Synthesizing Memory Models from Framework Sketches and Litmus Tests

James Bornholt
Emina Torlak       University of Washington
Memory consistency models define memory reordering behaviors on multiprocessors
Memory consistency models define memory reordering behaviors on multiprocessors...correctness of my compiler...

Compiler writers
Memory consistency models define memory reordering behaviors on multiprocessors

...correctness of my compiler...

Compiler writers

...rules to verify against...

Verification tools
Memory consistency models define memory reordering behaviors on multiprocessors

...correctness of my compiler...

Compiler writers

...rules to verify against...

Verification tools

...possible low-level behaviors...

Kernel/library developers
Memory consistency models define memory reordering behaviors on multiprocessors.

- Correctness of my compiler...
- Rules to verify against...
- Possible low-level behaviors...

Compiler writers
Verification tools
Kernel/library developers

Litmus tests and prose
Memory consistency models define memory reordering behaviors on multiprocessors

Compiler writers

Verification tools

Kernel/library developers

Litmus tests and prose

Formal specifications
Memory consistency models define memory reordering behaviors on multiprocessors.

- Correctness of my compiler...
- Rules to verify against...
- Possible low-level behaviors...

Compiler writers

Verification tools

Kernel/library developers

Litmus tests and prose

Formal specifications

- x86 [Sewell et al, CACM’10]
- PowerPC [Alglave et al, CAV’10, etc]
- ARM [Flur et al, POPL’16]
MemSynth

Litmus tests

Formal specifications
MemSynth

Litmus tests

Synthesize specifications

Formal specifications
MemSynth

Litmus tests

Framework sketch

Synthesize specifications

Formal specifications
MemSynth

1. Framework sketch
2. Synthesize specifications
3. Detect ambiguities
4. Formal specifications

Litmus tests
MemSynth

- Framework sketch
- Synthesize specifications
- Detect ambiguities
- Formal specifications
- Litmus tests
MemSynth

Synthesize specifications

Detect ambiguities
MemSynth

Framework sketches
define a class of memory models
MemSynth

Framework sketches
define a class of memory models

MemSynth engine
verification, equivalence, synthesis, ambiguity
MemSynth

Framework sketches
define a class of memory models

MemSynth engine
verification, equivalence, synthesis, ambiguity

Results
synthesize real-world memory model specs
Memory models and framework sketches
Litmus tests illustrate memory model behavior

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( X = 1 )</td>
<td>3 ( Y = 1 )</td>
</tr>
<tr>
<td>2 ( r1 = Y )</td>
<td>4 ( r2 = X )</td>
</tr>
</tbody>
</table>

Can \( r1 = 0 \land r2 = 0 \)?
Litmus tests illustrate memory model behavior

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (X = 1)</td>
<td>3 (Y = 1)</td>
</tr>
<tr>
<td>2 (r1 = Y)</td>
<td>4 (r2 = X)</td>
</tr>
</tbody>
</table>

Can \(r1 = 0 \land r2 = 0\)?

**Sequential consistency**: no
**Litmus tests illustrate memory model behavior**

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 X = 1</td>
<td>3 Y = 1</td>
</tr>
<tr>
<td>2 r1 = Y</td>
<td>4 r2 = X</td>
</tr>
</tbody>
</table>

Can $r1 = 0 \land r2 = 0$?

**Sequential consistency:** no

**x86:** yes!
Litmus tests illustrate memory model behavior

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 X = 1</td>
<td>3 Y = 1</td>
</tr>
<tr>
<td>2 r1 = Y</td>
<td>4 r2 = X</td>
</tr>
</tbody>
</table>

Can r1 = 0 ∧ r2 = 0?

**Sequential consistency:** no

**x86:** yes!

A memory model M is a set of constraints that define the possible executions (outcomes) of a program.
Litmus tests illustrate memory model behavior

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $X = 1$</td>
<td>3 $Y = 1$</td>
</tr>
<tr>
<td>2 $r_1 = Y$</td>
<td>4 $r_2 = X$</td>
</tr>
</tbody>
</table>

Can $r_1 = 0 \land r_2 = 0$?

**Sequential consistency:** no

**x86:** yes!

A **memory model** $M$ is a set of constraints that define the possible executions (outcomes) of a program.

Memory model $M$ **allows** litmus test $T$ if there exists an execution that satisfies $M$’s constraints.
Litmus tests illustrate memory model behavior

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 X = 1</td>
<td>3 Y = 1</td>
</tr>
<tr>
<td>2 r1 = Y</td>
<td>4 r2 = X</td>
</tr>
</tbody>
</table>

Can r1 = 0 ∧ r2 = 0?

