Armed cats

Jade Alglave

February 12, 2015

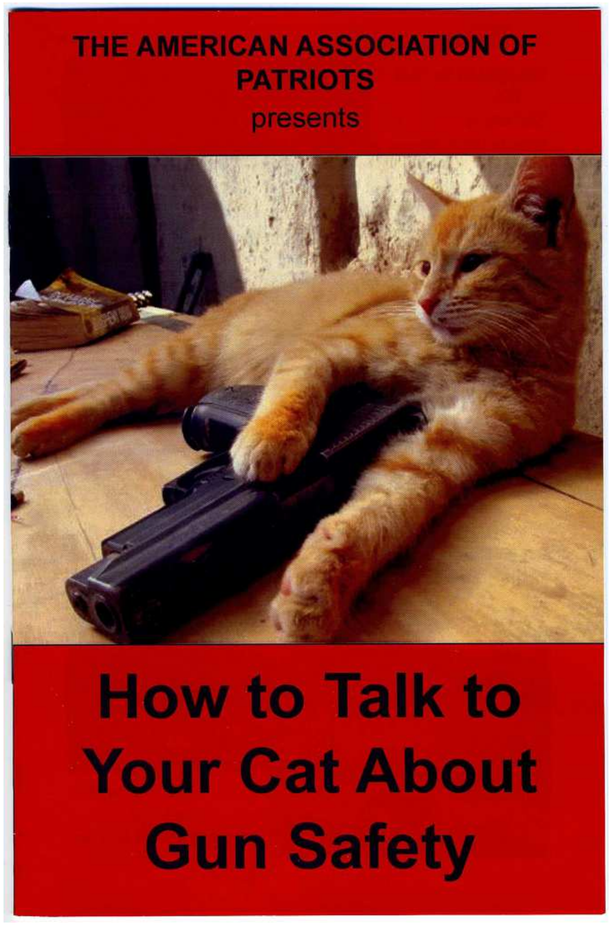

THE AMERICAN ASSOCIATION OF PATRIOTS
presents
How to Talk to Your Cat About Gun Safety

Figure 1: A real book I have at home

1 Preamble

This note presents a certain number of ARM models, developed by Luc Maranget and myself. Most of the text is heavily inspired by our Herding Cats paper [2].

In Sec. 2, I give the axiomatic ARM model that we propose in [2], in our home made `cat` format. I also give a table that summarises which scenarios are forbidden by which axiom of our model.

In Sec. 3, 4, 5, 6, I detail each axiom and scenarios.

In Sec. 7, I give an equivalent ARM model in `cat` format, that is somewhat more simple.

In Sec. 8, I argue that one might not prefer the reduced ARM model, given that recent work with Luc Maranget seems to indicate that the initial model of [2] is easier to extend to, e.g., ARMv8 features.

I have put all the models discussed in this note, as well as some litmus tests, on the web interface of our herd simulator:

http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=MP

It might useful to go through this note with the web interface at hand.

2 At a glance

axiom	forbidden scenarios
SC PER LOCATION	coWW, coRW1, coRW2, coWR, coRR
NO THIN AIR	lb+ppos
OBSERVATION	mp+lwfence+ppo, wrc+lwfence+ppo, isa2+lwfence+ppos
PROPAGATION	s+lwfence+ppo, 2+2w+lwfences, w+rw+2w+lwfences sb+ffences, rwc+ffences, iriw+ffences, r+ffences

Table 1: Summary

```
"ARM"

let com = rf | co | fr
acyclic po-loc | com as sc-per-location

let dp = addr | data
let rdw = po-loc & (fre;rfe)
let detour = po-loc & (coe ; rfe)
let ctrlcfence = ctrlisb

let ii0 = dp | rfi | rdw
let ic0 = 0
let ci0 = ctrlcfence | detour
let cc0 = dp | ctrl | (addr;po)

let rec ii = ii0 | ci | (ic;ci) | (ii;ii)
and ic = ic0 | ii | cc | (ic;cc) | (ii;ic)
and ci = ci0 | (ci;ii) | (cc;ci)
and cc = cc0 | ci | (ci;ic) | (cc;cc)

let ppo = ii & R*R | ic & R*W

let dmb.st = dmb.st & W*W
let dsb.st = dsb.st & W*W
let fence = dmb | dsb | dmb.st | dsb.st
let A-cumul = rfe;fence

let hb = ppo | fence | rfe
acyclic hb as no-thin-air

let prop-base = (fence | A-cumul);hb*
let prop = (prop-base & W*W)| (com*; prop-base*; fence; hb*)

irreflexive fre;prop;hb* as observation
acyclic co | prop as propagation
```

