Model Checking for Weakly Consistent Libraries

Michalis Kokologiannakis  Azalea Raad  Viktor Vafeiadis
June 24, 2019

Max Planck Institute for Software Systems (MPI-SWS)
How Do We Verify Concurrent Programs?

Stateless Model Checking (SMC): enumerates all executions without explicitly storing the visited states.

Challenges:
• State space explosion
• Weak memory
How Do We Verify Concurrent Programs?

Stateless Model Checking (SMC): enumerates all executions

- **without** explicitly *storing* the visited states

Challenges:

- State space explosion
- Weak memory
Challenge #1: State Space Explosion

\[\begin{align*}
\text{x} &= \text{y} = 0 \\
\text{x} &: = 1 \\
\text{y} &: = 42
\end{align*}\]
Challenge #1: State Space Explosion

\[ x = y = 0 \]

\[
\begin{array}{c}
  x := 1 \parallel x := 2 \parallel y := 42 \\
\end{array}
\]

Executions:
Challenge #1: State Space Explosion

\[
\begin{align*}
x &= y = 0 \\
x &:= 1 \quad \| \quad x := 2 \quad \| \quad y := 42
\end{align*}
\]

Executions:

\begin{itemize}
  \item SMC : 6
\end{itemize}
Challenge #1: State Space Explosion

\[
\begin{align*}
[x = y &= 0] \\
x &:= 1 \quad \| \quad x := 2 \quad \| \quad y := 42
\end{align*}
\]

Executions:

\begin{align*}
\text{SMC} & : 6 \\
\text{SMC+POR}^{\text{mo}} & : 2
\end{align*}
Challenge #1: State Space Explosion

\[ x = y = 0 \]

\[
\begin{align*}
  x &:= 1 \quad \parallel \quad x := 2 \quad \parallel \quad y := 42
\end{align*}
\]

Executions:

- SMC : 6
- SMC+POR\textsuperscript{mo} : 2
- SMC+POR\textsuperscript{porf} : 1
Challenge #2: Weak Memory Models

All current techniques are memory-model specific ⇒ with the exception of herd. What memory model properties are sufficient for efficient SMC?
Challenge #2: Weak Memory Models

All current techniques are memory-model specific
⇒ with the exception of herd
Challenge #2: Weak Memory Models

All current techniques are memory-model specific
⇒ with the exception of herd

What memory model properties are sufficient for efficient SMC?
Our contribution

• We present sufficient properties for efficient SMC

• GenMC: an SMC procedure

• parametric in the choice of the memory model

• sound, complete, optimal, and efficient
Our contribution

- We present sufficient properties for efficient SMC
Our contribution

• We present sufficient properties for efficient SMC
• **GenMC**: an SMC procedure
  • *parametric* in the choice of the memory model
Our contribution

• We present sufficient properties for efficient SMC
• **GenMC**: an SMC procedure
  • *parametric* in the choice of the memory model
  • sound, complete, optimal, and efficient
Our contribution

- We present sufficient properties for efficient SMC
- **GenMC**: an SMC procedure
  - parametric in the choice of the memory model (+ libraries!)
  - sound, complete, optimal, and efficient
Generic Model Checking
Goal: Enumerate all consistent execution graphs of $P$ for any memory model.
Goal: Enumerate all consistent execution graphs of $P$ for any memory model.

$x := 1 \parallel a := x \parallel x := 2$
**Goal:**Enumerate all consistent execution graphs of $P$ for any memory model.

$[x = 0]$

$x := 1 \parallel a := x \parallel x := 2$

$\begin{align*}
1 & \xrightarrow{\text{po}} \text{init} & 2 & \xrightarrow{\text{init}} & 3 & \xrightarrow{\text{init}} \\
W(x, 1) & \xrightarrow{\text{rf}} R(x) & W(x, 1) & \xrightarrow{\text{rf}} R(x) & W(x, 1) & \xrightarrow{\text{rf}} W(x, 2) \\
W(x, 2) & \xrightarrow{\text{rf}} R(x) & W(x, 2) & \xrightarrow{\text{rf}} W(x, 2) & R(x) & \xrightarrow{\text{rf}} W(x, 2)
\end{align*}$
Systematically Enumerate All Graphs

\[
\begin{align*}
x &= y = 0 \\
a &= y \\
x &= a \\
b &= xy \\
y &= b
\end{align*}
\]

\[R(y) \quad W(x, 42) \quad R(x) \quad W(y, 42)\]

