

# An ACL2 Mechanization of an Axiomatic Framework for Weak Memory

Benjamin Selfridge

The University of Texas at Austin

*benself@cs.utexas.edu*

May 1, 2015

# Outline

- Introduction
- Problematic programming examples
- Proposed memory model framework (Alglave et. al.)
- ACL2-based formalization of this framework
- Use of ACL2 to validate hand proofs
- Current work: x86-targeted approach

# Motivating Examples

Certain multiprocessor programs may behave differently when run on different architectures.

Let's consider two programs, and examine their possible outcomes.

# Example 1

	$P_0$		$P_1$
a)	$m_0 \leftarrow 1$	c)	$m_1 \leftarrow 1$
b)	$r_0 \leftarrow m_1$	d)	$r_1 \leftarrow m_0$

One might expect the following possible outcomes:

- 1  $r_0 = 1, r_1 = 1$  (a, c, b, d)
- 2  $r_0 = 0, r_1 = 1$  (a, b, c, d)
- 3  $r_0 = 1, r_1 = 0$  (c, d, a, b)

## Example 1

	$P_0$		$P_1$
a)	$m_0 \leftarrow 1$	c)	$m_1 \leftarrow 1$
b)	$r_0 \leftarrow m_1$	d)	$r_1 \leftarrow m_0$

One might expect the following possible outcomes:

- 1  $r_0 = 1, r_1 = 1$  (a, c, b, d)
- 2  $r_0 = 0, r_1 = 1$  (a, b, c, d)
- 3  $r_0 = 1, r_1 = 0$  (c, d, a, b)

We do *not* expect  $r_0 = 0, r_1 = 0$ ...

But this can happen on x86, PowerPC, and ARM!

## Example 2

$P_0$		$P_1$	
a)	$r_0 \leftarrow m_0$	c)	$r_1 \leftarrow m_1$
b)	$m_1 \leftarrow 1$	d)	$m_0 \leftarrow 1$

One might expect the following possible outcomes:

- 1  $r_0 = 0, r_1 = 0$  (a, c, b, d)
- 2  $r_0 = 0, r_1 = 1$  (a, b, c, d)
- 3  $r_0 = 1, r_1 = 0$  (c, d, a, b)

## Example 2

	$P_0$		$P_1$
a)	$r_0 \leftarrow m_0$	c)	$r_1 \leftarrow m_1$
b)	$m_1 \leftarrow 1$	d)	$m_0 \leftarrow 1$

One might expect the following possible outcomes:

- 1  $r_0 = 0, r_1 = 0$  (a, c, b, d)
- 2  $r_0 = 0, r_1 = 1$  (a, b, c, d)
- 3  $r_0 = 1, r_1 = 0$  (c, d, a, b)

We do *not* expect  $r_0 = 1, r_1 = 1$ .

This never happens on x86, but it *can* occur on PowerPC and ARM.

# What's going on?

- Modern MP architectures are not *sequentially consistent* (SC is too inefficient to implement)
- x86, Power, ARM all have “relaxed” or “weak” memory models
- They are all different
- Some are informally specified
- Many attempts to make them rigorous have been flawed (unsound w.r.t actual hardware)



# Herding Cats: Modelling, Simulation, Testing, and Data Mining for Weak Memory

JADE ALGLAVE, University College London

LUC MARANGET, INRIA

MICHAEL TAUTSCHNIG, Queen Mary University of London

We propose an axiomatic generic framework for modelling weak memory. We show how to instantiate this framework for Sequential Consistency (SC), Total Store Order (TSO), C++ restricted to release-acquire atomics, and Power. For Power, we compare our model to a preceding operational model in which we found a flaw. To do so, we define an operational model that we show equivalent to our axiomatic model.

We also propose a model for ARM. Our testing on this architecture revealed a behaviour later acknowledged as a bug by ARM, and more recently, 31 additional anomalies.

We offer a new simulation tool, called *herd*, which allows the user to specify the model of his choice in a concise way. Given a specification of a model, the tool becomes a simulator for that model. The tool relies on an axiomatic description; this choice allows us to outperform all previous simulation tools. Additionally, we confirm that verification time is vastly improved, in the case of bounded model checking.

Finally, we put our models in perspective, in the light of empirical data obtained by analysing the C and C++ code of a Debian Linux distribution. We present our new analysis tool, called *mole*, which explores a piece of code to find the weak memory idioms that it uses.