Figure 2: ARM cat file (`arm-from-paper.cat` on the web)

3 SC per location

```
let com = rf | co | fr
acyclic po-loc | com as sc-per-location
```

Figure 3: SC per location fragment of the ARM cat file

SC PER LOCATION ensures that the communications com cannot contradict po-loc (program order between events relative to the same memory location), i.e. acyclic(po-loc $\cup$ com). This axiom forbids exactly (as shown in [1, A.3 p. 184]) the five scenarios coWW, coRW1, coRW2, coWR, coRR given in Fig. 4.

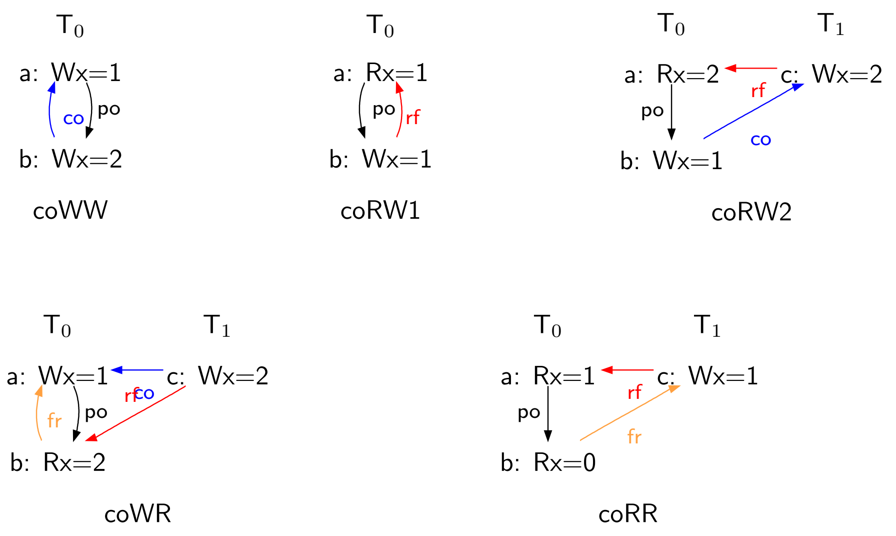


Figure 4: The five scenarios forbidden by SC PER LOCATION

The scenario coWW forces two writes to the same memory location x in program order to be in the same order in the coherence order co. The scenario coRW1 forbids a read from x to read from a po-subsequent write. The scenario coRW2 forbids the read a to read from a write c which is co-after a write b, if b is po-after a. The scenario coWR forbids a read b to read from a write c which is co-before a previous write a in program order. The scenario coRR imposes that if a read a reads from a write c, all subsequent reads in program order from the same location (e.g. the read b) read from c or a co-successor write.

On the web interface:

- coWW: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=coWW
- coRW1: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm-from-paper&cat=arm&litmus=coRW1
- coRW2: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=coRW2

- coWR: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=coWR
- coRR: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=coRR

4 No thin air

```
let hb = ppo | fence | rfe
acyclic hb as no-thin-air
```

Figure 5: No thin air fragment of the ARM cat file

NO THIN AIR defines a *happens-before* relation hb, defined as ppo $\cup$ fence$\cup$rfe, i.e. an event e_1 happens before another event e_2 if they are in preserved program order, or there is a fence in program order between them, or e_2 reads from e_1. NO THIN AIR requires the happens-before relation to be acyclic, which prevents values from appearing out of thin air. This axiom typically forbids load buffering scenarios such as lb+ppos as shown in Fig. 6.

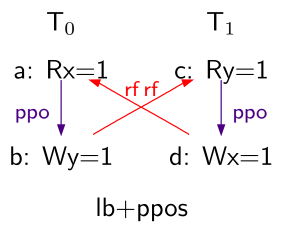


Figure 6: The load buffering scenario lb with ppo on both sides (forbidden)

T_0 reads from x and writes to y, imposing (for example) an address dependency between the two accesses, so that they cannot be reordered. Similarly T_1 reads from y and (dependently) writes to x. NO THIN AIR ensures that the two threads cannot communicate in a manner that creates a happens-before cycle, with the read from x on T_0 reading from the write to x on T_1, whilst T_1 reads from T_0's write to y.

