Corrections to the paper:
Overcoming Memory Weakness with Unified Fairness

Parosh Abdulla

Mohamed Faouzi Atig

Adwait Godbole

Shankaranarayanan Krishna

Mihir Vahanwala

[Published] Abdulla, P.A., Atig, M.F., Godbole, A., Krishna, S., Vahanwala, M. (2023). Overcoming Memory Weakness with Unified Fairness. In: Enea, C., Lal, A. (eds) Computer Aided Verification. CAV 2023. Lecture Notes in Computer Science, vol 13964. Springer, Cham. https://doi.org/10.1007/978-3-031-37706-8_10
[Full version] Abdulla, P.A., Atig, M.F., Godbole, A., Krishna, S., Vahanwala, M.: Overcoming memory weakness with unified fairness (2023). https://arxiv.org/abs/2305.17605

The inaccuracy

In Section 3.3 of the paper, and subsequently in Appendix A, we discuss the instantiation of the framework to various memory models.

However, the instantiations to FIFO and RMO are incorrect, and misrepresent the models. This was due to misunderstandings, for which we apologise.

It is also worth remarking here that the framework, as presented in the paper, applies only to models that prohibit reads from racing. The framework cannot be applied as is to models such as RMO and ARM.

However, this error does not take away from the main contribution of the paper, viz. identifying a perspective that naturally lends itself to reasonable and algorithmically amenable definitions of fairness and thus lays the foundation for the verification of liveness in the setting of weak memory.

Sketching an extension of the framework

In order to handle models like RMO and ARM, where reads are allowed to race (however, writes by the same process to the same variable still do not race), the framework can be augmented by a data structure that keeps a (heap-like) buffer of pending reads, with some constraints to keep track of which overtakes are prohibited by dependencies caused by program semantics.

In a transition, a pending read can be popped from the top of the (heap-)buffer, and justified using a write in a message channel visible to the process.

This retains the ideas central to the fairness definitions: we only need keep track of information that affects the execution's future (pending reads, writes yet to become stale), thereby leading to a notion of configuration size. The arguments that motivate bounding the configuration size still hold, and hence **Section 4, which was the main novelty, continues to hold** as well.

In this document…

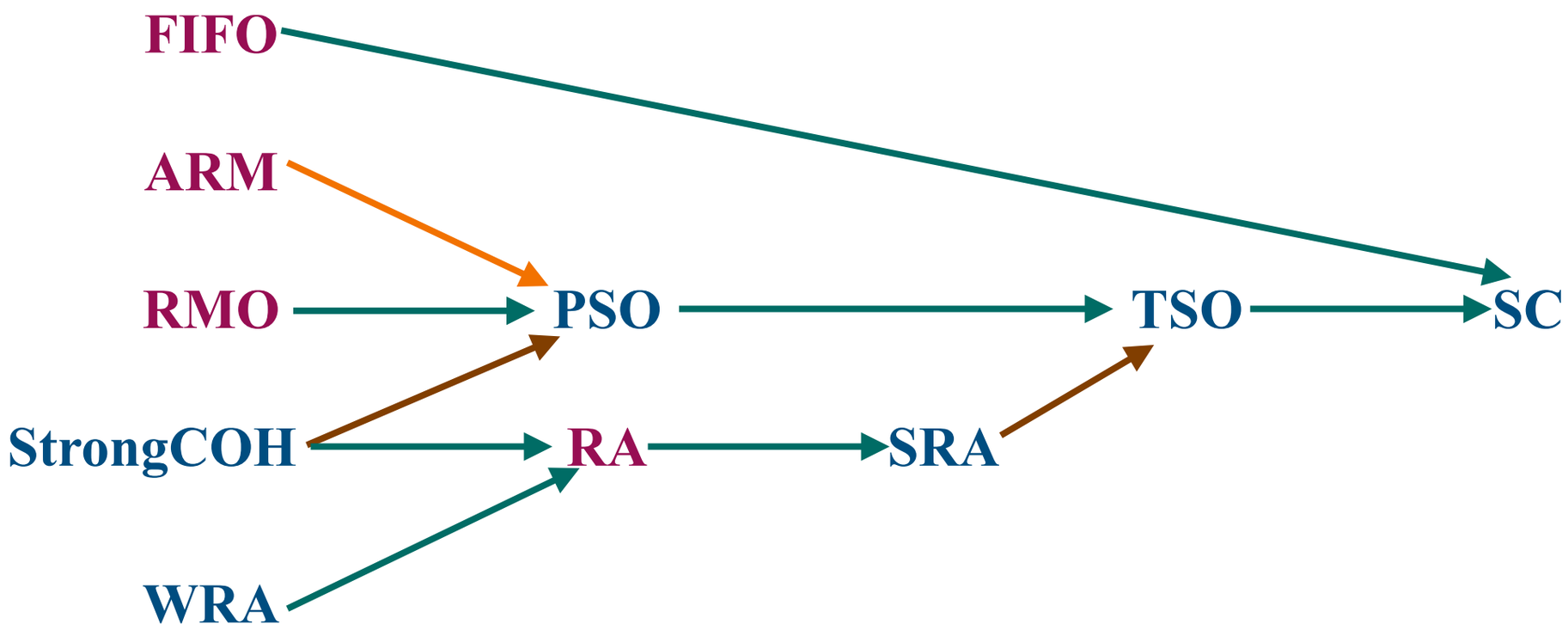
Precise corrections to the instantiations, and a discussion of the correct RMO and ARM, will be conducted in a carefully revised version

In this document, we shall address the inaccuracies in Section 3.3 of the original.

