checking axiomatic specifications of memory models

Emina Torlak · Mandana Vaziri · Julian Dolby

MIT SAT/SMT Summer School · June 16, 2011
Introduction

memory model

- contract between programmer and programming environment
- specifies which writes can be seen by a read

\[
\begin{array}{l}
x = 0, \ y = 0 \\
\text{r}_1 = x \\
y = 1 \\
\text{r}_2 = y \\
x = 1 \\
\text{r}_1 = \text{r}_2 = 1? \\
\end{array}
\]
Introduction

memory model

- contract between programmer and programming environment
- specifies which writes can be seen by a read
- described (in)formally by a set of axioms and litmus tests
Introduction

memory model

- contract between programmer and programming environment
- specifies which writes can be seen by a read
- described (in)formally by a set of axioms and litmus tests
- hard to design and reason about
MemSAT overview

- litmus test
- memory model
- finitization parameters
- legality witness
- proof of illegality
MemSAT overview

annotated java program with one or more assertions

P

memory model

MemSAT

finitization parameters

legality witness

proof of illegality
MemSAT overview

Annotated Java program with one or more assertions

Set of constraints in relational logic

Finitization parameters

MemSAT

Legality witness

Proof of illegality
MemSAT overview

- annotated java program with one or more assertions
- translate P to relational logic
- combine result with M
- set of constraints in relational logic
- finitization parameters
- legality witness
- proof of illegality
- $F(P, M) \rightarrow SAT$
MemSAT overview

* annotated java program with one or more assertions

P → F(P, M) → kodkod → legality witness

M → set of constraints in relational logic

→ finitization parameters

→ translate P to relational logic
→ combine result with M
→ solve combined constraints

→ proof of illegality
MemSAT overview

- annotated java program with one or more assertions
- set of constraints in relational logic
- translate P to relational logic
- combine result with M
- solve combined constraints
- finitization parameters
- model (solution) of the legality formula
- proof of illegality
MemSAT overview

- Annotated Java program with one or more assertions
- Set of constraints in relational logic
- \( F(P, M) \) parameters
  - Translate \( P \) to relational logic
  - Combine result with \( M \)
  - Solve combined constraints
- Model (solution) of the legality formula
- Minimal unsatisfiable core of the legality formula
Specifying a litmus test

\[ x = 0, y = 0 \]

\[ r1 = x \]
\[ y = 1 \]
\[ r2 = y \]
\[ x = 1 \]

\[ r1 == r2 == 1? \]

```
public class Test0 {
    static int x = 0;
    static int y = 0;

    @thread
    public static void thread1() {
        final int r1 = x;
        y = 1;
        assert r1 == 1;
    }

    @thread
    public static void thread2() {
        final int r2 = y;
        x = 1;
        assert r2 == 1;
    }
}
```

- control flow
- synchronize
- method calls
- field and array accesses
- assertions
Specifying a memory model
Specifying a memory model

- constants
- variables
- relational logic
  - first order logic ($\forall$, $\exists$, $\land$, $\lor$, $\neg$)
  - relational algebra ($\times$, $\cup$, $\cap$, $\setminus$)
  - bitvector arithmetic ($+$, $-$, $\times$, $/$)

5
Specifying a memory model

- relational constants capture static properties of a program
  - co, control flow
  - to, thread order

- first order logic (\( \forall, \exists, \land, \lor, \neg \))
- relational algebra (\( \cdot, \cup, \cap, \div, \times, \subseteq \))
- bitvector arithmetic (+, -, *, /)

relational constants
variables

relational logic
Specifying a memory model

relational constants capture static properties of a program
  › co, control flow
  › to, thread order

first order logic ($\forall$, $\exists$, $\land$, $\lor$, $\neg$)
relational algebra ($\cdot$, $\cup$, $\cap$, $\setminus$, $\times$, $\subseteq$)
bitvector arithmetic ($+$, $-$, $\ast$, $/$)

$x = 0, y = 0$

<table>
<thead>
<tr>
<th>r1 = x</th>
<th>r2 = y</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = 1$</td>
<td>$x = 1$</td>
</tr>
</tbody>
</table>
Specifying a memory model

relational constants capture static properties of a program
  - co, control flow
  - to, thread order

first order logic ($\forall$, $\exists$, $\wedge$, $\vee$, $\neg$)
relational algebra ($\cdot$, $\cup$, $\cap$, $\div$, $\times$, $\subseteq$)
bitevector arithmetic (+, -, *, /)

constants

variables

relational logic

$t_0$: $x = 0, y = 0$
$r1 = x$
$r2 = y$
y = 1$
x = 1$

$t_1$

$t_2$
Specifying a memory model

relational constants capture static properties of a program
- co, control flow
- to = { ⟨t₀, t₁⟩, ⟨t₀, t₂⟩ }

x = 0, y = 0   t₀
r₁ = x
y = 1
x = 1

r₂ = y

first order logic (∨, ∃, ∧, ∨, ¬)
relational algebra (·, ∪, ∩, ∖, ×, ⊆)
bitvector arithmetic (+, −, *, /)

constants

variables

relational logic
Specifying a memory model

relational constants capture static properties of a program
  - co, control flow
  - to = \{ \langle t0, t1 \rangle, \langle t0, t2 \rangle \}

relational logic

first order logic (∀, ∃, ∧, ∨, ¬)
relational algebra (., ∪, ∩, ∖, ×, ⊆)
bitvector arithmetic (+, -, *, /)

variables

relational variables capture runtime properties of a program
  - A, set of all executed actions
  - W, maps reads to seen writes
  - V, maps writes to written values
  - l, maps reads/writes to locations
  - m, maps locks/unlocks to monitors

x = 0, y = 0
r1 = x
y = 1
x = 1
Specifying a memory model

relational constants capture static properties of a program
  ‣ co, control flow
  ‣ to = \{ \langle t0, t1 \rangle, \langle t0, t2 \rangle \}

first order logic (\forall, \exists, \land, \lor, \neg)
relational algebra (\cdot, \cup, \cap, \setminus, \times, \subseteq)
bitvector arithmetic (+, -, *, /)

relational logic

constants

variables

a00: start
a01: write(x, 0)
a02: write(y, 0)
a03: end

r1 = x
r2 = y
y = 1
x = 1

t1

t2

relational variables capture runtime properties of a program
  ‣ A, set of all executed actions
  ‣ W, maps reads to seen writes
  ‣ V, maps writes to written values
  ‣ l, maps reads/writes to locations
  ‣ m, maps locks/unlocks to monitors
Specifying a memory model

relational constants capture static properties of a program
  - co, control flow
  - to = { \langle t0, t1 \rangle, \langle t0, t2 \rangle }

relational logic

first order logic (\forall, \exists, \land, \lor, \neg)
relational algebra (\cdot, \cup, \cap, \setminus, \times, \subseteq)
bitvector arithmetic (+, -, *, /)

relational variables capture runtime properties of a program
  - A, set of all executed actions
  - W, maps reads to seen writes
  - V, maps writes to written values
  - l, maps reads/writes to locations
  - m, maps locks/unlocks to monitors

a00: start
a01: write(x, 0)
a02: write(y, 0)
a03: end

a10: start
a11: read(x, 0)
a12: write(y, 1)
a13: end

r2 = y
x = l

\text{t2}
Specifying a memory model

relational constants capture static properties of a program
  - co, control flow
  - to = { ⟨t0, t1⟩, ⟨t0, t2⟩ }

first order logic (∀, ∃, ∧, ∨, ¬)
relational algebra (., ∪, ∩, ∖, ×, ⊆)
bitvector arithmetic (+, −, *, /,)

relational logic

relational variables capture runtime properties of a program
  - A, set of all executed actions
  - W, maps reads to seen writes
  - V, maps writes to written values
  - l, maps reads/writes to locations
  - m, maps locks/unlocks to monitors
Specifying a memory model

relational constants capture static properties of a program
- co, control flow
- to = { (t0, t1), (t0, t2) }

