# Brief Announcement: Generalising Concurrent Correctness to Weak Memory

**Simon Doherty**[1]
Department of Computer Science, University of Sheffield, UK
s.doherty@sheffield.ac.uk

**Brijesh Dongol**[2]
Department of Computer Science, University of Surrey, Guildford, UK

**Heike Wehrheim**
Department of Computer Science, Paderborn University, Paderborn, Germany

**John Derrick**
Department of Computer Science, University of Sheffield, UK

## Abstract

Correctness conditions like linearizability and opacity describe some form of atomicity imposed on concurrent objects. In this paper, we propose a correctness condition (called causal atomicity) for concurrent objects executing in a weak memory model, where the histories of the objects in question are partially ordered. We establish compositionality and abstraction results for causal atomicity and develop an associated refinement-based proof technique.

**2012 ACM Subject Classification** Theory of computation → Concurrency, Theory of computation → Shared memory algorithms

**Keywords and phrases** Weak Memory, Concurrent Object, Execution Structure

**Digital Object Identifier** 10.4230/LIPIcs.DISC.2018.45

## 1 Foundations

Correctness conditions like *linearizability* for data structures and *opacity* for software transactional memory (STM) specify atomicity for concurrent objects. They are developed under the assumption that the underlying memory model is sequentially consistent. Our objective here is to generalise such notions to weak memory models. We develop a new notion of correctness: *casual atomicity*[3], which we show is compositional and ensures client abstraction. Causal atomicity applies to both concurrent objects and transactional memory, thus it encompasses both linearizability and opacity. Causal linearizability is covered earlier [1]; here we specialise causal atomicity to *causal opacity* and develop a proof technique for it.

Our framework is based on object executions given as *execution structures* [4]. They describe the usual precedence order between events as well as a communication relation. Informally, a communication relation between two events arises when some low-level operation of the first event synchronises with some operation belonging to the second event. Events describe e.g. transactions in an STM or operations on a data structure, and their

[1] Simon Doherty and John Derrick are funded by EPSRC Grant EP/M017044/1.
[2] Funded by EPSRC Grant EP/R019045/1.
[3] This notion is related to, but different from causal atomicity defined by Farzan and Madhusudan [3]. For example, their notion is not compositional.

implementations give arise to the execution of low-level operations. In prior work [1], we have used this technique to define a causal version of linearizability. Here, we generalise the methods and define a notion of atomicity that is also applicable to transactional memory.

▶ **Definition 1.** An *execution structure* is a tuple $(E, \longrightarrow, \dashrightarrow)$ consisting of a finite set of events $E$, a strict order $\longrightarrow \subseteq E \times E$ and a relation $\dashrightarrow \subseteq E \times E$ satisfying:

1. If $t_1 \longrightarrow t_2$, then both $t_1 \dashrightarrow t_2$ and $\neg(t_2 \dashrightarrow t_1)$.
2. If $t_1 \longrightarrow t_2 \dashrightarrow t_3$ or $t_1 \dashrightarrow t_2 \longrightarrow t_3$, then $t_1 \dashrightarrow t_3$.
3. If $t_1 \longrightarrow t_2 \dashrightarrow t_3 \longrightarrow t_4$, then $t_1 \longrightarrow t_4$.

Like other concurrent correctness conditions, we employ some sequential object on a fixed alphabet to compare the executions of the concurrent object against. A sequential object $\mathbb{S}$ specifies a set of *legal* sequential executions denoted $\mathit{legal}_{\mathbb{S}}$. Syntactically, each element of $\mathit{legal}_{\mathbb{S}}$ is a sequence of events that are labelled with operations that the object provides.

## 2 Contributions

**Causal atomicity.** When comparing concurrent executions against sequential ones, some key orders of the concurrent execution need to be preserved. In our case, these are the relations $\longrightarrow$ and $\dashrightarrow$ of the execution structure. We say $<$ is a *logical order* of an execution structure $\mathbb{E} = (E, \longrightarrow, \dashrightarrow)$ iff $< \subseteq E \times E$ is a strict order such that $\longrightarrow \subseteq < \subseteq \dashrightarrow$. For a partial order $< \subseteq E \times E$, we let $\mathit{LE}(<) = \{w \in E^* \mid < \subseteq \twoheadrightarrow_w\}$ be the set of *linear extensions* of $<$, where $\twoheadrightarrow_w$ is the total order corresponding to the (total) order on the elements of $w$.

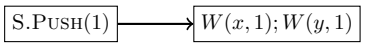
▶ **Definition 2.** Let $\mathbb{S}$ be a sequential object. An execution structure $\mathbb{E}$ is *causally atomic* w.r.t. $\mathbb{S}$ iff there exists a logical order $<$ of $\mathbb{E}$ such that $\mathit{LE}(<) \subseteq \mathit{legal}_{\mathbb{S}}$.

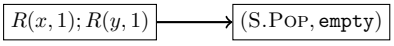
The logical order at least needs to contain the *precedence order* $\longrightarrow$ and at most the *communication relation* $\dashrightarrow$. Note that the logical order (induced by a specification) can introduce additional communication in an implementation (see example below).

**Compositionality.** A key requirement on every correctness condition is *compositionality*: when a program employs operations from two different concurrent objects, these objects individually satisfy the correctness condition if, and only if, their combined usage also satisfies the correctness condition. Technically, we use a composition operator $\otimes$ on sequential objects to compute the interleavings of sequential executions, and an operator $|$ to restrict execution structures to events from a given alphabet.