Categories and Subject Descriptors: B.3.2 [**Shared Memory**]; C.0 [**Hardware/Software Interfaces**]

General Terms: Theory, Experimentation, Verification

# Herding Cats: Modelling, Simulation, Testing, and Data Mining for Weak Memory (Alglave, Maranget, Tautschnig)

This paper attempts to unify these various memory models.

- Develops a common language (“framework”) for weak memory specs
- General enough to capture: SC, x86, Power, ARM

*“We believe that these models would benefit from stating principles that underpin weak memory as a whole, not just one particular architecture or language.”*

This work actually exposed bugs in ARM implementations!

# Herding Cats: Overview

## Description:

- Given a concurrent program, a multitude of *executions* are possible
- Executions are represented as directed graphs
  - ▶ Nodes are read/write *events*
  - ▶ Edges represent *dependencies* between events
- Different architectures “allow” different subsets of executions

# Herding Cats: Events

Two types of events: reads & writes

Components of an event:

- Type (read/write)
- Memory address
- Value read/written
- Processor ID

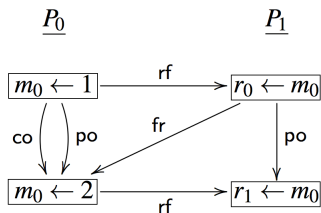
# Herding Cats: Executions

## Definition

An *execution* is a tuple  $(\mathbb{E}, po, rf, co, fr)$  where  $\mathbb{E}$  is a set of events (nodes) and  $po, rf, co, fr$  are relations (edges) on  $\mathbb{E}$ .

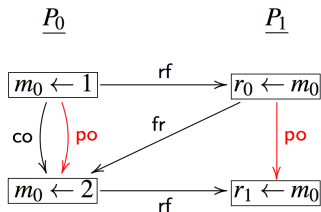
Let's look at these four relations in a little more detail.

# Herding Cats: Example of the four relations



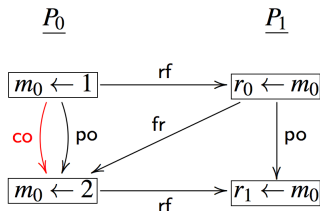
What does all this mean?

# Herding Cats: po (program order)



$e \xrightarrow{\text{po}} e'$ :  $e$  precedes  $e'$  in program order.

## Herding Cats: co (coherence order)

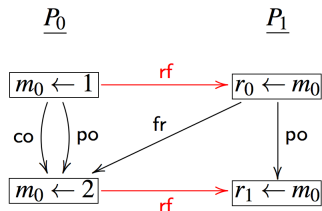


$w \xrightarrow{\text{co}} w'$ : write  $w$  becomes globally visible before  $w'$ .

Only applies to writes *at the same location*.



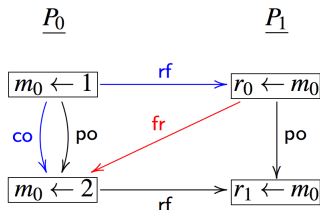
# Herding Cats: rf (read-from)



$w \xrightarrow{rf} r$ : write  $w$  supplied the value that  $r$  reads.

Only applies to events *at the same location*.

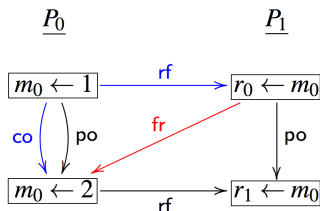
## Herding Cats: fr (from-read)



$r \xrightarrow{\text{fr}} w$ : read  $r$  “reads from” a write that precedes  $w$  in coherence order.

Only applies to writes *at the same location*.

## Herding Cats: fr (from-read)



$r \xrightarrow{fr} w$ : read  $r$  “reads from” a write that precedes  $w$  in coherence order.

Only applies to writes *at the same location*.

Actually,  $fr$  is defined in terms of  $rf$  and  $co$ :  $fr = rf^{-1} \circ co$ .

# Herding Cats: Sequential Consistency

Sequential consistency: “The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.”

In the “Herding Cats” framework, this condition is stated as

$$\text{acyclic}(\text{po} \cup \text{co} \cup \text{rf} \cup \text{fr})$$

## Herding Cats: Sequential Consistency

Sequential consistency: “The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.”

In the “Herding Cats” framework, this condition is stated as

$$\text{acyclic}(po \cup co \cup rf \cup fr)$$

**Most modern architectures do not satisfy this constraint.** (So, SC is not an *axiom* of this framework.)

# Herding Cats: SC-Per-Location

We do not have SC in most architectures.