Can the reads of this program read 42? The number of executions may be infinite!
Systematically Enumerate All Graphs

\[ x = y = 0 \]

\[
\begin{align*}
  a &:= y & b &:= x \\
  x &:= a & y &:= b
\end{align*}
\]

Can the reads of this program read 42?
The number of executions may be infinite!
Systematically Enumerate All Graphs

\[ x = y = 0 \]

\[
\begin{align*}
  a & := y \\
  b & := x \\
  x & := a \\
  y & := b
\end{align*}
\]

Can the reads of this program read 42?
Systematically Enumerate All Graphs

\[ x = y = 0 \]

\[
\begin{align*}
  a &:= y \\
  x &:= a
\end{align*}
\]

\[
\begin{align*}
  b &:= x \\
  y &:= b
\end{align*}
\]

R(y) \quad R(x)

po \quad rf

W(x, 42) \quad W(y, 42)

Can the reads of this program read 42?
Systematically Enumerate All Graphs

\[ x = y = 0 \]

\[
\begin{align*}
a & := y \\
x & := a \\
b & := x \\
y & := b
\end{align*}
\]

Can the reads of this program read 42?

The number of executions may be infinite!
Goal: Enumerate all consistent execution graphs of $P$ for any memory model
Goal: Enumerate all consistent execution graphs of \( P \) for any memory model, where

- \( po \cup rf \) is acyclic
Checking Consistency At Each Step
Checking Consistency At Each Step

\[
\begin{align*}
[x = 0] \\
x := 1 \parallel a := x \parallel x := 2
\end{align*}
\]
Checking Consistency At Each Step

\[ x = 0 \]

\[
\begin{array}{c}
\text{x := 1} \\
\text{a := x} \\
\text{x := 2}
\end{array}
\]
Checking Consistency At Each Step

\[ x := 0 \]

\[ x := 1 \parallel a := x \parallel x := 2 \]
Checking Consistency At Each Step

\[
\begin{array}{c|c|c}
[x = 0] & & \\
\hline
x := 1 & a := x & x := 2 \\
\end{array}
\]

\[
\begin{array}{c}
& 1 \quad \text{[init]} \quad \text{po} \quad W(x, 1) \quad R(x) \quad W(x, 2) \\
& 2 \quad \text{[init]} \quad W(x, 1) \quad \text{rf} \quad R(x) \quad W(x, 2) \\
& 3 \quad \text{[init]} \quad W(x, 1) \quad \text{rf} \quad R(x) \quad W(x, 2) \\
& 2 \quad \text{[init]} \quad W(x, 1) \quad \text{rf} \quad R(x) \\
\end{array}
\]
Checking Consistency At Each Step

\[
\begin{array}{c|c|c}
[x = 0] & 1 & 2 \\
\hline
x := 1 & a := x & x := 2 \\
\end{array}
\]

1. \[ \text{po} \downarrow \text{init} \]
   \[ W(x, 1) \rightarrow R(x) \rightarrow W(x, 2) \]

2. \[ \text{init} \]
   \[ W(x, 1) \rightarrow R(x) \rightarrow W(x, 2) \]

3. \[ \text{init} \]
   \[ W(x, 1) \rightarrow R(x) \rightarrow W(x, 2) \]

4. \[ \text{init} \]
   \[ W(x, 1) \rightarrow R(x) \rightarrow W(x, 2) \]
Goal: Enumerate all consistent execution graphs of $P$ for any memory model, where

- $po \cup rf$ is acyclic
**GenMC: Generic Model Checking**

**Goal:** Enumerate all consistent execution graphs of $P$ for *any* memory model, where

- $po \cup rf$ is acyclic
- Each execution can be obtained from some linear extension of $po \cup rf$
Goal: Enumerate all consistent execution graphs of $P$ for any memory model, where