We specify sources defining the models we consider, briefly justify relative strengths when comparable, and give litmus tests demonstrating increasing strength, or incomparability.

Model	See for definition and/or alternate semantics
Sequential Consistency (SC)	[1] Leslie Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. IEEE Trans. Comput. C-28,9 (Sept. 1979), 690-691.
Total Store Order (TSO)	[2] Mohamed Faouzi Atig, Ahmed Bouajjani, Sebastian Burckhardt, and Madanlal Musuvathi. 2010. On the verification problem for weak memory models. SIGPLAN Not. 45, 1 (January 2010), 7–18. https://doi.org/10.1145/1707801.1706303, [4, Section 8, Appendix D]
Strong Release-Acquire (SRA)	[3, Sections 3-4] Ori Lahav and Udi Boker. 2022. What's Decidable About Causally Consistent Shared Memory? ACM Trans. Program. Lang. Syst. 44, 2, Article 8 (June 2022), 55 pages. https://doi.org/10.1145/3505273
Release-Acquire (RA)	[3, Sections 3-4], [5, Section 2.2]
Partial Store Order (PSO)	[4, Section 8, Appendix D] CORPORATE SPARC International, Inc. 1994. The SPARC architecture manual (version 9). Prentice-Hall, Inc., USA. [2]
Weak Release-Acquire (WRA)	[3, Sections 3-4]
Strong Coherence (StrongCOH)	[5, Section 2.3] Ori Lahav, Egor Namakonov, Jonas Oberhauser, Anton Podkopaev, and Viktor Vafeiadis. 2021. Making weak memory models fair. Proc. ACM Program. Lang. 5, OOPSLA, Article 98 (October 2021), 27 pages. https://doi.org/10.1145/3485475
ARM	[6, Sections 4, 6] Jade Alglave, Will Deacon, Richard Grisenthwaite, Antoine Hacquard, and Luc Maranget. Armed Cats: Formal Concurrency Modelling at Arm. ACM Trans. Program. Lang. Syst. 43, 2, Article 8 (June 2021), 54 pages. https://doi.org/10.1145/3458926
Relaxed Memory Ordering (RMO)	[4, Section 8, Appendix D]
FIFO Consistency	[7] Lipton, R., Sandberg, J.: PRAM: a scalable shared memory. Technical report CS-TR-180-88, Princeton University (1988) [8] Ahamad, M., Neiger, G., Burns, J.E. *et al*. Causal memory: definitions, implementation, and programming. *Distrib Comput* **9**, 37–49 (1995). https://doi.org/10.1007/BF01784241

Relative Strength of Memory Models: An arrow from A to B denotes that all behaviours of B are allowed by A.



Blue denotes that the underlying reachability is decidable, purple denotes it is undecidable.

Turquoise arrows indicate that relative strength follows from **design**.
The **orange** arrow indicates the enforcement of **acquire semantics** on reads.
Brown arrows indicate the enforcement of **multi copy atomicity** on the memory model.

(1) {SRA, FIFO}, not {RMO, ARM}

```
x = 1  ||  y = 1  ||  a1 = y  //1        ||  b1 = x  //1
                      membar              ||  membar
                      a2 = x  //0         ||  b2 = y  //0
                      membar              ||  membar
                      a3 = x  //1         ||  b3 = y  //1
```

(2) TSO, not FIFO

```
x = 1          ||  a1 = x  //1
y = 1          ||  x = 3
x = 2          ||  a2 = y  //0
y = 2          ||  a3 = y  //2
               ||  a4 = x  //3
```

(3) ARM, not {RMO, FIFO, StrongCOH, WRA}

```
x = 1
a = x  //0
```

(4) RMO, not {ARM, FIFO, StrongCOH, WRA}

```
a1 = x  //1    ||  b1 = y  //1
a2 = *a1  //0  ||  b2 = *b1  //0
y = 1          ||  x = 1
```

(5) PSO, not {FIFO, WRA}

```
x = 1  ||  a1 = y  //1
y = 1  ||  a2 = x  //0
```

(6) FIFO, not WRA

```
x = 1  ||  a = x  //1  ||  b1 = y  //1
       ||  y = 1       ||  b2 = x  //0
```

(7) {FIFO, WRA, RMO, ARM}, not StrongCOH

```
x = 1  ||  x = 2  ||  a1 = x  //1  ||  b1 = x  //2
                      a2 = x  //2  ||  b2 = x  //1
```

(8) RA, not SRA

```
x = 1          ||  y = 1
y = 2          ||  x = 2
a = y  //1     ||  b = x  //1
```

The instruction **`a = *b`** reads the data at memory location whose address is stored in register **`b`** into register **`a`**

Entry in Column A, Row B is number j: Litmus test j is allowed by A but prohibited by B

	SC	TSO	SRA	RA	PSO	WRA	StrongCOH	ARM	RMO	FIFO
SC	-	2	2	2	2	2	2	3	4	1
TSO	**X**	-	1	1	5	1	1	3	4	1
SRA	X	**X**	-	**8**	5	7	5	3	4	6
RA	X	X	**X**	-	5	7	5	3	4	6
PSO	X	**X**	1	1	-	7	1	3	4	1
WRA	X	X	X	**X**	**5**	-	5	**3**	**4**	**6**
StrongCOH	X	X	X	**X**	**X**	**7**	-	**3**	**4**	**7**
ARM	X	X	**1**	1	**X**	1	1	-	**4**	**1**
RMO	X	X	**1**	1	**X**	1	1	**3**	-	**1**
FIFO	**X**	**2**	2	2	2	2	2	**3**	**4**	-

Entry in Column A, Row B is X: Any behaviour allowed by A is also allowed by B