first order logic (\forall, \exists, \land, \lor, \neg)
relational algebra (., \cup, \cap, \div, \times, \subseteq)
bitvector arithmetic (+, -, *, /)

relational variables capture runtime properties of a program
- A = { ⟨a00⟩, ⟨a01⟩, ..., ⟨a23⟩ }
- W, maps reads to seen writes
- V, maps writes to written values
- l, maps reads/writes to locations
- m, maps locks/unlocks to monitors
Specifying a memory model

relational constants capture static properties of a program
- co, control flow
- to = \{ \langle t0, t1 \rangle, \langle t0, t2 \rangle \}

first order logic (\forall, \exists, \land, \lor, \neg)
relational algebra (\cdot, \cup, \cap, \setminus, \times, \subseteq)
bitvector arithmetic (+, -, *, \slash)

relational logic

constants

variables

a00: start
a01: write(x, 0)
a02: write(y, 0)
a03: end

a10: start
a11: read(x, 0)
a12: write(y, 1)
a13: end

a20: start
a21: read(y, 1)
a22: write(x, 1)
a23: end

A = \{ \langle a00 \rangle, \langle a01 \rangle, ..., \langle a23 \rangle \}
W = \{ \langle a11, a01 \rangle, \langle a21, a12 \rangle \}
V, maps writes to written values
I, maps reads/writes to locations
m, maps locks/unlocks to monitors

relational variables capture runtime properties of a program
Specifying a memory model

relational constants capture static properties of a program
- co, control flow
- to = {〈t0, t1〉, 〈t0, t2〉}

first order logic (\(\forall, \exists, \land, \lor, \neg\))
relational algebra (\(., \cup, \cap, \setminus, \times\))
bitvector arithmetic (+, -, *, /)

variables

relational variables capture runtime properties of a program
- \(A = \{\langle a00\rangle, \langle a01\rangle, ..., \langle a23\rangle\}\)
- \(W = \{\langle a11, a01\rangle, \langle a21, a12\rangle\}\)
- \(V = \{\langle a01, 0\rangle, \langle a02, 0\rangle, \langle a12, 1\rangle, \langle a22, 1\rangle\}\)
- l, maps reads/writes to locations
- m, maps locks/unlocks to monitors
Specifying a memory model

relational constants capture
static properties of a program
› co, control flow
› to = {〈t0, t1〉, 〈t0, t2〉}

first order logic (∨, ∃, ∧, ∨, ¬)
relational algebra (., ∪, ∩, /, ×, ⊆)
bitevector arithmetic (+, -, *, /, )

relational variables capture runtime
properties of a program
› A = {〈a00〉, 〈a01〉, ..., 〈a23〉}
› W = {〈a11, a01〉, 〈a21, a12〉}
› V = {〈a01, 0〉, 〈a02, 0〉, 〈a12, 1〉, 〈a22, 1〉}
› l = {〈a01, x〉, 〈a02, y〉, ..., 〈a22, x〉}
› m, maps locks/unlocks to monitors
Specifying a memory model

relational constants capture static properties of a program
- co, control flow
- to = \{ (t₀, t₁), (t₀, t₂) \}

first order logic (\forall, \exists, \land, \lor, \neg)
relational algebra (., \cup, \cap, \setminus, \times, \subseteq)
bitvector arithmetic (+, -, \ast, /,)

relational variables capture runtime properties of a program
- A = \{ \langle a₀₀ \rangle, \langle a₀₁ \rangle, ..., \langle a₂₃ \rangle \}
- W = \{ \langle a₁₁, a₀₁ \rangle, \langle a₂₁, a₁₂ \rangle \}
- V = \{ \langle a₀₁, 0 \rangle, \langle a₀₂, 0 \rangle, \langle a₁₂, 1 \rangle, \langle a₂₂, 1 \rangle \}
- l = \{ \langle a₀₁, x \rangle, \langle a₀₂, y \rangle, ..., \langle a₂₂, x \rangle \}
- m = \{ \}

a₀₀: start
a₀₁: write(x, 0)
a₀₂: write(y, 0)
a₀₃: end

a₁₀: start
a₁₁: read(x, 0)
a₁₂: write(y, 1)
a₁₃: end

a₂₀: start
a₂₁: read(y, 1)
a₂₂: write(x, 1)
a₂₃: end
Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. Execution order is total,
2. antisymmetric, and
3. transitive.
4. It respects the control flow and
5. thread order.
6. Reads cannot see out of order writes.
7. No write interferes between a read and the write seen by that read.
Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. $\forall \ i, j: A \ | \ i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i]$  
   Execution order is total,
2. antisymmetric, and
3. transitive.
4. It respects the control flow and
5. thread order.
6. Reads cannot see out of order writes.
7. No write interferes between a read and the write seen by that read.
Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. \( \forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i] \)
2. \( \forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i] \)
3. transitive.
4. It respects the control flow and thread order.
5. Reads cannot see out of order writes.
6. No write interferes between a read and the write seen by that read.

Execution order is total, antisymmetric, and transitive.

Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. \( \forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i] \)
2. \( \forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i] \)
3. transitive.
4. It respects the control flow and thread order.
5. Reads cannot see out of order writes.
6. No write interferes between a read and the write seen by that read.
Example: sequential consistency

**interleaved semantics**

all statements appear to execute in a total order that agrees with the program text

1. \( \forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i] \)
2. \( \forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg\text{ord}[j, i] \)
3. \( \forall i, j, k: A \mid (\text{ord}[i, j] \land \text{ord}[j, k]) \Rightarrow \text{ord}[i, k] \)

Execution order is total, antisymmetric, and transitive.

4. It respects the control flow and thread order.
5. Reads cannot see out of order writes.
6. No write interferes between a read and the write seen by that read.
Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. \( \forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i] \)
2. \( \forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg\text{ord}[j, i] \)
3. \( \forall i, j, k: A \mid (\text{ord}[i, j] \land \text{ord}[j, k]) \Rightarrow \text{ord}[i, k] \)
4. \( \forall i, j: A \mid (t[i] = t[j] \land \text{co}^{+}[i, j]) \Rightarrow \text{ord}[i, j] \)
5. thread order.
6. Reads cannot see out of order writes.
7. No write interferes between a read and the write seen by that read.

Execution order is total,
antisymmetric, and
transitive.
It respects the control flow and
Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i]$
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg\text{ord}[j, i]$
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \land \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$
4. $\forall i, j: A \mid (t[i] = t[j] \land \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$
5. $\forall i, j: A \mid (t[i] \neq t[j] \land \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$
6. Reads cannot see out of order writes.
7. No write interferes between a read and the write seen by that read.

Execution order is total, antisymmetric, and transitive.
It respects the control flow and thread order.
Example: sequential consistency

- **Interleaved semantics**
  all statements appear to execute in a total order that agrees with the program text

1. \( \forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i] \)
2. \( \forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg\text{ord}[j, i] \)
3. \( \forall i, j, k: A \mid (\text{ord}[i, j] \land \text{ord}[j, k]) \Rightarrow \text{ord}[i, k] \)
4. \( \forall i, j: A \mid (t[i] = t[j] \land \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j] \)
5. \( \forall i, j: A \mid (t[i] \neq t[j] \land \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j] \)
6. \( \forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]] \)
7. No write interferes between a read and the write seen by that read.

Execution order is total, antisymmetric, and transitive.
It respects the control flow and thread order.
Reads cannot see out of order writes.
Example: sequential consistency

interleaved semantics

all statements appear to execute in a total order that agrees with the program text

1. \( \forall i, j: A | i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i] \)
2. \( \forall i, j: A | \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i] \)
3. \( \forall i, j, k: A | (\text{ord}[i, j] \land \text{ord}[j, k]) \Rightarrow \text{ord}[i, k] \)
4. \( \forall i, j: A | (t[i] = t[j] \land \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j] \)
5. \( \forall i, j: A | (t[i] \neq t[j] \land \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j] \)
6. \( \forall k: A \cap \text{Read} | \neg \text{ord}[k, W[k]] \)
7. \( \forall k: A \cap \text{Read}, j: A \cap \text{Write} | \neg (l[k] = l[j] \land \text{ord}[W[k], j] \land \text{ord}[j, k]) \)

Execution order is total, antisymmetric, and transitive.