**Sequential consistency:** no

**x86:** yes!

A memory model $M$ is a set of constraints that define the possible executions (outcomes) of a program.
Memory models, formally

Common formalizations based on relational logic

Example for sequential consistency:

$$\text{no} ^{(ws + fr + po + rf + fences)} \& \text{idem}$$

[Alglave et al, CAV’10]
Memory models, formally

Common formalizations based on relational logic

Example for sequential consistency:

\[ \text{no} \ ^\wedge (ws + fr + po + rf + fences) \ & \ iden \]

Memory model M allows test T: \[ \exists \ E. \ M(T,E) \]

Binary relations over program instructions

[Alglave et al, CAV’10]
Memory models, formally

Common formalizations based on relational logic

Example for sequential consistency:

\[ \neg \left( ws + fr + po + rf + fences \right) \land \text{idem} \]
Memory models, formally

Common formalizations based on \textit{relational logic}

Example for \textit{sequential consistency}:

\[
\text{no} \ ^{(ws + fr + po + rf + fences)} \& \text{idem}
\]

Binary relations over program instructions

Memory model \( M \) allows test \( T \):

\[
\exists E. \ M(T,E)
\]

[Alglave et al, CAV’10]
Memory models, formally

Common formalizations based on relational logic

Example for sequential consistency:

\[ \text{no} \ ^\land (\text{ws} + \text{fr} + \text{po} + \text{rf} + \text{fences}) \ \land \ \text{idem} \]

From program syntax

Binary relations over program instructions

Memory model \( M \) allows test \( T: \exists E. M(T,E) \)

[Alglave et al, CAV’10]
Memory models, formally

Common formalizations based on relational logic

Example for sequential consistency:

\[ \neg \left( \text{ws} + \text{fr} + \text{po} + \text{rf} + \text{fences} \right) \land \text{idem} \]

From program syntax

Binary relations over program instructions

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( X = 1 )</td>
<td>3 ( Y = 1 )</td>
</tr>
<tr>
<td>2 ( r1 = Y )</td>
<td>4 ( r2 = X )</td>
</tr>
</tbody>
</table>

Can \( r1 = 0 \land r2 = 0 \)?

[Alglave et al, CAV’10]
Memory models, formally

Common formalizations based on relational logic

Example for sequential consistency:

\[
\text{no} \quad (^{(ws + fr + po + rf + fences)} \quad \& \quad \text{idem})
\]

From program syntax:

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( X = 1 )</td>
<td>3 ( Y = 1 )</td>
</tr>
<tr>
<td>2 ( r1 = Y )</td>
<td>4 ( r2 = X )</td>
</tr>
</tbody>
</table>

Can \( r1 = 0 \land r2 = 0 \)?

Program order:

\( \text{po} = \{ (1, 2), (3, 4) \} \)

[Alglave et al., CAV’10]
Memory models, formally

Common formalizations based on relational logic

Example for sequential consistency:

\[ \text{no} \ \neg^\land (ws + fr + po + rf + fences) \ \& \ \text{idem} \]

Part of execution; implicitly existentially quantified

Program order:

\[ \text{po} = \{(1, 2), (3, 4)\} \]

[Alglave et al, CAV’10]
Framework sketches

A framework sketch defines the search space for synthesizing a memory model $M$ by including holes in constraints

\[ \text{no} \ ^\land (ws + fr + po + rf + fences) \ & \ iden \]
Framework sketches

A framework sketch defines the search space for synthesizing a memory model $M$ by including $holes$ in constraints

$$\text{no} \ ^{\text{Expression holes}} \ (ws + fr + ?? + ?? + ??) \ & \ iden$$
Framework sketches

A framework sketch defines the search space for synthesizing a memory model $M$ by including $holes$ in constraints

$$no ^{(ws + fr + ?? + ?? + ?? ) \& iden}$$

Framework sketches are the key design tool for synthesizing memory model specifications — they define the “interesting” candidate models
Memory model frameworks

\[ \text{no} \, \wedge (ws + fr + ?? + ?? + ??) \, \& \, \text{idem} \]

[Alglave et al, CAV’10]
Memory model frameworks

\[ \text{no} \quad \wedge (w_{s} + f_{r} + \text{ppo} + \text{grf} + \text{fences}) \quad \& \quad \text{idem} \]

- Preserved program order (same-thread reorderings)
- Global reads from (inter-thread order)
- Fence cumulativity (for Power, ARM, etc)