On the web interface:

- lb+addrs: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=LB+addrs
- lb+addrs: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=LB+dmbs
- lb+addrs: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=LB+dsb+addr

Note that address and data dependencies are observationally different.

Consider the variant lb+addrs+ww in Fig. 7: it is forbidden by our definition of ppo, since we put `addr;po` in ppo (see line 14 of the ARM cat file of Fig. 2). However, if we replace address dependencies by data dependencies in Fig. 7, to produce the lb+datas+ww variant, the scenario is allowed by our model, and observed on ARM hardware (see http://diy.inria.fr/cats/data-addr/).

On the web interface:

- lb+addrs+ww: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=LB+addrs+WW

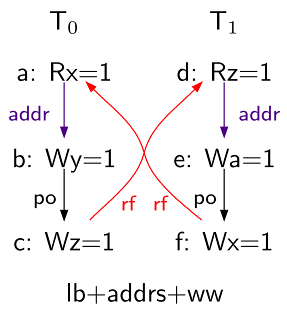


Figure 7: The lb+addrs+ww variant of the load buffering pattern lb (forbidden)

- lb+addrs: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=LB+datas+WW

5 Observation

```
let fence = dmb | dsb | dmb.st | dsb.st
let A-cumul = rfe;fence
...
let prop-base = (fence | A-cumul);hb*
let prop = (prop-base & W*W)| (com*; prop-base*; fence; hb*)

irreflexive fre;prop;hb* as observation
```

Figure 8: Observation fragment of the ARM cat file

OBSERVATION constrains the value of reads. If a write a to x and a write b to y are ordered by the propagation order prop, and if a read c reads from the write of y, then any read d from x which happens after c (i.e. $(c, d) \in$ hb) cannot read from a write that is co-before the write a (i.e. $(d, a) \notin$ fr).

This axiom typically forbids the scenarios mp+fence+ppo, wrc+fence+ppo, isa2+fence+ppos as shown in Fig. 9, 10 and 11.

5.1 Message passing (mp+fence+ppo)

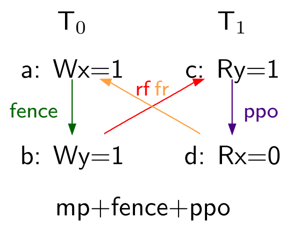


Figure 9: The message passing scenario mp with fence and ppo (forbidden)

T_0 writes a message in x, then sets up a flag in y, so that when T_1 sees the flag (via its read from y), it can read the message in x. For this scenario to behave as intended, following the message passing protocol described above, the write to x needs to be propagated to the reading thread before the write to y.

The protocol would also be ensured with two fences (mp+fences)

On the web interface:

- mp+dsb+addr: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=MP+dmb+addr
- mp+dmbs: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=MP+dmbs

5.2 Write-to-read causality (**wrc+fence+ppo**)

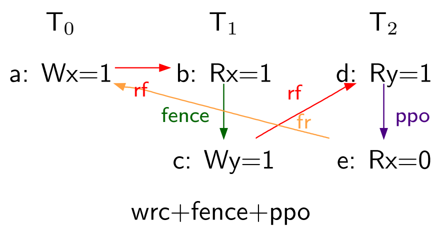


Figure 10: The write-to-read causality scenario with fence and ppo (forbidden)

This scenario distributes the message passing over three threads instead of two, and illustrates the A-cumulativity of the fence on T$_1$, namely the `A-cumul` fragment of the cat file in Fig. 2.

On the web interface:

- wrc+dsb+addr: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=WRC+dmb+addr
- wrc+dmb+addr: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=WRC+dsb+addr

5.3 Power ISA2 (isa2+fence+ppos)

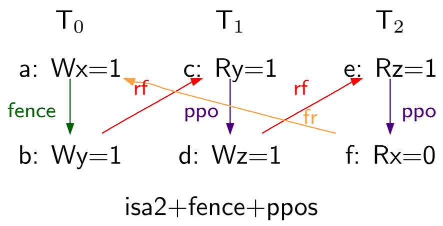


Figure 11: The scenario isa2 with fence and ppo twice (forbidden)

This scenario, given in Fig. 11), distributes the message passing over three threads like wrc+fence+ppo, but keeping the writes to x and y on the same thread.