It respects the control flow and thread order.

Reads cannot see out of order writes.

No write interferes between a read and the write seen by that read.
Example:  Java memory model

committing semantics

an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. \( \forall i: [1..k] \mid C_i \subseteq A_i \)
2. \( \forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \leftrightarrow \text{hb}[W[r], r]) \land \neg \text{hb}[r, W[r]] \)
3. \( \forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V \)
4. \( \forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W \)
5. \( \forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}[W_i[r], r] \)
6. \( \forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1} \)
7. \( \forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \land \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1} \)
Example: Java memory model

An execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions.

1. \( \forall i: [1..k] \mid C_i \subseteq A_i \)
2. \( \forall i: [1..k] \), \( r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \Leftrightarrow \text{hbi}[W[r], r]) \wedge \neg \text{hb}[r, W[r]] \)
3. \( \forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V \)
4. \( \forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W \)
5. \( \forall i: [1..k] \), \( r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}[W_i[r], r] \)
6. \( \forall i: [1..k] \), \( r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1} \)
7. \( \forall i: [1..k] \), \( y: C_i \), \( x: A_i \mid (y \in \text{Special} \wedge \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1} \)

Initial execution: reads can only see writes that happen-before them.
Example: Java memory model

committing semantics

an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. $\forall i: [1..k] | C_i \subseteq A_i$
2. $\forall i: [1..k], r: C_i \cap \text{Read} | (\text{hb}[W[r], r] \Leftrightarrow \text{hb}[W[r], r]) \land \neg \text{hb}[r, W[r]]$
3. $\forall i: [1..k] | C_i \triangleleft V_i = C_i \triangleleft V$
4. $\forall i: [1..k] | C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W$
5. $\forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} | \text{hb}[W_i[r], r]$
6. $\forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} | W_i[r] \subseteq C_{i-1}$
7. $\forall i: [1..k], y: C_i, x: A_i | (y \subseteq \text{Special} \land \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1}$

initial execution: reads can only see writes that happen-before them
Example: Java memory model

### Committing semantics

An execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions.

1. \( \forall i: [1..k] \mid C_i \subseteq A_i \)
2. \( \forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \leftrightarrow \text{hb}_i[W[r], r]) \wedge \neg \text{hb}_i[r, W[r]] \)
3. \( \forall i: [1..k] \mid C_i \propto V_i = C_i \propto V \)
4. \( \forall i: [1..k] \mid C_{i-1} \propto W_i = C_{i-1} \propto W \)
5. \( \forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}_i[W[i][r], r] \)
6. \( \forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W[i][r] \subseteq C_{i-1} \)
7. \( \forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \wedge \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1} \)

\( i^{th} \text{ execution: committed reads can see committed writes; other reads must see writes that happen-before them} \)
Example: Java memory model

Committing semantics

An execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions.

1. \( \forall i: [1..k] \mid C_i \subseteq A_i \)
2. \( \forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \Leftrightarrow \text{hb}_{i}[W[r], r]) \land \neg \text{hb}_{i}[r, W[r]] \)
3. \( \forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V \)
4. \( \forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W \)
5. \( \forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}_{i}[W[r], r] \)
6. \( \forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W[r] \subseteq C_{i-1} \)
7. \( \forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \land \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1} \)

I\( ^{th}\) execution: committed reads can see committed writes; other reads must see writes that happen-before them.
Example: Java memory model

committing semantics

an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. \( \forall i: [1..k] \mid C_i \subseteq A_i \)
2. \( \forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \leftrightarrow \text{hb}[W[r], r]) \land \neg \text{hb}[r, W[r]] \)
3. \( \forall i: [1..k] \mid C_i \triangleleft V_i = C_i \triangleleft V \)
4. \( \forall i: [1..k] \mid C_{i-1} \triangleleft W_i = C_{i-1} \triangleleft W \)
5. \( \forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}[W[r], r] \)
6. \( \forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1} \)
7. \( \forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \land \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1} \)
Example: Java memory model

committing semantics
an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. \( \forall i: [1..k] \mid C_i \subseteq A_i \)
2. \( \forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \Leftrightarrow \text{hb}[W[r], r]) \land \neg \text{hb}[r, W[r]] \)
3. \( \forall i: [1..k] \mid C_i \triangleright V_i = C_i \triangleright V \)
4. \( \forall i: [1..k] \mid C_{i-1} \triangleright W_i = C_{i-1} \triangleright W \)
5. \( \forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}[W_i[r], r] \)
6. \( \forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1} \)
7. \( \forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \land \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1} \)

\( i^{th} \) execution: committed reads can see committed writes; other reads must see writes that happen-before them
Example: Java memory model

committing semantics
an execution is legal if it can be derived by committing and executing actions in a sequence of speculative executions

1. \( \forall i: [1..k] \mid C_i \subseteq A_i \)
2. \( \forall i: [1..k], r: C_i \cap \text{Read} \mid (\text{hb}[W[r], r] \leftrightarrow \text{hb}_i[W[r], r]) \land \neg \text{hb}_i[r, W[r]] \)
3. \( \forall i: [1..k] \mid C_i \sqsubseteq V_i = C_i \sqsubseteq V \)
4. \( \forall i: [1..k] \mid C_{i-1} \sqsubseteq W_i = C_{i-1} \sqsubseteq W \)
5. \( \forall i: [1..k], r: (A_i \setminus C_i) \cap \text{Read} \mid \text{hb}_i[W[r], r] \)
6. \( \forall i: [1..k], r: (C_i \setminus C_{i-1}) \cap \text{Read} \mid W_i[r] \subseteq C_{i-1} \)
7. \( \forall i: [1..k], y: C_i, x: A_i \mid (y \subseteq \text{Special} \land \text{hb}[x, y]) \Rightarrow x \subseteq C_{i-1} \)
Witness of legality (model)

\[ x = 0, y = 0 \]

\[
\begin{array}{c|c}
\text{r1} & \text{r2} \\
\hline
x & y \\
y & x \\
\end{array}
\]

witness: an execution of the program that satisfies both the assertions and the memory model constraints.
Proof of illegality (minimal core)

\[ x = 0, y = 0 \]

\[
\begin{array}{c|c}
 r_1 &= x \\
y &= 1 \\
\hline
 r_1 \equiv l & \& r_2 \equiv l?
\end{array}
\]

1. \( \forall i, j: A \mid i \neq j \Rightarrow ord[i, j] \lor ord[j, i] \)  \textcolor{red}{\text{SC}}
2. \( \forall i, j: A \mid ord[i, j] \Rightarrow \neg ord[j, i] \)
3. \( \forall i, j, k: A \mid (ord[i, j] \land ord[j, k]) \Rightarrow ord[i, k] \)
4. \( \forall i, j: A \mid (t[i] = t[j] \land co^+[i, j]) \Rightarrow ord[i, j] \)
5. \( \forall i, j: A \mid (t[i] \neq t[j] \land to^+[t[i], t[j]]) \Rightarrow ord[i, j] \)
6. \( \forall k: A \cap \text{Read} \mid \neg ord[k, W[k]] \)
7. \( \forall k: A \cap \text{Read, j: A \cap Write} \mid \neg (l[k] = l[j] \land ord[W[k], j] \land ord[j, k]) \)

\[ V[a_01] = 0 \]
\[ V[a_02] = 0 \]

\[ V[W[a_{11}]] = 1 \]
\[ V[W[a_{21}]] = 1 \]

\( \forall i, j: A \mid i \neq j \Rightarrow ord[i, j] \lor ord[j, i] \)
\( \forall i, j, k: A \mid (ord[i, j] \land ord[j, k]) \Rightarrow ord[i, k] \)
\( \forall i, j: A \mid (t[i] = t[j] \land co^+[i, j]) \Rightarrow ord[i, j] \)
\( \forall k: A \cap \text{Read} \mid \neg ord[k, W[k]] \)

