The Benefits of Duality in Verifying Concurrent Programs under TSO[*]

Parosh Aziz Abdulla[1], Mohamed Faouzi Atig[2], Ahmed Bouajjani[3], and Tuan Phong Ngo[4]

1 **Uppsala University, Sweden**
`parosh@it.uu.se`

2 **Uppsala University, Sweden**
`mohamed_faouzi.atig@it.uu.se`

3 **IRIF, Université Paris Diderot & IUF, France**
`abou@liafa.univ-paris-diderot.fr`

4 **Uppsala University, Sweden**
`tuan-phong.ngo@it.uu.se`

Abstract

We address the problem of verifying safety properties of concurrent programs running over the TSO memory model. Known decision procedures for this model are based on complex encodings of store buffers as lossy channels. These procedures assume that the number of processes is fixed. However, it is important in general to prove correctness of a system/algorithm in a parametric way with an arbitrarily large number of processes. In this paper, we introduce an alternative (yet equivalent) semantics to the classical one for the TSO model that is more amenable for efficient algorithmic verification and for extension to parametric verification. For that, we adopt a *dual* view where *load buffers* are used instead of store buffers. The flow of information is now from the memory to load buffers. We show that this new semantics allows (1) to simplify drastically the safety analysis under TSO, (2) to obtain a spectacular gain in efficiency and scalability compared to existing procedures, and (3) to extend easily the decision procedure to the parametric case, which allows to obtain a new decidability result, and more importantly, a verification algorithm that is more general and more efficient in practice than the one for bounded instances.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Weak Memory Models, Reachability Problem, Parameterized Verification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2016.5

1 Introduction

Most modern processor architectures execute instructions in an out-of-order manner to gain efficiency. In the context of *sequential* programming, this out-of-order execution is transparent to the programmer since one can still work under the Sequential Consistency (SC) model [24]. However, this is not true when we consider concurrent processes that share the memory. In fact, it turns out that concurrent algorithms such as mutual exclusion and producer-consumer protocols may not behave correctly any more. Therefore, program verification is a relevant (and difficult) task in order to prove correctness under the new semantics. The inadequacy of the interleaving semantics has led to the invention of new program semantics, so called

[*] This work was supported in part by the Swedish Research Council and carried out within the Linnaeus centre of excellence UPMARC, Uppsala Programming for Multicore Architectures Research Center.

Weak (or relaxed) Memory Models (WMM), by allowing permutations between certain types of memory operations [7, 20, 8]. Total Store Ordering (TSO) is one of the the most common models, and it corresponds to the relaxation adopted by Sun's SPARC multiprocessors [28] and formalizations of the x86-tso memory model [26, 27]. These models put an unbounded perfect (non-lossy) *store buffer* between each process and the main memory where a store buffer carries the pending store operations of the process. When a process performs a store operation, it appends it to the end of its buffer. These operations are propagated to the shared memory non-deterministically in a FIFO manner. When a process reads a variable, it searches its buffer for a pending store operation on that variable. If no such a store operation exists, it fetches the value of the variable from the main memory. Verifying programs running on the TSO memory model poses a difficult challenge since the unboundedness of the buffers implies that the state space of the system is infinite even in the case where the input program is finite-state. Decidability of safety properties has been obtained by constructing equivalent models that replace the perfect store buffer by *lossy* channels [11, 12, 2]. However, these constructions are complicated and involve several ingredients that lead to inefficient verification procedures. For instance, they require each message inside a lossy channel to carry (instead of a single store operation) a full snapshot of the memory representing a local view of the memory contents by the process. Furthermore, the reductions involve non-deterministic guessing the lossy channel contents. The guessing is then resolved either by consistency checking [11] or by using explicit pointer variables (each corresponding to one process) inside the buffers [2], causing a serious state space explosion problem.

In this paper, we introduce a novel semantics which we call the *dual TSO* semantics. Our aim is to provide an alternative (and equivalent) semantics that is more amenable for efficient algorithmic verification. The main idea is to have *load buffers* that contain pending load operations (more precisely, values that will potentially be taken by forthcoming load operations) rather than store buffers (that contain store operations). The flow of information will now be in the reverse direction, i.e., store operations are performed by the processes atomically on the main memory, while values of variables are propagated non-deterministically from the memory to the load buffers of the processes. When a process performs a load operation it can fetch the value of the variable from the head of its load buffer. We show that the dual semantics is equivalent to the original one in the sense that any given set of processes will reach the same set of local states under both semantics. The dual semantics allows us to understand the TSO model in a totally different way compared to the classical semantics. Furthermore, the dual semantics offers several important advantages from the point of view of formal reasoning and program verification. First, the dual semantics allows transforming the load buffers to *lossy* channels without adding the costly overhead that was necessary in the case of store buffers. This means that we can apply the theory of *well-structured systems* [6, 5, 21] in a straightforward manner leading to a much simpler proof of decidability of safety properties. Second, the absence of extra overhead means that we obtain more efficient algorithms and better scalability (as shown by our experimental results). Finally, the dual semantics allows extending the framework to perform *parameterized verification* which is an important paradigm in concurrent program verification. Here, we consider systems, e.g., mutual exclusion protocols, that consist of an arbitrary number of processes. The aim of parameterized verification is to prove correctness of the system regardless of the number of processes. It is not obvious how to perform parameterized verification under the classical semantics. For instance, extending the framework of [2], would involve an unbounded number of pointer variables, thus leading to channel systems with unbounded message alphabets. In contrast, as we show in this paper, the simple nature of the dual semantics allows a straightforward extension of our verification algorithm to the case of parameterized verification. This is the first time a decidability result is established for the parametrized verification of programs running over WMM. Notice that this result is taking into account two sources of infinity: the number of processes, and the size of the buffers.

Based on our framework, we have implemented a tool and applied it to a large set of benchmarks. The experiments demonstrate the efficiency of the dual semantics compared to the classical one (by two order of magnitude in average), and the feasibility of parametrized verification in the former case. In fact, besides its theoretical generality, parametrized verification is practically crucial in this setting: as our experiments show, it is much more efficient than verification of bounded-size instances (starting from a number of components of 3 or 4), especially concerning memory consumption (which is the critical resource).

Related Work. There have been a lot of works related to the analysis of programs running under WMM (e.g., [25, 22, 23, 17, 2, 15, 16, 13, 14, 29]). Some of these works propose precise analysis techniques for checking safety properties or stability of finite-state programs under WMM (e.g., [2, 13, 19, 4]). Others propose stateless model-checking techniques for programs under TSO and PSO (e.g., [1, 30, 18]). Different other techniques based on monitoring and testing have also been developed during these last years (e.g., [15, 16, 25]). There are also a number of efforts to design bounded model checking techniques for programs under WMM (e.g., [9, 29, 14]) which encode the verification problem in SAT/SMT.

The closest works to ours are those presented in [2, 11, 3, 12] which provide precise and sound techniques for checking safety properties for finite-state programs running under TSO. However, as stated in the introduction, these techniques are complicated and can not be extended, in a straightforward manner, to the verification of parameterized systems (as it is the case of the developed techniques for the dual TSO semantics).

In Section 7, we experimentally compare our techniques with Memorax [2, 3] which is the only precise and sound tool for checking safety properties for programs under TSO.

2 Preliminaries

Let Σ be a finite alphabet. We use Σ^* (resp. Σ^+) to denote the set of all *words* (resp. non-empty words) over Σ. Let ϵ be the empty word. The length of a word $w \in \Sigma^*$ is denoted by $|w|$ (and in particular $|\epsilon| = 0$). For every $i : 1 \leq i \leq |w|$, let $w(i)$ be the symbol at position i in w. For $a \in \Sigma$, we write $a \in w$ if a appears in w, i.e., $a = w(i)$ for some $i : 1 \leq i \leq |w|$.