[Alglave et al, CAV’10]
Memory model frameworks

\[
\text{no} \ ^\sim (ws + fr + \textbf{ppo} + \textbf{grf} + \textbf{fences}) \ & \text{iden}
\]

- Preserved program order (same-thread reorderings)
- Global reads from (inter-thread order)
- Fence cumulativity (for Power, ARM, etc)

Sequential consistency

[Alglave et al, CAV’10]
Memory model frameworks

\[ \text{no} \ ^\wedge (ws + fr + ppo + grf + fences) \ & \ \text{iden} \]

- **Preserved program order** (same-thread reorderings)
- **Global reads from** (inter-thread order)
- **Fence cumulativity** (for Power, ARM, etc)

**Sequential consistency**
- \( po \)
- \( rf \)
- \( \emptyset \)

**Total store order (x86)**
- \( po - (Wr\rightarrow Rd) \)
- \( rf \ & \ \text{SameThd} \)
- \( \emptyset \)

[Alglave et al, CAV’10]
Memory model frameworks are common

- **Global time relational model**
  [Alglave et al, CAV’10]

- **Axiomatic “must-not-reorder” functions**
  [Mador-Haim et al, DAC’11]

- **Executable distributed consistency models**
  [Yang et al, IPDPS’04]

...
Ocelot: relational logic with holes

A relational logic DSL with synthesis support

Built on the Rosette solver-aided language [Torlak & Bodik, PLDI’14]

Expression holes for a synthesizer to complete

\[ \texttt{no} ^{\texttt{(ws + fr + ?? + ?? + ?? ) & iden}} \]

Available as a Racket package: raco pkg install ocelot
Ocelot: relational logic with holes

A relational logic DSL with synthesis support

Built on the Rosette solver-aided language [Torlak & Bodik, PLDI’14]

Expression holes for a synthesizer to complete

\[ \text{no} \ ^{(\text{ws} + \text{fr} + \text{??} + \text{??} + \text{??} )} \ & \text{idem} \]

Available as a Racket package: raco pkg install ocelot
Ocelot: relational logic with holes

A relational logic DSL with synthesis support

Built on the Rosette solver-aided language [Torlak & Bodik, PLDI’14]

Expression holes for a synthesizer to complete

\[ \text{no} \quad \neg (\text{ws} + \text{fr} + \text{??} + \text{??} + \text{??}) \quad \& \quad \text{idem} \]

Completions are expressions in relational logic with chosen operators, terminals, and depth.

operators = \{+, \&\}
terminals = \{po, ws\}
depth = 1

Available as a Racket package: raco pkg install ocelot
Ocelot: relational logic with holes

A relational logic DSL with synthesis support

Built on the Rosette solver-aided language [Torlak & Bodik, PLDI’14]

Expression holes for a synthesizer to complete

no ^ (ws + fr + ?? + ?? + ?? ) & iden

Completions are expressions in relational logic with chosen operators, terminals, and depth.

operators = {+, &}
terminals = {po, ws}
depth = 1

Available as a Racket package: raco pkg install ocelot
Queries

- Verification
- Equivalence
- Synthesis
- Ambiguity
Verification and equivalence

Common queries for automated memory model reasoning tools

Herd [Alglave et al, CAV’10]; MemAlloy [Wickerson et al, POPL’17]; etc.
**Verification and equivalence**

Common queries for automated memory model reasoning tools

**Herd** [Alglave et al, CAV’10]; **MemAlloy** [Wickerson et al, POPL’17]; etc.

\[ \exists E. M(T,E) \]

- Litmus test
- Memory model
- **VERIFY**
- SAT or UNSAT

Memory model M allows test T:
Verification and equivalence

Common queries for automated memory model reasoning tools

Herd [Alglave et al, CAV’10]; MemAlloy [Wickerson et al, POPL’17]; etc.

Litmus test $\rightarrow$ VERIFY $\rightarrow$ SAT or UNSAT

Memory model

Reduces to SAT (since litmus tests are loop-free)

Memory model M allows test T: $\exists E. M(T,E)$
Verification and equivalence

Common queries for automated memory model reasoning tools

Herd [Alglave et al, CAV’10]; MemAlloy [Wickerson et al, POPL’17]; etc.

- Litmus test
- Memory model
  → **VERIFY**
  → SAT or UNSAT
  Reduces to SAT (since litmus tests are loop-free)

- Memory model $M_A$
- Memory model $M_B$
  → **EQUIV**
  → Litmus test or UNSAT
Verification and equivalence

Common queries for automated memory model reasoning tools

Herd [Alglave et al, CAV’10]; MemAlloy [Wickerson et al, POPL’17]; etc.