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed
Proof of illegality (minimal core)

SC

<table>
<thead>
<tr>
<th>$x = 0, y = 0$</th>
<th>$r_1 = x$</th>
<th>$r_2 = y$</th>
<th>$y = 1$</th>
<th>$x = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1 == 1 \land \land r_2 == 1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. $\forall i, j: A \mid i \neq j \Rightarrow ord[i, j] \lor ord[j, i]$ [SC]
2. $\forall i, j: A \mid ord[i, j] \Rightarrow \neg ord[j, i]$ 
3. $\forall i, j, k: A \mid (ord[i, j] \land ord[j, k]) \Rightarrow ord[i, k]$ 
4. $\forall i, j: A \mid (t[i] = t[j] \land co^*[i, j]) \Rightarrow ord[i, j]$ 
5. $\forall i, j: A \mid (t[i] \neq t[j] \land to^*[t[i], t[j]]) \Rightarrow ord[i, j]$ 
6. $\forall k: A \cap Read \mid \neg ord[k, W[k]]$ 
7. $\forall k: A \cap Read, j: A \cap Write \mid \neg (l[k] = l[j] \land ord[W[k], j] \land ord[j, k])$ 

$V[a_{01}] = 0$
$V[a_{02}] = 0$

$V[W[a_{11}]] = 1$
$V[W[a_{21}]] = 1$

$\forall i, j: A \mid i \neq j \Rightarrow ord[i, j] \lor ord[j, i]$

$\forall i, j, k: A \mid (ord[i, j] \land ord[j, k]) \Rightarrow ord[i, k]$ 

$\forall i, j: A \mid (t[i] = t[j] \land co^*[i, j]) \Rightarrow ord[i, j]$ 

$\forall k: A \cap Read \mid \neg ord[k, W[k]]$

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed
Proof of illegality (minimal core)

<table>
<thead>
<tr>
<th>x = 0, y = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1 = x</td>
</tr>
<tr>
<td>y = 1</td>
</tr>
<tr>
<td>r2 = y</td>
</tr>
<tr>
<td>r1 == 1 &amp;&amp; r2 == 1?</td>
</tr>
</tbody>
</table>

1. $\forall i, j: A \mid i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i]$  
2. $\forall i, j: A \mid \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i]$  
3. $\forall i, j, k: A \mid (\text{ord}[i, j] \land \text{ord}[j, k]) \Rightarrow \text{ord}[i, k]$  
4. $\forall i, j: A \mid (t[i] = t[j] \land \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j]$  
5. $\forall i, j: A \mid (t[i] \neq t[j] \land \text{to}^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j]$  
6. $\forall k: A \cap \text{Read} \mid \neg \text{ord}[k, W[k]]$  
7. $\forall k: A \cap \text{Read}, j: A \cap \text{Write} \mid \neg (\lceil k \rceil = \lceil j \rceil \land \text{ord}[W[k], j] \land \text{ord}[j, k])$

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed.
Proof of illegality (minimal core)

\[
\begin{align*}
\begin{array}{c|c}
 x = 0, y = 0 \\
r1 = x & r2 = y \\
y = 1 & x = 1 \\
r1==1 \land r2==1?
\end{array}
\end{align*}
\]

\[V[a_{01}] = 0\]
\[V[a_{02}] = 0\]
\[V[W[a_{11}]] = 1\]
\[V[W[a_{21}]] = 1\]
\[
\begin{align*}
\forall i, j: A \mid i \neq j & \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i] \\
\forall i, j: A \mid \text{ord}[i, j] & \Rightarrow \neg\text{ord}[j, i] \\
\forall i, j, k: A \mid (\text{ord}[i, j] \land \text{ord}[j, k]) & \Rightarrow \text{ord}[i, k] \\
\forall i, j: A \mid (t[i] = t[j] \land \text{co}^+[i, j]) & \Rightarrow \text{ord}[i, j] \\
\forall i, j: A \mid (t[i] \neq t[j] \land \text{to}^+[t[i], t[j]]) & \Rightarrow \text{ord}[i, j] \\
\forall k: A \cap \text{Read} \mid \neg\text{ord}[k, W[k]] \\
\forall k: A \cap \text{Read}, j: A \cap \text{Write} \mid \\
\neg (l[k] = l[j] \land \text{ord}[W[k], j] \land \text{ord}[j, k])
\end{align*}
\]

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed
Proof of illegality (minimal core)

| x = 1, y = 0 |
| r1 = x | r2 = y |
| y = 1 |
| r1 == 1 && r2 == 1? |

a01: write(x, 1)
a02: write(y, 0)
a11: read(x, 1)
a12: write(y, 1)
a21: read(y, 1)
a22: write(x, 1)

1. \( \forall i, j: A | i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i] \) - SC
2. \( \forall i, j: A | \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i] \)
3. \( \forall i, j, k: A | (\text{ord}[i, j] \land \text{ord}[j, k]) \Rightarrow \text{ord}[i, k] \)
4. \( \forall i, j: A | (t[i] = t[j] \land co^+[i, j]) \Rightarrow \text{ord}[i, j] \)
5. \( \forall i, j: A | (t[i] \neq t[j] \land to^+[t[i], t[j]]) \Rightarrow \text{ord}[i, j] \)
6. \( \forall k: A \land \text{Read} \land \neg \text{ord}[k, W[k]] \)
7. \( \forall k: A \land \text{Read}, j: A \land \text{Write} | \neg (\exists k | \exists j | \text{ord}[W[k], j] \land \text{ord}[j, k] ) \)

\[ V[a_{01}] = 0 \]
\[ V[a_{02}] = 0 \]
\[ V[W[a_{11}]] = 1 \]
\[ V[W[a_{22}]] = 1 \]

\( \forall i, j: A | i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i] \)
\( \forall i, j, k: A | (\text{ord}[i, j] \lor \text{ord}[j, k]) \Rightarrow \text{ord}[i, k] \)
\( \forall i, j: A | (t[i] = t[j] \lor co^+[i, j]) \Rightarrow \text{ord}[i, j] \)
\( \forall k: A \land \text{Read} | \neg \text{ord}[k, W[k]] \)

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed
Proof of illegality (minimal core)

<table>
<thead>
<tr>
<th>x = 0, y = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1 = x</td>
</tr>
<tr>
<td>y = 1</td>
</tr>
<tr>
<td>r2 = y</td>
</tr>
<tr>
<td>x = 1</td>
</tr>
<tr>
<td>r2 = 1?</td>
</tr>
</tbody>
</table>

a01: write(x, 0)
a02: write(y, 0)
a11: read(x, 0)
a12: write(y, 1)
a21: read(y, 1)
a22: write(x, 1)

V[a01] = 0
V[a02] = 0
V[W[a11]] = 1
V[W[a21]] = 1
∀ i, j: A | i ≠ j ⇒ ord[i, j] ∨ ord[j, i]
∀ i, j, k: A | (ord[i, j] ∧ ord[j, k]) ⇒ ord[i, k]
∀ i, j: A | (t[i] = t[j] ∧ co*[i, j]) ⇒ ord[i, j]
∀ k: A ∩ Read | ¬ ord[k, W[k]]
∀ k: A ∩ Read, j: A ∩ Write |

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed
Proof of illegality (minimal core)

\[
\begin{array}{c|c}
 x = 0, y = 0 & \text{a01: write}(x, 0) \\
 r1 = x & \text{a02: write}(y, 0) \\
 y = 1 & \text{a12: write}(y, 1) \\
 x = 1 & \text{a21: read}(y, 1) \\
 r1 == 1 & \text{a22: write}(x, 1) \\
 r2 == 1 & \text{a11: read}(x, 1) \\
\end{array}
\]

\[
\begin{align*}
V[a01] &= 0 \\
V[a02] &= 0 \\
V[W[a11]] &= 1 \\
V[W[a21]] &= 1 \\
\forall i, j: & A | i \neq j \Rightarrow \text{ord}[i, j] \lor \text{ord}[j, i] \\
\forall i, j: & A | \text{ord}[i, j] \Rightarrow \neg \text{ord}[j, i] \\
\forall i, j, k: & A | (\text{ord}[i, j] \land \text{ord}[j, k]) \Rightarrow \text{ord}[i, k] \\
\forall i, j: & A | (t[i] \neq t[j] \land \text{co}^+[i, j]) \Rightarrow \text{ord}[i, j] \\
\forall k: & A \cap \text{Read} | \neg \text{ord}[k, W[k]] \\
\forall k: & A \cap \text{Read}, j: A \cap \text{Write} | \\
& \neg (l[k] = l[j] \land \text{ord}[W[k], j] \land \text{ord}[j, k])
\end{align*}
\]

minimal core: an unsatisfiable subset of the program and memory model constraints that becomes satisfiable if one of its members is removed
Approach