Given two words u and v over Σ, we use $u \preceq v$ to denote that u is a (not necessarily contiguous) subword of v (i.e., if there is an injection $h : \{1, \ldots, |u|\} \mapsto \{1, \ldots, |v|\}$ such that: (1) $h(i) < h(j)$ for all $i < j$, and (2) for every $i \in \{1, \ldots, |u|\}$, we have $u(i) = v(h(i))$).

Given a subset $\Sigma' \subseteq \Sigma$ and a word $w \in \Sigma^*$, we use $w|_{\Sigma'}$ to denote the projection of w over Σ', i.e., the word obtained from w by erasing all the symbols that are not in Σ'.

Let A and B be two sets and let $f : A \mapsto B$ be a total function from A to B. We use $f[a \hookleftarrow b]$ to denote the function f' such that $f'(a) = b$ and $f'(a') = f(a')$ for all $a' \neq a$.

A transition system $\mathcal{T}$ is a tuple $\left(\mathtt{C}, \mathtt{Init}, \mathtt{Act}, \cup_{a \in \mathtt{Act}} \xrightarrow{a}\right)$ where $\mathtt{C}$ is a (potentially infinite) set of *configurations*, $\mathtt{Init} \subseteq \mathtt{C}$ is a set of *initial configurations*, $\mathtt{Act}$ is a set of actions, and for every $a \in \mathtt{Act}$, $\xrightarrow{a} \subseteq \mathtt{C} \times \mathtt{C}$ is a *transition relation*. We use $c \xrightarrow{a} c'$ to denote that $(c, c') \in \xrightarrow{a}$. Let $\rightarrow = \cup_{a \in \mathtt{Act}} \xrightarrow{a}$. A *run* π of $\mathcal{T}$ is of the form $c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} c_n$, where $c_i \xrightarrow{a_{i+1}} c_{i+1}$ for all $i : 0 \leq i < n$. Then, we write $c_0 \xrightarrow{\pi} c_n$. We use *target* (π) to denote the configuration c_n. The run π is said to be a *computation* if $c_0 \in \mathtt{Init}$. Two runs $\pi_1 = c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} c_m$ and $\pi_2 = c_{m+1} \xrightarrow{a_{m+2}} c_{m+2} \xrightarrow{a_{m+3}} \cdots \xrightarrow{a_n} c_n$ are *compatible* if $c_m = c_{m+1}$. Then, we write

$\pi_1 \bullet \pi_2$ to denote the run $\pi = c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} c_m \xrightarrow{a_{m+2}} c_{m+2} \xrightarrow{a_{m+3}} \cdots \xrightarrow{a_n} c_n$. For two configurations c and c', we use $c \xrightarrow{*} c'$ to denote that $c \xrightarrow{\pi} c'$ for some run π. A configuration c is said to be *reachable* in $\mathcal{T}$ if $c_0 \xrightarrow{*} c$ for some $c_0 \in \texttt{Init}$, and a set C of configurations is said to be *reachable* in $\mathcal{T}$ if some $c \in C$ is reachable in $\mathcal{T}$.

3 Concurrent Systems

In this section, we define the syntax we use for *concurrent programs*, a model for representing communicating concurrent processes. Communication between processes is performed through a shared memory that consists of a finite number of shared variables (over finite domains) to which all processes can read and write. Then we recall the classical TSO semantics including the transition system it induces and its reachability problem. Next, we introduce the *Dual TSO* semantics and its induced transition system. Finally, we state the equivalence between the two semantics (for a given concurrent program, we can reduce its reachability problem under the classical TSO to its reachability problem under Dual TSO and vice-versa).

3.1 Syntax

Let V be a finite data domain and X be a finite set of variables. We assume w.l.o.g. that V contains the value 0. Let $\Omega(X, V)$ be the smallest set of memory operations that contains (1) *"no operation"* nop, (2) *read operation* $\mathsf{r}(x, v)$, (3) *write operation* $\mathsf{w}(x, v)$, (4) *fence operation* fence, and (5) *atomic read-write operation* $\mathsf{arw}(x, v, v')$, where $x \in X$, and $v, v' \in V$.

A *concurrent system* is a tuple $\mathcal{P} = (A_1, A_2, \ldots, A_n)$ where for every $p : 1 \leq p \leq n$, A_p is a finite-state automaton describing the behavior of the process p. The automaton A_p is defined as a triple $\left(Q_p, q_p^{init}, \Delta_p\right)$ where Q_p is a finite set of *local states*, $q_p^{init} \in Q_p$ is the *initial* local state, and $\Delta_p \subseteq Q_p \times \Omega(X, V) \times Q_p$ is a finite set of *transitions*. We define $P = \{1, \ldots, n\}$ to be the set of process IDs, $Q := \cup_{p \in P} Q_p$ to be the set of all local states and $\Delta := \cup_{p \in P} \Delta_p$ to be the set of all transitions.

3.2 Classical TSO semantics

In the following, we recall the semantics of concurrent systems under the classical TSO model as formalized in [26, 27]. To do that, we define the set of configurations and the induced transition relation. Let $\mathcal{P} = (A_1, A_2, \ldots, A_n)$ be a concurrent system.

Configurations. A *TSO-configuration* c is a triple $(\mathbf{q}, \mathbf{b}, mem)$ where (1) $\mathbf{q} : P \mapsto Q$ is the *global state* of $\mathcal{P}$ mapping each process $p \in P$ to a local state in Q_p (i.e., $\mathbf{q}(p) \in Q_p$), (2) $\mathbf{b} : P \mapsto (X \times V)^*$ gives the content of the store buffer of each process, and (3) $mem : X \mapsto V$ defines the value of each shared variable. Observe that the store buffer of each process contains a sequence of write operations, where each write operation is defined by a pair, namely a variable x and a value v that is assigned to x. The initial TSO-configuration c_{init} is defined by the tuple $(\mathbf{q}_{init}, \mathbf{b}_{init}, mem_{init})$ where, for all $p \in P$ and $x \in X$, we have that $\mathbf{q}_{init}(p) = q_p^{init}$, $\mathbf{b}_{init}(p) = \epsilon$ and $mem_{init}(x) = 0$. In other words, each process is in its initial local state, all the buffers are empty, and all the variables in the shared memory are initialized to 0. We use $\mathtt{C}_{TSO}$ to denote the set of TSO-configurations.

Transition Relation. The transition relation $\rightarrow_{TSO}$ between TSO-configurations is given by a set of rules, described in Figure 1. Here we informally explain these rules. A nop transition $(q, \mathsf{nop}, q') \in \Delta_p$ changes only the local state of the process p from q to q'. A

$$\frac{t = (q, \mathsf{nop}, q') \qquad \mathbf{q}(p) = q}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}, mem)}\ \textsf{Nop}$$

$$\frac{t = (q, \mathsf{w}(x, v), q') \qquad \mathbf{q}(p) = q}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}\left[p \hookleftarrow (x, v) \cdot \mathbf{b}(p)\right], mem)}\ \textsf{Write}$$

$$\frac{t = \mathsf{update}_p}{(\mathbf{q}, \mathbf{b}\left[p \hookleftarrow \mathbf{b}(p) \cdot (x, v)\right], mem) \xrightarrow{t}_{TSO} (\mathbf{q}, \mathbf{b}, mem\left[x \hookleftarrow v\right])}\ \textsf{Update}$$

$$\frac{t = (q, \mathsf{r}(x, v), q') \qquad \mathbf{q}(p) = q \qquad \mathbf{b}(p)|_{\{x\} \times V} = (x, v) \cdot w}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}, mem)}\ \textsf{Read-Own-Write}$$

$$\frac{t = (q, \mathsf{r}(x, v), q') \qquad \mathbf{q}(p) = q \qquad \mathbf{b}(p)|_{\{x\} \times V} = \epsilon \qquad mem(x) = v}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}, mem)}\ \textsf{Read Memory}$$

$$\frac{t = (q, \mathsf{arw}(x, v, v'), q') \qquad \mathbf{q}(p) = q \qquad \mathbf{b}(p) = \epsilon \qquad mem(x) = v}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}, mem\left[x \hookleftarrow v'\right])}\ \textsf{ARW}$$

$$\frac{t = (q, \mathsf{fence}, q') \qquad \mathbf{q}(p) = q \qquad \mathbf{b}(p) = \epsilon}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{TSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}, mem)}\ \textsf{Fence}$$

