Gradual Consistency Checking

Rachid Zennou[1,2(✉)], Ahmed Bouajjani[1], Constantin Enea[1], and Mohammed Erradi[2]

[1] Université de Paris, IRIF, CNRS, 75013 Paris, France
rachid.zennou@gmail.com, {abou,cenea}@irif.fr
[2] ENSIAS, University Mohammed V, Rabat, Morocco
mohamed.erradi@gmail.com

Abstract. We address the problem of checking that computations of a shared memory implementation (with write and read operations) adheres to some given consistency model. It is known that checking conformance to Sequential Consistency (SC) for a given computation is NP-hard, and the same holds for checking Total Store Order (TSO) conformance. This poses a serious issue for the design of scalable verification or testing techniques for these important memory models. In this paper, we tackle this issue by providing an approach that avoids hitting systematically the worst-case complexity. The idea is to consider, as an intermediary step, the problem of checking weaker criteria that are as strong as possible while they are still checkable in polynomial time (in the size of the computation). The criteria we consider are new variations of causal consistency suitably defined for our purpose. The advantage of our approach is that in many cases (1) it can catch violations of SC/TSO early using these weaker criteria that are efficiently checkable, and (2) when a computation is causally consistent (according to our newly defined criteria), the work done for establishing this fact simplifies significantly the work required for checking SC/TSO conformance. We have implemented our algorithms and carried out several experiments on realistic cache-coherence protocols showing the efficiency of our approach.

1 Introduction

This paper addresses the problem of checking whether a given implementation of a shared memory offers the expected consistency guarantees to its clients which are concurrent programs composed of several threads running in parallel. Indeed, users of a memory need to see it as an abstract object allowing to perform concurrent reads and writes over a set of variables, which conform to some *memory model* defining the valid visible sequences of such operations. Various memory models can be considered in this context. Sequential Consistency (SC) [24] is the model where operations can be seen as atomic, executing according to some

This work is supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 678177).

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11562, pp. 267–285, 2019.
https://doi.org/10.1007/978-3-030-25543-5_16

interleaving of the operations issued by the different threads, while preserving the order in which these operations were issued by each of the threads. This fundamental model offers strong consistency in the sense that for each write operation, when it is issued by a thread, it is immediately visible to all the other threads. Other weaker memory models are adopted in order to meet performance and/or availability requirements in concurrent/distributed systems. One of the most widely used models in this context is Total Store Order (TSO) [29]. In this model, writes can be delayed, which means that after a write is issued, it is not immediately visible to all threads (except for the thread that issued it), and it is committed later after some arbitrary delay. However, writes issued by the same thread are committed in the same order are they were issued, and when a write is committed it becomes visible to all the other threads simultaneously. TSO is implemented in hardware but also in a distributed context over a network [22].

Implementing shared memories that are both highly performant and correct with respect to a given memory model is an extremely hard and error prone task. Therefore, checking that a given implementation is indeed correct from this point of view is of paramount importance. In this paper we address the issue of checking that a given execution of a shared memory implementation is consistent, and we consider as consistency criteria the cases of SC and TSO.

Checking SC or TSO conformance is known to be NP-complete [18,21]. This is due to the fact that in order to justify that the execution is consistent, one has to find a total order between the writes which explains the read operations happening along the computation. It can be proved that one cannot avoid enumerating all the possible total orders between writes, in the worst case. The situation is different for other weaker criteria such as Causal Consistency (CC) and its different variations, which have been shown to be checkable in polynomial time (in the the size of the computation) [6]. In fact, CC imposes fewer constraints than SC/TSO on the order between writes, and the way it imposes these constraints is "deterministic", in the sense that they can be derived from the history of the execution by applying a least fixpoint computation (which can be encoded for instance, as a standard DATALOG program). All these complexity results hold under the assumption that each value is written at most once, which is without loss of generality for implementations which are data-independent [31], i.e., their behavior doesn't depend on the concrete values read or written in the program. Indeed, any buggy behavior of such implementations can be exposed in executions satisfying this assumption [1].

The intrinsic hardness of the problem of checking SC/TSO poses a crucial issue for the design of scalable verification or testing techniques for these important consistency models. Tackling this issue requires the development of practical approaches that can work well (with polynomial complexity) when the instance of the problem does not need to generate the worst case (exponential) complexity.

[1] All the CC variations become NP-complete without the assumption that each value is written at most once [6]. This holds for the variations of CC we introduce in this paper as well.

The purpose of this paper is to propose such an approach. The idea is to reduce the amount of "nondeterminism" in searching for the write orders in order to establish SC/TSO conformance. For that, our approach for SC is to consider a weaker consistency model called CCM (for Convergent Causal Memory), that is "as strong as possible" while being polynomial time checkable. In fact CCM is stronger than both causal memory [2,26] (CM) and causal convergence [7] (CCv), two other well-known variations of causal consistency. Then, if CCM is already violated by the given computation then we can conclude that the computation does not satisfy the stronger criterion SC. Here the hope is that in practice many computations violating SC can be caught already at this stage using a polynomial time check. Now, in the case that the computation does not violate CCM, we exploit the fact that establishing CCM already imposes a set of constraints on the order between writes. We show that these constraints form a partial order which *must* be a subset of any total write order that would witness for SC conformance. Therefore, at this point, it is enough to find an extension of this partial write order, and the hope is that in many practical cases, this set of constraints is already large enough, letting only a small number of pairs of writes to be ordered in order to check SC conformance. For the case of TSO, we proceed in the same way, but we consider a different intermediary polynomial time checkable criterion called *weak* CCM (wCCM). This is due to the fact that some causality constraints need to be relaxed in order to take into account the program order relaxations of TSO, that allow reads to overtake writes. The definitions of the new criteria CCM and wCCM we use in our approach are quite subtile. Ensuring that these criteria are "as strong as possible" by including all possible order constraints on pairs of writes that can be computed (in polynomial time) using a least fixpoint calculation, while still ensuring that they are weaker than SC/TSO, and proving this fact, is not trivial.

As a proof of concept, we implemented our approach for checking SC/TSO and applied it to executions extracted from realistic cache coherence protocols within the Gem5 simulator [5] in system emulation mode. This evaluation shows that our approach scales better than a direct encoding of the axioms defining SC and TSO [3] into boolean satisfiability. We also show that the partial order of writes imposed by the stronger criteria CCM and wCCM leaves only a small percentage of writes unordered (6.6% in average) in the case that the executions are valid, and most SC/TSO violations are also CCM/wCCM violations.

2 Sequential Consistency and TSO

We consider multi-threaded programs over a set of shared variables $\mathsf{Var} = \{x, y, \ldots\}$. Threads issue read and write operations. Assuming an unspecified set of values Val and a set of operation identifiers OId, we let

$$\mathsf{Op} = \{\mathsf{read}_i(x, v), \mathsf{write}_i(x, v) : i \in \mathsf{OId}, x \in \mathsf{Var}, v \in \mathsf{Val}\}$$

be the set of operations reading a value v or writing a value v to a variable x. We omit operation identifiers when they are not important. The set of read,

resp., write, operations is denoted by $\mathbb{R}$, resp., $\mathbb{W}$. The set of read, resp., write, operations in a set of operations O is denoted by $\mathbb{R}(O)$, resp., $\mathbb{W}(O)$. The variable accessed by an operation o is denoted by $\mathsf{var}(o)$.