Program → preprocessor → translator → constraint assembler → bounds assembler → solver

Memory model

finitization parameters
Approach

finitize $P$ and convert it to an intermediate form

Program

preprocessor

translator

constraint assembler

bounds assembler

solver

Memory model

finitization parameters

$I(P)$

finitize $P$ and convert it to an intermediate form

Program

preprocessor

translator

constraint assembler

bounds assembler

solver

Memory model

finitization parameters

$I(P)$
Approach

Program

Memory model

preprocessor

finitize P and convert it to an intermediate form

translator

translate I(P) to a relational representation

constraint assembler

finitization parameters

bounds assembler

solver

I(P)

R(P)
Approach

1. Finitize $P$ and convert it to an intermediate form.
2. Translate $I(P)$ to a relational representation.
3. Combine $R(P)$ and $M$ into the legality formula $F(P, M)$.
4. Solve the bounds assembler.

Program

Preprocessor

Translator

Constraint assembler

Bounds assembler

Solver

Memory model

Finitization parameters
Approach

Finitize \( P \) and convert it to an intermediate form

Translate \( I(P) \) to a relational representation

Combine \( R(P) \) and \( M \) into the legality formula

Compute a set of bounds on the search space
Approach

Program → WALA → translator

Memory model → finitization parameters

WALA → translator

MemSAT contributions

modularity → efficiency

bound assembler → bounds assembler

kodkod + MiniSat
public class Test1 {
    static int x = 0;
    static int y = 0;

    @thread
    public static void thread1() {
        final int r1 = x;
        if (r1 != 0)
            y = r1;
        else
            y = 1;
        assert r1==1;
    }

    @thread
    public static void thread2() {
        final int r2 = y;
        x = 1;
        assert r2==1;
    }
}
public class Test1 {
    static int x = 0;
    static int y = 0;

    @thread
    public static void thread1() {
        final int r1 = x;
        if (r1 != 0)
            y = r1;
        else
            y = 1;
        assert r1 == 1;
    }

    @thread
    public static void thread2() {
        final int r2 = y;
        x = 1;
        assert r2 == 1;
    }
}
public class Test1 {
    static int x = 0;
    static int y = 0;

    @thread
    public static void thread1() {
        final int r1 = x;
        if (r1 != 0)
            y = r1;
        else
            y = 1;
        assert r1==1;
    }

    @thread
    public static void thread2() {
        final int r2 = y;
        x = 1;
        assert r2==1;
    }
}

control flow
00 start
  01 write(x, 0)
  02 write(y, 0)
  03 end

thread order
10 start
  11 r1=read(x)
  12 branch(r1!=0)
  T
  13 write(y, r1)
  F
  14 write(y, 1)
  15 assert(r1==1)
  16 end
20 start
  21 r2=read(y)
  22 write(x, 1)
  23 assert(r2==1)
  24 end
public class Test1 {
    static int x = 0;
    static int y = 0;

    @thread
    public static void thread1() {
        final int r1 = x;
        if (r1 != 0)
            y = r1;
        else
            y = 1;
        assert r1 == 1;
    }

