Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Non-interference and local correctness in transactional memory[☆,☆☆]

Petr Kuznetsov[a,1], Sathya Peri[b,*,2]

[a] Télécom ParisTech, Paris, France
[b] CSE Dept, IIT Hyderabad, Hyderabad, India

A R T I C L E I N F O

Article history:
Received 14 April 2014
Received in revised form 25 November 2015
Accepted 15 June 2016
Available online xxxx

Keywords:
Software transactional memory
Correctness criterion
Opacity
Non-interference
Permissiveness

A B S T R A C T

Transactional memory promises to make concurrent programming tractable and efficient by allowing the user to assemble sequences of actions in atomic *transactions* with all-or-nothing semantics. It is believed that, by its very virtue, transactional memory must ensure that all *committed* transactions constitute a serial execution respecting the real-time order. In contrast, aborted or incomplete transactions should not "take effect." But what does "not taking effect" mean exactly?

It seems natural to expect that aborted or incomplete transactions do not appear in the global serial execution, and, thus, no committed transaction can be affected by them. We investigate another, less obvious, feature of "not taking effect" called *non-interference*: aborted or incomplete transactions should not force any other transaction to abort. In the strongest form of non-interference that we explore in this paper, by removing a subset of aborted or incomplete transactions from the history, we should not be able to turn an aborted transaction into a committed one without violating the correctness criterion.

We show that non-interference is, in a strict sense, not *implementable* with respect to the popular criterion of opacity that requires *all* transactions (be they committed, aborted or incomplete) to witness the same global serial execution. In contrast, when we only require *local* correctness, non-interference is implementable. Informally, a correctness criterion is local if it only requires that every transaction can be serialized along with (a subset of) the transactions committed before its last event (aborted or incomplete transactions ignored). We give a few examples of local correctness properties, including the recently proposed criterion of virtual world consistency, and present a simple though efficient implementation that satisfies non-interference and *local opacity*.

1. Introduction

Transactional memory (TM) promises to make concurrent programming efficient and tractable. The programmer simply represents a sequence of instructions that should appear atomic as a speculative *transaction* that may either *commit* or

☆ A preliminary version of this paper has been published in ICDCN 2014.
☆☆ This work was supported in part by DAAD-IIT Faculty Exchange Program.
* Corresponding author.
E-mail addresses: petr.kuznetsov@telecom-paristech.fr (P. Kuznetsov), sathya_p@iith.ac.in (S. Peri).
[1] The author is supported by the Agence Nationale de la Recherche, under grant agreement ANR-14-CE35-0010-01, project DISCMAT.
[2] The author is supported in part by NetApp Faculty Fellowship.

http://dx.doi.org/10.1016/j.tcs.2016.06.021
0304-3975/© 2016 Elsevier B.V. All rights reserved.

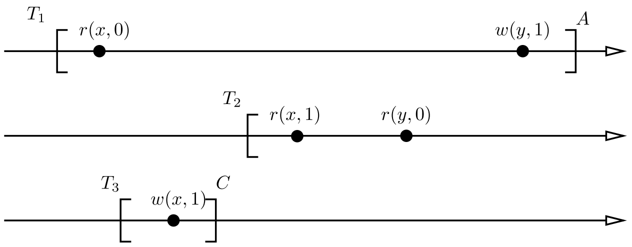

Fig. 1. An opaque-permissive opaque but not opaque-non-interfering history: T_2 forces T_1 to abort.

abort. It is usually expected that a TM *serializes* all committed transactions, i.e., makes them appear as in some sequential execution. An implication of this requirement is that no committed transaction can read values written by a transaction that is aborted or might abort in the future. Intuitively, this is a desirable property because it does not allow a write performed within a transaction to get "visible" as long as there is a chance for the transaction to abort.