On the web interface:

- isa2+dmb+addr+addr: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=ISA2+dmb+addr+addr
- isa2+dsb+data+addr: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=ISA2+dsb+data+addr

6 Propagation

```
31 let prop-base = (fence | A-cumul);hb*
32 let prop = (prop-base & W*W)| (com*; prop-base*; fence; hb*)
...
35 acyclic co | prop as propagation
```

Figure 12: Propagation fragment of the ARM cat file

PROPAGATION constrains the order in which writes to memory are propagated to the other threads, so that it does not contradict the coherence order, i.e. acyclic(co ∪ prop). Fences sometimes constrain the propagation of writes, as in the cases of mp (see Fig. 9), wrc (see Fig. 10) or isa2 (see Fig. 11). They can also, in combination with the coherence order, create new ordering constraints.

6.1 s+fence+ppo

In the scenario s+fence+ppo, T_1 reads from y, reading the value stored by the write b, and then writes to x. A fence on T_0 forces the write a to x to propagate to T_1 before the write b to y. Thus, as T_1 sees the write b by reading its value (read c) and as the write d is forced to occur by a dependency (ppo) after the read c, that write d cannot co-precede the write a.

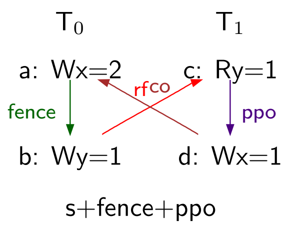


Figure 13: The scenario s with fence and ppo (forbidden)

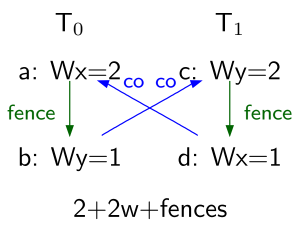
On the web interface:

- s+dmb+addr: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=S+dmb+addr
- s+dsb.st+addr: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=S+dsb.st+addr

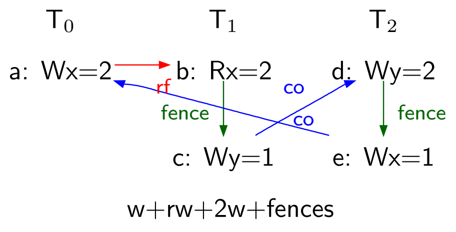
6.2 2+2w+fences and w+rw+2w+fences

The 2+2w+fences pattern (given in Fig. 14(a)) is for us the archetypal illustration of coherence order and fences interacting to yield new ordering constraints. This scenario is forbidden.

The w+rw+2w+fences scenario in Fig. 14(b) distributes 2+2w+fences over three threads. This scenario is to 2+2w+fences what wrc is to mp. Thus, just as in the case of mp and wrc, the fence plays a cumulative role, which ensures that 2+2w and w+rw+2w respond to the fence in the same way.



(a) The scenario 2+2w+fences



(b) The scenario w+rw+2w+fences

Figure 14: Two similar scenarios with fences (forbidden)

On the web interface:

- 2+2w+dmb+dmb.st: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=2+2W+dmb+dmb.st
- 2+2w+dmb+dsb: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=2+2W+dmb+dsb

6.3 Store buffering (sb+fences)

Consider the store buffering (sb+fences) scenario given in Fig. 15.

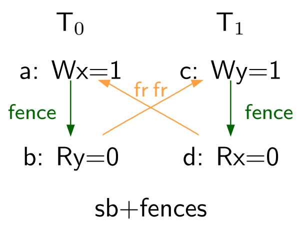


Figure 15: The store buffering scenario sb with fences (forbidden)

We need a fence on each side to prevent the reads b and d from reading the initial state.

On the web interface:

- sb+dmb+dmb.st: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=SB+dmb+dmb.st
- sb+dmb+dsb: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=SB+dmb+dsb

6.4 Read-to-write causality (rwc+fences)

The read-to-write causality scenario rwc+fences (see Fig. 16) distributes the sb scenario over three threads with a read b from x between the write a of x and the read c of y. It is to sb what wrc is to mp, thus responds to fences in the same way as sb. Hence it needs two fences too.