Figure 1 The transition relation $\rightarrow_{TSO}$ under TSO. Here $p \in P$ and $t \in \Delta_p \cup \left\{\mathsf{update}_p\right\}$ where update_p is an operation that updates the memory using the oldest message in the buffer of process p.

write transition $(q, \mathsf{w}(x, v), q') \in \Delta_p$ adds the message (x, v) to the tail of the store buffer of the process p. A memory update transition update_p can be performed at any time by removing the oldest message in the store buffer of the process p and updating the memory accordingly. For a read transition $(q, \mathsf{r}(x, v), q') \in \Delta_p$, if the store buffer of the process p contains some write operations to x, then the read value v must correspond to the value of the most recent such a write operation. Otherwise the value v of x is fetched from the memory. A fence transition may be performed by a process p only if its store buffer is empty. Finally, an atomic read-write transition $(q, \mathsf{arw}(x, v, v'), q') \in \Delta_p$ can be performed by the process p only if its store buffer is empty. This operation checks then whether the value of x is v and changes it to v'.

We use $c \rightarrow_{TSO} c'$ to denote that $c \xrightarrow{t}_{TSO} c'$ for some $t \in \Delta \cup \Delta'$ where $\Delta' := \left\{\mathsf{update}_p |\ p \in P\right\}$. The transition system induced by $\mathcal{P}$ under the classical TSO semantics is then given by $\mathcal{T}_{TSO} = (\mathtt{C}_{TSO}, \{\mathtt{c}_{init}\}, \Delta \cup \Delta', \rightarrow_{TSO})$.

The TSO Reachability Problem. A global state $\mathbf{q}_{target}$ is said to be reachable in $\mathcal{T}_{TSO}$ iff there is a TSO-configuration c of the form $(\mathbf{q}_{target}, \mathbf{b}, mem)$, with $\mathbf{b}(p) = \epsilon$ for all $p \in P$, such that c is reachable in $\mathcal{T}_{TSO}$. The reachability problem for the concurrent system $\mathcal{P}$ under TSO asks, for a given global state $\mathbf{q}_{target}$, whether $\mathbf{q}_{target}$ is reachable in $\mathcal{T}_{TSO}$. Observe that, in the definition of the reachability problem, we require that the buffers of the configuration c must be empty instead of being arbitrary. This is only for sake of simplicity and does not constitute a restriction. Indeed, we can easily show that the "arbitrary buffer" reachability problem is reducible to the "empty buffer" reachability problem.

3.3 Dual TSO semantics

In this section, we define the Dual TSO semantics. The model has a perfect FIFO *load* buffer between the main memory and each process. The *load* buffer is used to store *potential read* operations that will be performed by the process. Each message in the load buffer of a process p is either a pair of the form (x, v) or a triple of the form (x, v, own) where $x \in X$ and $v \in V$. A message of the form (x, v) corresponds to the fact that x had the value v in the shared memory. While a message (x, v, own) corresponds to the fact that the process p has written the value v to x. A write operation $\mathsf{w}(x, v)$ of the process p immediately updates the shared memory and a message of the form (x, v, own) is then appended to the tail of the load buffer of p. Read propagation is then performed by non-deterministically choosing a variable (let's say x and its value is v in the shared memory) and appending the message (x, v) to the tail of the load buffer of p. This propagation operation speculates on a read operation of p on x that will be performed later on. Moreover, any message at the head of the load buffer can be removed at any time. A read operation $\mathsf{r}(x, v)$ of the process p can be executed if the message at the head of the load buffer (i.e., the oldest one) of p is of the form (x, v) and there is no pending message of the form (x, v', own). In the case that the load buffer contains a message belonging to p (i.e., of the form (x, v', own)), the read value must correspond to the value of the most recent message belonging to p (implicitly, this allows to simulate the Read-Own-Write transitions). A fence means that the load buffer of p must be empty before p can continue. Let $\mathcal{P} = (A_1, A_2, \ldots, A_n)$ be a concurrent system.

Configurations. A *DTSO-configuration* c is a triple $(\mathbf{q}, \mathbf{b}, mem)$ where $\mathbf{q} : P \mapsto Q$ is the *global state* of $\mathcal{P}$, $\mathbf{b} : P \mapsto ((X \times V) \cup (X \times V \times \{own\}))^*$ is the content of the load buffer, and $mem : X \mapsto V$ gives the value of each variable. The initial DTSO-configuration c^D_{init} is defined by $(\mathbf{q}_{init}, \mathbf{b}_{init}, mem_{init})$ where, for all $p \in P$ and $x \in X$, we have that $\mathbf{q}_{init}(p) = q^{init}_p$, $\mathbf{b}_{init}(p) = \epsilon$ and $mem_{init}(x) = 0$. We use C_{DTSO} to denote the set of DTSO-configurations.

Transition Relation. The transition relation $\rightarrow_{DTSO}$ induced by the Dual TSO semantics is given in Figure 2. This relation is induced by members of Δ and $\Delta_{aux} := \{\mathsf{propagate}^x_p, \mathsf{delete}_p \mid p \in P,\ x \in X\}$. $\mathsf{propagate}^x_p$ is an operation that speculates on a read operation of p over x that will be executed later. This is done by appending the message (x, v) to the tail of the load buffer of p where v is the current value of x in the shared memory. The operation delete_p removes the oldest message in the load buffer of process p. A write operation $\mathsf{w}(x, v)$ updates the memory and appends the message (x, v, own) to the tail of the load buffer. A read operation $\mathsf{r}(x, v)$ checks first if the load buffer contains a message of the form (x, v', own), and in that case the read value v should correspond to the value of the most recent message of that form. If there is no message on the variable x belonging to p in its load buffer then the value v of x is fetched from the message at the head of its load buffer.

We use $c \rightarrow_{DTSO} c'$ to denote that $c \xrightarrow{t}_{DTSO} c'$ for some $t \in \Delta \cup \Delta_{aux}$. The transition system induced by $\mathcal{P}$ under the Dual TSO semantics is then given by $\mathcal{T}_{DTSO} = (\mathsf{C}_{DTSO}, \{\mathsf{c}^{\mathsf{D}}_{init}\}, \Delta \cup \Delta_{\mathsf{aux}}, \rightarrow_{DTSO})$.