Consistency criteria like SC or TSO are formalized on an abstract view of an execution called *history*. A history includes a set of write or read operations ordered according to a (partial) *program order* po which order operations issued by the same thread. Most often, po is a union of sequences, each sequence containing all the operations issued by some thread. Then, we assume that the history includes a *write-read* relation which identifies the write operation writing the value returned by each read in the execution. Such a relation can be extracted easily from executions where each value is written at most once. Since shared-memory implementations (or cache coherence protocols) are data-independent [31] in practice, i.e., their behavior doesn't depend on the concrete values read or written in the program, any potential buggy behavior can be exposed in such executions.

Definition 1. *A* history $\langle O, \mathsf{po}, \mathsf{wr}\rangle$ *is a set of operations* O *along with a strict partial* program order po *and a* write-read *relation* $\mathsf{wr} \subseteq \mathbb{W}(O) \times \mathbb{R}(O)$*, such that the inverse of* wr *is a total function and if* $(\mathit{write}(x, v), \mathit{read}(x', v')) \in \mathsf{wr}$*, then* $x = x'$ *and* $v = v'$*.*

We assume that every history includes a write operation writing the initial value of variable x, for each variable x. These write operations precede all other operations in po. We use h, h_1, h_2, ... to range over histories.

We now define the SC and TSO memory models (we use the same definitions as in the formal framework developed by Alglave et al. [3]). Given a *history* $h = \langle O, \mathsf{po}, \mathsf{wr}\rangle$ and a variable x, a *store order on* x is a strict total order ww_x on the write operations $\mathsf{write}\_(x, \_)$ in O. A *store order* is a union of store orders ww_x, one for each variable x used in h. A *history* $\langle O, \mathsf{po}, \mathsf{wr}\rangle$ is *sequentially consistent* (SC, for short) if there exists a *store order* ww such that $\mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}$ is acyclic. The *read-write* relation rw is defined by $\mathsf{rw} = \mathsf{wr}^{-1} \circ \mathsf{ww}$ (where $\circ$ denotes the standard relation composition).

The definition of TSO relies on three additional relations: (1) the ppo relation which excludes from the program order pairs formed of a write and respectively, a read operation, i.e., $\mathsf{ppo} = \mathsf{po} \setminus (\mathbb{W}(O) \times \mathbb{R}(O))$, (2) the $\mathsf{po\text{-}loc}$ relation which is a restriction of po to operations accessing the same variable, i.e., $\mathsf{po\text{-}loc} = \mathsf{po} \cap \{(o, o') \mid \mathsf{var}(o) = \mathsf{var}(o')\}$, and (3) the write-read external relation wr_e which is a restriction of the write-read relation to pairs of operations in different threads (not related by program order), i.e., $\mathsf{wr}_e = \mathsf{wr} \cap \{(o, o') \mid (o, o') \notin \mathsf{po} \text{ and } (o', o) \notin \mathsf{po}\}$. Then, we say that a history satisfies TSO if there exists a *store order* ww such that $\mathsf{po\text{-}loc} \cup \mathsf{wr}_e \cup \mathsf{ww} \cup \mathsf{rw}$ and $\mathsf{ppo} \cup \mathsf{wr}_e \cup \mathsf{ww} \cup \mathsf{rw}$ are both acyclic.

Notice that the formal definition of the TSO given above is equivalent to the formal operational model of TSO that consists in considering that each thread has a store buffer, and then, each write issued by a thread is first sent to its store buffer before being committed to the memory later in a nondeterministic way. To read a value on some variable x, a thread first checks if it there is still

a write on x pending in its own buffer and in this case it takes the value of the last such as write, otherwise it fetches the value of x in the memory.

3 Checking Sequential Consistency

We define an algorithm for checking whether a history satisfies SC which enforces a polynomially-time checkable criterion weaker than SC, a variation of causal consistency, in order to construct a *partial* store order, i.e., one in which not all the writes on the same variable are ordered. This partial store order is then completed until it orders every two writes on the same variable using a standard backtracking enumeration. This approach is efficient when the number of writes that remain to be ordered using the backtracking enumeration is relatively small, a hypothesis confirmed by our experimental evaluation (see Sect. 5.).

The variation of causal consistency mentioned above, called *convergent causal memory* (CCM, for short), is stronger than existing variations [6] while still being polynomially-time checkable (and weaker than SC). Its definition uses several relations between read and write operations which are analogous or even exactly the same relations used to define those variations. Section 3.1 recalls the existing notions of causal consistency as they are defined in [6] (using the so called "bad-pattern" characterization introduced in that paper), Sect. 3.2 introduces CCM, while Sect. 3.3 presents our algorithm for checking SC.

3.1 Causal Consistency

The weakest variation of causal consistency, called *weak causal consistency* (CC, for short), requires that any two causally-dependent values are observed in the same order by all threads, where causally-dependent means that either those values were written by the same thread (i.e., the corresponding writes are ordered by po), or that one value was written by a thread after reading the other value, or any transitive composition of such dependencies. Values written concurrently by two threads can be observed in any order, and even-more, this order may change in time. A *history* $\langle O, \mathsf{po}, \mathsf{wr} \rangle$ satisfies CC if $\mathsf{po} \cup \mathsf{wr} \cup \mathsf{rw}[\mathsf{co}]$ is acyclic where $\mathsf{co} = (\mathsf{po} \cup \mathsf{wr})^+$ is called the *causal relation*. The *read-write* relation $\mathsf{rw}[\mathsf{co}]$ induced by the causal relation is defined by

$$(\mathsf{read}(x, v), \mathsf{write}(x, v')) \in \mathsf{rw}[\mathsf{co}] \text{ iff } (\mathsf{write}(x, v), \mathsf{write}(x, v')) \in \mathsf{co} \text{ and } (\mathsf{write}(x, v), \mathsf{read}(x, v)) \in \mathsf{wr}, \text{ for some } \mathsf{write}(x, v)$$

The read-write relation $\mathsf{rw}[\mathsf{co}]$ is a variation of rw from the definition of SC/TSO where the store order ww is replaced by the projection of co on pairs of writes. In general, given a binary relation R on operations, $R_{\mathbb{WW}}$ denotes the projection of R on pairs of writes on the same variable. Then,

Definition 2. *The read-write relation* $\mathsf{rw}[R]$ *induced by a relation* R *is defined by* $\mathsf{rw}[R] = \mathsf{wr}^{-1} \circ R_{\mathbb{WW}}$.

Causal convergence (CCv, for short) is a strengthening of CC where concurrent values are required to be observed in the same order by all threads.

