

Safe Open-Nested Transactions

Bradley C. Kuzmaul

Charles E. Leiserson

Jim Sukha

MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA 02139, USA

Open-nested transactions [2–5] have been proposed as a loophole for transactional memory (TM) to increase concurrency on highly contended resources in transactional programs. Programs that use open nesting can be difficult to reason about because open nesting breaks serializability at the level of memory semantics. Evidence suggests that an unconstrained use of open nesting cannot be encapsulated, i.e., that programmers may need to be aware of whether subroutines contain open-nested transactions. Despite this adverse outlook, we are optimistic that serializability can be recovered abstractly in a TM system by providing the programmer with simple but unobtrusive guidelines and language constructs that provably encapsulate the effects of open-nested transactions.

Although ordinary TM with only closed nesting presents the programmer with simple semantics, these semantics provide no way for the programmer to tell the TM system to ignore a conflict which abstractly he/she deems as safe. Proposed hardware open nesting mechanisms (e.g., [2–4]) allow programmers to avoid such conflicts by breaking physical serializability. By allowing some nonserializable behaviors, however, these mechanisms also permit other arguably anomalous behaviors, even in the case where all transactions commit [1].

For example, Figure 1 shows a program execution in which transaction *A* is not physically serializable. After *A*'s open-nested transaction *B* commits, it is possible for a concurrent transaction *C* to depend on values committed by *B* (e.g., *w*), and then itself commit and change values (e.g., *y*) which the remainder of *A* depends on. Existing hardware open nesting proposals do not detect a conflict between *A* and *C*; therefore, these systems permit a program execution in which *A* is neither physically serializable before or after *C*.

TM systems which guarantee only a transactional memory model of *prefix race-freedom* permit program executions such as the one in Figure 1. Our prior work [1] formally defines prefix race-freedom and shows that the open-nesting semantics of McDonald *et al.* [2] implements this memory model. We also prove that prefix race-freedom and serializability are equivalent for programs with only closed-nested transactions, but different for programs with open-nested transactions.

Unlike TM without open nesting, a system that guarantees only prefix race-freedom no longer provides guarantees of modularity and composability. With unconstrained sharing of data between transactions, the correctness of a transaction *A* with an open-nested transaction *B* no longer depends locally on *A*'s code because other transactions *C* may interleave with *A*'s execution. In fact, this problem extends to any of *A*'s ancestor transactions. Thus, a programmer using a subroutine needs to be aware whether that subroutine contains open-nested transactions. Without higher-level abstractions, programmers can not easily reason about their code because the effects of open-nested transactions cannot be encapsulated.

One way to eliminate some anomalous program behaviors is to use the structure of an object-oriented language to limit data sharing between transactions. For example, Ni *et al.* [5] describes a software TM implementation of open nest-

```
// Thread 1                                // Thread 2
1 atomic { // A                             6 atomic { // C
2   int a = x;                               7   w++;
3   open_atomic { // B                       8   y = w;
4     w++;                                     9   }
5   }
10  int b = y;                                // w, x, y, and z are
11  z = x + y;                                // global variables
12 }
```

Figure 1. A program execution allowed by hardware open nesting in which transaction *A* is not serializable.

ing which breaks physical serializability, but provides some high-level language constructs to enable encapsulation. This system requires programmers to incorporate high-level application semantics when using open transactions; in particular, programmers must specify constructs such as compensating transactions and abstract locks. Although this system uses language constructs to prohibit some anomalous behaviors for open nesting, it is unclear under what conditions the system can provably guarantee that the effects of open-nested transactions are fully encapsulated. Since this system's semantics are still complicated, use of open nesting in this system appears to be limited to expert programmers writing libraries.

Theoretical models can provide a foundation for understanding how modularity and composability might be provided in a TM system with loopholes. A theoretical exploration should facilitate the design of a structured programming model that provides programmers with provable guarantees they can use to reason about their code. Moreover, by understanding the abstract semantics of transactional memory with open nesting, we believe that other unsolved problems in TM, such as how to incorporate I/O within a transaction, can be fruitfully addressed.

REFERENCES

- [1] K. Agrawal, C. E. Leiserson, and J. Sukha. Memory models for open-nested transactions. In *MSPC*, Oct. 2006.
- [2] A. McDonald, J. Chung, B. D. Carlstrom, C. Cao Minh, H. Chafi, C. Kozyrakis, and K. Olukotun. Architectural semantics for practical transactional memory. In *ISCA*, June 2006.
- [3] M. J. Moravan, J. Bobba, K. E. Moore, L. Yen, M. D. Hill, B. Liblit, M. M. Swift, and D. A. Wood. Supporting nested transactional memory in LogTM. In *ASPLOS*, pages 359–370, New York, 2006.
- [4] J. E. B. Moss and A. L. Hosking. Nested transactional memory: Model and architecture sketches. In *Science of Computer Programming*, volume 63, pages 186–201. Elsevier, Dec 2006.
- [5] Y. Ni, V. Menon, A. Adl-Tabatabai, A. L. Hosking, R. L. Hudson, J. E. B. Moss, B. Saha, and T. Shpeisman. Open nesting in software transactional memory. In *PPOPP*, Mar. 2007.