`acyclic(poUcoUrfUfr)`

# Herding Cats: SC-Per-Location

We do not have SC in most architectures.

$$\text{acyclic}(\text{po} \cup \text{co} \cup \text{rf} \cup \text{fr})$$

However, we do have a similar condition, “SC-Per-Location,” in *all* modern commercial architectures.

$$\text{acyclic}(\text{po-loc} \cup \text{co} \cup \text{rf} \cup \text{fr})$$

Note: po-loc is po, restricted to events *at the same address*.

# Herding Cats: SC-Per-Location

We do not have SC in most architectures.

$$\text{acyclic}(\text{po} \cup \text{co} \cup \text{rf} \cup \text{fr})$$

However, we do have a similar condition, “SC-Per-Location,” in *all* modern commercial architectures.

$$\text{acyclic}(\text{po-loc} \cup \text{co} \cup \text{rf} \cup \text{fr})$$

Note: po-loc is po, restricted to events *at the same address*.

SC-Per-Location is one of the four axioms of the Herding Cats framework.



# Our research

To internalize all of this material, we did the following:

- Formalized these concepts in the ACL2 theorem prover
  - ▶ Events, executions
  - ▶ All of the axioms (including SC-Per-Location)
- Used ACL2 to check a hand proof of a theorem first presented in Alglave's thesis
  - ▶ The theorem is about the SC-Per-Location axiom
  - ▶ Our ACL2 proof is **simpler** than the one initially presented!

# Our research

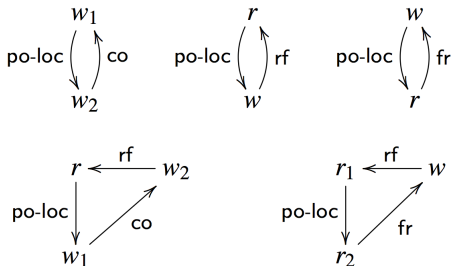
To internalize all of this material, we did the following:

- Formalized these concepts in the ACL2 theorem prover
  - ▶ Events, executions
  - ▶ All of the axioms (including SC-Per-Location)
- Used ACL2 to check a hand proof of a theorem first presented in Alglave's thesis
  - ▶ The theorem is about the SC-Per-Location axiom
  - ▶ Our ACL2 proof is **simpler** than the one initially presented!

Let's look at the theorem we verified using ACL2, and then we will provide a sketch of the proof itself.

# Theorem about SC-Per-Location

In her dissertation, Alglave demonstrated that SC-Per-Location was equivalent to prohibiting the following five patterns in an execution:



In other words, if there exists a cycle in  $\text{po-loc} \cup \text{co} \cup \text{rf} \cup \text{fr}$ , then one of these patterns exists somewhere in the graph.

## Theorem about SC-Per-Location

From “Herding Cats”: Define  $\text{com} = \text{co} \cup \text{rf} \cup \text{fr}$ , and let  $\text{com}^+$  be the (irreflexive) transitive closure of  $\text{com}$ .

### Lemma

Let  $E = (\mathbb{E}, \text{po}, \text{co}, \text{rf})$  be an execution. Then we have

$$\text{com}^+ = \text{co} \cup \text{rf} \cup \text{fr} \cup (\text{co}; \text{rf}) \cup (\text{fr}; \text{rf}).$$

Note:  $\text{com}^+$  actually IS irreflexive, because  $\text{com}$  is always acyclic. (We re-proved this lemma in ACL2, although it was first proved in Alglave’s thesis.)

# Theorem about SC-Per-Location

In other words,  $e_1 \xrightarrow{\text{com}^+} e_2$  iff. one of the following are true:

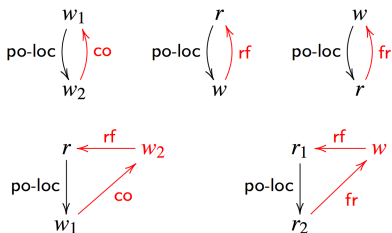
- $e_1 \xrightarrow{\text{co}} e_2$
- $e_1 \xrightarrow{\text{rf}} e_2$
- $e_1 \xrightarrow{\text{fr}} e_2$
- $\exists e_3$  s.t.  $e_1 \xrightarrow{\text{co}} e_3 \xrightarrow{\text{rf}} e_2$
- $\exists e_3$  s.t.  $e_1 \xrightarrow{\text{fr}} e_3 \xrightarrow{\text{rf}} e_2$