A *history* $\langle O, \mathsf{po}, \mathsf{wr} \rangle$ satisfies CCv if it satisfies CC and $\mathsf{po} \cup \mathsf{wr} \cup \mathsf{cf}$ is acyclic where the *conflict relation* cf is defined by

$$(\mathsf{write}(x, v), \mathsf{write}(x, v')) \in \mathsf{cf} \text{ iff } (\mathsf{write}(x, v), \mathsf{read}(x, v')) \in \mathsf{co} \text{ and } (\mathsf{write}(x, v'), \mathsf{read}(x, v')) \in \mathsf{wr}, \text{ for some } \mathsf{read}(x, v')$$

The conflict relation relates two writes w_1 and w_2 when w_1 is causally related to a read taking its value from w_2. The definition of CCM, our new variation of causal consistency, relies on a generalization of the conflict relation where a different relation is used instead of co. Given a binary relation R on operations, $R_{\mathbb{WR}}$ denotes the projection of R on pairs of writes and reads on the same variable, respectively.

Definition 3. *The conflict relation* $\mathsf{cf}[R]$ *induced by a relation* R *is defined by* $\mathsf{cf}[R] = R_{\mathbb{WR}} \circ \mathsf{wr}^{-1}$.

t_0:	t_1:
$\mathsf{write}(x, 1)$	$\mathsf{write}(x, 2)$
$\mathsf{read}(x, 2)$	$\mathsf{read}(x, 1)$

(a) CM but not CCv nor wCCM

t_0:	t_1:
$\mathsf{write}(z, 1)$	$\mathsf{write}(x, 2)$
$\mathsf{write}(x, 1)$	$\mathsf{read}(z, 0)$
$\mathsf{write}(y, 1)$	$\mathsf{read}(y, 1)$
	$\mathsf{read}(x, 2)$

(b) CCv, wCCM and TSO but not CM

t_0:	t_1:
$\mathsf{write}(x, 1)$	$\mathsf{write}(y, 1)$
$\mathsf{write}(x, 2)$	$\mathsf{write}(y, 2)$
$\mathsf{read}(y, 1)$	$\mathsf{read}(y, 2)$
	$\mathsf{read}(x, 1)$

(c) CM and CCv but not CCM

t_0:	t_1:
$\mathsf{write}(x, 1)$	$\mathsf{write}(x, 2)$
$\mathsf{read}(y, 0)$	$\mathsf{read}(y, 0)$
$\mathsf{write}(y, 1)$	$\mathsf{write}(y, 2)$
$\mathsf{read}(x, 1)$	$\mathsf{read}(x, 2)$

(d) CCM but not SC

Fig. 1. Histories with two threads used to compare different consistency models. Operations of the same thread are aligned vertically.

Finally, *causal memory* (CM, for short) is a strengthening of CC where roughly, concurrent values are required to be observed in the same order by a thread during its entire execution. Differently from CCv, this order can differ from one thread to another. Although this intuitive description seems to imply that CM is weaker than CCv, the two models are actually incomparable. For instance, the history in Fig. 1a is allowed by CM, but not by CCv. It is not allowed by CCv because reading 1 from x in the first thread implies that it observed $\mathsf{write}(x, 1)$ after $\mathsf{write}(x, 2)$ while reading 2 from x in the second thread

implies that it observed $\mathsf{write}(x,2)$ after $\mathsf{write}(x,1)$. While this is allowed by CM where different threads can observe concurrent writes in different orders, it is not allowed by CCv. Then, the history in Fig. 1b is CCv but not CM. It is not allowed by CM because reading the initial value 0 from z implies that $\mathsf{write}(x,1)$ is observed after $\mathsf{write}(x,2)$ while reading 2 from x implies that $\mathsf{write}(x,2)$ is observed after $\mathsf{write}(x,1)$ ($\mathsf{write}(x,1)$ must have been observed because the same thread reads 1 from y and the writes on x and y are causally related). However, under CCv, a thread simply reads the most recent value on each variable and the order in which these values are ordered using timestamps for instance is independent of the order in which variables are read in a thread, e.g., reading 0 from z doesn't imply that the timestamp of $\mathsf{write}(x,2)$ is smaller than the timestamp of $\mathsf{write}(x,1)$. This history is admitted by CCv assuming that the order in which $\mathsf{write}(x,1)$ and $\mathsf{write}(x,2)$ are observed is $\mathsf{write}(x,1)$ before $\mathsf{write}(x,2)$.

Let us give the formal definition of CM. Let h=$\langle O, \mathsf{po}, \mathsf{wr}\rangle$ be a history. For every operation o in h, let hb_o be the smallest transitive relation such that:

1. if two operations are causally related, and each one causally related to o, then they are related by hb_o, i.e., $(o_1,o_2) \in \mathsf{hb}_o$ if $(o_1,o_2) \in \mathsf{co}$, $(o_1,o) \in \mathsf{co}$, and $(o_2,o) \in \mathsf{co}^*$ (where co^* is the reflexive closure of co), and
2. two writes w_1 and w_2 are related by hb_o if w_1 is hb_o-related to a read taking its value from w_2, and that read is done by the same thread executing o and before o (this scenario is similar to the definition of the conflict relation above), i.e., $(\mathsf{write}(x,v), \mathsf{write}(x,v')) \in \mathsf{hb}_o$ if $(\mathsf{write}(x,v), \mathsf{read}(x,v')) \in \mathsf{hb}_o$, $(\mathsf{write}(x,v'), \mathsf{read}(x,v')) \in \mathsf{wr}$, and $(\mathsf{read}(x,v'), o) \in \mathsf{po}^*$, for some $\mathsf{read}(x,v')$.

A history $\langle O, \mathsf{po}, \mathsf{wr}\rangle$ satisfies CM if it satisfies CC and for each operation o in the history, the relation hb_o is acyclic.

Bouajjani et al. [6] show that the problem of checking whether a history satisfies CC, CCv, or CM is polynomial time. This result is a straightforward consequence of the above definitions, since the union of relations required to be acyclic can be computed in polynomial time from the relations po and wr which are fixed in a given history. In particular, the union of these relations can be computed by a DATALOG program.

3.2 Convergent Causal Memory

We define a new variation of causal consistency which builds on causal memory, but similar to causal convergence it enforces that all threads agree on an order in which to observe values written by concurrent (causally-unrelated) writes, and also, it uses a larger read-write relation. A history $\langle O, \mathsf{po}, \mathsf{wr}\rangle$ satisfies *convergent causal memory* (CCM, for short) if $\mathsf{po} \cup \mathsf{wr} \cup \mathsf{pww} \cup \mathsf{rw}[\mathsf{pww}]$ is acyclic, where the *partial store order* pww is defined by

$$\mathsf{pww} = (\mathsf{hb}_{\mathbb{WW}} \cup \mathsf{cf}[\mathsf{hb}])^+ \text{ with } \mathsf{hb} = \big(\bigcup_{o \in O} \mathsf{hb}_o\big)^+.$$

The partial store order pww contains the ordering constraints between writes in all relations hb_o used to defined causal memory, and also, the conflict relation

induced by this set of constraints (a weaker version of conflict relation was used to define causal convergence).

As a first result, we show that all the variations of causal consistency in Sect. 3.1, i.e., CC, CCv and CM, are strictly weaker than CCM.

