{\mathtt{mf}} \mathsf{q}_7$, since she emptied the buffer one turn earlier. Player A continues to q_8. If she does not perform any updates, or updates more than one message, i.e. at least one $\langle \mathsf{y}, 1 \rangle$ and one $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m} \rangle$ buffer message (for some channel message m), then player B can immediately win in her next turn. So, we proceed assuming that player A updates exactly one message $\langle \mathsf{y}, 1 \rangle$. Then, player B takes the transition $\mathsf{q}_8 \xrightarrow{\mathtt{wr}(\mathsf{y},0)} \mathsf{q}_9$ and immediately updates this message to the memory. This leaves player A with the only possibility of moving to q_{10}. As usual, she cannot update any buffer message without losing. Player B continues with $\mathsf{q}_{10} \xrightarrow{\mathtt{mf}} \mathsf{q}_{11}$. Player A has to perform $\mathsf{q}_{11} \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{rd}},\perp)} \mathsf{q}_{12}$ and may update this message, but nothing else. Next, player B moves to q_1 and empties the buffer of Proc^2. Since player A cannot move to q_2 without losing, she takes the transition $\mathsf{h} \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{rd}},\perp)} \mathsf{s}_{k+1}$. This is enabled since the buffer of Proc^2 was last emptied.

To summarise the changes: Proc^1 transitioned from s_k to s_{k+1}, while Proc^2 is back in state q_1. The buffer of Proc^1 has updated its two oldest messages, which correspond to the last letter in w_k, and the buffer of Proc^2 has been emptied. All variables have their default values.

We conclude that the game is in configuration c_{k+1}, which completes the proof by induction. In summary, starting from c_0, the game either reaches the configuration $\mathsf{c}_n \in \mathsf{C}_F$, or another configuration in C_F. In the latter case, player B already won, but in the former case, she takes the transition $\mathsf{s}_n \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}},\top)} \mathsf{h}_F$, updates this message, and finally reaches $\mathsf{q}_1 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}},\top)} \mathsf{q}_F \in \mathsf{Q}^{\mathcal{P}}_F$ in her next turn.

C.2. **From a Winning Strategy in AB-TSO to $\mathcal{L}$-Reachability.** For the other direction, suppose that player B has a winning strategy. Consider a strategy of player A that avoids reaching $\mathsf{q}_F \in \mathsf{Q}^2 \cap \mathsf{Q}^{\mathcal{P}}_F$ whenever possible. Furthermore, to support our argumentation we can assume arbitrary behaviour of player A, since the strategy of player B must be winning in any case.

Let $\mathsf{s}_0, \dots, \mathsf{s}_n$ be the sequence of channel states that are visited by Proc^1 during the run induced by these two strategies. Since player B uses a winning strategy, this sequence is finite. We will show by induction that for each $k = 0, \dots, n$, the run contains the game configuration $\mathsf{c}_k = \langle \mathcal{S}_k, \mathcal{B}_k, \mathcal{M}_k \rangle_B \in \mathsf{C}_B$, where:

(1) $\mathcal{S}_k := \langle \mathsf{s}_k, \mathsf{q}_1 \rangle$
(2) $\mathcal{B}_k := \langle \langle \langle \mathsf{y}, 1 \rangle, \langle \mathsf{x}_{\mathtt{wr}}, \mathsf{w}_k[1] \rangle, \dots, \langle \mathsf{y}, 1 \rangle, \langle \mathsf{x}_{\mathtt{wr}}, \mathsf{w}_k[|\mathsf{w}_k|] \rangle \rangle, \varepsilon \rangle$, where $\mathsf{w}_k \in \mathsf{M}^*$
(3) $\mathcal{M}_k := \{ \mathsf{x}_{\mathtt{wr}} \mapsto \bot, \mathsf{x}_{\mathtt{rd}} \mapsto \bot, \mathsf{y} \mapsto 0 \}$
(4) If $k > 0$, there is a label op_k such that $\langle \mathsf{s}_{k-1}, \mathsf{w}_{k-1} \rangle \xrightarrow{\mathsf{op}_k} \langle \mathsf{s}_k, \mathsf{w}_k \rangle$.

The induction base case clearly holds true, since c_0 is the initial configuration of the game. So, suppose that the claim holds for some arbitrary but fixed $0 \le k < n$. By the induction hypothesis, we can assume that the game is in configuration c_k. Consider the different moves player B can make.

First, she may take a transition $\mathsf{s}_k \xrightarrow{\mathtt{skip}} \mathsf{h}$. She surely does not update any buffer messages during that move, since this would enable player A to take $\mathsf{q}_1 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}},\mathsf{m})} \mathsf{q}_L$ and win. So, we assume that player A moves along $\mathsf{h} \xrightarrow{\mathtt{skip}} \mathsf{s}'$. By the construction of Proc^1, the path $\mathsf{s}_k \xrightarrow{\mathtt{skip}} \mathsf{h} \xrightarrow{\mathtt{skip}} \mathsf{s}'$ implies the existence of a transition $\mathsf{s}_k \xrightarrow{\mathtt{skip}} \mathsf{s}'$. We define $\mathsf{s}_{k+1} := \mathsf{s}'$ and $\mathsf{w}_{k+1} := \mathsf{w}_k$ and conclude that the game is now in configuration c_{k+1}.

