An ACL2 Mechanization of an Axiomatic Weak Memory Model

Ben Selfridge

February 28, 2014
Outline

1. Introduction
   - Multiprocessor Reasoning
   - Weak Memory
   - Goals of this talk

2. An Axiomatic Weak Memory Model
   - Concurrent Executions
   - SC-Per-Location

3. ACL2 Mechanization

4. Conclusion
Outline

1 Introduction
   - Multiprocessor Reasoning
   - Weak Memory
   - Goals of this talk

2 An Axiomatic Weak Memory Model
   - Concurrent Executions
   - SC-Per-Location

3 ACL2 Mechanization

4 Conclusion
Multiprocessor Reasoning

Goal: Analysis of programs written for multiple processors with a shared memory
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
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
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.
Bridging the Gap

How could we “bridge the gap” between these two approaches?
Bridging the Gap

How could we “bridge the gap” between these two approaches?

One strategy:
Bridging the Gap

How could we “bridge the gap” between these two approaches?

One strategy:

1. **As we execute** our model (i.e. using an oracle), simultaneously **construct one of these mathematical objects**
Bridging the Gap

How could we “bridge the gap” between these two approaches?

One strategy:

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

2. Demonstrate, for all programs and oracles, any object produced by such an execution **satisfies certain structural properties**
Bridging the Gap

How could we “bridge the gap” between these two approaches?

One strategy:

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

2. Demonstrate, for all programs and oracles, **any object** 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 object**
Complication: Weak Memory

- Practical MP architectures do not satisfy *sequential consistency*
Complication: Weak Memory

- Practical MP architectures do not satisfy *sequential consistency*
- Instead, they satisfy some weaker properties
Complication: Weak Memory

- Practical MP architectures do not satisfy *sequential consistency*
- 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:

Goal of this talk

What this talk is about:

- A partial description of one particular axiomatic memory framework\(^1\)

---

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

Introduction

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

Outline

1. Introduction
   - Multiprocessor Reasoning
   - Weak Memory
   - Goals of this talk

2. An Axiomatic Weak Memory Model
   - Concurrent Executions
   - SC-Per-Location

3. ACL2 Mechanization

4. Conclusion
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 \((E, po, rf, co)\), where \(E\) is a set of events and \(po, rf, \) and \(co\) are relations on \(E\) satisfying
Concurrent Executions

Definition

An execution is a tuple \((E, po, rf, co)\), where \(E\) is a set of events and \(po, rf,\) and \(co\) are relations on \(E\) satisfying

- \(po\) is a total order on events in the same process
Definition

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

- \(\text{po}\) is a total order on events in the same process
- \(\text{co}\) is a total order on writes to the same location
An \textit{execution} is a tuple \((\mathcal{E}, \text{po}, \text{rf}, \text{co})\), where \(\mathcal{E}\) is a set of events and \(\text{po}, \text{rf}, \) and \(\text{co}\) are relations on \(\mathcal{E}\) satisfying:

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

Definition

An execution is a tuple \((E, po, rf, co)\), where \(E\) is a set of events and \(po\), \(rf\), and \(co\) are relations on \(E\) 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 \(w \xrightarrow{rf} r\) and \(val(w) = val(r)\)

\(po\) is “program order”, \(co\) is “coherence order”, \(rf\) is “read-from”
Concurrent Executions

Define $fr = rf^{-1} \circ co$ to represent a write that must come after a read.
Define $fr = rf^{-1} \circ co$ 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.
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)**: “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

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

---

**[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

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

Modern architectures do not satisfy this constraint.

\[^2\text{[3] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Transactions on Computers, September 1979}\]
Although we don’t usually have full sequential consistency, we do have an analogous notion that is enforced by most modern architectures:

\[
\text{acyclic}(\text{pol} \cup \text{co} \cup \text{rf} \cup \text{fr}),
\]

where \text{pol} is \text{po} restricted to events at the same memory location.
Although we don’t usually have full sequential consistency, we do have an analogous notion that is enforced by most modern architectures:

\[
\text{acyclic}(\text{pol} \cup \text{co} \cup \text{rf} \cup \text{fr}),
\]

where \text{pol} is \text{po} restricted to events at the same memory location.

We refer to this condition as \textit{SC-Per-Location}.
SC-Per-Location is equivalent to prohibiting the following five patterns:

\[
\begin{align*}
&\text{w}_1 \xrightarrow{\text{pol}} \text{co} \quad \text{w}_2 \\
&\text{r} \xrightarrow{\text{pol}} \text{fr} \\
&\text{rf} \xleftarrow{\text{pol}} \text{co} \\
&\text{w}_1 \\
&\text{w}_2 \\
&\text{r} \xrightarrow{\text{pol}} \text{rf} \\
&\text{w} \\
&\text{r}_1 \xrightarrow{\text{pol}} \text{fr} \\
&\text{w} \\
&\text{r}_2
\end{align*}
\]
SC-Per-Location is equivalent to prohibiting the following five patterns:

\[ \begin{align*}
&\text{pol} \xrightarrow{\text{co}} \textit{w}_1 \\
&\text{pol} \xrightarrow{\textit{w}_2} \text{co} \\
&\text{pol} \xrightarrow{\textit{fr}} \textit{w} \\
&\text{pol} \xrightarrow{\textit{rf}} \textit{r} \\
&\text{pol} \xrightarrow{\textit{rf}} \textit{w}_2 \\
&\text{pol} \xrightarrow{\textit{co}} \textit{w}_1 \\
&\text{pol} \xrightarrow{\textit{fr}} \textit{r}_1 \\
&\text{pol} \xrightarrow{\textit{fr}} \textit{r}_2 \\
&\text{pol} \xrightarrow{\textit{fr}} \textit{r}_2
\end{align*} \]

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

1. Introduction
   - Multiprocessor Reasoning
   - Weak Memory
   - Goals of this talk

2. An Axiomatic Weak Memory Model
   - Concurrent Executions
   - SC-Per-Location

3. ACL2 Mechanization

4. Conclusion
Outline

1 Introduction
   - Multiprocessor Reasoning
   - Weak Memory
   - Goals of this talk

2 An Axiomatic Weak Memory Model
   - Concurrent Executions
   - SC-Per-Location

3 ACL2 Mechanization

4 Conclusion
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
Conclusion

- We presented a (partial) mechanization in ACL2 of an axiomatic model of weak memory.
- This included a new proof of an equivalence theorem.
- I plan to investigate how this model (or a similar one) can be used practically for MP code proofs.
References