Lemma 1. *If a history satisfies CCM, then it satisfies CC, CCv and CM.*

Proof. Let $h = \langle O, \mathsf{po}, \mathsf{wr} \rangle$ be a history satisfying CCM. By the definition of hb, we have that $\mathsf{co}_{\mathbb{WW}} \subseteq \mathsf{hb}_{\mathbb{WW}}$. Indeed, any two writes o_1 and o_2 related by co are also related by hb_{o_2}, which by the definition of hb, implies that they are related by $\mathsf{hb}_{\mathbb{WW}}$. Then, by the definition of pww, we have that $\mathsf{hb}_{\mathbb{WW}} \subseteq \mathsf{pww}$. This implies that $\mathsf{rw}[\mathsf{co}] \subseteq \mathsf{rw}[\mathsf{pww}]$ (by definition, $\mathsf{rw}[\mathsf{co}] = \mathsf{rw}[\mathsf{co}_{\mathbb{WW}}]$). Therefore, the acyclicity of $\mathsf{po} \cup \mathsf{wr} \cup \mathsf{pww} \cup \mathsf{rw}[\mathsf{pww}]$ implies that its subset $(\mathsf{po} \cup \mathsf{wr} \cup \mathsf{rw}[\mathsf{co}]$ is also acyclic, which means that h satisfies CC. Also, it implies that $\mathsf{po} \cup \mathsf{wr} \cup \mathsf{cf}[\mathsf{hb}]$ is acyclic (the last term of the union is included in pww), which by $\mathsf{co} \subseteq \mathsf{hb}$, implies that $\mathsf{po} \cup \mathsf{wr} \cup \mathsf{cf}[\mathsf{co}]$ is acyclic, and thus, h satisfies CCv. The fact that h satisfies CM follows from the fact that h satisfies CC (since $\mathsf{po} \cup \mathsf{wr}$ is acyclic) and hb is acyclic ($\mathsf{hb}_{\mathbb{WW}}$ is included in pww and the rest of the dependencies in hb are included in $\mathsf{po} \cup \mathsf{wr}$). □

The reverse of the above lemma doesn't hold. Figure 1c shows a history which satisfies CM and CCv, but it is not CCM. To show that this history does not satisfy CCM we use the fact that pww relates any two writes which are ordered by program order. Then, we get that $\mathsf{read}(x, 1)$ and $\mathsf{write}(x, 2)$ are related by $\mathsf{rw}[\mathsf{pww}]$ (because $\mathsf{write}(x, 1)$ is related by write-read with $\mathsf{read}(x, 1)$), which further implies that $(\mathsf{read}(x, 1), \mathsf{read}(y, 1)) \in \mathsf{rw}[\mathsf{pww}] \circ \mathsf{po}$. Similarly, we have that $(\mathsf{read}(y, 1), \mathsf{read}(x, 1)) \in \mathsf{rw}[\mathsf{pww}] \circ \mathsf{po}$, which implies that $\mathsf{po} \cup \mathsf{wr} \cup \mathsf{pww} \cup \mathsf{rw}[\mathsf{pww}]$ is *not* acyclic, and therefore, the history does not satisfy CCM. The fact that this history satisfies CM and CCv follows easily from definitions.

Next, we show that CCM is weaker than SC, which will be important in our algorithm for checking whether a history satisfies SC.

Lemma 2. *If a history satisfies SC, then it satisfies CCM.*

Proof. Using the definition of CCM, Let $h = \langle O, \mathsf{po}, \mathsf{wr} \rangle$ be a history satisfying SC. Then, there exists a *store order* ww such that $\mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}]$ is acyclic. We show that the two relations $\mathsf{hb}_{\mathbb{WW}}$ and $\mathsf{cf}[\mathsf{hb}]$, whose union constitutes pww, are both included in ww. We first prove that $\mathsf{hb} \subseteq (\mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}])^+$ by structural induction on the definition of hb_o:

1. if $(o_1, o_2) \in \mathsf{co} = (\mathsf{po} \cup \mathsf{wr})^+$, then clearly, $(o_1, o_2) \in (\mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}])^+$,
2. if $(\mathsf{write}(x, v), \mathsf{read}(x, v')) \in (\mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}])^+$ and there is $\mathsf{read}(x, v')$ such that $(\mathsf{write}(x, v'), \mathsf{read}(x, v')) \in \mathsf{wr}$, then $(\mathsf{write}(x, v), \mathsf{write}(x, v')) \in \mathsf{ww}$. Otherwise, assuming by contradiction that $(\mathsf{write}(x, v'), \mathsf{write}(x, v)) \in \mathsf{ww}$, we get that $(\mathsf{read}(x, v'), \mathsf{write}(x, v)) \in \mathsf{rw}[\mathsf{ww}]$ (by the definition of $\mathsf{rw}[\mathsf{ww}]$ using the hypothesis $(\mathsf{write}(x, v'), \mathsf{read}(x, v')) \in \mathsf{wr}$). Note that the latter implies that $\mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}]$ is cyclic.

Since $\mathsf{hb} \subseteq (\mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}])^+$, we get that $\mathsf{hb}_{\mathsf{WW}} \subseteq \mathsf{ww}$. Also, since $\mathsf{cf}[(\mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}])^+] \subseteq (\mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}])^+$ (using a similar argument as in point (2) above), we get that $\mathsf{cf}[\mathsf{hb}] \subseteq (\mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}])^+$.

Finally, since $\mathsf{pww} \subseteq \mathsf{ww}$, we get that $(\mathsf{po} \cup \mathsf{wr} \cup \mathsf{pww} \cup \mathsf{rw}[\mathsf{pww}])^+ \subseteq (\mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}])^+$, which implies that the acyclicity of the latter implies the acyclicity of the former. Therefore, h satisfies CCM. □

The reverse of the above lemma doesn't hold. For instance, the history in Fig. 1d is not SC but it is CCM. This history admits a partial store order pww where the writes in different threads are not ordered.

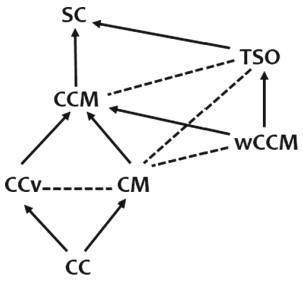


Fig. 2. Relationships between consistency models. Directed arrows denote the "weaker-than" relation while dashed lines connect incomparable models.

The left side of Fig. 2 (ignoring wCCM and TSO) summarizes the relationships between the consistency models presented in this section.

The partial store order pww can be computed in polynomial time (in the size of the input history). Indeed, the hb_o relations can be computed using a least fixpoint calculation that converges in at most a quadratic number of iterations and acyclicity can be decided in polynomial time. Therefore,

Theorem 1. *Checking whether a history satisfies CCM is polynomial time in the size of the history.*

3.3 An Algorithm for Checking Sequential Consistency

Algorithm 1 checks whether a given history satisfies sequential consistency. As a first step, it checks whether the given history satisfies CCM. If this is not the case, then, by Lemma 2, the history does not satisfy SC as well, and the algorithm returns *false*. Otherwise, it enumerates store orders which extend the partial store order pww, until finding one that witnesses for satisfaction of SC. The history is a violation to SC iff no such store order is found. The soundness of this last step is implied by the proof of Lemma 2, which shows that pww is included in any store order ww witnessing for SC satisfaction.

