On Checking Correctness of Concurrent Data Structures

Ahmed Bouajjani
LIAFA, Univ Paris Diderot - Paris 7

Joint work with

Michael Emmi
IMDEA

Constantin Enea
LIAFA, U Paris Diderot - P7

Jad Hamza

Workshop on Software Correctness and Reliability — ETH Zürich, 2014
Concurrent Data Structures

Methods Implementation

Low Level Representation

Push
Pop
Empty

T1
Push(0)
Pop(1)

... 

Tn
Push(1)
Pop(0)
Empty(true)
Different atomicity levels

Client view:
- Operations are atomic
- Thread executions are interleaved

Implementation:
- Naive solution: Coarse-grain locking
- Performances: Avoid coarse-grain locking
- => Execution intervals may overlap
Observational Refinement

For every Client, Client $\times$ Impl included in Client $\times$ Spec
Linearizability

[Herlihy, Wing, 1990]

Valid sequence in the sequential specification

- reorder call/return events, while preserving returns $\rightarrow$ calls
- find “linearization points” within execution time intervals
- $\Rightarrow$ match a sequential execution

Linearizability implies Observational Refinement

[Filipovic et al. 2010]
Complexity

Finite number of threads, regular specification

• Checking linearizability is in \textit{EXPSPACE} [Alur et al. 1996]

• Lower bound: \textit{EXSPACE}-hardness [Jad Hamza ’14]

• Checking linearizability of a single execution is \textit{NP-complete} [Gibbons, Korach, 1997]
Unbounded Number of Threads?

[B, Emmi, Enea, Hamza, ESOP’13]

Finite-state threads, regular specifications

- Linearizability is **undecidable**

  Reachability in 2-counter machines reducible to non-linearizability

- **Static Linearizability**:  
  - Fixed linearizations points, except for read-only methods  
  - Relevant for a wide class of implementations

- **Static Linearizability is decidable**  
  - Reduction to state reachability  
  - Reachability in FSM/VASS for fixed/unbounded number of threads  
  - P/EXP-SPACE-complete for a fixed/unbounded number of threads
Efficient detection of OR violations?
[B, Emmi, Enea, Hamza, POPL’15]

- Parameterized under-approximation schema?
- Tractable refinement checking?
- Good coverage?
Efficient detection of OR violations?

[B, Emmi, Enea, Hamza, POPL’15]

• Parametrized under-approximation schema?
• Tractable refinement checking?
• Good coverage?

• Characterizing OR as a History Inclusion Problem
• Histories are a special partial orders: Interval orders
• => Interval-length bounding

• Efficient implementation using counting representations
• => Symbolic representation of sets of histories
• => Checking “correctness” of single history is Polynomial
• => Reduction to a reachability problem
• => Scalable dynamic and static analysis techniques

• Small bounds are needed
Programs, Libraries, and Observational Refinement

- Fix a set of methods, and set of local actions $A$
- Operation = pair of a call + return actions of a method
- Execution = sequence of actions + calls & returns
- Program = set of executions over $A$ + calls & returns
- Library = set of executions over calls & returns
- Observational Refinement:

$$L_1 \text{ refines } L_2 \iff (P \times L_1)|_A \text{ is a subset of } (P \times L_2)|_A, \text{ for every } P$$
Histories

History of an execution $e$:

$$H(e) = (\text{OP}(e), <)$$ is a partial order s.t.

$$O_1 < O_2 \quad \text{iff} \quad \text{Return}(O_1) \text{ is before } \text{Call}(O_2) \text{ in } e$$
Histories as Interval Orders

- **Interval Orders** = partial order \((O, <)\) such that 
  \((o_1 < o_1' \text{ and } o_2 < o_2')\) implies \((o_1 < o_2' \text{ or } o_2 < o_1')\)

- For every execution \(e\), \(H(e)\) is an interval order
History Inclusion vs OR vs Linearizability