The Dual TSO Reachability Problem. The reachability problem for $\mathcal{P}$ under the Dual TSO semantics is defined in a similar manner to the case of TSO. A global state $\mathbf{q}_{target}$ is said to be reachable in $\mathcal{T}_{DTSO}$ iff there is a DTSO-configuration c of the form $(\mathbf{q}_{target}, \mathbf{b}, mem)$, with $\mathbf{b}(p) = \epsilon$ for all $p \in P$, such that c is reachable in $\mathcal{T}_{DTSO}$. Then, the reachability problem

$$\frac{t = (q, \mathsf{nop}, q') \qquad \mathbf{q}(p) = q}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{DTSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}, mem)} \quad \text{Nop}$$

$$\frac{t = (q, \mathsf{w}(x, v), q') \qquad \mathbf{q}(p) = q}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{DTSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}\left[p \hookleftarrow (x, v, own) \cdot \mathbf{b}(p)\right], mem\left[x \hookleftarrow v\right])} \quad \text{Write}$$

$$\frac{t = \mathsf{propagate}_p^x \qquad mem(x) = v}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{DTSO} (\mathbf{q}, \mathbf{b}\left[p \hookleftarrow (x, v) \cdot \mathbf{b}(p)\right], mem)} \quad \text{Propagate}$$

$$\frac{t = \mathsf{delete}_p \qquad |m| = 1}{(\mathbf{q}, \mathbf{b}\left[p \hookleftarrow \mathbf{b}(p) \cdot m\right], mem) \xrightarrow{t}_{DTSO} (\mathbf{q}, \mathbf{b}, mem)} \quad \text{Delete}$$

$$\frac{t = (q, \mathsf{r}(x, v), q') \qquad \mathbf{q}(p) = q \qquad \mathbf{b}(p)|_{\{x\} \times V \times \{own\}} = (x, v, own) \cdot w}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{DTSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}, mem)} \quad \text{Read-Own-Write}$$

$$\frac{t = (q, \mathsf{r}(x, v), q') \qquad \mathbf{q}(p) = q \qquad \mathbf{b}(p)|_{\{x\} \times V \times \{own\}} = \epsilon \qquad \mathbf{b}(p) = (x, v) \cdot w}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{DTSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}, mem)} \quad \text{Read from buffer}$$

$$\frac{t = (q, \mathsf{arw}(x, v, v'), q') \qquad \mathbf{q}(p) = q \qquad \mathbf{b}(p) = \epsilon \qquad mem(x) = v}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{DTSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}, mem\left[x \hookleftarrow v'\right])} \quad \text{ARW}$$

$$\frac{t = (q, \mathsf{fence}, q') \qquad \mathbf{q}(p) = q \qquad \mathbf{b}(p) = \epsilon}{(\mathbf{q}, \mathbf{b}, mem) \xrightarrow{t}_{DTSO} (\mathbf{q}\left[p \hookleftarrow q'\right], \mathbf{b}, mem)} \quad \text{Fence}$$

Figure 2 The induced transition relation $\rightarrow_{DTSO}$ under the Dual TSO semantics. Here $p \in P$ and $t \in \Delta_p \cup \Delta'_p$ where $\Delta'_p := \left\{\mathsf{propagate}_p^x, \mathsf{delete}_p \middle| \, x \in X\right\}$.

consists in checking whether $\mathbf{q}_{target}$ is reachable in $\mathcal{T}_{DTSO}$. The following theorem states equivalence of the reachability problems under the TSO and Dual TSO semantics.

▶ **Theorem 1.** *A global state $\mathbf{q}_{target}$ is reachable in $\mathcal{T}_{TSO}$ iff $\mathbf{q}_{target}$ is reachable in $\mathcal{T}_{DTSO}$.*

4 The Dual TSO Reachability Problem

In this section, we show the decidability of the Dual TSO reachability problem by making use of the framework of *Well-Structured Transition Systems* (WSTS) [5, 21]. First, we briefly recall the framework of WSTS and then we instantiate it to show the decidability of the Dual TSO reachability problem.

4.1 Well-Structured Transition Systems

Let $\mathcal{T} = \left(\mathtt{C}, \mathtt{Init}, \mathtt{Act}, \cup_{\mathtt{a} \in \mathtt{Act}} \xrightarrow{a}\right)$ be a transition system. Let $\sqsubseteq$ be a *well-quasi ordering* on $\mathtt{C}$. Recall that a well-quasi ordering on $\mathtt{C}$ is a binary relation over $\mathtt{C}$ that is reflexive and transitive and for every infinite sequence $(c_i)_{i \geq 0}$ of elements in $\mathtt{C}$ there exist $i, j \in \mathbb{N}$ such that $i < j$ and $c_i \sqsubseteq c_j$. A set $\mathtt{U} \subseteq \mathtt{C}$ is called *upward closed* if for every $c \in \mathtt{U}$ and $c' \in \mathtt{C}$ with $c \sqsubseteq c'$, we have $c' \in \mathtt{U}$. It is known that every upward closed set $\mathtt{U}$ can be characterised by a finite *minor set* $\mathtt{M} \subseteq \mathtt{U}$ such that: (i) for every $c \in \mathtt{U}$ there is $c' \in \mathtt{M}$ such that $c' \sqsubseteq c$, and (ii) if $c, c' \in \mathtt{M}$ and $c \sqsubseteq c'$ then $c = c'$. We use $\mathtt{min}$ to denote the function which for a given upward closed set $\mathtt{U}$ returns its minor set.

Let $\mathtt{D} \subseteq \mathtt{C}$. The upward closure of $\mathtt{D}$ is defined as $\mathtt{D} \uparrow := \{c' \in \mathtt{C} |\ \exists c \in \mathtt{D} \text{ with } c \sqsubseteq c'\}$. We also define the set of predecessors of $\mathtt{D}$ as $\mathtt{Pre}_{\mathcal{T}}(\mathtt{D}) := \{c |\ \exists c_1 \in \mathtt{D}, c \to c_1\}$. For a finite set of configurations $\mathtt{M} \subseteq \mathtt{C}$, we use *minpre* $(\mathtt{M})$ to denote $\mathtt{min}\,(\mathtt{Pre}_{\mathcal{T}}(\mathtt{M}\uparrow) \cup \mathtt{M}\uparrow)$.

The transition relation $\to$ is said to be *monotonic* wrt. $\sqsubseteq$ if, given $c_1, c_2, c_3 \in \mathtt{C}$ s.t. $c_1 \to c_2$ and $c_1 \sqsubseteq c_3$, we can compute a configuration $c_4 \in \mathtt{C}$ and a run π s.t. $c_3 \xrightarrow{\pi} c_4$ and $c_2 \sqsubseteq c_4$. The pair $(\mathcal{T}, \sqsubseteq)$ is called *monotonic transition system* if $\to$ is *monotonic* wrt. $\sqsubseteq$.

Given a *finite* set of configurations $\mathtt{M} \subseteq \mathtt{C}$, the *coverability problem* of $\mathtt{M}$ in the monotonic transition system $(\mathcal{T}, \sqsubseteq)$ asks whether the set $\mathtt{M}\uparrow$ is reachable in $\mathcal{T}$. For the decidability of this problem the following **three conditions** are sufficient: **(i)** For every two configurations c_1 and c_2, it is decidable whether $c_1 \sqsubseteq c_2$, **(ii)** for every $c \in \mathtt{C}$, we can check whether $\{c\}\uparrow \cap \mathtt{Init} \neq \emptyset$, and **(iii)** for every $c \in \mathtt{C}$, the set *minpre* (c) is finite and computable.

The solution for the coverability problem as suggested in [5, 21] is based on a backward analysis approach. It is shown that starting from a finite set $\mathtt{M}_0 \subseteq \mathtt{C}$, the sequence $(\mathtt{M}_i)_{i \geq 0}$ with $\mathtt{M}_{i+1} := \textit{minpre}\,(\mathtt{M}_i)$, for $i \geq 0$ reaches a fixpoint and is computable.