Another option for player A is to take the transition $\mathsf{s}_k \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}},\mathsf{m})} \mathsf{h}$. The argumentation continues exactly as in the previous paragraph, the only difference is that $\mathsf{s}_k \xrightarrow{!\mathsf{m}} \mathsf{s}'$ and $\mathsf{w}_{k+1} := \mathsf{m} \bullet \mathsf{w}_k$.

Since all other transitions in Proc^1 are disabled, the last possibility for player B to move is in Proc^2. She moves from q_1 to q_2. We can assume that she updates exactly one message from the buffer of Proc^1, since otherwise player A could win in her next turn. Instead, player A has to follow $\mathsf{q}_2 \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}},\mathsf{m})} \mathsf{q}_{\mathsf{m}}$. In the following, we assume that player A performs no buffer updates unless stated otherwise. Player B takes the transition $\mathsf{q}_{\mathsf{m}} \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{rd}},\mathsf{m})} \mathsf{q}_4$. She might immediately update this message, but nothing else, following the same argumentation as previously. Player A continues with $\mathsf{q}_4 \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}},\bot)} \mathsf{q}_5$, we can assume that she also empties the buffer of Proc^2. Moving to q_6 is losing for player B, thus we can safely say that she performs $\mathsf{s}_k \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{rd}},\mathsf{m})} \mathsf{h}$ instead, which is the only enabled transition in Proc^1. Player A then has to proceed in Proc^2 and the play continues there until reaching q_{11}.

We note the following observations, based on the fact that player A cannot force a win. First, in q_7, the variable y apparently has value 0, otherwise player A would win. This means that up to this point, no message from the buffer of Proc^1 has been updated (other than the single $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m}\rangle$ from before). Second, after moving to q_8, player A performs such a buffer update to prevent player B from immediately winning. Third, in q_{11}, the value of $\mathsf{x}_{\mathtt{wr}}$ in the memory must be $\perp$, since otherwise player A could win with $\mathsf{q}_{11} \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{wr}},\mathsf{m})} \mathsf{q}_L$ for some channel message m. This shows that since the transition $\mathsf{q}_6 \xrightarrow{\mathtt{mf}} \mathsf{q}_7$, no further $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m}'\rangle$-message has been updated. In summary, we conclude that during the whole process, exactly the two messages $\langle \mathsf{x}_{\mathtt{wr}}, \mathsf{m}\rangle$ and $\langle \mathsf{y}, 1\rangle$ have been updated to the memory.

We continue with the execution from q_{11}. Player A performs $\mathsf{q}_{11} \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{rd}},\perp)} \mathsf{q}_{12}$ and immediately updates this message to the memory. Player B either takes either $\mathsf{h} \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{rd}},\perp)} \mathsf{s}'$ in Proc^1 or $\mathsf{q}_{11} \xrightarrow{\mathtt{skip}} \mathsf{q}_1$ in Proc^2. We assume that player A takes the other one, respectively.

We define $\mathsf{s}_{k+1} := \mathsf{s}'$ and w_{k+1} by $\mathsf{w}_{k+1} \bullet \mathsf{m} := \mathsf{w}_k$. The path $\mathsf{s}_k \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{rd}},\mathsf{m})} \mathsf{h} \xrightarrow{\mathtt{rd}(\mathsf{x}_{\mathtt{rd}},\perp)} \mathsf{s}_{k+1}$ implies that $\langle \mathsf{s}_k, \mathsf{w}_k\rangle \xrightarrow{?\mathsf{m}} \langle \mathsf{s}_{k+1}, \mathsf{w}_{k+1}\rangle$. Furthermore, the observations above imply that the game is in configuration c_{k+1} as defined in the beginning.

Now, we investigate how the run can continue after c_n. Player B cannot start to simulate a skip or send operation, since we can assume that player A will move to s', which contradicts the initial assumption that s_n is the last channel state visited in this run. The same holds true for a receive operation: Following the argumentation from above, we have already seen that player A can either force a win, or lead the game into completing the simulation of the receive operation. The only possibility left is that s_n is a final channel state and player B takes the transition $\mathsf{s}_n \xrightarrow{\mathtt{wr}(\mathsf{x}_{\mathtt{wr}},\top)} \mathsf{h}_F$.

In summary, we constructed a path through $\mathcal{L}$ from $\langle \mathsf{s}_0, \varepsilon\rangle$ to $\langle \mathsf{s}_n, \mathsf{w}_n\rangle$, where $\mathsf{s}_n \in \mathsf{S}_F$. This shows that the set of final states of $\mathcal{L}$ is reachable.

C.3. **Undecidability.**

Theorem C.2. *The safety problem for the AB-TSO game is undecidable.*

Proof. This follows directly from Theorem C.1 and the undecidability of the state reachability problem for perfect channel systems [BZ83]. □