An ACL2 Mechanization of an Axiomatic Weak Memory Model

Ben Selfridge

July 13, 2014

Outline

Outline

Multiprocessor Reasoning

Goal: **Analysis of programs** written for **multiple processors** with a shared memory

Multiprocessor Reasoning

Two conceivable approaches:

Multiprocessor Reasoning

Two conceivable approaches:

- **Operational** - Create a **model** of a multiprocessor machine (e.g. in ACL2) and **mechanically prove** that certain properties of the program hold

Multiprocessor Reasoning

Two conceivable approaches:

- **Operational** - Create a **model** of a multiprocessor machine (e.g. in ACL2) and **mechanically prove** that certain properties of the program hold
  - Use an oracle to model non-determinism of scheduler

Multiprocessor Reasoning

Two conceivable approaches:

- **Operational** - Create a **model** of a multiprocessor machine (e.g. in ACL2) and **mechanically prove** that certain properties of the program hold
  - Use an oracle to model non-determinism of scheduler
- **Axiomatic** - derive a set of **mathematical objects** from the program and prove theorems about the **structure** of those objects

Multiprocessor Reasoning

Two conceivable approaches:

- **Operational** - Create a **model** of a multiprocessor machine (e.g. in ACL2) and **mechanically prove** that certain properties of the program hold
  - Use an oracle to model non-determinism of scheduler
- **Axiomatic** - derive a set of **mathematical objects** from the program and prove theorems about the **structure** of those objects

Both approaches have certain advantages and disadvantages

Complication: Weak Memory

- Practical MP architectures do not satisfy *sequential consistency* (one reason: write buffers)

Complication: Weak Memory

- Practical MP architectures do not satisfy *sequential consistency* (one reason: write buffers)
- Instead, they satisfy some weaker properties

Complication: Weak Memory

- Practical MP architectures do not satisfy *sequential consistency* (one reason: write buffers)
- Instead, they satisfy some weaker properties
- **Axiomatic Memory Models** attempt to capture the weaker consistency guarantees of most modern architectures as axioms

Goal of this talk

What this talk is about:

[1] Jade Alglave, Luc Maranget, and Michael Tautschnig. *Herding Cats - Modelling, simulation, testing, and data-mining for weak memory.* To appear in TOPLAS 2014. http://arxiv.org/abs/1308.6810

Goal of this talk

What this talk is about:

- A partial description of one particular axiomatic memory framework[1]

[1] Jade Alglave, Luc Maranget, and Michael Tautschnig. *Herding Cats - Modelling, simulation, testing, and data-mining for weak memory.* To appear in TOPLAS 2014. http://arxiv.org/abs/1308.6810

Goal of this talk

What this talk is about:

- A partial description of one particular axiomatic memory framework[1]
- An ACL2 mechanization of this framework

[1] Jade Alglave, Luc Maranget, and Michael Tautschnig. *Herding Cats - Modelling, simulation, testing, and data-mining for weak memory.* To appear in TOPLAS 2014. http://arxiv.org/abs/1308.6810

Goal of this talk

What this talk is about:

- A partial description of one particular axiomatic memory framework[1]
- An ACL2 mechanization of this framework
- A new proof of a nice equivalence result for this framework, and a mechanization of this proof

[1] Jade Alglave, Luc Maranget, and Michael Tautschnig. *Herding Cats - Modelling, simulation, testing, and data-mining for weak memory.* To appear in TOPLAS 2014. http://arxiv.org/abs/1308.6810

Goal of this talk

What this talk is about:

- A partial description of one particular axiomatic memory framework[1]
- An ACL2 mechanization of this framework
- A new proof of a nice equivalence result for this framework, and a mechanization of this proof
- Ultimate goal: utilize this, or a similar, axiomatic framework to reason about an executable MP model

[1] Jade Alglave, Luc Maranget, and Michael Tautschnig. *Herding Cats - Modelling, simulation, testing, and data-mining for weak memory.* To appear in TOPLAS 2014. http://arxiv.org/abs/1308.6810

Outline

Execution

An *execution* of a sequential program is a sequence of events that results from running the program on a particular set of inputs (or with a particular starting configuration).

Execution

An *execution* of a sequential program is a sequence of events that results from running the program on a particular set of inputs (or with a particular starting configuration).

How does this translate to concurrent programs?

Concurrent Executions

With multiple processors, an execution is not necessarily a linear sequence.

Concurrent Executions

With multiple processors, an execution is not necessarily a linear sequence.