- $po \cup rf$ is acyclic
- Each execution can be obtained from some every linear extension of $po \cup rf$
Goal: Enumerate all consistent execution graphs of $P$ for any memory model, where

- $po \cup rf$ is acyclic
- Consistency is prefix-closed
Fixing The Construction Order

\[ x = 0 \]

\[
\begin{array}{l}
x := 1 \parallel a := x \parallel x := 2
\end{array}
\]
Fixing The Construction Order

<table>
<thead>
<tr>
<th>[x = 0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>x := 1</td>
</tr>
</tbody>
</table>

[init]
Fixing The Construction Order

\[ x = 0 \]

\[
x := 1 \parallel a := x \parallel x := 2
\]

\[
W(x, 1)
\]
Fixing The Construction Order

[init]

\[ x := 0 \]

\[ x := 1 \parallel a := x \parallel x := 2 \]

\[ W(x, 1) \xrightarrow{rf} R(x) \]
Fixing The Construction Order

\[ x = 0 \]
\[ x := 1 \quad a := x \quad x := 2 \]
Fixing The Construction Order

\[ x = 0 \]

\[ x := 1 \parallel a := x \parallel x := 2 \]

\[ \text{[init]} \]

\[ W(x, 1) \rightarrow R(x) \rightarrow W(x, 2) \]

\[ \text{[init]} \]

\[ W(x, 1) \rightarrow R(x) \rightarrow W(x, 2) \]
Fixing The Construction Order

\[
\begin{align*}
[x = 0] \\
x := 1 & \quad a := x & \quad x := 2
\end{align*}
\]

\[
\begin{tikzpicture}
\node (W1) at (0,0) {$W(x, 1)$};
\node (R) at (1,0) {$R(x)$};
\node (W2) at (2,0) {$W(x, 2)$};
\node (init1) at (-1,1) {[init]};
\node (init2) at (1,1) {[init]};
\draw[->] (init1) -- (W1);
\draw[->] (init2) -- (R);
\draw[->] (R) -- (W2);
\end{tikzpicture}
\]
Fixing The Construction Order

\[
\begin{array}{c}
[x = 0] \\
x := 1 \parallel a := x \parallel x := 2
\end{array}
\]
Fixing The Construction Order

<table>
<thead>
<tr>
<th>x = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x := 1 ∧ a := x ∧ x := 2</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
[x = 0] & \quad \text{[init]} \\
x := 1 & \quad a := x & \quad x := 2 \\
\end{align*}
\]
Fixing The Construction Order

\[ x = 0 \]
\[ x := 1 \parallel a := x \parallel x := 2 \]
Fixing The Construction Order

\[ x = 0 \]
\[
\begin{align*}
x &:= 1 \parallel a := x \parallel x := 2
\end{align*}
\]

\[ W(x, 1) \quad R(x) \quad W(x, 2) \quad W(x, 1) \quad R(x) \quad W(x, 2) \quad W(x, 1) \quad R(x) \quad W(x, 2) \]
**Goal:** Enumerate all consistent execution graphs of $P$ for *any* memory model, where

- $po \cup rf$ is acyclic
- Consistency is *prefix-closed*
**Goal:** Enumerate all consistent execution graphs of $P$ for *any* memory model, where

- $po \cup rf$ is acyclic
- Consistency is prefix-closed
- The memory model is extensible
Goal: Enumerate all consistent execution graphs of $P$ for any memory model, where

- $p_0 \cup rf$ is acyclic
- Consistency is prefix-closed
- The memory model is extensible

These are fulfilled by SC, TSO, PSO, RC11
**Goal:** Enumerate all consistent execution graphs of $P$ for *any* memory model, where

- $po \cup rf$ is acyclic
- Consistency is prefix-closed
- The memory model is extensible

These are fulfilled by SC, TSO, PSO, RC11 but *not* by POWER and ARM
# Handling Locks

| $\text{init}(l)$ |  
|------------------|---
| $\text{lock}(l)$ | $\text{lock}(l)$  
| $\text{unlock}(l)$ | $\text{unlock}(l)$  

### Handling Locks

<table>
<thead>
<tr>
<th>init(l)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock(l)</td>
</tr>
<tr>
<td>unlock(l)</td>
</tr>
</tbody>
</table>