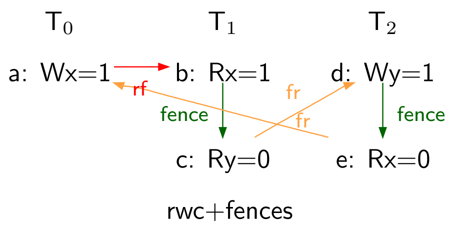


Figure 16: The read-to-write causality scenario rwc with fences (forbidden)

- rwc+dmb+dmb.st: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=RWC+dmb+dmb.st
- rwc+dmb+dsb: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=RWC+dmb+dsb

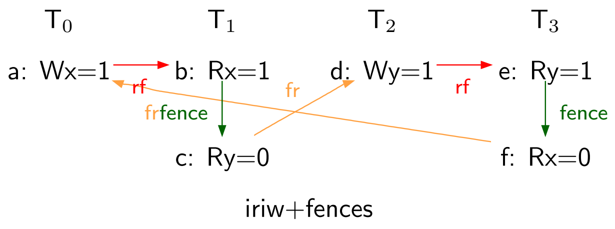


Figure 17: The independent reads from independent writes scenario iriw with fences (forbidden)

6.5 Independent reads of independent writes (**iriw+fences**)

ARM documentation [4] forbids iriw+dmb (see Fig. 17).

- iriw+dmbs: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=IRIW+dmbs

6.6 **r+fences**

In the r+fences scenario, the thread T_0 writes to x (event a) and then to y (event b). T_1 writes to y and reads from x. A fence on each thread forces the write a to x to propagate to T_1 before the write b to y. Thus if the write b is co-before the write c on T_1, the read d of x on T_1 cannot read from a write that is co-before the write a.

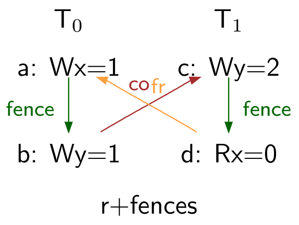


Figure 18: The scenario r with fences (forbidden)

- r+dmb+dmb.st: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=R+dmb+dmb.st
- r+dmb+dsb: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=R+dmb+dsb
- r+dmbs: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-from-paper&litmus=R+dmbs

7 ARM redux

Here I give an equivalent cat file. On the web interface: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-redux&litmus=MP

```
"ARM redux (equivalent to ARM)"

let com = rf | co | fr
acyclic po-loc | com as sc-per-location

let dp = addr | data
let rdw = po-loc & (fre;rfe)
let detour = po-loc & (coe ; rfe)
let ctrlcfence = ctrlisb

let ii0 = dp | rfi | rdw
let ic0 = 0
let ci0 = ctrlcfence | detour
let cc0 = dp | ctrl | (addr;po)

let rec ii = ii0 | ci | (ic;ci) | (ii;ii)
and ic = ic0 | ii | cc | (ic;cc) | (ii;ic)
and ci = ci0 | (ci;ii) | (cc;ci)
and cc = cc0 | ci | (ci;ic) | (cc;cc)

let ppo = ii & R*R | ic & R*W

let dmb.st = dmb.st & W*W
let dsb.st = dsb.st & W*W
let fence = dmb|dsb|dmb.st|dsb.st

let hb = ppo | fence | rfe
acyclic hb as no-thin-air

let prop-redux = com*; fence; hb*
acyclic prop-redux as propagation-redux
```

Figure 19: Another cat file, equivalent to the one of Fig. 2 (`arm-redux.cat` on the web)

One can prove that the model described in Fig. 2 is equivalent to the model of Fig. 19. Let us do that.

7.1 An intermediate lemma

Consider the model of Fig. 2. Let us observe that `fre;prop;hb*` is included in `prop`. This means that `propagation` implies `observation`.

7.2 From Fig. 2 to Fig. 19

We need to show that the two checks `observation` and `propagation` of Fig. 2 imply the check `propagation-redux` of Fig. 19. By the lemma above, we simply need to show that the check `propagation` implies the check `propagation-redux`.

This means that, if `co | prop` is acyclic, then we can prove that `prop-redux` is acyclic. Let's proceed by contradiction, and take a cycle in `prop-redux`: `(x,x)` in `(com*;fence;hb*)+`.

Thus `(x,x)` belongs to `(com*;prop-base*;fence;hb*)+`, i.e. to `(prop)+`. Hence we reach a contradiction as there cannot be any cycle in `co | prop` by `propagation`, hence no cycle in `prop`.