Instead, we represent it as a graph, consisting of a collection of **events** with various kinds of directed edges.

Events

Definition

An *event* is a read or a write.

Events

Components of an event:

Events

Components of an event:

- Type (read or write)

Events

Components of an event:

- Type (read or write)
- Memory location

Events

Components of an event:

- Type (read or write)
- Memory location
- Value read or written

Events

Components of an event:

- Type (read or write)
- Memory location
- Value read or written
- Process number

Concurrent Executions

Definition

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

Concurrent Executions

Definition

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

- po is a total order on events in the same process

Concurrent Executions

Definition

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

- po is a total order on events in the same process
- co is a total order on writes to the same location

Concurrent Executions

Definition

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

- po is a total order on events in the same process
- co is a total order on writes to the same location
- rf is a relation from writes to reads s.t. for each read r, there is exactly one write w such that $w \xrightarrow{\mathsf{rf}} r$ and $\mathsf{val}(w) = \mathsf{val}(r)$

Concurrent Executions

Definition

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

- po is a total order on events in the same process
- co is a total order on writes to the same location
- rf is a relation from writes to reads s.t. for each read r, there is exactly one write w such that $w \xrightarrow{\mathsf{rf}} r$ and $\mathsf{val}(w) = \mathsf{val}(r)$

po is "program order", co is "coherence order", rf is "read-from"

Concurrent Executions

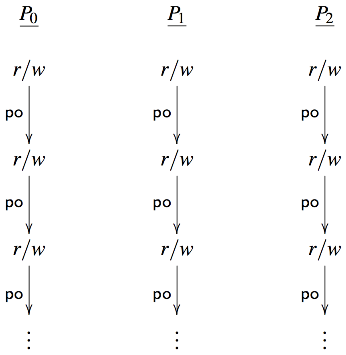
Define $\mathsf{fr} = \mathsf{rf}^{-1} \circ \mathsf{co}$ to represent a write that must come after a read

Concurrent Executions

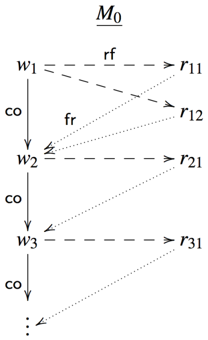
Define $\mathsf{fr} = \mathsf{rf}^{-1} \circ \mathsf{co}$ to represent a write that must come after a read

co, rf, and fr are *per-location* dependencies; they relate events which occur at the same memory location only

Concurrent Executions



(a) The per-process view.



(b) The per-location view.

Sequential consistency (SC)

Sequential consistency[2]: "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."

[2] [3] Leslie Lamport. *How to make a multiprocessor computer that correctly executes multiprocess programs.* IEEE Transactions on Computers, September 1979

Sequential consistency (SC)

Sequential consistency[2]: "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 our framework, we interpret this as the condition

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

[2] [3] Leslie Lamport. *How to make a multiprocessor computer that correctly executes multiprocess programs.* IEEE Transactions on Computers, September 1979

Sequential consistency (SC)

Sequential consistency[2]: "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 our framework, we interpret this as the condition

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

Modern architectures do not satisfy this constraint.

[2] [3] Leslie Lamport. *How to make a multiprocessor computer that correctly executes multiprocess programs.* IEEE Transactions on Computers, September 1979

SC-Per-Location

Although we don't usually have full sequential consistency, we do have an **analogous notion** that is enforced by most modern architectures:

$$\mathsf{acyclic}(\mathsf{pol} \cup \mathsf{co} \cup \mathsf{rf} \cup \mathsf{fr}),$$

where pol is po restricted to events at the same memory location.

SC-Per-Location

Although we don't usually have full sequential consistency, we do have an **analogous notion** that is enforced by most modern architectures:

$$\mathsf{acyclic}(\mathsf{pol} \cup \mathsf{co} \cup \mathsf{rf} \cup \mathsf{fr}),$$

where pol is po restricted to events at the same memory location.

We refer to this condition as *SC-Per-Location*.

SC-Per-Location

Although we don't usually have full sequential consistency, we do have an **analogous notion** that is enforced by most modern architectures:

$$\mathsf{acyclic}(\mathsf{pol} \cup \mathsf{co} \cup \mathsf{rf} \cup \mathsf{fr}),$$

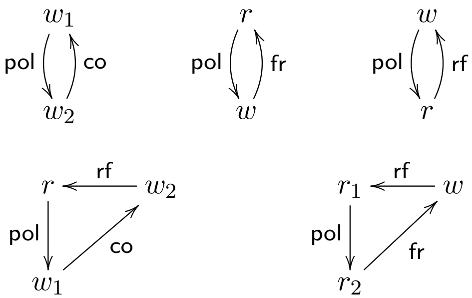
where pol is po restricted to events at the same memory location.