4.2 Dual TSO Transition System is a WSTS

In this section, we instantiate the framework of WSTS to show the following result:

▶ **Theorem 2.** *The Dual TSO reachability problem is decidable.*

The rest of this section is devoted to the proof of the above theorem. Let $\mathcal{P} = (A_1, A_2, \ldots, A_n)$ be a concurrent system (as defined in Section 3). Let $\mathcal{T}_{DTSO} = (\mathtt{C}_{DTSO}, \{\mathtt{c}^{\mathtt{D}}_{init}\}, \Delta \cup \Delta_{\mathtt{aux}}, \to_{DTSO})$ be the transition system induced by $\mathcal{P}$ under the Dual TSO semantics (as defined in Section 3.3). In the following, we will first define a well-quasi ordering $\sqsubseteq$ on the set of DTSO-configurations (Lemma 4) such that for every two configurations c_1 and c_2, it is decidable whether $c_1 \sqsubseteq c_2$. Then we show that the transition system induced under the Dual TSO semantics is monotonic wrt. to $\sqsubseteq$ (Lemma 5). We will show also that the Dual TSO reachability problem for $\mathcal{P}$ can be reduced to the coverability problem in the monotonic transition system $(\mathcal{T}_{DTSO}, \sqsubseteq)$ (Lemma 6). (Observe that this reduction is needed since we require that the load buffers are empty when defining the Dual TSO reachability problem.) The second sufficient condition (i.e., checking whether the upward closed set $\{c\}\uparrow$, with c is a DTSO-configuration, contains an initial configuration) for the decidability of the coverability problem is trivial. This check boils down to verifying whether c is an initial configuration. Finally, the computability of the set of minimal configurations for the set of predecessors of any upward closed set is stated by the following lemma:

▶ **Lemma 3.** *For any configuration c, we can compute minpre$(\{c\})$.*

Well-quasi Ordering. In the following, we define a well-quasi ordering $\sqsubseteq$ on $\mathtt{C}_{DTSO}$. Let us first introduce some notations and definitions. Consider a word
$w \in ((X \times V) \cup (X \times V \times \{own\}))^*$ representing the content of a load buffer. We define an operation that divides w into a number of fragments according to the most-recent own-messages concerning each variable. We define
$[w]_{own} := (w_1, (x_1, v_1, own), w_2, \ldots, w_m, (x_m, v_m, own), w_{m+1})$, where the following conditions are satisfied: (1) $x_i \neq x_j$ if $i \neq j$, (2) if $(x, v, own) \in w_i$ then $x = x_j$ for some $j < i$ (i.e., the most recent own-message on x_j occurs at position j), and (3) $w = w_1 \cdot (x_1, v_1, own) \cdot w_2 \cdots w_m \cdot (x_m, v_m, own) \cdot w_{m+1}$ (i.e., the fragments correspond to the given word w).

Let $w, w' \in ((X \times V) \cup (X \times V \times \{own\}))^*$ be two words. Let us assume that $[w]_{own} = (w_1, (x_1, v_1, own), w_2, \ldots, w_r, (x_r, v_r, own), w_{r+1})$, and $[w']_{own} = (w'_1, (x'_1, v'_1, own), w'_2, \ldots, w'_m, (x'_m, v'_m, own), w'_{m+1})$. We write $w \sqsubseteq w'$ to denote that the following conditions are satisfied: (i) $r = m$, (ii) $x'_i = x_i$ and $v'_i = v_i$ for all $i : 1 \leq i \leq m$, and (iii) $w_i \preceq w'_i$ for all $i : 1 \leq i \leq m+1$.

Consider two DTSO-configurations $c = (\mathbf{q}, \mathbf{b}, mem)$ and $c' = (\mathbf{q}', \mathbf{b}', mem')$, we extend the ordering $\sqsubseteq$ to configurations as follows: $c \sqsubseteq c'$ iff the following conditions are satisfied: (i) $\mathbf{q} = \mathbf{q}'$, (ii) $\mathbf{b}(p) \sqsubseteq \mathbf{b}'(p)$ for all process $p \in P$, and (iii) $mem' = mem$. The following lemma shows that $\sqsubseteq$ is indeed a well-quasi ordering.

▶ **Lemma 4.** *The relation $\sqsubseteq$ is a well-quasi ordering over $\mathtt{C}_{DTSO}$. Furthermore, for every two DTSO-configurations c_1 and c_2, it is decidable whether $c_1 \sqsubseteq c_2$.*

Monotonicity. Given configurations $c_1 = (\mathbf{q}_1, \mathbf{b}_1, mem_1), c_2 = (\mathbf{q}_2, \mathbf{b}_2, mem_2), c_3 = (\mathbf{q}_3, \mathbf{b}_3, mem_3) \in \mathtt{C}_{DTSO}$ such that $c_1 \xrightarrow{t}_{DTSO} c_2$ for some transition $t \in \Delta_p \cup \{\mathsf{propagate}_p^x, \mathsf{delete}_p |\ x \in X\}$, with $p \in P$, and $c_1 \sqsubseteq c_3$, we will show that it is possible to compute a configuration $c_4 \in \mathtt{C}_{DTSO}$ and a run π such that $c_3 \xrightarrow{\pi}_{DTSO} c_4$ and $c_2 \sqsubseteq c_4$. To that aim, we first show that it is possible from c_3 to reach a configuration c'_3, by performing a certain number of delete_p operations, such that the process p will have the same last message in its load buffer in the configurations c_1 and c'_3 while $c_1 \sqsubseteq c'_3$. Then, from the configuration c'_3, the process p can perform the same transition t as c_1 did in order to reach the configuration c_4 such that $c_2 \sqsubseteq c_4$. Let us assume that $[\mathbf{b}_1(p)]_{own}$ is of the form $\langle w_1, (x_1, v_1, own), w_2, \ldots, w_m, (x_m, v_m, own), w_{m+1}\rangle$, and $[\mathbf{b}_3(p)]_{own}$ is of the form $\langle w'_1, (x'_1, v'_1, own), w'_2, \ldots, w'_m, (x'_m, v'_m, own), w'_{m+1}\rangle$. We define the word $w \in ((X \times V) \cup (X \times V \times \{own\}))^*$ to be the longest word such that $w'_{m+1} = w' \cdot w$ with $w_{m+1} \preceq w'$. Observe that in this case we have either $w_{m+1} = w' = \epsilon$ or $w'(|w'|) = w_{m+1}(|w_{m+1}|)$. Then, after executing a certain number $|w|$ of delete_p operations from the configuration c_3, one can obtain a configuration $c'_3 = (\mathbf{q}_3, \mathbf{b}'_3, mem_3)$ such that $\mathbf{b}_3 = \mathbf{b}'_3\, [p \hookleftarrow \mathbf{b}'_3(p) \cdot w]$. As a consequence, we have $c_1 \sqsubseteq c'_3$. Furthermore, since c_1 and c'_3 have the same global state, the same memory valuation, the same sequence of most-recent own messages concerning each variable, and the same last message in the load buffer of p, c'_3 can perform the transition t and reaches a configuration c_4 such that $c_2 \sqsubseteq c_4$. The following lemma shows that $(\mathcal{T}_{DTSO}, \sqsubseteq)$ is monotonic system.

▶ **Lemma 5.** *The relation $\rightarrow_{DTSO}$ is monotonic wrt. $\sqsubseteq$.*

From Reachability to Coverability. Let $\mathbf{q}_{target}$ be a global state of $\mathcal{P}$ and $\mathtt{M}_{target}$ be the set of DTSO-configurations of the form $c = (\mathbf{q}_{target}, \mathbf{b}, mem)$ with $\mathbf{b}(p) = \epsilon$ for all $p \in P$. Next, we show that the reachability problem of $\mathbf{q}_{target}$ in $\mathcal{T}_{DTSO}$ can be reduced to the coverability problem of $\mathtt{M}_{target}$ in $(\mathcal{T}_{DTSO}, \sqsubseteq)$. Recall that $\mathbf{q}_{target}$ in $\mathcal{T}_{DTSO}$ iff $\mathtt{M}_{target}$ is reachable in $\mathcal{T}_{DTSO}$. Let us assume that $\mathtt{M}_{target}\!\uparrow$ is reachable in $\mathcal{T}_{DTSO}$. This means that there is a configuration $c \in \mathtt{M}_{target}\!\uparrow$ which is reachable in $\mathcal{T}_{DTSO}$. Let us assume that c is of the form $(\mathbf{q}_{target}, \mathbf{b}, mem)$. Then, from the configuration c, it is possible to reach the configuration $c' = (\mathbf{q}_{target}, \mathbf{b}', mem)$, with $\mathbf{b}'(p) = \epsilon$ for all $p \in P$, by performing a sequence of delete_p operations to empty the load buffer of each process. It is then easy to see that $c' \in \mathtt{M}_{target}$ and so $\mathtt{M}_{target}$ is reachable in $\mathcal{T}_{DTSO}$. The other direction of the following lemma is trivial since $\mathtt{M}_{target} \subseteq \mathtt{M}_{target}\!\uparrow$.