Theorem 2. *Algorithm 1 returns true iff the input history h satisfies SC.*

Input: A history $h = \langle O, \mathsf{po}, \mathsf{wr} \rangle$
Output: *true* iff h satisfies SC

```
1 if po ∪ wr ∪ pww ∪ rw[pww] is cyclic then
2 |   return false;
3 end
4 foreach ww ⊃ pww do
5 |   if po ∪ wr ∪ ww ∪ rw[ww] is acyclic then
6 |   |   return true;
7 |   end
8 end
9 return false;
```

Algorithm 1. Checking SC conformance.

4 Checking Conformance to the TSO Model

We consider now the problem of checking whether a history satisfies TSO. Following the approach developed above for SC, we define a polynomial time checkable criterion, based on a (different) variation of causal consistency that is suitable for the case of TSO. This allows to reduce the number of pairs of writes for which an order must be guessed in order to establish conformance to TSO.

The case of TSO requires the definition of a new intermediary consistency model because CCM is based on a causality order that includes the program order po which is relaxed in the context of TSO, compared to the SC model. Indeed, CCM is *not* weaker than TSO as shown by the history in Fig. 1b (note that this does not imply that other variations of causal consistency, CC and CCv, are also not weaker than TSO). This history satisfies TSO because, based on its operational model, the operation $\mathsf{write}(x, 2)$ of thread t_1 can be delayed (pending in the store buffer of t_1) until the end of the execution. Therefore, after executing $\mathsf{read}(z, 0)$, all the writes of thread t_0 are committed to the main memory so that thread t_1 can read 1 from y and 2 from x (it is obliged to read the value of x from its own store buffer). This history is not admitted by CCM because it is not admitted by the weaker causal consistency variation CM. Figure 3 shows a history admitted by CCM but not by TSO. Indeed, under TSO, both t_2 and t_3 should see the writes on x and y performed by t_0 and t_1, respectively, in the same order. This is not the case, because t_2 "observes" the write on x before the write on y (since it reads 0 from y) and t_3 "observes" the write on y before the write on x (since it reads 0 from x). This history is admitted by CCM because the two writes are causally independent and they concern different variables. We mention that TSO and CM are also incomparable. For instance, the history in Fig. 1a is allowed by CM, but not by TSO. The history in Fig. 1b is admitted by TSO, but not by CM.

Next, we define a weakening of CCM, called *weak convergent causal memory* (wCCM), which is also weaker than TSO. The model wCCM is based on causality relations induced by the relaxed program orders ppo and po-loc instead of po, and the external write-read relation instead of the full write-read relation.

t_0:	t_1:	t_2:	t_3:
$\mathsf{write}(x,1)$	$\mathsf{write}(y,1)$	$\mathsf{read}(x,1)$	$\mathsf{read}(y,1)$
		$\mathsf{read}(y,0)$	$\mathsf{read}(x,0)$

Fig. 3. A history admitted by wCCM and CCM but not by TSO.

4.1 Weak Convergent Causal Memory

First, we define two causality relations relative to the partial program orders in the definition of TSO and the external write-read relation: For $\pi \in \{\mathsf{ppo}, \mathsf{po\text{-}loc}\}$, let $\mathsf{co}^\pi = (\pi \cup \mathsf{wr}_e)^+$. We also consider a notion of conflict that is defined in terms of the external write-read relation as follows: For a given relation R, let $\mathsf{cf}_e[R] = R_{\mathbb{WR}} \circ \mathsf{wr}_e^{-1}$.

Then, given a history $\langle O, \mathsf{po}, \mathsf{wr}\rangle$, we define for each operation o two happens-before relations $\mathsf{hb}_o^{\mathsf{ppo}}$ and $\mathsf{hb}_o^{\mathsf{po\text{-}loc}}$. The definition of these relations is similar to the one of hb_o (from causal memory), the differences being that po is replaced by ppo and $\mathsf{po\text{-}loc}$ respectively, co is replaced by $\mathsf{co}^{\mathsf{ppo}}$ and $\mathsf{co}^{\mathsf{po\text{-}loc}}$ respectively, and wr is replaced by wr_e. Therefore, for $\pi \in \{\mathsf{ppo}, \mathsf{po\text{-}loc}\}$, hb_o^π is is the smallest transitive relation such that:

1. $(o_1, o_2) \in \mathsf{hb}_o^\pi$ if $(o_1, o_2) \in \mathsf{co}^\pi$, $(o_1, o) \in \mathsf{co}^\pi$, and $(o_2, o) \in (\mathsf{co}^\pi)^*$, and
2. $(\mathsf{write}(x,v), \mathsf{write}(x,v')) \in \mathsf{hb}_o^\pi$ if $(\mathsf{write}(x,v), \mathsf{read}(x,v')) \in \mathsf{hb}_o^\pi$, and $(\mathsf{write}(x,v'), \mathsf{read}(x,v')) \in \mathsf{wr}$ and $(\mathsf{read}(x,v'), o) \in \pi^*$, for some $\mathsf{read}(x,v')$.

Let $\mathsf{hb}^\pi = (\bigcup_{o \in O} \mathsf{hb}_o^\pi)^+$, for $\pi \in \{\mathsf{ppo}, \mathsf{po\text{-}loc}\}$, and let $\mathsf{whb} = (\mathsf{hb}_o^{\mathsf{ppo}} \cup \mathsf{hb}_o^{\mathsf{po\text{-}loc}})^+$. Then, the weak partial store order is defined as follows:

$$\mathsf{wpww} = (\mathsf{whb}_{\mathbb{WW}} \cup \mathsf{cf}_e[\mathsf{hb}^{\mathsf{po\text{-}loc}}] \cup \mathsf{cf}_e[\mathsf{hb}^{\mathsf{ppo}}])^+$$

Then, we say that a history $\langle O, \mathsf{po}, \mathsf{wr}\rangle$ satisfies *weak convergent causal memory* (wCCM) if both relations:

$$\mathsf{ppo} \cup \mathsf{wr}_e \cup \mathsf{wpww} \cup \mathsf{rw}[\mathsf{wpww}] \text{ and } \mathsf{po\text{-}loc} \cup \mathsf{wr}_e \cup \mathsf{wpww} \cup \mathsf{rw}[\mathsf{wpww}]$$

are acyclic.