We refer to this condition as *SC-Per-Location*.

It is **one** of the axioms used in Alglave et. al. [2]

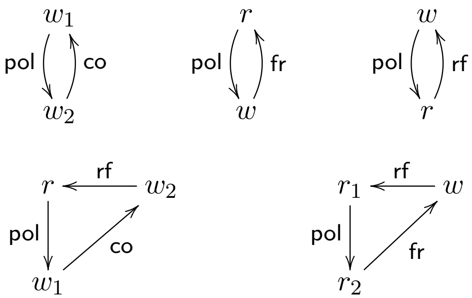
SC-Per-Location

SC-Per-Location is equivalent to prohibiting the following five patterns:



SC-Per-Location

SC-Per-Location is equivalent to prohibiting the following five patterns:



We formalized SC-Per-Location in ACL2 and proved this equivalence.

Outline

Outline

Conclusion

- We presented a (partial) mechanization in ACL2 of an axiomatic model of weak memory

Conclusion

- We presented a (partial) mechanization in ACL2 of an axiomatic model of weak memory
- This included a new proof of an equivalence theorem

Where is this going?

Where is this going?

I plan to integrate an "axiomatic" approach like this with an operational semantics in the following manner:

Where is this going?

I plan to integrate an “axiomatic” approach like this with an operational semantics in the following manner:

1. **As we execute** our model (i.e. using an oracle), simultaneously **construct one of these graphs**

Where is this going?

I plan to integrate an “axiomatic” approach like this with an operational semantics in the following manner:

1. **As we execute** our model (i.e. using an oracle), simultaneously **construct one of these graphs**
2. Demonstrate, for all programs and oracles, **any graph** produced by such an execution **satisfies certain structural properties**

Where is this going?

I plan to integrate an "axiomatic" approach like this with an operational semantics in the following manner:

1. **As we execute** our model (i.e. using an oracle), simultaneously **construct one of these graphs**
2. Demonstrate, for all programs and oracles, **any graph** produced by such an execution **satisfies certain structural properties**
3. To prove a program has property P, show that **any execution** of that program that **fails to satisfy** P will produce an **invalid graph**

Where is this going?

I plan to integrate an "axiomatic" approach like this with an operational semantics in the following manner:

1. **As we execute** our model (i.e. using an oracle), simultaneously **construct one of these graphs**
2. Demonstrate, for all programs and oracles, **any graph** produced by such an execution **satisfies certain structural properties**
3. To prove a program has property P, show that **any execution** of that program that **fails to satisfy** P will produce an **invalid graph**

An "invalid" graph could be, for instance, one that violates *SC-Per-Location*

Where is this going?

Ultimate goal:

Where is this going?

Ultimate goal:

- Verify multiprocessor code with an *executable* model (in ACL2)

Where is this going?

Ultimate goal:

- Verify multiprocessor code with an *executable* model (in ACL2)
- But we don't want to reason directly about store buffers!

Where is this going?

Ultimate goal:

- Verify multiprocessor code with an *executable* model (in ACL2)
- But we don't want to reason directly about store buffers!
- Unite the model with a higher-level reasoning strategy to handle weak memory more transparently

References

1. Jade Alglave. *A Shared Memory Poetics.* Ph.D. Dissertation. Université Paris 7, 2010.

2. Jade Alglave, Luc Maranget, and Michael Tautschnig. *Herding Cats - Modelling, simulation, testing, and data-mining for weak memory.* To appear in TOPLAS 2014. http://arxiv.org/abs/1308.6810

3. Leslie Lamport. *How to make a multiprocessor computer that correctly executes multiprocess programs.* IEEE Transactions on Computers, C-28(9):690691. September 1979

Operational vs. Axiomatic

- **Operational** semantics have a **closer connection to the actual architecture** being modeled, whereas an axiomatic approach makes a lot of assumptions

Operational vs. Axiomatic

- **Operational** semantics have a **closer connection to the actual architecture** being modeled, whereas an axiomatic approach makes a lot of assumptions
- **Axiomatic** models can be **easier to reason about**; fully modeling an MP architecture can be messy from a theorem-proving perspective