▶ **Lemma 6.** *$\mathtt{M}_{target}\!\uparrow$ is reachable in $\mathcal{T}_{DTSO}$ iff $\mathtt{M}_{target}$ is reachable in $\mathcal{T}_{DTSO}$.*

5 Parameterized Concurrent Systems

Let V be a finite data domain and X be a finite set of variables ranging over V. A *parameterized concurrent system* (or simply a *parameterized system*) consists of an unbounded number of identical processes running under the Dual TSO semantics. Formally, a parameterized system $\mathcal{S}$ is defined by a finite-state automaton $A = \left(Q, q^{init}, \Delta\right)$ uniformly describing the behavior of each process. An *instance* of $\mathcal{S}$ is a concurrent system $\mathcal{P} = (A_1, A_2, \ldots, A_n)$, for some $n \in \mathbb{N}$, where for every $p : 1 \leq p \leq n$, we have $A_p = A$. In other words, it consists of a finite set of processes each running the same code defined by A. Observe that considering that all processes run the same code is not a real restriction. In fact, the case where the processes run (finitely many) different finite-state automata $A_1, A_2, \ldots, A_m$ can be easily encoded in our model by constructing an extended automaton A which represents the *union* of these automata $A_1, A_2, \ldots, A_m$. We use $\mathit{Inst}(\mathcal{S})$ to denote all possible instances of $\mathcal{S}$. We use $\mathcal{T}_{\mathcal{P}} = (\mathtt{C}_{\mathcal{P}}, \mathtt{Init}_{\mathcal{P}}, \mathtt{Act}_{\mathcal{P}}, \rightarrow_{\mathcal{P}})$ to denote the transition system induced by an instance $\mathcal{P}$ of $\mathcal{S}$ under the Dual TSO semantics.

A parameterized configuration α is a pair (P, c) where $P = \{1, \ldots, n\}$, with $n \in \mathbb{N}$, is the set of process IDs and c is a DTSO-configuration of an instance $\mathcal{P}$ of $\mathcal{S}$ of the form $(A_1, A_2, \ldots, A_n)$. The parameterized configuration $\alpha = (P, c)$ is said to be initial if c is an initial configuration of $\mathcal{P}$ (i.e., $c \in \mathtt{Init}_{\mathcal{P}}$). We use $\mathtt{C}$ (resp. $\mathtt{Init}$) to denote the set of all the parameterized configurations (resp. initial configurations) of $\mathcal{S}$.

Let $\mathtt{Act}$ denote the set of actions of all possible instances of $\mathcal{S}$ (i.e., $\mathtt{Act} = \cup_{\mathcal{P} \in \mathit{Inst}(\mathcal{S})} \mathtt{Act}_{\mathcal{P}}$). We define a transition relation $\rightarrow$ on parameterized configurations such that $(P, c) \xrightarrow{t} (P', c')$ for some action $t \in \mathtt{Act}$ iff $P' = P$ and there is an instance $\mathcal{P}$ of $\mathcal{S}$ such that $t \in \mathtt{Act}_{\mathcal{P}}$ and $c \rightarrow_{\mathcal{P}} c'$. The transition system induced by $\mathcal{S}$ is given by $\mathcal{T} = (\mathtt{C}, \mathtt{Init}, \mathtt{Act}, \rightarrow)$.

In the following we extend the definition of the Dual TSO reachability problem to the case of parameterized systems. A global state $\mathbf{q}_{target} : P' \mapsto Q$ is said to be reachable in $\mathcal{T}$ if there exists a parameterized configuration $\alpha = (P, (\mathbf{q}, \mathbf{b}, \mathit{mem}))$, with $\mathbf{b}(p) = \epsilon$ for all $p \in P$, such that α is reachable in $\mathcal{T}$ and $\mathbf{q}_{target}(1) \cdots \mathbf{q}_{target}(|P'|) \preceq \mathbf{q}(1) \cdots \mathbf{q}(|P|)$. Then, the reachability problem consists in checking whether $\mathbf{q}_{target}$ is reachable in $\mathcal{T}$. In other words, the Dual TSO reachability problem for parameterized systems asks whether there is an instance of the parameterized system that reaches a configuration with a number of processes in certain given local states.

6 Decidability of the Parameterized Verification Problem

We prove hereafter the following fact:

▶ **Theorem 7.** *The Dual TSO reachability problem for parameterized systems is decidable.*

Let $\mathcal{S} = \left(Q, q^{init}, \Delta\right)$ be a parameterized system and $(\mathtt{C}, \mathtt{Init}, \mathtt{Act}, \rightarrow)$ be its induced transition system. The proof of Theorem 7 is done by instantiating the framework of WSTS. Following that framework, we will first define an ordering that we denote by $\trianglelefteq$ on the set of parameterized configurations and show the monotonicity of the the relation $\rightarrow$ wrt. this ordering (see Lemma 9 and Lemma 10). Then we will show that the Dual TSO reachability problem for $\mathcal{S}$ can be reduced to the coverability problem in the monotonic transition system $(\mathcal{T}, \trianglelefteq)$ (Lemma 11). The second sufficient condition (i.e., checking whether the upward closed set $\{\alpha\}\uparrow$, with α is a parameterized configuration, contains an initial configuration) for the decidability of the coverability problem is trivial. This check boils down to whether the configuration α is initial. Finally, the last sufficient condition (i.e., computing the set of minimal configurations for the set of predecessors of any upward closed set) for the decidability of the coverability problem is stated by the following lemma:

▶ **Lemma 8.** *For any parameterized configuration α, we can compute $minpre(\{\alpha\})$.*

Well-quasi Ordering. Let $\alpha = (P, (\mathbf{q}, \mathbf{b}, mem))$ and $\alpha' = (P', (\mathbf{q}', \mathbf{b}', mem'))$ be two parameterized configurations. We define the ordering $\trianglelefteq$ on the set of parameterized configurations as follows: $\alpha \trianglelefteq \alpha'$ iff the following conditions are satisfied: (1) $mem = mem'$ and (2) there is an injection $h : \{1, \ldots, |P|\} \mapsto \{1, \ldots, |P'|\}$ such that (i) $p < p'$ implies $h(p) < h(p')$, and (ii) for every $p \in \{1, \ldots, |P|\}$, $\mathbf{q}(p) = \mathbf{q}'(h(p))$ and $\mathbf{b}(p) \sqsubseteq \mathbf{b}'(h(p))$. The following lemma states that $\trianglelefteq$ is indeed a well-quasi ordering.

▶ **Lemma 9.** *The relation $\trianglelefteq$ is a well-quasi ordering over* C. *Furthermore, for every two parameterized configurations α and α', it is decidable whether $\alpha \trianglelefteq \alpha'$.*