1. ![Diagram 1](#)

2. ![Diagram 2](#)
Results
### An Interesting Example

<table>
<thead>
<tr>
<th></th>
<th>Nidhugg</th>
<th>RCMC</th>
<th>GENMC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(SC)</td>
<td>(SC^0)</td>
<td>(RC11^{mo})</td>
</tr>
<tr>
<td><strong>lamport(2)</strong></td>
<td>.13</td>
<td>.10</td>
<td>.04</td>
</tr>
<tr>
<td><strong>lamport(3)</strong></td>
<td>7.53</td>
<td>4.49</td>
<td>5.40</td>
</tr>
<tr>
<td><strong>lamport(4)</strong></td>
<td>⊟</td>
<td>⊟</td>
<td>⊟</td>
</tr>
</tbody>
</table>


- ⊟ = tool did not finish within 2 days

  All times are in seconds.
# An Interesting Example

<table>
<thead>
<tr>
<th></th>
<th>NIDHRUGG</th>
<th>RCMC</th>
<th>GENMC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SC</td>
<td>SC^o</td>
<td>RC11^mo</td>
</tr>
<tr>
<td>lamport(2)</td>
<td>.13</td>
<td>.10</td>
<td>.04</td>
</tr>
<tr>
<td>lamport(3)</td>
<td>7.53</td>
<td>4.49</td>
<td>5.40</td>
</tr>
<tr>
<td>lamport(4)</td>
<td>❕</td>
<td>❕</td>
<td>❕</td>
</tr>
</tbody>
</table>

❖ = tool did not finish within 2 days

All times are in seconds
Equivalence Partitionings

CoA

CasW

inc

NwIr
Equivalence Partitionings (202 benchmarks)
More in the paper

- Detailed description of the algorithm
- Formalization of memory model assumptions
- More benchmarks and evaluation
Conclusions

Summary

- **Sound, complete, and optimal** SMC procedure for memory models that are:
  - $\text{po} \cup \text{rf}$-acyclic
  - prefix-closed
  - extensible
- **GENMC** can be **exponentially faster** than existing tools
- **GENMC** is **available** at [github.com/MPI-SWS/genmc](http://github.com/MPI-SWS/genmc)
Summary

- **Sound, complete, and optimal** SMC procedure for memory models that are:
  - po ∪ rf-acyclic
  - prefix-closed
  - extensible
- **GenMC** can be **exponentially faster** than existing tools
- **GenMC** is **available** at github.com/MPI-SWS/genmc

Future work

- Can we relax the memory-model assumptions?

Thank You!
Why Extensibility Is Necessary

\[ x = y = 0 \]

\[ a := x \parallel b := y \parallel x := 42 \]
Why Extensibility Is Necessary

\[
\begin{align*}
[x = y = 0] \\
& a := x \parallel b := y \parallel x := 42
\end{align*}
\]

Under a memory model that dictates the following:

“If a read of \( y \) reads 0, then there cannot be a read of \( x \) that also reads 0”
Why Extensibility Is Necessary

\[
\begin{align*}
[x = y = 0] \\
a := x & \parallel b := y & x := 42
\end{align*}
\]

Under a memory model that dictates the following:

“If a read of y reads 0, then there cannot be a read of x that also reads 0”
### Why Extensibility Is Necessary

<table>
<thead>
<tr>
<th>[x = y = 0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a := x</td>
</tr>
</tbody>
</table>

Under a memory model that dictates the following:

“If a read of y reads 0, then there cannot be a read of x that also reads 0”
Why Extensibility Is Necessary

\[
\begin{align*}
[x = y = 0] \\
a := x; b := y; x := 42
\end{align*}
\]

Under a memory model that dictates the following:

“If a read of \( y \) reads 0, then there cannot be a read of \( x \) that also reads 0”
Why Extensibility Is Necessary

\[ x = y = 0 \]

\[ a := x \parallel b := y \parallel x := 42 \]

Under a memory model that dictates the following:

“If a read of \( y \) reads 0, then there cannot be a read of \( x \) that also reads 0”
Why Extensibility Is Necessary

\[ x = y = 0 \]

\[ a := x \parallel b := y \parallel x := 42 \]