History Inclusion vs OR

\[ L_1 \text{ refines } L_2 \text{ iff } H(L_1) \text{ is a subset of } H(L_2) \]

i.e.,

OR is equivalent to History Inclusion
History Inclusion vs OR vs Linearizability

History Inclusion vs OR

$L_1$ refines $L_2$ iff $H(L_1)$ is a subset of $H(L_2)$

i.e.,

OR is equivalent to History Inclusion

OR vs Linearizability

$L_1$ is linearizable w.r.t $L_2$ iff

$H(L_1)$ is a subset of $H(L_2)$,

when $L_2$ is atomic
Abstracting Histories

Weakening relation

\[ h_1 \leq h_2 \text{ (} h_1 \text{ is weaker than } h_2 \text{)} \]

iff

\[ h_1 \text{ has less constraints than } h_2 \]

Key lemma:

If \( h_1 \leq h_2 \) and \( h_2 \) is in \( H(L) \), then \( h_1 \) is in \( H(L) \) too

i.e., \( H(L) \) is \( \leq \)-downward closed
Approximation Schema for detecting OR violations

Parametrized weakening function $A_k$, for any $k \geq 0$, s.t.

- $A_k(h) \leq h$
- $A_0(h) \leq A_1(h) \leq A_2(h) \leq \ldots \leq h$
- There is a $k$ s.t. $h \leq A_k(h)$
- Checking if $A_k(h)$ is in $H(L)$ decidable in polynomial time
Approximation Schema for detecting OR violations

Parametrized weakening function $A_k$, for any $k \geq 0$, s.t.

1. $A_k(h) \leq h$
2. $A_0(h) \leq A_1(h) \leq A_2(h) \leq \ldots \leq h$
3. There is a $k$ s.t. $h \leq A_k(h)$
4. Checking if $A_k(h)$ is in $H(L)$ decidable in polynomial time

Approximating History Inclusion

1. Choose a parameter $k \geq 0$
2. Is there an $h$ in $H(L_1)$ s.t. $A_k(h)$ is not in $H(L_2)$?
3. $\leq$-DC $\Rightarrow$ If $A_k(h)$ not in $H(L_2)$, then $h$ is not in $H(L_2)$
A Bounding Concept for Histories

Let $h = (O, <)$ be an Interval Order (history in our case)

Notion of length:

• Past of an operation: $\text{past}(o) = \{o' : o' < o\}$
• Lemma [Rabinovitch’78]:
  The set $\{\text{past}(o) : o \text{ in } O\}$ is linearly ordered
• The $length$ of the order = number of pasts - 1
A Bounding Concept for Histories

Let \( h = (\mathcal{O}, \prec) \) be an Interval Order (history in our case)

Notion of length:

- Past of an operation: \( \text{past}(o) = \{ o' : o' \prec o \} \)
- Lemma [Rabinovitch’78]:
  The set \( \{ \text{past}(o) : o \in \mathcal{O} \} \) is linearly ordered
- The \textit{length} of the order = number of pasts - 1

Bounded interval-length approximation

\( A_k \) maps each \( h \) to some \( h' \leq h \) of length \( k \)

\( \Rightarrow A_k \) keeps precise the information (bounds) about the \( k \) last intervals
Canonical Representation of Interval Orders

- Mapping $I : O \rightarrow [n]^2$ where $n = \text{length}(h)$ [Greenough '76]
- $I(o) = [i, j]$, with $i, j \leq n$, such that
  
  $i = |\{\text{past}(o') : o' < o\}|$ and
  
  $j = |\{\text{past}(o') : \text{not} \ (o < o') \& \ \text{past}(o') \neq \text{past}(o)\}|$

$\text{push}(1)$

$\text{push}(2)$

$\text{push}(3)$

$\text{pop}(1)$

$\text{pop}(2)$

$\text{pop}(3)$

$I(\text{push}(1)) = [0, 0]$  
$I(\text{push}(3)) = [3, 3]$

$I(\text{pop}(1)) = [1, 1]$  
$I(\text{pop}(3)) = [1, 3]$

$I(\text{push}(2)) = [2, 2]$  
$I(\text{pop}(2)) = [4, 4]$
Counting Representation of Interval Orders

Count the number of occurrences of each operation in each interval

• $h = (O, <)$ an IO with canonical representation $I:O \rightarrow [n]^2$
• Then, $\prod(h)$ is the multi-set $\{ (o, I(o)) : o \text{ in } O \}$
• and $\prod(L) = \{ \prod(h) : h \text{ in } L \}$