Lemma 3. *If a history satisfies TSO, then it satisfies wCCM.*

Proof. Let $h = \langle O, \mathsf{po}, \mathsf{wr}\rangle$ be a history satisfying TSO. Then, there exists a store order ww such that $\mathsf{po\text{-}loc} \cup \mathsf{wr}_e \cup \mathsf{ww} \cup \mathsf{rw}$ and $\mathsf{ppo} \cup \mathsf{wr}_e \cup \mathsf{ww} \cup \mathsf{rw}$ are both acyclic. The fact that

$$\mathsf{hb}^{\mathsf{po\text{-}loc}} \subseteq (\mathsf{po\text{-}loc} \cup \mathsf{wr}_e \cup \mathsf{ww} \cup \mathsf{rw})^+ \text{ and } \mathsf{hb}^{\mathsf{ppo}} \subseteq (\mathsf{ppo} \cup \mathsf{wr}_e \cup \mathsf{ww} \cup \mathsf{rw})^+$$

can be proved by structural induction like in the case of SC (the step of the proof showing that $\mathsf{hb} \subseteq \mathsf{po} \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}]$). Then, since ww is a total order on writes on the same variable, we get that the projection of whb (the transitive closure of the union of $\mathsf{hb}^{\mathsf{po\text{-}loc}}$ and $\mathsf{hb}^{\mathsf{ppo}}$) on pairs of writes on the same variable

is included in ww. Therefore, $\mathsf{whb}_{\mathbb{WW}} \subseteq \mathsf{ww}$. Then, since $\mathsf{cf}_e[R^\pi] \subseteq R^\pi$ for each $R^\pi = (\pi \cup \mathsf{wr}_e \cup \mathsf{ww} \cup \mathsf{rw})^+$ with $\pi \in \{\mathsf{ppo}, \mathsf{po\text{-}loc}\}$ and since each $\mathsf{cf}_e[R^\pi]$ relates only writes on the same variable, we get that each $\mathsf{cf}_e[R^\pi]$ is included in ww. This implies that $\mathsf{wpww} \subseteq \mathsf{ww}$.

Finally, since $\mathsf{wpww} \subseteq \mathsf{ww}$, we get that $(\pi \cup \mathsf{wr} \cup \mathsf{wpww} \cup \mathsf{rw}[\mathsf{wpww}])^+ \subseteq (\pi \cup \mathsf{wr} \cup \mathsf{ww} \cup \mathsf{rw}[\mathsf{ww}])^+$, for each $\pi \in \{\mathsf{ppo}, \mathsf{po\text{-}loc}\}$. In each case, the acyclicity of the latter implies the acyclicity of the former. Therefore, h satisfies wCCM.

Input: A history $h = \langle O, \mathsf{po}, \mathsf{wr} \rangle$
Output: *true* iff h satisfies TSO

```
1 if ppo ∪ wr_e ∪ wpww ∪ rw[wpww] or po-loc ∪ wr_e ∪ pww ∪ rw[wpww] is cyclic then
2 |   return false;
3 end
4 foreach ww ⊃ wpww do
5 |   if ppo ∪ wr_e ∪ ww ∪ rw[ww] and po-loc ∪ wr_e ∪ ww ∪ rw[ww] are acyclic then
6 |   |   return true;
7 |   end
8 end
9 return false;
```

Algorithm 2. Checking TSO conformance.

The reverse of the above lemma does not hold. Indeed, it can be easily seen that wCCM is weaker than CCM (since wpww is included in pww) and the history in Fig. 3, which satisfies CCM but not TSO (as explained in the beginning of the section), is also an example of a history that satisfies wCCM but not TSO. Then, wCCM is incomparable to CM. For instance, the history in Fig. 1b is allowed by wCCM (since it is allowed by TSO as explained in the beginning of the section) but not by CM. Also, since CCM is stronger than CM, the history in Fig. 3 satisfies CM but not wCCM (since it does not satisfy TSO). These relationships are summarized in Fig. 2. Establishing the precise relation between CC/CCv and TSO is hard because they are defined using one, resp., two, acyclicity conditions. We believe that CC and CCv are weaker than TSO, but we don't have a formal proof.

Finally, it can be seen that, similarly to pww, the weak partial store order wpww can be computed in polynomial time, and therefore:

Theorem 3. *Checking whether a history satisfies wCCM is polynomial time in the size of the history.*

4.2 An Algorithm for Checking TSO Conformance

The algorithm for checking TSO conformance for a given history is given in Fig. 2. It starts by checking whether the history violates the weaker consistency

model wCCM. If yes, it returns false. If not, it starts enumerating the orders between the writes that are not related by the weak partial store order wpww until it founds one that allows establishing TSO conformance and in this case it returns true. Otherwise it returns false.

Theorem 4. *Algorithm 2 returns true iff the input history h satisfies TSO.*

5 Experimental Evaluation

To demonstrate the practical value of the theory developed in the previous sections, we argue that our algorithms are efficient and scalable. We experiment with both SC and TSO algorithms, investigating their running time compared to a standard encoding of these models into boolean satisfiability on a benchmark obtained by running realistic cache coherence protocols within the Gem5 simulator [5] in system emulation mode.

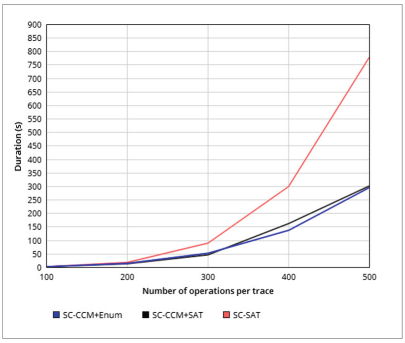
Histories are generated with random clients of the following cache coherence protocols included in the Gem5 distribution: MI, MEOSI Hammer, MESI Two Level, and MEOSI AMD Base. The randomization process is parametrized by the number of cpus (threads) and the total number of read-/write operations. We ensure that every value is written at most once.

We have compared two variations of our algorithms for checking SC/TSO with a standard encoding of SC/TSO into boolean satisfiability (named X-SAT where X is SC or TSO). The two variations differ in the way in which the partial store order pww dictated by CCM is completed to a total store order ww as required by SC/TSO: either using standard enumeration (named X-CCM+Enum where X is SC or TSO) or using a SAT solver (named X-CCM+SAT where X is SC or TSO).

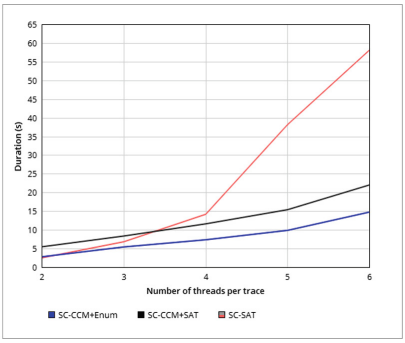
The computation of the partial store order pww is done using an encoding of its definition into a DATALOG program. The inductive definition of hb_o supports an easy translation to DATALOG rules, and the same holds for the union of two relations, or their composition. We used Clingo [19] to run DATALOG programs.

5.1 Checking SC

Figure 4 reports on the running time of the three algorithms while increasing the number of operations or cpus. All the histories considered in this experiment satisfy SC. This is intended because valid histories force our algorithms to enumerate extensions of the partial store order (SC violations may be detected while checking CCM). The graph on the left pictures the evolution of the running time when increasing the number of operations from 100 to 500, in increments of 100 (while using a constant number of 4 cpus). For each number of operations, we have considered 200 histories and computed the average running time. The graph on the right shows the running time when increasing the number of cpus from 2 to 6, in increments of 1. For x cpus, we have limited the number of operations to $50x$. As before for each number of cpus, we have considered 200 histories and computed



(a) Checking SC while varying the number of operations.