▶ **Theorem 3.** *Let $\mathbb{S}_1$ and $\mathbb{S}_2$ be sequential objects with disjoint alphabets $\Sigma_1$ and $\Sigma_2$ and let $\mathbb{E} = (E, \longrightarrow, \dashrightarrow)$ be an execution structure. Then $\mathbb{E}_{|\Sigma_1}$ and $\mathbb{E}_{|\Sigma_2}$ are causally atomic w.r.t. $\mathbb{S}_1$ and $\mathbb{S}_2$, respectively, iff $\mathbb{E}$ is causally atomic w.r.t. $\mathbb{S}_1 \otimes \mathbb{S}_2$.*

For example, consider the execution structure below, which comprises a stack object $S$ and an STM, where we assume memory values are initialised to 0. The event $W(x,1); W(y,1)$ corresponds to a transaction consisting of a write to $x$ followed by a write to $y$. Similarly, $R(x,1); R(y,1)$ corresponds to a transaction comprising a read of $x$ followed by a read of $y$.




The execution should not be allowed since there is no total ordering of events that respects the existing precedence order $\longrightarrow$. Causal atomicity necessitates a communication relation $W(x,1); W(y,1) \dashrightarrow R(x,1); R(y,1)$ which, together with axiom **A3** induces order

S.Push(1) $\longrightarrow$ (S.Pop, `empty`). Thus, when considering the full execution structure the composition execution restricted to the stack object is not causally atomic. A more detailed example composition of concurrent objects is given in [1].

**Abstraction.** Another property of causal atomicity is *abstraction*, which formalises a notion of substitutability: when a client uses the operations that an object provides in its interface, then it should not be able to distinguish the implementation from its sequential specification. To formalise this, we represent a client $\mathcal{C}$ as a set of *client executions*, each of which is simply a partial order over events labelled with operations from the alphabet of the object and other client-local events such as reads and writes to client variables. Suppose $\mathbb{O}$ is a set of execution structures over $\Sigma$. The *client-object composition* of $\mathcal{C}$ and $\mathbb{O}$ is the set $\mathcal{C}[\![\mathbb{O}]\!] = \{\prec \in \mathcal{C} \mid \exists\, ((dom(\prec) \cup ran(\prec)) \cap \Sigma, \longrightarrow, \dashrightarrow) \in \mathbb{O}.\ \prec_{|\Sigma} \subseteq \longrightarrow\}$.

Given a sequential object $\mathbb{S}$, we let $CA[\mathbb{S}]$ be the set of execution structures that are causally atomic w.r.t. $\mathbb{S}$. Furthermore, we lift sequential objects to sets of execution structures by letting $E[\mathbb{S}]$ be the set of execution structures such that there is some $w \in legal_{\mathbb{S}}$ where *both* the precedence order and communication relation is the total order $\twoheadrightarrow_w$. Our goal is to compare, for any client $\mathcal{C}$, the client-object composition $\mathcal{C}[\![CA[\mathbb{S}]]\!]$ with the corresponding composition with the sequential object, $\mathcal{C}[\![E[\mathbb{S}]]\!]$. To do so, we define a notion of *observational refinement*, denoted $\sqsubseteq$, such that $\mathcal{C}[\![CA[\mathbb{S}]]\!] \sqsubseteq \mathcal{C}[\![E[\mathbb{S}]]\!]$ holds if for every execution in $\mathcal{C}[\![CA[\mathbb{S}]]\!]$, there is an *observationally equivalent* execution in $\mathcal{C}[\![E[\mathbb{S}]]\!]$. Our notion of observational equivalence requires that any pair of equivalent executions must have compatible orders when restricted to client-local events. We then prove the following.

▶ **Theorem 4.** *If $\mathcal{C}$ is a client and $\mathbb{S}$ a sequential object, then $\mathcal{C}[\![CA[\mathbb{S}]]\!] \sqsubseteq \mathcal{C}[\![E[\mathbb{S}]]\!]$.*

**Transactional Memory.** We use causal atomicity to obtain a correctness condition for transactional memory, which we call *causal opacity*. To do so, we first define a *transactional sequential object*, denoted $\mathbb{T}$, whose alphabet is made up of entire *blocks* of reads and writes, and whose semantics requires that each block executes atomically. Causal opacity itself is a condition on transactional operations rather than atomic blocks, thus allowing concurrent transactions. It is defined in terms a transformation called *$\mu$-abstraction* [4], that combines these operations into a block from the alphabet of $\mathbb{T}$. An execution structure is said to be causally opaque whenever its $\mu$-abstraction is causally atomic w.r.t. $\mathbb{T}$.

We show how to verify causal opacity using a sequential object, $\mathbb{C}$, such that the $\mu$-abstraction of every execution structure in $CA[\mathbb{C}]$ is causally atomic w.r.t. $\mathbb{T}$. The object $\mathbb{C}$ is adapted from TMS2 [2], and like its predecessor, is given in an operational fashion which enables simulation-based refinement proofs.

## References


1 S. Doherty, B. Dongol, H. Wehrheim, and J. Derrick. Making Linearizability Compositional for Partially Ordered Executions. In *iFM*, volume 11023 of *LNCS*, 2018.

2 S. Doherty, L. Groves, V. Luchangco, and M. Moir. Towards formally specifying and verifying transactional memory. *Formal Asp. Comput.*, 25(5):769–799, 2013.

3 A. Farzan and P. Madhusudan. Causal atomicity. In *CAV*, volume 4144 of *LNCS*, pages 315–328. Springer, 2006.

4 L. Lamport. On interprocess communication. part I: basic formalism. *Distributed Computing*, 1(2):77–85, 1986.