7.3 From Fig. 19 to Fig. 2

We need to show that the check `propagation-redux` of Fig. 19 imply the two checks `observation` and `propagation` of Fig. 2. By the lemma above, we simply need to show that the check `propagation-redux` implies the check `propagation`.

This means that, if `prop-redux` is acyclic, then we can prove that `co | prop` is acyclic.

Let's proceed by contradiction, and take `(x,x)` in `(co | prop)+`, i.e. a cycle in `co | prop`. Recall that `prop = prop-base & W*W | (com*; prop-base*; fence; hb*)`.

Observe that both `prop-base & W*W` and `com*; prop-base*;fence;hb*` are included in `com*;prop-base`, so that a cycle in `co | prop` is also a cycle in `co | com*;prop-base`. Since `co` is acyclic by `sc-per-location`, we have that a cycle in `co | prop` is a cycle in `com*;prop-base`. Now, `rfe` is included in `com*`, so that `com*;prop-base` is equal to `com*;fence;hb*`.

Hence we reach a contradiction: if we have a cycle in `co | prop`, then we have a cycle in `com*;fence;hb*`, which is impossible by `propagation-redux`.

8 Some comments

Even though it is possible to come up with a redux version of the ARM cat file of Fig. 2, I am not sure that it would be such a great idea. The reason is as follows: to integrate ldrex and strex on the one hand, and v8 features on the other hand, I think we need that separation of concerns between `observation` and `propagation`. The next sections expand on that.

8.1 With ldrex and strex

In experiments with Luc Maranget, we came up with several possible models to integrate ldrex and strex, two of which we show below in Fig. 20 and 21. Both models are experimentally sound, but they are underlied by different principles. This is a typical case where discussion with architects would help us decide which model is best, i.e. fits the architects' intents best.

8.1.1 Atomics contributing to `propagation`

In the first model, we take atomics to contribute to the `prop` order, but only in the `propagation` check (see lines 35 and 36 in Fig. 20). This means that atomics can be used to forbid e.g. 2+2w (see Fig. 14(a)) or sb (see Fig. 15), but not mp (see Fig. 9) or wrc (see Fig. 10).

On the web interface: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-with-ldrex-and-strex1&litmus=MP

8.1.2 Atomics as lightweight fences

In the second model, we take atomics to be lightweight fences, à la `lwsync` on Power. This means that they have cumulativity properties (see lines 27 and 28 of Fig. 21), and contribute to the `prop` order, both in the `observation` and `propagation` checks (see lines 36 and 37 in Fig. 21). This means that atomics can be used to forbid e.g. mp (see Fig. 9) or wrc (see Fig. 10), but also 2+2w (see Fig. 14(a)) or sb (see Fig. 15).

On the web interface: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-with-ldrex-and-strex2&litmus=MP

8.2 With v8

In recent work with Luc Maranget, we came up with several possible models to integrate v8 features, i.e. dmb.ld, dsb.ld, load acquire and store release. We give one possible model in Fig. 22, which is based on our reading of the ARM v8 manual [3, p. B2-85 to B2-88], and educated guesses.

We would need to be able to test ARM v8 chips to adjust our model, as discuss it with architects.

On the web interface: http://virginia.cs.ucl.ac.uk/herd-web-prerelease/?book=armed-cats&language=arm&cat=arm-with-v8&litmus=MP

```
"ARM with ldrex and strex (experimental 1)"

let com = rf | co | fr
acyclic po-loc | com as sc-per-location

let dp = addr | data
let rdw = po-loc & (fre;rfe)
let detour = po-loc & (coe ; rfe)
let ctrlcfence = ctrlisb

let ii0 = dp | rfi | rdw
let ic0 = 0
let ci0 = ctrlcfence | detour
let cc0 = dp | ctrl | (addr;po)

let rec ii = ii0 | ci | (ic;ci) | (ii;ii)
and ic = ic0 | ii | cc | (ic;cc) | (ii;ic)
and ci = ci0 | (ci;ii) | (cc;ci)
and cc = cc0 | ci | (ci;ic) | (cc;cc)

let ppo = ii & R*R | ic & R*W

let dmb.st = dmb.st & W*W
let dsb.st = dsb.st & W*W
let fence = dmb|dsb|dmb.st|dsb.st
let A-cumul = rfe;fence

let hb = ppo | fence | rfe
acyclic hb as no-thin-air

let prop-base = (fence | A-cumul);hb*
let prop = (prop-base & W*W)| (com*; prop-base*; fence; hb*)

irreflexive fre;prop;hb* as observation
let aa = WW(AA(po))
acyclic co | prop | aa as propagation
empty atom & (fre;coe) as atomic
```