    @thread
    public static void thread2() {
        final int r2 = y;
        x = 1;
        assert r2 == 1;
    }
}

```
public class Test1 {
    static int x = 0;
    static int y = 0;

    @thread
    public static void thread1() {
        final int r1 = x;
        if (r1 != 0)
            y = r1;
        else
            y = 1;
        assert r1==1;
    }

    @thread
    public static void thread2() {
        final int r2 = y;
        x = 1;
        assert r2==1;
    }
}

Preprocessing
Translation

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

translate I(P) to a relational representation

R(P)
Translation

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Translation

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td></td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

maps reads, writes, locks, and unlocks to relations representing locations that are accessed

\[
\begin{array}{|c|c|c|c|}
\hline
s & \text{Loc} & \text{Val} & \text{Guard} \\
\hline
00 & & & \\
01 & x & & \\
02 & y & & \\
03 & & & \\
10 & & & \\
11 & x & & \\
12 & & & \\
13 & y & & \\
14 & y & & \\
15 & & & \\
16 & & & \\
20 & & & \\
21 & & & y \\
22 & & & x \\
23 & & & \\
24 & & & \\
\hline
\end{array}
\]

maps reads, writes, locks, and unlocks to relations representing locations that are accessed
Translation

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

00 start
01 write(x, 0)
02 write(y, 0)
03 end

10 start
11 r1=read(x)
12 branch(r1!=0)
13 write(y, r1)
14 write(y, 1)
15 assert(r1==1)
16 end

20 start
21 r2=read(y)
22 write(x, 1)
23 assert(r2==1)
24 end

maps reads, writes, locks, and unlocks to relations representing locations that are accessed.

<table>
<thead>
<tr>
<th>s</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>x</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>x</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>y</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>x</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>y</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>y</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>y</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>y</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>x</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>x</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

relational constants that represent fields: \( x = \{<x>\} \) and \( y = \{<y>\} \)
Translation

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>16</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>x</td>
<td>Bits(0)</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>y</td>
<td>Bits(0)</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>y</td>
<td>Bits(1)</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>x</td>
<td>Bits(1)</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

maps writes and asserts to relational encodings of the values written or asserted
Translation

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>x</td>
<td>Bits(0)</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>y</td>
<td>Bits(0)</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

maps writes and asserts to relational encodings of the values written or asserted

relational variable that acts as a placeholder for the value read into r1

T
F

10 start
11 r1=read(x)
12 branch(r1!=0)
13 write(y, r1)
14 write(y, 1)
15 assert(r1==1)
16 end

20 start
21 r2=read(y)
22 write(x, 1)
23 assert(r2==1)
24 end

r1

r1=Bits(1)

y

y=Bits(1)

x=Bits(1)

r2=Bits(1)
Translation

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

**Translation Diagram**

<table>
<thead>
<tr>
<th>s</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>x</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>02</td>
<td>y</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>y</td>
<td>Bits(1)</td>
<td>r1=Bits(0)</td>
</tr>
<tr>
<td>14</td>
<td>y</td>
<td>Bits(1)</td>
<td>r1=Bits(0)</td>
</tr>
<tr>
<td>15</td>
<td>r1</td>
<td></td>
<td>r1!=Bits(0)</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>x</td>
<td>Bits(1)</td>
<td>r2=Bits(1)</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Maps statements to formulas that encode their guards.
Translation

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>x</td>
<td>T</td>
</tr>
<tr>
<td>01</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>02</td>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td>03</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>10</td>
<td>x</td>
<td>T</td>
</tr>
<tr>
<td>11</td>
<td>r1==0</td>
<td>T</td>
</tr>
<tr>
<td>12</td>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td>13</td>
<td>r1≠Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>14</td>
<td>Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>15</td>
<td>r1=Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>16</td>
<td>y</td>
<td>T</td>
</tr>
<tr>
<td>20</td>
<td>x</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>22</td>
<td>r2==1</td>
<td>T</td>
</tr>
<tr>
<td>23</td>
<td>Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>24</td>
<td>r2=Bits(1)</td>
<td>T</td>
</tr>
</tbody>
</table>
Constraint assembly

<table>
<thead>
<tr>
<th>s</th>
<th>s guard</th>
<th>guard</th>
<th>00 start</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1! = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>x</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>01</td>
<td>y</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>x</td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>11</td>
<td>y</td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>y</td>
<td>r1</td>
<td>T</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>r1=Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>r1=Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>y</td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>x</td>
<td>Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

construct the legality formula for R(P) and M

\[ F(P, M) \]
F(R(P), E) \land 
F_\alpha(R(P), E) \land 
\land_{1 \leq i \leq k} F(R(P), E_i) \land 
M(E, E_1, \ldots, E_k)
The witness execution $E$ respects the sequential semantics of $P$.

$$F(R(P), E) \land \\
F_{\alpha}(R(P), E) \land \\
\land_{1 \leq i \leq k} F(R(P), E_i) \land \\
M(E, E_1, \ldots, E_k)$$
The witness execution $E$ respects the sequential semantics of $P$

$$F(R(P), E) \land F_\alpha(R(P), E) \land \bigwedge_{1 \leq i \leq k} F(R(P), E_i) \land M(E, E_1, \ldots, E_k)$$

E executes and satisfies the assertions in $P$
Constraint assembly

<table>
<thead>
<tr>
<th>s</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>x</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>01</td>
<td>y</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>02</td>
<td>y</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>x</td>
<td>r1</td>
<td>T</td>
</tr>
<tr>
<td>11</td>
<td>y</td>
<td>Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>12</td>
<td>y</td>
<td>Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>r1</td>
<td>T</td>
</tr>
<tr>
<td>14</td>
<td>y</td>
<td>Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>r1</td>
<td>T</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>y</td>
<td>Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>x</td>
<td>Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The witness execution $E$ respects the sequential semantics of $P$:

$$F(R(P), E) \land F_\alpha(R(P), E) \land \bigwedge_{1 \leq i \leq k} F(R(P), E_i) \land M(E, E_1, \ldots, E_k)$$

Each speculative execution $E_i$ respects the sequential semantics of $P$:

$E$ executes and satisfies the assertions in $P$.

The table shows the locations, values, and guards for each step in the execution.
### Constraint assembly

#### Table

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td>x</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>y</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>02</td>
<td></td>
<td>y</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>11</td>
<td>r1≠0</td>
<td>x</td>
<td>Bits(0)</td>
<td>F</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>y</td>
<td>Bits(1)</td>
<td>F</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>y</td>
<td>Bits(0)</td>
<td>T</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>y</td>
<td>Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>x</td>
<td>Bits(1)</td>
<td>T</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
</tbody>
</table>

#### Note

- The witness execution $E$ respects the sequential semantics of $P$.
- $F(R(P), E) \land F_\alpha(R(P), E) \land \bigwedge_{1 \leq i \leq k} F(R(P), E_i) \land M(E, E_1, \ldots, E_k)$