# Theorem about SC-Per-Location

In other words,  $e_1 \xrightarrow{\text{com}^+} e_2$  iff. one of the following are true:

- $e_1 \xrightarrow{\text{co}} e_2$
- $e_1 \xrightarrow{\text{rf}} e_2$
- $e_1 \xrightarrow{\text{fr}} e_2$
- $\exists e_3$  s.t.  $e_1 \xrightarrow{\text{co}} e_3 \xrightarrow{\text{rf}} e_2$
- $\exists e_3$  s.t.  $e_1 \xrightarrow{\text{fr}} e_3 \xrightarrow{\text{rf}} e_2$

But wait! This corresponds to the five patterns prohibited by SC-Per-Location:



# Theorem about SC-Per-Location

This lemma suggests an alternate definition of SC-Per-Location, corresponding to the five forbidden patterns.

## Definition

An execution  $E = (\mathbb{E}, \text{po}, \text{co}, \text{rf})$  satisfies the property *SC-Per-Location-2* if

$$\forall x, y \in \mathbb{E}, x \xrightarrow{\text{po-loc}} y \implies \neg(y \xrightarrow{\text{com}^+} x)$$

i.e. no two events be related by po-loc in one direction and  $\text{com}^+$  in the other direction.

The above definition is the same one used in the equivalence proof Alglave's thesis - we still have not seen the "new" aspect of *our* proof just yet.

# Theorem about SC-Per-Location: Formal statement

## Theorem

*Let  $E$  be an execution. Then  $E$  satisfies SC-Per-Location if and only if  $E$  satisfies SC-Per-Location-2. In other words:*

$$\text{acyclic}(\text{po-loc} \cup \text{co} \cup \text{rf} \cup \text{fr})$$

*if and only if*

$$\forall x, y \in \mathbb{E}, x \xrightarrow{\text{po-loc}} y \implies \neg(y \xrightarrow{\text{com}^+} x)$$

Although this theorem was proved in Alglave's dissertation, we re-proved it independently using a simpler strategy, and checked our proof in ACL2.



# Theorem about SC-Per-Location: Our Proof

## Theorem

*Let  $E$  be an execution. Then  $E$  satisfies SC-Per-Location if and only if  $E$  satisfies SC-Per-Location-2.*

Proof Sketch: It is clear that SC-Per-Location implies SC-Per-Location-2, because all five patterns are actually cycles in  $\text{po-loc} \cup \text{co} \cup \text{rf} \cup \text{fr}$ . To prove the reverse implication, we proceed by contrapositive.

Assume SC-Per-Location does not hold for an execution - i.e., assume there is a cycle in  $\text{po-loc} \cup \text{co} \cup \text{rf} \cup \text{fr}$ . In particular, this is also a cycle in  $\text{po-loc} \cup \text{com}^+$ . We must show that there exist two events  $x, y \in \mathbb{E}$  such that  $x \xrightarrow{\text{po-loc}} y$  and  $y \xrightarrow{\text{com}^+} x$ .

We demonstrate inductively that if the cycle has length  $k > 2$ , we can always construct a shorter one. Since any cycle of length 2 has the form  $x \xrightarrow{\text{po-loc}} y \xrightarrow{\text{com}^+} x$ , we are done.

# Ben's Publications So Far

# Summary

## Current work

The work presented here helped inspire our current research activities.




- We implemented the x86-TSO memory model directly using ACL2
- We also built a simple instruction semantics using this model
  - ▶ Not x86 - much simpler
  - ▶ However, same underlying memory model (TSO)
- We have written several simple programs on this model and proven they satisfy certain properties

# Future Work

What next?

- Specify exactly which programs are DRF (data-race free) on our x86-like model
- If a program is data-race free, it only has SC behavior
- This would be a huge win
  - ▶ We wouldn't have to think about weak memory anymore
  - ▶ Move on to bigger and better things – proving real programs correct!
- Lots of previous work on this problem to mine from - it's just a matter of incorporating it into the model

# References

-  Owens et al. (2009)  
x86-TSO: A Rigorous and Usable Programmer's Model for x86 Multiprocessors  
*TPHOLs'09*, 391 – 407.
-  Sewell et al. (2010)  
x86-TSO: A Rigorous and Usable Programmer's Model for x86 Multiprocessors  
*Commun. ACM* 53(7), 89 – 97.
-  Sorin et al. (2011)  
A Primer on Memory Consistency and Cache Coherence  
*Morgan & Claypool*