Memory model \( M \) allows test \( T: \exists E. M(T,E) \)

Litmus test \( \Rightarrow \) VERIFY \( \Rightarrow \) SAT or UNSAT

Reduces to SAT (since litmus tests are loop-free)

Memory model \( M_A \) \( \Rightarrow \) EQUIV \( \Rightarrow \) Litmus test or UNSAT

UNSAT = bounded equivalence ("equivalent up to tests of size \( k \)"")
Synthesis

Find a memory model consistent with a set of litmus tests

Allowed litmus tests → SYNTH → Memory model
Forbidden litmus tests
Framework sketch
Synthesis

Find a memory model consistent with a set of litmus tests
Synthesis

Find a memory model consistent with a set of litmus tests

x86

Framework sketch
Synthesis

Find a memory model consistent with a set of litmus tests

2 allowed tests

8 forbidden tests

Framework sketch
Synthesis

Find a memory model consistent with a set of litmus tests

x86

2 allowed tests

8 forbidden tests

SYNTH

Total store order

Framework sketch
Synthesis

Find a memory model consistent with a set of litmus tests

Allowed litmus tests

Forbidden litmus tests

Memory model

∃ E. M(T,E)

T⁺ → M

T⁻ → Framework sketch
Synthesis

Find a memory model consistent with a set of litmus tests

- **Allowed litmus tests**
  \[ T^+ \Rightarrow \bigwedge_{T \in T^+} \exists E. M(T,E) \]

- **Forbidden litmus tests**
  \[ T^- \Rightarrow \]

**Memory model**
\[ M \]

**Framework sketch**
Synthesis

Find a memory model consistent with a set of litmus tests

 Allowed litmus tests

\[ T^+ \Rightarrow \bigwedge_{T \in T^+} \exists E. M(T,E) \]

Forbidden litmus tests

\[ T^- \Rightarrow \bigwedge_{T \in T^-} \forall E. \neg M(T,E) \]

\[ \exists E. M(T,E) \]

Memory model

Framework sketch
Synthesis

Find a memory model consistent with a set of litmus tests

 Allowed litmus tests

\[ T^+ \rightarrow \bigwedge_{T \in T^+} \exists E. M(T,E) \]

Forbidden litmus tests

\[ T^- \rightarrow \bigwedge_{T \in T^-} \forall E. \neg M(T,E) \]

Solved incrementally, like counterexample-guided inductive synthesis (CEGIS)

Memory model

\[ \exists E. M(T,E) \]

Framework sketch
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

Key idea: after synthesis, is there a different memory model that explains the tests?
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

Key idea: after synthesis, is there a different memory model that explains the tests?

Allowed litmus tests
Forbidden litmus tests

AMBIG
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

**Key idea:** after synthesis, is there a different memory model that explains the tests?

- **Allowed litmus tests**
- **Forbidden litmus tests**
- **Memory model \( M_A \)**
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

**Key idea:** after synthesis, is there a different memory model that explains the tests?

- Allowed litmus tests
- Forbidden litmus tests
- Memory model $M_A$

Framework sketch
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

**Key idea:** after synthesis, is there a *different* memory model that explains the tests?

- **Allowed litmus tests**
- **Forbidden litmus tests**
- **Memory model** $M_A$

**Framework sketch**

**Memory model** $M_B$

**Litmus test**
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

Key idea: after synthesis, is there a different memory model that explains the tests?

Allowed litmus tests

Forbidden litmus tests

Memory model $M_A$

Framework sketch

Memory model $M_B$

Litmus test

The new memory model must be \textit{semantically different} from the input: $M_A$ and $M_B$ must disagree about a new test $T$

Similar to oracle-guided synthesis [Jha et al, ICSE’10]
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $X = 1$</td>
<td>3. $Y = 1$</td>
</tr>
<tr>
<td>2. $r1 = Y$</td>
<td>4. $r2 = X$</td>
</tr>
</tbody>
</table>

Can $r1 = 0 \land r2 = 0$?

Total store order (x86)
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $X = 1$</td>
<td>3 $Y = 1$</td>
</tr>
<tr>
<td>2 $r_1 = Y$</td>
<td>4 $r_2 = X$</td>
</tr>
</tbody>
</table>

Can $r_1 = 0 \land r_2 = 0$?