But is this all we can do if we do not want aborted or incomplete transactions to "take effect"? We observe that there is a more subtle side of the "taking effect" phenomenon that is usually not taken into consideration. An incomplete or aborted transaction may cause another transaction to abort. Suppose we have an execution in which an aborted transaction T cannot be committed without violating correctness of the execution, but if we remove some incomplete or aborted transactions, then T can be committed. This property, originally highlighted in [1], is called *non-interference*.

Thus, ideally, a TM must "insulate" transactions that are aborted or might abort in the future from producing any effect, either by affecting reads of other transactions or by provoking forceful aborts.

Defining non-interference. Consider first non-interference as a characteristics of an *implementation*. For example, we may say that a TM implementation M is non-interfering if removing an aborted or incomplete transaction from a *history* (a sequence of events on the TM interface) of M would still result in a history of M. We observe that many existing TM implementations that employ *commit-time* lock acquisition or version update (e.g., TL2 [2] or NOrec [3]) are non-interfering in this sense. Some *encounter-time* implementations, such as TinySTM [4], are not non-interfering. But restricting ourselves by the set of histories of a given implementation, we do not seem to derive any insights about *non-interference* itself.

This paper rather focuses on non-interference as a characteristics of a *correctness criterion*, which results in a much stronger restriction on implementations. We intend to understand whether this strong notion of non-interference is achievable and at what cost, which we believe to be a challenging theoretical question. For a given correctness criterion P, a TM implementation M is P-non-interfering if removing one aborted transaction from any history of M does not allow committing another aborted transaction while still preserving P. We observe that P-non-interference produces a subset of *permissive* [5] with respect to P histories. This is not difficult to see if we recall that in a permissive (with respect to P) history, no aborted transaction can be turned into a committed one while still satisfying P.

Therefore, what we end up with is a very strong restriction on the set of histories. In particular, when we focus on *opaque* histories [6,7], we observe that non-interference gives a *strict* subset of permissive opaque histories. Opacity requires that all transactions (be they committed, aborted, or incomplete) constitute a consistent sequential execution in which every read returns the latest committed written value. This is a strong requirement, because it expects every transaction (even aborted or incomplete) to witness the same sequential execution. Indeed, there exist permissive opaque histories that do not provide non-interference: some aborted transactions force other transactions to abort.

For example, consider the history in Fig. 1. Here the very fact that the incomplete operation T_2 read the "new" (written by T_3) value in object x and the "old" (initial) value in object y prevents an updating transaction T_1 from committing. Suppose that T_1 commits. Then T_2 can only be *serialized* (put in the global sequential order) after T_3 and before T_1, while T_1 can only be serialized before T_3. Thus, we obtain a cycle which prevents any serialization. Therefore, the history does not provide opaque-non-interference: by removing T_2 we can commit T_1 by still allowing a correct serialization T_1, T_3. But the history is permissive with respect to opacity: no transaction aborts without a reason!

This example can be used to show that opaque-non-interference is, in a strict sense, *non-implementable*. Every opaque permissive implementation that guarantees that every transactional operation (*read*, *write*, *tryCommit* or *tryAbort*) completes if it runs in the absence of concurrency (note that it can complete with an *abort* response), may be brought to the scenario above, where the only option for T_1 in its last event is *abort*.

Local correctness. But are there weaker definitions of TM correctness that allow for non-interfering implementations? Intuitively, the problem with the history in Fig. 1 is that T_2 should be consistent with a global order of *all* transactions. But what if we only expect every transaction T to be consistent *locally*, i.e., to fit to *some* serialization composed of the transactions that committed before T terminates? This way a transaction does not have to account for transactions that are aborted or incomplete at the moment it completes. Consequently, local serializations for different transactions do not have to be mutually consistent.

For example, the history in Fig. 1, assuming that T_1 commits, is still *locally* opaque: the local serialization of T_2 would simply be $T_3 \cdot T_2$, while T_1 (assuming it commits) and T_3 would both be consistent with the serialization $T_1 \cdot T_3$.