Monotonicity. Let $\alpha_1 = (P, (\mathbf{q}_1, \mathbf{b}_1, mem_1))$, $\alpha_2 = (P, (\mathbf{q}_2, \mathbf{b}_2, mem_2))$ and $\alpha_3 = (P', (\mathbf{q}_3, \mathbf{b}_3, mem_3))$ be parameterized configurations. Furthermore, we assume that $\alpha_1 \trianglelefteq \alpha_3$ and $\alpha_1 \xrightarrow{t} \alpha_2$ for some transition t. Since $\alpha_1 \trianglelefteq \alpha_3$, there is an injection function $h : \{1, \ldots, |P|\} \mapsto \{1, \ldots, |P'|\}$ such that (i) $p < p'$ implies $h(p) < h(p')$, and (ii) for every $p \in \{1, \ldots, |P|\}$, $\mathbf{q}_1(p) = \mathbf{q}_3(h(p))$ and $\mathbf{b}_1(p) \sqsubseteq \mathbf{b}_3(h(p))$. We define the parameterized configuration α' from α_3 by only keeping the local state and load buffers of processes in $h(P)$. Formally, $\alpha' = (P, (\mathbf{q}', \mathbf{b}', mem'))$ is defined as follows: (i) $mem' = mem_3$ and (ii) for every $p \in \{1, \ldots, |P|\}$, $\mathbf{q}'(p) = \mathbf{q}_3(h(p))$ and $\mathbf{b}'(p) = \mathbf{b}_3(h(p))$. (Observe that $(\mathbf{q}_1, \mathbf{b}_1, mem_1) \sqsubseteq (\mathbf{q}', \mathbf{b}', mem')$). Since the relation $\rightarrow_{DTSO}$ is monotonic wrt. the ordering $\sqsubseteq$ (see Lemma 5), there is a Dual TSO-configuration $(\mathbf{q}'', \mathbf{b}'', mem'')$ such that $(\mathbf{q}', \mathbf{b}', mem') \rightarrow^*_{DTSO} (\mathbf{q}'', \mathbf{b}'', mem'')$ and $(\mathbf{q}_2, \mathbf{b}_2, mem_2) \sqsubseteq (\mathbf{q}'', \mathbf{b}'', mem'')$. Consider now the parameterized configuration $\alpha_4 = (P', (\mathbf{q}_4, \mathbf{b}_4, mem_4))$ such that $mem'' = mem_4$, (ii) for every $p \in \{1, \ldots, |P|\}$, $\mathbf{q}''(p) = \mathbf{q}_4(h(p))$ and $\mathbf{b}''(p) = \mathbf{b}_4(h(p))$, and (iii) for $p \in (\{1, \ldots, |P'|\} \setminus \{h(1), \ldots, h(|P|)\})$, we have $\mathbf{q}_4(p) = \mathbf{q}_3(p)$ and $\mathbf{b}_4(p) = \mathbf{b}_3(p)$. It is easy then to see that $\alpha_2 \trianglelefteq \alpha_4$ and $\alpha_3 \rightarrow^* \alpha_4$. Hence, we obtain the following result:

▶ **Lemma 10.** *The relation $\rightarrow$ is monotonic wrt. $\trianglelefteq$.*

From Reachability to Coverability. Let $\mathbf{q}_{target} : P' \mapsto Q$ be a global state. Let $\mathtt{M}_{target}$ be the set of parameterized configurations of the form $\alpha = (P', (\mathbf{q}_{target}, \mathbf{b}, mem))$ with $\mathbf{b}(p) = \epsilon$ for all $p \in P'$. In the following, we show that $\mathtt{M}_{target}\uparrow$ is reachable in $\mathcal{T}$ iff there is a parameterized configuration $\alpha = (P, (\mathbf{q}, \mathbf{b}, mem))$, with $\mathbf{b}(p) = \epsilon$ for all $p \in P$, such that α is reachable in $\mathcal{T}$ and $\mathbf{q}_{target}(1) \cdots \mathbf{q}_{target}(|P'|) \preceq \mathbf{q}(1) \cdots \mathbf{q}(|P|)$.

Let us assume that there is a parameterized configuration $\alpha = (P, (\mathbf{q}, \mathbf{b}, mem))$, with $\mathbf{b}(p) = \epsilon$ for all $p \in P$, such that α is reachable in $\mathcal{T}$ and $\mathbf{q}_{target}(1) \cdots \mathbf{q}_{target}(|P'|) \preceq \mathbf{q}(1) \cdots \mathbf{q}(|P|)$. It is then easy to show that $\alpha \in \mathtt{M}_{target}\uparrow$.

Now let us assume that there is $\alpha' = (P'', (\mathbf{q}', \mathbf{b}', mem')) \in \mathtt{M}_{target}\uparrow$ which is reachable in $\mathcal{T}$. From the configuration α', it is possible to reach the configuration $\alpha'' = (P'', (\mathbf{q}', \mathbf{b}'', mem'))$, with $\mathbf{b}''(p) = \epsilon$ for all $p \in P''$, by performing a sequence of delete_p operations to empty the load buffer of each process. Since $\alpha' \in \mathtt{M}_{target}\uparrow$, we have $\mathbf{q}_{target}(1) \cdots \mathbf{q}_{target}(|P'|) \preceq \mathbf{q}'(1) \cdots \mathbf{q}'(|P''|)$. Hence, α'' is a witness of the state reachability problem.

▶ **Lemma 11.** *$\mathbf{q}_{target}$ is reachable in $\mathcal{T}$ iff $\mathtt{M}_{target}\uparrow$ is reachable in $\mathcal{T}$.*

Table 1 Comparison between Dual-TSO and Memorax: The columns #P, #T and #C give the number of processes, the running time in seconds and the number of generated configurations, respectively. If a tool runs out of time, we put TO in the #T column and • in the #C column.

Program	#P	Dual-TSO		Memorax	
		#T	#C	#T	#C
SB	5	0.3	10641	559.7	10515914
LB	3	0.0	2048	71.4	1499475
WRC	4	0.0	1507	63.3	1398393
ISA2	3	0.0	509	21.1	226519
RWC	5	0.1	4277	61.5	1196988
W+RWC	4	0.0	1713	83.6	1389009
IRIW	4	0.0	520	34.4	358057
Nbw_w_wr	2	0.0	222	10.7	200844
Sense_rev_bar	2	0.1	1704	0.8	20577
Dekker	2	0.1	5053	1.1	19788
Dekker_simple	2	0.0	98	0.0	595
Peterson	2	0.1	5442	5.2	90301
Peterson_loop	2	0.2	7632	5.6	100082
Szymanski	2	0.6	29018	1.0	26003
MP	4	0.0	883	TO	•
Ticket_spin_lock	3	0.9	18963	TO	•
Bakery	2	2.6	82050	TO	•
Dijkstra	2	0.2	8324	TO	•
Lamport_fast	3	17.7	292543	TO	•
Burns	4	124.3	2762578	TO	•

7 Experimental Results

We have implemented our techniques described in Section 4 and Section 6 in an open-source tool called Dual-TSO[1]. The tool checks the state reachability problems for (parameterized) concurrent systems under the Dual TSO semantics. We compare our tool with Memorax [2, 3] which is the *only precise and sound tool* for deciding the state reachability problem of concurrent systems under TSO. Observe that Memorax cannot handle parameterized verification. All experiments are performed on an Intel x86-32 Core2 2.4 Ghz machine and 4GB of RAM.

In the following, we present two sets of results. The first set concerns the comparison of Dual-TSO with Memorax (see Table 1). The second set shows the benefit of the parameterized verification compared to the use of the state reachability when increasing the number of processes (see Figure 3 and Table 2). Our examples are from [2, 10, 13, 4, 25]. In all experiments, we set up the time out to 600 seconds.

Table 1 presents a comparison between Dual-TSO and Memorax on a representative sample of 20 benchmarks. In all these examples, Dual-TSO and Memorax return the same result for the state reachability problem (except 6 examples where Memorax runs out of time). In the examples where the two tools return, Dual-TSO out-performs Memorax and generates fewer configurations (and so uses less memory). Indeed, Dual-TSO is 600 times faster than

[1] https://www.it.uu.se/katalog/tuang296/dual-tso

Table 2 Parameterized verification with Dual-TSO.