$H(e)$ is in $H(L)$ iff $\prod(H(e))$ is in $\prod(H(L))$
Reduction to Reachability with Counters

\[ H(L_1) \text{ subset of } H(L_2) \]

iff

\[ \Pi(H(L_1)) \text{ subset of } \Pi(H(L_2)) \]

- Consider only k-bounded-length histories
- Track histories of \( L_1 \) using a finite number of counters
- Use an arithmetic-based representation of \( \Pi(H(L_2)) \)
- Check that \( \Pi(H(L_2)) \) is invariant
- \( \Pi(H(L)) \) can be provided for common data structures
- \( \Pi(H(L)) \) can be automatically constructed if \( L \) is a context-free set of executions. (Use Parikh's theorem.)
- \( \Rightarrow \) dynamic and static analysis
Experimental Results: Coverage

Comparison of violations covered with $k \leq 4$

- Data point: Counts in logarithmic scale over all executions (up to 5 preemptions) on Scal’s nonblocking bounded-reordering queue with $\leq 4$ enqueue and $\leq 4$ dequeue
- White: traditional linearizability checker (e.g., Line-up)
- x-axis: increasing number of executions (1023-2359292)
Experimental Results: Runtime Monitoring

Comparison of runtime overhead between Linearization-based monitoring and Operation counting

- Data point: runtime on logarithmic scale, normalised on unmonitored execution time
- Scal’s nonblocking Michael-Scott queue, 10 enqueue and 10 dequeue operations.
- x-axis is ordered by increasing number of operations
## Experimental Results: Static Analysis

<table>
<thead>
<tr>
<th>Library</th>
<th>Bug</th>
<th>P</th>
<th>k</th>
<th>m</th>
<th>n</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael-Scott Queue</td>
<td>B1 (head)</td>
<td>2x2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>24.76s</td>
</tr>
<tr>
<td>Michael-Scott Queue</td>
<td>B1 (tail)</td>
<td>3x1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>45.44s</td>
</tr>
<tr>
<td>Treiber Stack</td>
<td>B2</td>
<td>3x4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>52.59s</td>
</tr>
<tr>
<td>Treiber Stack</td>
<td>B3 (push)</td>
<td>2x2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>24.46s</td>
</tr>
<tr>
<td>Treiber Stack</td>
<td>B3 (pop)</td>
<td>2x2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>15.16s</td>
</tr>
<tr>
<td>Elimination Stack</td>
<td>B4</td>
<td>4x1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>317.79s</td>
</tr>
<tr>
<td>Elimination Stack</td>
<td>B5</td>
<td>3x1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>222.04s</td>
</tr>
<tr>
<td>Elimination Stack</td>
<td>B2</td>
<td>3x4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>434.84s</td>
</tr>
<tr>
<td>Lock-coupling Set</td>
<td>B6</td>
<td>1x2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>11.27s</td>
</tr>
<tr>
<td>LFDS Queue</td>
<td>B7</td>
<td>2x2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>77.00s</td>
</tr>
</tbody>
</table>

- Static detection of injected refinement violations with CSeq & CBMC.
- Program Pij with i and j invocations to the push and pop methods, explore n-round round-robin schedules with m loop iterations unrolled, with monitor for Ak.
- Bugs: (B1) non-atomic lock, (B2) ABA bug, (B3) non-atomic CAS operation, (B4) misplaced brace, (B5) forgotten assignment, (B6) misplaced
Conclusion/Future work

- Characterization of Observational Refinement
- Bounding concept based on the notion of interval-length
- OR checking —> Reachability problem, using counting
- Suitable bounding concept: low complexity, small bounds
- Application in Dynamic and Static Analysis

- Alternatives to reachability/counting representations?
- Linearizability for special classes of concurrent objects?
- Weak memory models?

- Distributed (replicated) data types?
- Weaker consistency notions must be investigated (eventual consistency, causal consistency, etc.)
Papers

Linearizability: (this talk)
• A. B., M. Emmi, C. Enea, J. Hamza: Verifying Concurrent Programs against Sequential Specifications. ESOP 2013.

Eventual Consistency

Weak Memory Models:
• A. B., E. Derevenetc, R. Meyer: Checking and Enforcing Robustness against TSO. ESOP 2013