(b) Checking SC while varying the number of cpus.

Fig. 4. Checking SC for valid histories.

the average running time. As it can be observed, our algorithms scale much better than the SAT encoding and interestingly enough, the difference between an explicit enumeration of pww extensions and one using a SAT solver is not significant. Note that even small improvements on the average running time provide large speedups when taking into account the whole testing process, i.e., checking consistency for a possibly large number of (randomly-generated) executions. For instance, the work on McVerSi [13], which focuses on the complementary problem of finding clients that increase the probability of uncovering bugs, shows that exposing bugs in some realistic cache coherence implementations requires even 24 h of continuous testing.

Since the bottleneck in our algorithms is given by the enumeration of pww extensions, we have measured the percentage of pairs of writes that are *not* ordered by pww. Thus, we have considered a random sample of 200 histories (with 200 operations per history) and evaluated this percentage to be just 6.6%, which is surprisingly low. This explains the net gain in comparison to a SAT encoding of SC, since the number of pww extensions that need to be enumerated is quite low. As a side remark, using CCv instead of CCM in the algorithms above leads to a drastic increase in the number of unordered writes. For the same random sample of 200 histories, we conclude that using CCv instead of CCM leaves 57.75% of unordered writes in average which is considerably bigger than the percentage of unordered writes when using CCM.

We have also evaluated our algorithms on SC violations. These violations were generated by reordering statements from the MI implementation, e.g., swapping the order of the actions `s_store_hit` and `p_profileHit` in the transition `transition(M, Store)`. As an optimization, our implementation checks gradually the weaker variations of causal consistency CC and CCv before checking CCM. This is to increase the chances of returning in the case of a violation (a violation to CC/CCv is also a violation to CCM and SC). We have considered 1000 histories with 100 to 400 operations and 2 to 8 cpus, equally distributed in function

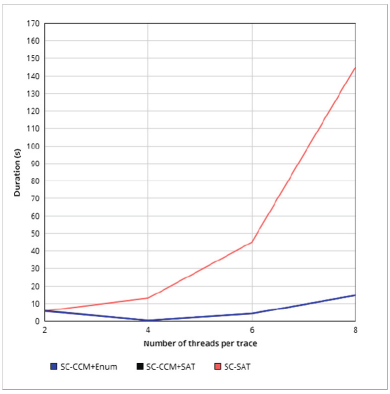
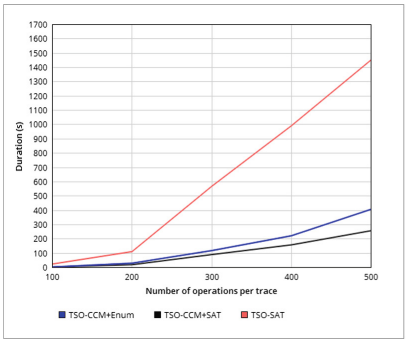
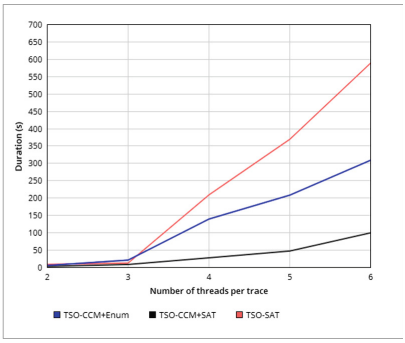


Fig. 5. Checking SC for invalid histories while increasing the number of cpus.



(a) Checking TSO while varying the number of operations.



(b) Checking TSO while varying the number of cpus.

Fig. 6. Checking TSO for valid histories.

of the number of cpus. Figure 5 reports on the evolution of the average running time. Since these histories happen to all be CCM violations, SC-CCM+Enum and SC-CCM+SAT have the same running time. As an evaluation of our optimization, we have found that 50% of the histories invalidate weaker variations of causal consistency, CC or CCv.

5.2 Checking TSO

We have evaluated our TSO algorithms on the same set of histories used for SC in Fig. 4. Since these histories satisfy SC, they satisfy TSO as well. As in the case of SC, our algorithms scale better than the SAT encoding. However, differently from SC, the enumeration of wpww extensions using a SAT solver outperforms

the explicit enumeration. Since this difference was more negligible in the case of SC, it seems that the SAT variation is generally better.

6 Related Work

While several static techniques have been developed to prove that a shared-memory implementation (or cache coherence protocol) satisfies SC [1,4,9–12,17,20,23,27,28] few have addressed dynamic techniques such as testing and runtime verification (which scale to more realistic implementations). From the complexity standpoint, Gibbons and Korach [21] showed that checking whether a history is SC is NP-hard while Alur et al. [4] showed that checking SC for finite-state shared-memory implementations (over a bounded number of threads, variables, and values) is undecidable [4]. The fact that checking whether a history satisfies TSO is also NP-hard has been proved by Furbach et al. [18].

There are several works that addressed the testing problem for related criteria, e.g., linearizability. While SC requires that the operations in a history be explained by a linearization that is consistent with the program order, linearizability requires that such a linearization be also consistent with the real-time order between operations (linearizability is stronger than SC). The works in [25,30] describe monitors for checking linearizability that construct linearizations of a given history incrementally, in an online fashion. This incremental construction cannot be adapted to SC since it strongly relies on the specificities of linearizability. Line-Up [8] performs systematic concurrency testing via schedule enumeration, and offline linearizability checking via linearization enumeration. The works in [15,16] show that checking linearizability for some particular class of ADTs is polynomial time. Emmi and Enea [14] consider the problem of checking weak consistency criteria, but their approach focuses on specific relaxations in those criteria, falling back to an explicit enumeration of linearizations in the context of a criterion like SC or TSO. Bouajjani et al. [6] consider the problem of checking causal consistency. They formalize the different variations of causal consistency we consider in this work and show that the problem of checking whether a history satisfies one of these variations is polynomial time.

The complementary issue of test generation, i.e., finding clients that increase the probability of uncovering bugs in shared memory implementations, has been approached in the McVerSi framework [13]. Their methodology for checking a criterion like SC lies within the context of white-box testing, i.e., the user is required to annotate the shared memory implementation with events that define the store order in an execution. Our algorithms have the advantage that the implementation is treated as a black-box requiring less user intervention.

7 Conclusion

We have introduced an approach for checking the conformance of a computation to SC or to TSO, a problem known to be NP-hard. The idea is to avoid an explicit enumeration of the exponential number of possible total orders between writes in

order to solve these problems. Our approach is to define weaker criteria that are as strong as possible but still polynomial time checkable. This is useful for (1) early detection of violations, and (2) reducing the number of pairs of writes for which an order must be found in order to check SC/TSO conformance. Morally, the approach consists in being able to capture an "as large as possible" partial order on writes that can be computed in polynomial time (using a least fixpoint calculation), and which is a subset of any total order witnessing SC/TSO conformance. Our experimental results show that this approach is indeed useful and performant: it allows to catch most of violations early using an efficient check, and it allows to compute a large kernel of write constraints that reduces significantly the number of pairs of writes that are left to be ordered in an enumerative way. Future work consists in exploring the application of this approach to other correctness criteria that are hard to check such a serializability in the context of transactional programs.