- Each speculative execution $E_i$ respects the sequential semantics of $P$.
- $E$ and all $E_i$ respect the memory model constraints.
Constraint assembly

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>⊤</td>
<td></td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>01</td>
<td>x</td>
<td>⊤</td>
<td>Bits(0)</td>
<td>⊤</td>
</tr>
<tr>
<td>02</td>
<td>y</td>
<td>⊤</td>
<td>Bits(0)</td>
<td>⊤</td>
</tr>
<tr>
<td>03</td>
<td></td>
<td>⊤</td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>⊤</td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>11</td>
<td>x</td>
<td>⊤</td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>13</td>
<td>y</td>
<td>⊤</td>
<td>Bits(1)</td>
<td>⊤</td>
</tr>
<tr>
<td>14</td>
<td>y</td>
<td>⊤</td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>⊤</td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>21</td>
<td>y</td>
<td>⊤</td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>22</td>
<td>x</td>
<td>⊤</td>
<td>Bits(1)</td>
<td>⊤</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>⊤</td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>⊤</td>
<td>⊤</td>
</tr>
</tbody>
</table>

\[ F(R(P), E) \land F_\alpha(R(P), E) \land \bigwedge_{1 \leq i \leq k} F(R(P), E_i) \land M(E, E_1, \ldots, E_k) \]
Constraint assembly: $F_\alpha(R(P), E)$

<table>
<thead>
<tr>
<th>$s$</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td></td>
<td>$\top$</td>
</tr>
<tr>
<td>01</td>
<td>$x$</td>
<td>Bits(0)</td>
<td>$\top$</td>
</tr>
<tr>
<td>02</td>
<td>$y$</td>
<td>Bits(0)</td>
<td>$\top$</td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
<td>$\top$</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>$\top$</td>
</tr>
<tr>
<td>11</td>
<td>$x$</td>
<td></td>
<td>$\top$</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>$\top$</td>
</tr>
<tr>
<td>13</td>
<td>$y$</td>
<td>$r_1$</td>
<td>$\top$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$r_1 \neq \text{Bits}(0)$</td>
</tr>
<tr>
<td>14</td>
<td>$y$</td>
<td>Bits(1)</td>
<td>$\top$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$r_1$</td>
<td>$\top$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$r_1 = \text{Bits}(1)$</td>
<td>$\top$</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td>$\top$</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td>$\top$</td>
</tr>
<tr>
<td>20</td>
<td>$y$</td>
<td>Bits(1)</td>
<td>$\top$</td>
</tr>
<tr>
<td>21</td>
<td>$x$</td>
<td>Bits(1)</td>
<td>$\top$</td>
</tr>
<tr>
<td>22</td>
<td>$x$</td>
<td>Bits(1)</td>
<td>$\top$</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td>$\top$</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>$\top$</td>
</tr>
</tbody>
</table>

$A$ set of all executed actions
$W$ maps reads to seen writes
$V$ maps writes to written values
$l$ maps writes to written values
$m$ maps locks/unlocks to monitors

$F(R(P), E)$ ∧ $F_\alpha(R(P), E)$ ∧ $\bigwedge_{1 \leq i \leq k} F(R(P), E_i)$ ∧ $M(E, E_1, \ldots, E_k)$
Constraint assembly: \( F_\alpha(R(P), E) \)

<table>
<thead>
<tr>
<th>s</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 start</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01 write(x, 0)</td>
<td>x</td>
<td>Bits(0)</td>
<td>( a_{01} )</td>
</tr>
<tr>
<td>02 write(y, 0)</td>
<td>y</td>
<td>Bits(0)</td>
<td>( a_{02} )</td>
</tr>
<tr>
<td>03 end</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 start</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 r1=read(x)</td>
<td>x</td>
<td></td>
<td>( a_{10} )</td>
</tr>
<tr>
<td>12 branch(r1!=0)</td>
<td>y</td>
<td></td>
<td>( a_{11} )</td>
</tr>
<tr>
<td>13 write(y, r1)</td>
<td></td>
<td>Bits(1)</td>
<td></td>
</tr>
<tr>
<td>14 write(y, 1)</td>
<td></td>
<td>Bits(1)</td>
<td></td>
</tr>
<tr>
<td>15 assert(r1==1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 end</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 start</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21 r2=read(y)</td>
<td>y</td>
<td></td>
<td>( a_{20} )</td>
</tr>
<tr>
<td>22 write(x, 1)</td>
<td>x</td>
<td></td>
<td>( a_{21} )</td>
</tr>
<tr>
<td>23 assert(r2==1)</td>
<td></td>
<td>Bits(1)</td>
<td></td>
</tr>
<tr>
<td>24 end</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A = set of all executed actions
\( W \) maps reads to seen writes
\( V \) maps writes to written values
\( l \) maps writes to written values
\( m \) maps locks/unlocks to monitors

\[
F(R(P), E) \land F_\alpha(R(P), E) \land \bigwedge_{1 \leq i \leq k} F(R(P), E_i) \land M(E, E_1, \ldots, E_k)
\]

relational variable \( a_{ij} \) represents the action performed if E executes the statement ij
Constraint assembly: \( F_\alpha(R(P), E) \)

\[
\begin{align*}
A & \text{ set of all executed actions} \\
W & \text{ maps reads to seen writes} \\
V & \text{ maps writes to written values} \\
l & \text{ maps writes to written values} \\
m & \text{ maps locks/unlocks to monitors}
\end{align*}
\]

\[
F(P, E) \land \bigwedge_{1 \leq i \leq k} F(P, E_i) \land M(E, E_1, \ldots, E_k)
\]
**Constraint assembly:** $F_\alpha(R(P), E)$

<table>
<thead>
<tr>
<th>s</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>start</td>
<td></td>
<td>$a_{00}$</td>
</tr>
<tr>
<td>01</td>
<td>write(x, 0)</td>
<td>x Bits(0)</td>
<td>T $a_{01}$</td>
</tr>
<tr>
<td>02</td>
<td>write(y, 0)</td>
<td>y Bits(0)</td>
<td>T $a_{02}$</td>
</tr>
<tr>
<td>03</td>
<td>end</td>
<td></td>
<td>$a_{03}$</td>
</tr>
<tr>
<td>10</td>
<td>start</td>
<td></td>
<td>$a_{10}$</td>
</tr>
<tr>
<td>11</td>
<td>r1=read(x)</td>
<td>x</td>
<td>T $a_{11}$</td>
</tr>
<tr>
<td>12</td>
<td>branch(r1!=0)</td>
<td>y r1 Bits(1)</td>
<td>T $a_{13}$</td>
</tr>
<tr>
<td>13</td>
<td>write(y, r1)</td>
<td>y</td>
<td>T $a_{14}$</td>
</tr>
<tr>
<td>14</td>
<td>write(y, 1)</td>
<td>r1 Bits(1)</td>
<td>T $a_{16}$</td>
</tr>
<tr>
<td>15</td>
<td>assert(r1==1)</td>
<td>y r1=Bits(1)</td>
<td>T $a_{20}$</td>
</tr>
<tr>
<td>16</td>
<td>end</td>
<td></td>
<td>$a_{21}$</td>
</tr>
<tr>
<td>20</td>
<td>start</td>
<td></td>
<td>$a_{22}$</td>
</tr>
<tr>
<td>21</td>
<td>r2=read(y)</td>
<td>y Bits(1)</td>
<td>T $a_{24}$</td>
</tr>
<tr>
<td>22</td>
<td>write(x, 1)</td>
<td>x r2 Bits(1)</td>
<td>T $a_{24}$</td>
</tr>
<tr>
<td>23</td>
<td>assert(r2==1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>end</td>
<td></td>
<td>$a_{24}$</td>
</tr>
</tbody>
</table>

$F(R(P), E)$ $\land$ $V[W[a_{11}]]=$Bits(1) $\land$ $V[W[a_{21}]]=$Bits(1) $\land$ $\land_{1\leq i \leq k} F(R(P), E_i) \land$ $M(E, E_1, \ldots, E_k)$
Constraint assembly: \( F(R(P), E) \)

\[
F(R(P), E) \land \\
\land_1^k \left( F(R(P), E_i) \land \\
\land \left( \left( a_{13} \neq \text{Bits}(0) \right) \land \\
\left( a_{14} = \text{Bits}(0) \right) \right) \land \\
M(E, E_1, \ldots, E_k) \right)
\]

A set of all executed actions
\( W \) maps reads to seen writes
\( V \) maps writes to written values
\( l \) maps writes to written values
\( m \) maps locks/unlocks to monitors
Constraint assembly: \( F(R(P), E) \)

\[
\bigwedge_{s \in P} F(s, R(P), E) \land \\
A = a_{00} \cup \ldots \cup a_{24} \land \\
V[W[a_{11}]] = \text{Bits}(1) \land \\
V[W[a_{21}]] = \text{Bits}(1) \land \\
\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \land \\
M(E, E_1, \ldots, E_k)
\]
Constraint assembly: $F(R(P), E)$

00 start
01 write(x, 0)
02 write(y, 0)
03 end
10 start
11 r1=read(x)
12 branch(r1!=0)
13 write(y, r1)
14 write(y, 1)
15 assert(r1==1)
16 end
20 start
21 r2=read(y)
22 write(x, 1)
23 assert(r2==1)
24 end

$F(R(P), E) \wedge F(s, R(P), E) \wedge \bigwedge_{1 \leq i \leq k} F(R(P), E_i) \wedge M(E, E_1, \ldots, E_k)$

- $A$ set of all executed actions
- $W$ maps reads to seen writes
- $V$ maps writes to written values
- $l$ maps writes to written values
- $m$ maps locks/unlocks to monitors

0 or 1 action performed
action performed iff the guard is true
no other statement performs the same action
action location is valid
action value is valid

$V[W[a_{11}]]$
Constraint assembly: $F(R(P), E)$

$A$ set of all executed actions
$W$ maps reads to seen writes
$V$ maps writes to written values
$l$ maps writes to written values
$m$ maps locks/unlocks to monitors

$\wedge_{s \in P} F(s, R(P), E) \land$

$A = a_{00} \cup \ldots \cup a_{24} \land$

$V[W[a_{11}]] = \text{Bits}(1) \land$

$V[W[a_{21}]] = \text{Bits}(1) \land$

$\wedge_{1 \leq i \leq k} F(R(P), E_i) \land$

$M(E, E_1, \ldots, E_k)$

- 0 or 1 action performed
- action performed iff the guard is true
- no other statement performs the same action
- action location is valid
- action value is valid
Constraint assembly: $F(R(P), E)$