Figure 20: A first ARM model with ldrex and strex (`arm-with-ldrex-and-strex1.cat` on the web)

```
"ARM with ldrex and strex (experimental 2)"

let com = rf | co | fr
acyclic po-loc | com as sc-per-location

let dp = addr | data
let rdw = po-loc & (fre;rfe)
let detour = po-loc & (coe ; rfe)
let ctrlcfence = ctrlisb

let ii0 = dp | rfi | rdw
let ic0 = 0
let ci0 = ctrlcfence | detour
let cc0 = dp | ctrl | (addr;po)

let rec ii = ii0 | ci | (ic;ci) | (ii;ii)
and ic = ic0 | ii | cc | (ic;cc) | (ii;ic)
and ci = ci0 | (ci;ii) | (cc;ci)
and cc = cc0 | ci | (ci;ic) | (cc;cc)

let ppo = ii & R*R | ic & R*W

let dmb.st = dmb.st & W*W
let dsb.st = dsb.st & W*W
let ffence = dmb|dsb|dmb.st|dsb.st
let lwfence = WW(AA(po))
let fences = ffence|lwfence
let A-cumul = rfe;fences

let hb = ppo | fences | rfe
acyclic hb as no-thin-air

let prop-base = (fences | A-cumul);hb*
let prop = (prop-base & W*W)| (com*; prop-base*; ffence; hb*)

irreflexive fre;prop;hb* as observation
acyclic co | prop as propagation
empty atom & (fre;coe) as atomic
```

Figure 21: A second ARM model with ldrex and strex (`arm-with-ldrex-and-strex2.cat` on the web)

```
"ARM with ldrex and strex (experimental 1) and v8"

let com = rf | co | fr
acyclic po-loc | com as sc-per-location

let dp = addr | data
let rdw = po-loc & (fre;rfe)
let detour = po-loc & (coe ; rfe)
let ctrlcfence = ctrlisb

let ii0 = dp | rfi | rdw
let ic0 = 0
let ci0 = ctrlcfence | detour
let cc0 = dp | ctrl | (addr;po)

let rec ii = ii0 | ci | (ic;ci) | (ii;ii)
and ic = ic0 | ii | cc | (ic;cc) | (ii;ic)
and ci = ci0 | (ci;ii) | (cc;ci)
and cc = cc0 | ci | (ci;ic) | (cc;cc)

let ppo = ii & R*R | ic & R*W

enum attribute = 'none || 'st || 'ld
let dmb(t) = match t with
 || 'none -> dmb
 || 'st -> dmb & W*W
 || 'ld -> dmb & (R*R | R*W)
 end
let all-dmb = dmb('none) | dmb('st) | dmb('ld)
let all-dsb = all-dmb
let acq-po = po & (acquire * (W|R))
let po-rel = ((W|R) * release)
let fence = all-dmb | all-dsb | acq-po | po-rel
let cumul-fence = all-dmb | all-dsb | po-rel
let A-cumul = rfe;cumul-fence

let hb = ppo | fence | rfe
acyclic hb as no-thin-air

let prop-base = (fence | A-cumul);hb*
let prop = (prop-base & W*W)| (com*; prop-base*; fence; hb*)

irreflexive fre;prop;hb* as observation
let aa = WW(AA(po))
acyclic co | prop | aa as propagation
empty atom & (fre;coe) as atomic
```

Figure 22: An ARM model with ldrex and strex, and v8 features (`arm-with-v8.cat` on the web)

References

[1] Jade Alglave. *A Shared Memory Poetics*. PhD thesis, Université Paris 7 and INRIA, 2010.

[2] Jade Alglave, Luc Maranget, and Michael Tautschnig. Herding cats: Modelling, simulation, testing, and data-mining for weak memory. *TOPLAS*, July 2014. Article No. 7, Issue 2.

[3] ARM Ltd. *ARM Architecture Reference Manual*
ARMv8, for ARMv8-A architecture profile
Beta, 2013.

[4] Richard Grisenthwaite. *ARM Barrier Litmus Tests and Cookbook*. ARM Ltd., 2009.