Program	Dual-TSO	
	#T	#C
SB	0.0	147
LB	0.6	1028
MP	0.0	149
WRC	0.8	618
ISA2	4.3	1539
RWC	0.2	293
W+RWC	1.5	828
IRIW	4.6	648

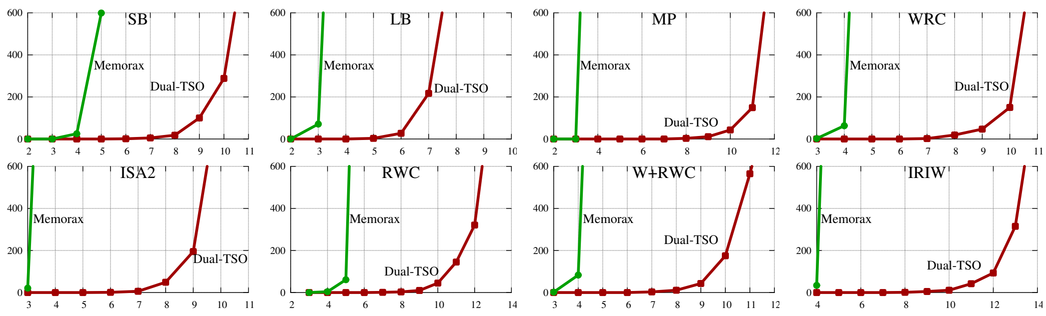

Figure 3 Running time of Memorax and Dual-TSO by increasing number of processes. The x axis is number of processes, the y axis is running time in seconds.

Memorax and generates 277 times fewer configurations on average.

The second set compares the scalability of Memorax and Dual-TSO while increasing the number of processes. The results are given in Fig. 3. We observe that Dual-TSO scales better than Memorax in all these examples. In fact, Memorax can only handle the examples with at most 5 processes. Table 2 presents the running time and the number of generated configurations when checking the state reachability problem for the parameterized version of these examples. We observe that the verification of these parameterized systems is much more efficient than verification of bounded-size instances (starting from a number of processes of 3 or 4), especially concerning memory consumption (which is given in terms of number of generated configurations). The reason behind is that the size of the generated minor sets in the analysis of a parameterized system is usually smaller than the size of the generated configurations during the analysis of an instance of the system with a large number of processes.

8 Conclusion

In this paper, we have presented an alternative (yet equivalent) semantics to the classical one for the TSO model. This new semantics allows us to understand the TSO model in a totally different way compared to the classical semantics. Furthermore, the proposed semantics offers several important advantages from the point of view of formal reasoning and program verification. First, the dual semantics allows transforming the load buffers to *lossy* channels without adding the costly overhead that was necessary in the case of store buffers. This means that we can apply the theory of *well-structured systems* [6, 5, 21] in a straightforward manner leading to a much simpler proof of decidability of safety properties. Second, the absence of extra overhead means that we obtain more efficient algorithms and better scalability (as shown by our experimental results). Finally, the dual semantics allows extending the framework to perform *parameterized verification* which is an important paradigm in concurrent program verification.

In the future, we plan to apply our techniques to more memory models and to combine with predicate abstraction for handling programs with unbounded data domain.

References

1 P. Abdulla, S. Aronis, M.F. Atig, B. Jonsson, C. Leonardsson, and K. Sagonas. Stateless model checking for TSO and PSO. In *TACAS*, volume 9035 of *LNCS*, pages 353–367. Springer, 2015.

2 P.A. Abdulla, M.F. Atig, Y.F. Chen, C. Leonardsson, and A. Rezine. Counter-example guided fence insertion under TSO. In *TACAS 2012*, pages 204–219, 2012.

3 P.A. Abdulla, M.F. Atig, Y.F. Chen, C. Leonardsson, and A. Rezine. Memorax, a precise and sound tool for automatic fence insertion under TSO. In *TACAS*, pages 530–536, 2013.

4 P.A. Abdulla, M.F. Atig, and N.T. Phong. The best of both worlds: Trading efficiency and optimality in fence insertion for TSO. In *ESOP 2015*, pages 308–332, 2015.

5 P.A. Abdulla, K. Cerans, B. Jonsson, and Y.K. Tsay. General decidability theorems for infinite-state systems. In *LICS'96*, pages 313–321. IEEE Computer Society, 1996.

6 Parosh Aziz Abdulla. Well (and better) quasi-ordered transition systems. *Bulletin of Symbolic Logic*, 16(4):457–515, 2010.

7 S. Adve and K. Gharachorloo. Shared memory consistency models: a tutorial. *Computer*, 29(12), 1996.

8 S. Adve and M. D. Hill. Weak ordering - a new definition. In *ISCA*, 1990.

9 J. Alglave, D. Kroening, and M. Tautschnig. Partial orders for efficient bounded model checking of concurrent software. In *CAV*, volume 8044 of *LNCS*, pages 141–157, 2013.

10 Jade Alglave, Luc Maranget, and Michael Tautschnig. Herding cats: Modelling, simulation, testing, and data mining for weak memory. *ACM TOPLAS*, 36(2):7:1–7:74, 2014.

11 M. F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. On the verification problem for weak memory models. In *POPL*, 2010.

12 M.F. Atig, A. Bouajjani, S. Burckhardt, and M. Musuvathi. What's decidable about weak memory models? In *ESOP*, volume 7211 of *LNCS*, pages 26–46. Springer, 2012.

13 Ahmed Bouajjani, Egor Derevenetc, and Roland Meyer. Checking and enforcing robustness against TSO. In *ESOP*, volume 7792 of *LNCS*, pages 533–553. Springer, 2013.

14 S. Burckhardt, R. Alur, and M. M. K. Martin. CheckFence: checking consistency of concurrent data types on relaxed memory models. In *PLDI*, pages 12–21. ACM, 2007.

15 Sebastian Burckhardt and Madanlal Musuvathi. Effective program verification for relaxed memory models. In *CAV*, volume 5123 of *LNCS*, pages 107–120. Springer, 2008.

16 Jacob Burnim, Koushik Sen, and Christos Stergiou. Testing concurrent programs on relaxed memory models. In *ISSTA*, pages 122–132. ACM, 2011.

17 A. Marian Dan, Y. Meshman, M. T. Vechev, and E. Yahav. Predicate abstraction for relaxed memory models. In *SAS*, volume 7935 of *LNCS*, pages 84–104. Springer, 2013.

18 Brian Demsky and Patrick Lam. Satcheck: Sat-directed stateless model checking for SC and TSO. In *OOPSLA 2015*, pages 20–36. ACM, 2015.

19 Egor Derevenetc and Roland Meyer. Robustness against Power is PSpace-complete. In *ICALP (2)*, volume 8573 of *LNCS*, pages 158–170. Springer, 2014.

20 M. Dubois, C. Scheurich, and F. A. Briggs. Memory access buffering in multiprocessors. In *ISCA*, 1986.

21 A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
22 Michael Kuperstein, Martin T. Vechev, and Eran Yahav. Automatic inference of memory fences. In *FMCAD*, pages 111–119. IEEE, 2010.
23 Michael Kuperstein, Martin T. Vechev, and Eran Yahav. Partial-coherence abstractions for relaxed memory models. In *PLDI*, pages 187–198. ACM, 2011.
24 L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Comp.*, C-28(9), 1979.
25 Feng Liu, Nayden Nedev, Nedyalko Prisadnikov, Martin T. Vechev, and Eran Yahav. Dynamic synthesis for relaxed memory models. In *PLDI '12*, pages 429–440, 2012.
26 S. Owens, S. Sarkar, and P. Sewell. A better x86 memory model: x86-tso. In *TPHOL*, 2009.
27 P. Sewell, S. Sarkar, S. Owens, F. Z. Nardelli, and M. O. Myreen. x86-tso: A rigorous and usable programmer's model for x86 multiprocessors. *CACM*, 53, 2010.
28 D. Weaver and T. Germond, editors. *The SPARC Architecture Manual Version 9*. PTR Prentice Hall, 1994.
29 Y. Yang, G. Gopalakrishnan, G. Lindstrom, and K. Slind. Nemos: A framework for axiomatic and executable specifications of memory consistency models. In *IPDPS*. IEEE, 2004.
30 N. Zhang, M. Kusano, and C. Wang. Dynamic partial order reduction for relaxed memory models. In *PLDI*, pages 250–259. ACM, 2015.