Under a memory model that dictates the following:

“If a read of y reads 0, then there cannot be a read of x that also reads 0”
Handling Locks

<table>
<thead>
<tr>
<th>[init(l)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a : lock(l)</td>
</tr>
<tr>
<td>a' : unlock(l)</td>
</tr>
</tbody>
</table>

\[
\begin{array}{c}
\text{init(l)} \\
\text{rf} \swarrow \\
a : lock(l)
\end{array} \quad \sim \quad \\
\begin{array}{c}
\\text{[init]} \\
\text{rf} \searrow \\
a : lock(l) \downarrow \\
a' : unlock(l)
\end{array}
\]
Handling Locks

\[
\begin{align*}
\text{[init(l)]} \\
\text{a : lock(l) } &\quad \text{b : lock(l)} \\
a' : unlock(l) &\quad b' : unlock(l)
\end{align*}
\]
Handling Locks

<table>
<thead>
<tr>
<th>[init(l)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a : lock(l)</td>
</tr>
<tr>
<td>a' : unlock(l)</td>
</tr>
</tbody>
</table>

\[\text{init(l)} \rightarrow \text{rf} \downarrow \text{a : lock(l)} \downarrow \text{a' : unlock(l)} \]

\[W[a] \downarrow\]
Handling Locks

<table>
<thead>
<tr>
<th>init(l)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a : lock(l)</td>
</tr>
<tr>
<td>a' : unlock(l)</td>
</tr>
</tbody>
</table>

\[ \text{init}(l) \]
\[ \text{rf} \quad \checkmark \]
\[ a : \text{lock}(l) \quad \downarrow \]
\[ a' : \text{unlock}(l) \]

\[ W[a] \]
\[ \frac{\perp}{b'} \quad \checkmark \]
Handling Locks

\[
\begin{align*}
\text{init}(l) & \quad \text{init}(l) \\
\text{rf} & \quad \text{rf} \\
\bowtie
\end{align*}
\]

\[
\begin{align*}
a : \text{lock}(l) \\
a' : \text{unlock}(l)
\end{align*}
\]

\[
\begin{align*}
b : \text{lock}(l) \\
b' : \text{unlock}(l)
\end{align*}
\]
<table>
<thead>
<tr>
<th></th>
<th>Nidhugg SC</th>
<th>Nidhugg TSO</th>
<th>Nidhugg PSO</th>
<th>RCMC RC11</th>
<th>RCMC W/RC11</th>
<th>GENMC MO</th>
<th>GENMC WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcs_spinlock(2)</td>
<td>.12</td>
<td>.09</td>
<td>.10</td>
<td>.05</td>
<td>.05</td>
<td>.05</td>
<td>.05</td>
</tr>
<tr>
<td>mcs_spinlock(3)</td>
<td>2.98</td>
<td>6.84</td>
<td>12.54</td>
<td>.84</td>
<td>.67</td>
<td>.89</td>
<td>.78</td>
</tr>
<tr>
<td>mcs_spinlock(4)</td>
<td>.68h</td>
<td>1.51h</td>
<td>3.32h</td>
<td>0.16h</td>
<td>0.15h</td>
<td>0.42h</td>
<td>0.26h</td>
</tr>
<tr>
<td>qspinlock(2)</td>
<td>.17</td>
<td>.11</td>
<td>.11</td>
<td>.04</td>
<td>.04</td>
<td>.04</td>
<td>.04</td>
</tr>
<tr>
<td>qspinlock(3)</td>
<td>10.93</td>
<td>18.20</td>
<td>23.43</td>
<td>2.13</td>
<td>2.08</td>
<td>1.10</td>
<td>1.12</td>
</tr>
<tr>
<td>seqlock(2)</td>
<td>.10</td>
<td>.09</td>
<td>.10</td>
<td>.04</td>
<td>.04</td>
<td>.04</td>
<td>.04</td>
</tr>
<tr>
<td>seqlock(3)</td>
<td>1.64</td>
<td>3.07</td>
<td>11.00</td>
<td>.49</td>
<td>.51</td>
<td>.37</td>
<td>.37</td>
</tr>
</tbody>
</table>