References

1. Abdulla, P.A., Haziza, F., Holík, L.: Parameterized verification through view abstraction. STTT **18**(5), 495–516 (2016). https://doi.org/10.1007/s10009-015-0406-x
2. Ahamad, M., Neiger, G., Burns, J.E., Kohli, P., Hutto, P.W.: Causal memory: definitions, implementation, and programming. Distrib. Comput. **9**(1), 37–49 (1995)
3. Alglave, J., Maranget, L., Tautschnig, M.: Herding cats: modelling, simulation, testing, and data mining for weak memory. ACM Trans. Program. Lang. Syst. **36**(2), 7:1–7:74 (2014). https://doi.org/10.1145/2627752
4. Alur, R., McMillan, K.L., Peled, D.A.: Model-checking of correctness conditions for concurrent objects. Inf. Comput. **160**(1–2), 167–188 (2000). https://doi.org/10.1006/inco.1999.2847
5. Binkert, N., et al.: The gem5 simulator. SIGARCH Comput. Archit. News **39**(2), 1–7 (2011). https://doi.org/10.1145/2024716.2024718
6. Bouajjani, A., Enea, C., Guerraoui, R., Hamza, J.: On verifying causal consistency. In: Castagna, G., Gordon, A.D. (eds.) Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18–20, 2017, pp. 626–638. ACM (2017). http://dl.acm.org/citation.cfm?id=3009888
7. Burckhardt, S.: Principles of Eventual Consistency. Now publishers, Boston, October 2014
8. Burckhardt, S., Dern, C., Musuvathi, M., Tan, R.: Line-up: a complete and automatic linearizability checker. In: Zorn, B.G., Aiken, A. (eds.) Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, Toronto, Ontario, Canada, 5–10 June 2010, pp. 330–340. ACM (2010). https://doi.org/10.1145/1806596.1806634
9. Clarke, E.M., et al.: Verification of the futurebus+ cache coherence protocol. In: Agnew, D., Claesen, L.J.M., Camposano, R. (eds.) Computer Hardware Description Languages and their Applications, Proceedings of the 11th IFIP WG10.2 International Conference on Computer Hardware Description Languages and their Applications - CHDL 1993, sponsored by IFIP WG10.2 and in cooperation with IEEE COMPSOC, Ottawa, Ontario, Canada, 26–28 April 1993. IFIP Transactions, vol. A-32, pp. 15–30. North-Holland (1993)

10. Delzanno, G.: Automatic verification of parameterized cache coherence protocols. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 53–68. Springer, Heidelberg (2000). https://doi.org/10.1007/10722167_8
11. Delzanno, G.: Constraint-based verification of parameterized cache coherence protocols. Formal Methods Syst. Des. **23**(3), 257–301 (2003)
12. Eiríksson, Á.T., McMillan, K.L.: Using formal verification/analysis methods on the critical path in system design: a case study. In: Wolper, P. (ed.) CAV 1995. LNCS, vol. 939, pp. 367–380. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60045-0_63
13. Elver, M., Nagarajan, V.: Mcversi: a test generation framework for fast memory consistency verification in simulation. In: 2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2016, Barcelona, Spain, 12–16 March 2016, pp. 618–630. IEEE Computer Society (2016). https://doi.org/10.1109/HPCA.2016.7446099
14. Emmi, M., Enea, C.: Monitoring weak consistency. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 487–506. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_26
15. Emmi, M., Enea, C.: Sound, complete, and tractable linearizability monitoring for concurrent collections. PACMPL **2**(POPL), 25:1–25:27 (2018). https://doi.org/10.1145/3158113
16. Emmi, M., Enea, C., Hamza, J.: Monitoring refinement via symbolic reasoning. In: Grove, D., Blackburn, S. (eds.) Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, 15–17 June, 2015, pp. 260–269. ACM (2015). https://doi.org/10.1145/2737924.2737983
17. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: 14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, 2–5 July 1999, pp. 352–359. IEEE Computer Society (1999). https://doi.org/10.1109/LICS.1999.782630
18. Furbach, F., Meyer, R., Schneider, K., Senftleben, M.: Memory-model-aware testing: a unified complexity analysis. ACM Trans. Embed. Comput. Syst. **14**(4), 63:1–63:25 (2015). https://doi.org/10.1145/2753761
19. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo = ASP + control: Preliminary report. CoRR abs/1405.3694 (2014). http://arxiv.org/abs/1405.3694
20. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM **39**(3), 675–735 (1992). https://doi.org/10.1145/146637.146681
21. Gibbons, P.B., Korach, E.: Testing shared memories. SIAM J. Comput. **26**(4), 1208–1244 (1997). https://doi.org/10.1137/S0097539794279614
22. Gotsman, A., Burckhardt, S.: Consistency models with global operation sequencing and their composition. In: Richa, A.W. (ed.) 31st International Symposium on Distributed Computing, DISC 2017, LIPIcs, 16–20 October 2017, Vienna, Austria, vol. 91, pp. 23:1–23:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017). https://doi.org/10.4230/LIPIcs.DISC.2017.23
23. Ip, C.N., Dill, D.L.: Better verification through symmetry. Formal Methods Syst. Des. **9**(1/2), 41–75 (1996). https://doi.org/10.1007/BF00625968
24. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Comput. **28**(9), 690–691 (1979). https://doi.org/10.1109/TC.1979.1675439
25. Lowe, G.: Testing for linearizability. Concurrency Comput. Pract. Experience 29(4) (2017). https://doi.org/10.1002/cpe.3928

26. Perrin, M., Mostefaoui, A., Jard, C.: Causal consistency: beyond memory. In: Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2016, pp. 26:1–26:12. ACM, New York (2016)
27. Pong, F., Dubois, M.: A new approach for the verification of cache coherence protocols. IEEE Trans. Parallel Distrib. Syst. **6**(8), 773–787 (1995). https://doi.org/10.1109/71.406955
28. Qadeer, S.: Verifying sequential consistency on shared-memory multiprocessors by model checking. IEEE Trans. Parallel Distrib. Syst. **14**(8), 730–741 (2003). https://doi.org/10.1109/TPDS.2003.1225053
29. Sewell, P., Sarkar, S., Owens, S., Nardelli, F.Z., Myreen, M.O.: x86-tso: a rigorous and usable programmer's model for x86 multiprocessors. Commun. ACM **53**(7), 89–97 (2010). https://doi.org/10.1145/1785414.1785443
30. Wing, J.M., Gong, C.: Testing and verifying concurrent objects. J. Parallel Distrib. Comput. **17**(1–2), 164–182 (1993). https://doi.org/10.1006/jpdc.1993.1015
31. Wolper, P.: Expressing interesting properties of programs in propositional temporal logic. In: Conference Record of the Thirteenth Annual ACM Symposium on Principles of Programming Languages, St. Petersburg Beach, Florida, USA, January 1986, pp. 184–193. ACM Press (1986). https://doi.org/10.1145/512644.512661