$\bigwedge_{s \in P} F(s, R(P), E) \land$

$A \quad \text{set of all executed actions}$

$W \quad \text{maps reads to seen writes}$

$V \quad \text{maps writes to written values}$

$l \quad \text{maps writes to written values}$

$m \quad \text{maps locks/unlocks to monitors}$

$|a_{13}| \leq 1 \land$

• action performed iff the guard is true

• no other statement performs the same action

• action location is valid

• action value is valid

$V[W[a_{11}]] = \text{Bits}(1) \land$

$V[W[a_{21}]] = \text{Bits}(1) \land$

$\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \land$

$M(E, E_1, \ldots, E_k)$
Constraint assembly: \( F(R(P), E) \)

\[
\begin{align*}
&\text{Constraint assembly: } F(R(P), E) \\
&\wedge \quad s \in P \quad F(s, R(P), E) & A \text{ set of all executed actions} \\
&\wedge V[W[a_{11}]] = \text{Bits}(1) & W \text{ maps reads to seen writes} \\
&\wedge V[W[a_{21}]] = \text{Bits}(1) & V \text{ maps writes to written values} \\
&\wedge 1 \leq i \leq k \quad F(R(P), E_i) & l \text{ maps writes to written values} \\
&\wedge M(E, E_1, \ldots, E_k) & m \text{ maps locks/unlocks to monitors} \\
\end{align*}
\]
Constraint assembly: $F(R(P), E)$

<table>
<thead>
<tr>
<th>$s$</th>
<th>Loc</th>
<th>Val</th>
<th>Guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>write(x, 0)</td>
<td>x</td>
<td>Bits(0)</td>
</tr>
<tr>
<td>02</td>
<td>write(y, 0)</td>
<td>y</td>
<td>Bits(0)</td>
</tr>
<tr>
<td>03</td>
<td>end</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>r1=read(x)</td>
<td>x</td>
<td>Bits(0)</td>
</tr>
<tr>
<td>12</td>
<td>branch(r1!=0)</td>
<td>y</td>
<td>Bits(0)</td>
</tr>
<tr>
<td>13</td>
<td>write(y, r1)</td>
<td>y</td>
<td>r1=Bits(0)</td>
</tr>
<tr>
<td>14</td>
<td>write(y, 1)</td>
<td>y</td>
<td>Bits(1)</td>
</tr>
<tr>
<td>15</td>
<td>assert(r1==1)</td>
<td>y</td>
<td>Bits(1)</td>
</tr>
<tr>
<td>16</td>
<td>end</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>r2=read(y)</td>
<td>y</td>
<td>Bits(1)</td>
</tr>
<tr>
<td>22</td>
<td>write(x, 1)</td>
<td>y</td>
<td>Bits(1)</td>
</tr>
<tr>
<td>23</td>
<td>assert(r2==1)</td>
<td>x</td>
<td>Bits(1)</td>
</tr>
<tr>
<td>24</td>
<td>end</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$A$ set of all executed actions
$W$ maps reads to seen writes
$V$ maps writes to written values
$l$ maps writes to written values
$m$ maps locks/unlocks to monitors

$\bigwedge_{s \in P} F(s, R(P), E) \land
A = a_{00} \cup \ldots \cup a_{24} \land
V[W[a_{11}]] = \text{Bits}(1) \land
V[W[a_{21}]] = \text{Bits}(1) \land
\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \land
M(E, E_1, \ldots, E_k)$
Constraint assembly: $F(R(P), E)$

- $A$ set of all executed actions
- $W$ maps reads to seen writes
- $V$ maps writes to written values
- $l$ maps writes to written values
- $m$ maps locks/unlocks to monitors

$\bigwedge_{s \in P} F(s, R(P), E) \land$

$A = a_{00} \cup \ldots \cup a_{24} \land$

$V[W[a_{11}]] = \text{Bits}(1) \land$

$V[W[a_{21}]] = \text{Bits}(1) \land$

$\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \land$

$M(E, E_1, \ldots, E_k)$
Constraint assembly: $F(R(P), E)$

- $A$ set of all executed actions
- $W$ maps reads to seen writes
- $V$ maps writes to written values
- $l$ maps writes to written values
- $m$ maps locks/unlocks to monitors

$$\bigwedge_{s \in P} F(s, R(P), E) \land$$

$$A = a_{00} \cup \ldots \cup a_{24} \land$$

$$V[W[a_{11}]] = \text{Bits}(1) \land$$

$$V[W[a_{21}]] = \text{Bits}(1) \land$$

$$\bigwedge_{1 \leq i \leq k} F(R(P), E_i) \land$$

$$M(E, E_1, \ldots, E_k)$$
Bounds assembly

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

compute a set of bounds on the search space

B(P, M)

F(P, M)
Bounds assembly

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r₁!=0</td>
</tr>
<tr>
<td>14</td>
<td>r₁==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

F(P, M)

10 start

11 r₁=read(x)
12 branch(r₁!=0)

T

13 write(y, r₁)
14 write(y, 1)

F

20 start

21 r₂=read(y)

21 r₂=read(y)
22 write(x, 1)
23 assert(r₂==1)
24 end

16 end

-8, 1, 2, 4, x, y, a₀₀, a₀₁, a₀₂, a₀₃, a₁₀, a₁₁, a₁₃, a₁₆, a₂₀, a₂₁, a₂₂, a₂₄

{...} ⊆ A ⊆ {...}
{...} ⊆ V ⊆ {...}
{...} ⊆ W ⊆ {...}
{...} ⊆ l ⊆ {...}
{...} ⊆ m ⊆ {...}
Bounds assembly: universe

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

finite universe of symbolic values from which the model, if any, is drawn

-8, 1, 2, 4, x, y, a00, a01, a02, a03, a10, a11, a13, a16, a20, a21, a22, a24

{...} ⊆ A ⊆ {...}
{...} ⊆ V ⊆ {...}
{...} ⊆ W ⊆ {...}
{...} ⊆ I ⊆ {...}
{...} ⊆ m ⊆ {...}
Bounds assembly: universe

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

F(P, M)

Finite universe of symbolic values from which the model, if any, is drawn

-8, 1, 2, 4, x, y, a00, a01, a02, a03, a10, a11, a13, a16, a20, a21, a22, a24

(...} ⊆ A ⊆ {...}

(...} ⊆ V ⊆ {...}

(...} ⊆ W ⊆ {...}

(...} ⊆ l ⊆ {...}

(...} ⊆ m ⊆ {...}
Bounds assembly: universe

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

finite universe of symbolic values from which the model, if any, is drawn
Bounds assembly: lower/upper bounds

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>21</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

- $8, 1, 2, 4, x, y, a00, a01, a02, a03, a10, a11, a13, a16, a20, a21, a22, a24$

...
Bounds assembly: lower/upper bounds

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

F(P, M)

- \[ s \text{ guard} \]
  - 00 start
  - 01 write(x, 0)
  - 02 write(y, 0)
  - 03 end

- \[ s \text{ maySee} \]
  - 11 {01, 22}
  - 21 {02, 13, 14}
  - \[ \{\ldots\} \subseteq A \subseteq \{\ldots\} \]
  - \[ \{\ldots\} \subseteq V \subseteq \{\ldots\} \]
  - \[ \{\ldots\} \subseteq W \subseteq \{\ldots\} \]
  - \[ \{\ldots\} \subseteq l \subseteq \{\ldots\} \]
  - \[ \{\ldots\} \subseteq m \subseteq \{\ldots\} \]

-8, 1, 2, 4, x, y, a00, a01, a02, a03, a10, a11, a13, a16, a20, a21, a22, a24
Bounds assembly: lower/upper bounds

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

F(P, M)

-8, 1, 2, 4, x, y, a00, a01, a02, a03, a10, a11, a13, a16, a20, a21, a22, a24

\{...\} \subseteq A \subseteq \{<a00>, <a01>, <a02>, <a03>, <a10>, <a11>, <a13>, <a16>, <a20>, <a21>, <a22>, <a24>\}

\{...\} \subseteq V \subseteq \{...\}

\{...\} \subseteq W \subseteq \{...\}

\{...\} \subseteq I \subseteq \{...\}

\{...\} \subseteq m \subseteq \{...\}

Bounds assembly: upper and lower bound on the value of each relation that appears in F(P, M)
Bounds assembly: lower/upper bounds

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td></td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

F(P, M)

-8, 1, 2, 4, x, y, a00, a01, a02, a03, a10, a11, a13, a16, a20, a21, a22, a24

upper and lower bound on the value of each relation that appears in F(P, M)
Bounds assembly: lower/upper bounds

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
<td>{02, 13, 14}</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>

F(P, M)

-8, 1, 2, 4, x, y, a00, a01, a02, a03, a10, a11, a13, a16, a20, a21, a22, a24

Upper and lower bound on the value of each relation that appears in F(P, M)
Bounds assembly: lower/upper bounds

<table>
<thead>
<tr>
<th>s</th>
<th>guard</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>r1!=0</td>
</tr>
<tr>
<td>14</td>
<td>r1==0</td>
</tr>
<tr>
<td>*</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>maySee</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>{01, 22}</td>
</tr>
<tr>
<td>21</td>
<td>{02, 13, 14}</td>
</tr>
</tbody>
</table>

\[
F(P, M) = \begin{cases} 
-8, 1, 2, 4, x, y, a00, a01, a02, a03, a10, a11, a13, a16, a20, a21, a22, a24 \\
\{<a00>, <a01>, <a02>, <a03>, <a10>, <a11>, <a16>, <a20>, <a21>, <a22>, <a24>\} \subseteq A \subseteq \{<a00>, <a01>, <a02>, <a03>, <a10>, <a11>, <a16>, <a20>, <a21>, <a22>, <a24>\} \\
\{\ldots\} \subseteq V \subseteq \{\ldots\} \\
\{\ldots\} \subseteq W \subseteq \{\ldots\} \\
\{\ldots\} \subseteq m \subseteq \{\ldots\}
\end{cases}
\]

upper and lower bound on the value of each relation that appears in \(F(P, M)\)
Results (highlights)

MemSAT performance on JMM causality tests

- **Original JMM**
  - validates 17 & 18 (Sevcik’08 ✗)
  - violates 19 & 20 (Aspinall’07 ✔)

- **Revised JMM**
  - overconstrained as given; fixed it
  - now validates all tests (Aspinall’07 ✔)
Conclusion

Practical checker for axiomatic specifications of memory models
- first tool to directly handle the current JMM
- first tool to provide minimal cores

Prior work (highlights)
- CheckFence hardcodes the memory model
- Nemos accepts simple axiomatic specs but no cores
- JMM checkers (e.g. OpMM) use operational approximations

Future work
- extend MemSAT to handle hardware memory models