Total store order (x86)

Is there another **semantically different** memory model that also allows this test?
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 X = 1</td>
<td>3 Y = 1</td>
</tr>
<tr>
<td>2 r1 = Y</td>
<td>4 r2 = X</td>
</tr>
</tbody>
</table>

Can r1 = 0 ∧ r2 = 0?

Partial store order (SPARC)

Total store order (x86)

Is there another semantically different memory model that also allows this test?
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

<table>
<thead>
<tr>
<th>Total store order (x86)</th>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 X = 1</td>
<td>3 Y = 1</td>
<td></td>
</tr>
<tr>
<td>2 r1 = Y</td>
<td>4 r2 = X</td>
<td></td>
</tr>
<tr>
<td>Can r1 = 0 ∧ r2 = 0?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Is there another semantically different memory model that also allows this test?

<table>
<thead>
<tr>
<th>Partial store order (SPARC)</th>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 X = 1</td>
<td>3 r1 = Y</td>
<td></td>
</tr>
<tr>
<td>2 Y = 1</td>
<td>4 r2 = X</td>
<td></td>
</tr>
<tr>
<td>Can r1 = 1 ∧ r2 = 0?</td>
<td></td>
<td>✓ PSO</td>
</tr>
</tbody>
</table>
The Synthesis-Ambiguity Cycle

Litmus tests
The Synthesis-Ambiguity Cycle

Documentation

Random/systematic generation

Architects

Litmus tests
The Synthesis-Ambiguity Cycle

Litmus tests
The Synthesis-Ambiguity Cycle

Litmus tests

Synth

Memory model specification
The Synthesis-Ambiguity Cycle

Litmus tests

Memory model specification

3 5 6
1 2 4
The Synthesis-Ambiguity Cycle

Litmus tests → SYNTH → AMBIG → Memory model specification

Unique memory model (within framework sketch)
Results
Synthesizing existing memory models

PowerPC

x86
Synthesizing existing memory models

PowerPC: 768 tests
[Alglave et al, CAV'10]

x86: 10 tests
Synthesizing existing memory models

<table>
<thead>
<tr>
<th>Platform</th>
<th>Tests</th>
<th>Synthesis</th>
<th>Search space</th>
</tr>
</thead>
<tbody>
<tr>
<td>PowerPC</td>
<td>768 tests</td>
<td>12 seconds</td>
<td>$2^{1406}$</td>
</tr>
<tr>
<td>x86</td>
<td>10 tests</td>
<td>2 seconds</td>
<td>$2^{624}$</td>
</tr>
</tbody>
</table>
Synthesizing existing memory models

PowerPC  
768 tests  
[Alglave et al, CAV’10]  
✓ 12 seconds  
Search space: $2^{1406}$

x86  
10 tests  
✓ 2 seconds  
Search space: $2^{624}$

Not equivalent to published model!
Synthesizing existing memory models

PowerPC

768 tests
[Alglave et al, CAV’10]

✓ 12 seconds
Search space: $2^{1406}$

Not equivalent to published model!

x86

10 tests

✓ 2 seconds
Search space: $2^{624}$

Not equivalent to TSO!
Synthesizing existing memory models

PowerPC
- 768 tests
- [Alglave et al, CAV'10]
- ✓ 12 seconds
- Search space: $2^{1406}$
- Not equivalent to published model!

x86
- 10 tests
- ✓ 2 seconds
- Search space: $2^{624}$
- Not equivalent to TSO!

Ambiguity
- 9 new tests
  - sync, lwsync, etc.
- 4 new tests
  - mfence, xchg
Other results

Implemented another framework sketch [Mador-Haim et al, DAC’11]

Found typo in paper; couldn’t fix by hand, but synthesized repair
Other results

Implemented another framework sketch [Mador-Haim et al, DAC’11]

  Found typo in paper; couldn’t fix by hand, but synthesized repair

Order of magnitude faster than the Alloy general-purpose relational solver for verification and equivalence

  Ocelot offers finer-grained control over relational constraints
Other results

Implemented another framework sketch [Mador-Haim et al, DAC’11]

  Found typo in paper; couldn’t fix by hand, but synthesized repair

Order of magnitude faster than the Alloy general-purpose relational solver for verification and equivalence

  Ocelot offers finer-grained control over relational constraints

Comparable performance to existing custom memory model tool for verification (Herd [Alglave et al, CAV’10])
MemSynth

Framework sketches
define a class of memory models

MemSynth engine
verification, equivalence, synthesis, ambiguity

Results
synthesize real-world memory model specs

memsynth.uwplse.org