On Checking Correctness of Concurrent Data Structures

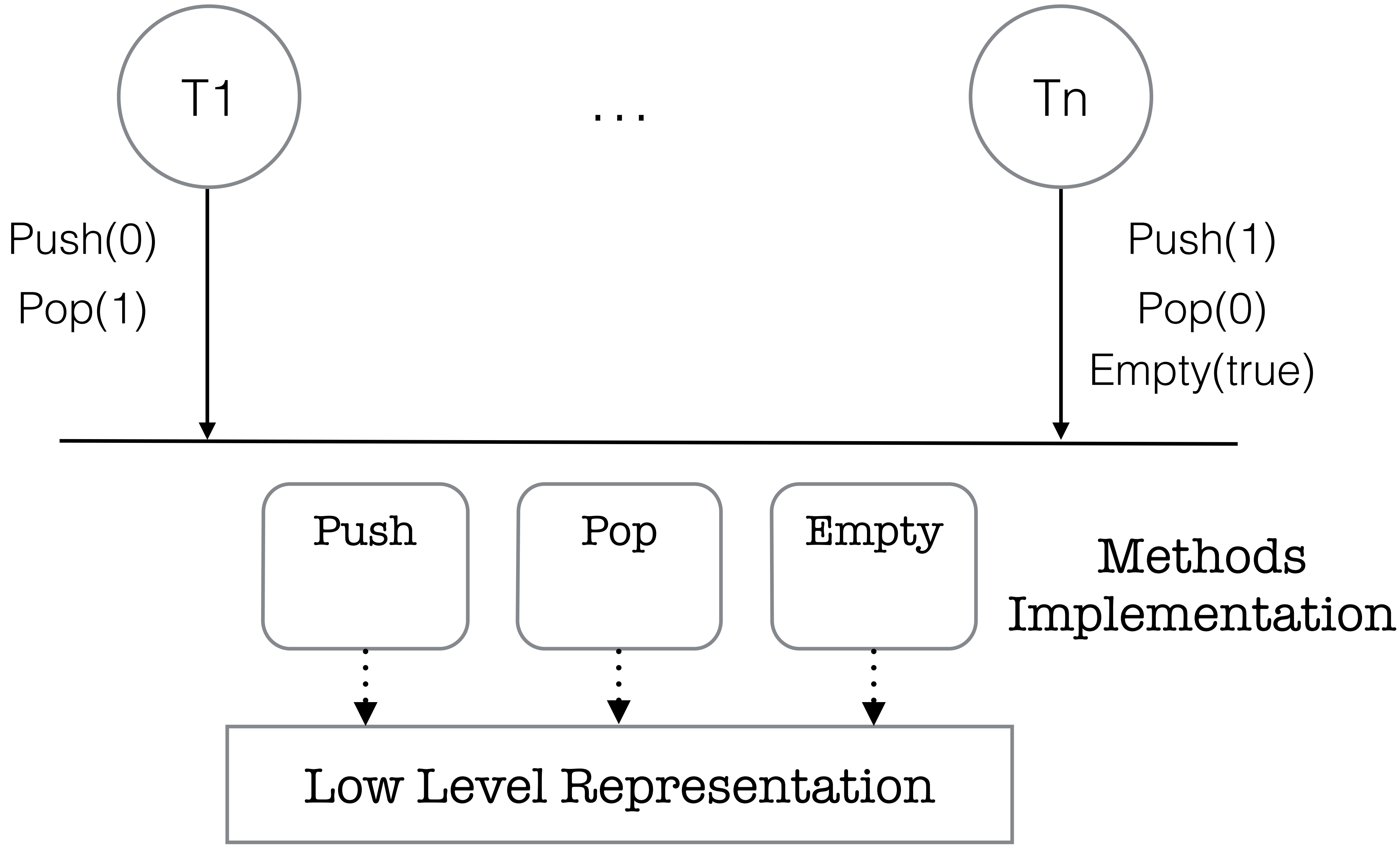
Ahmed Bouajjani
LIAFA, Univ Paris Diderot - Paris 7

Joint work with

Michael Emmi — IMDEA
Constantin Enea — LIAFA, U Paris Diderot - P7
Jad Hamza — LIAFA, U Paris Diderot - P7

Workshop on Software Correctness and Reliability — ETH Zürich, 2014

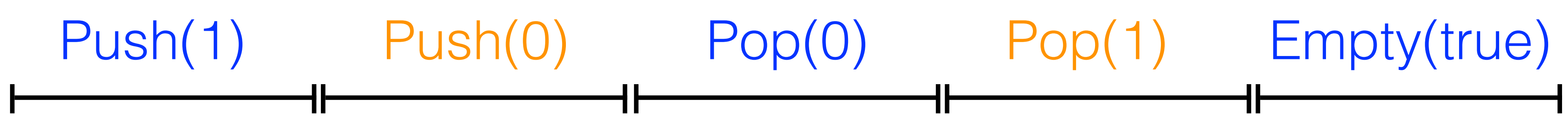
Concurrent Data Structures



Different atomicity levels

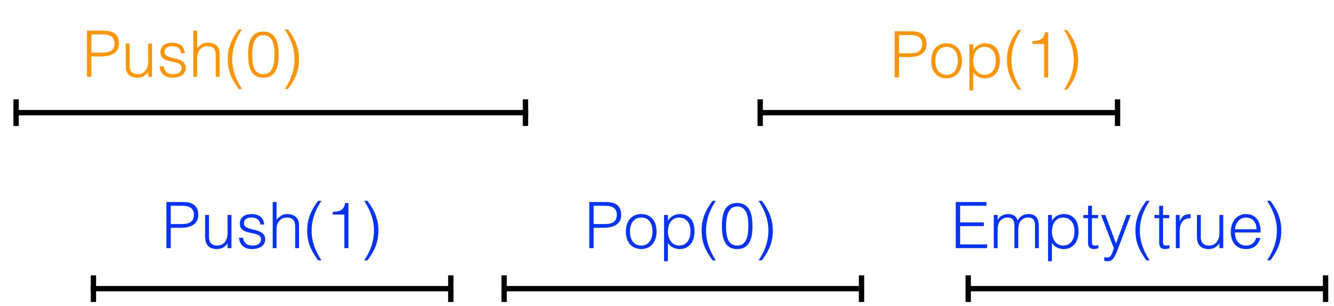
Client view:

- Operations are atomic
- Thread executions are interleaved

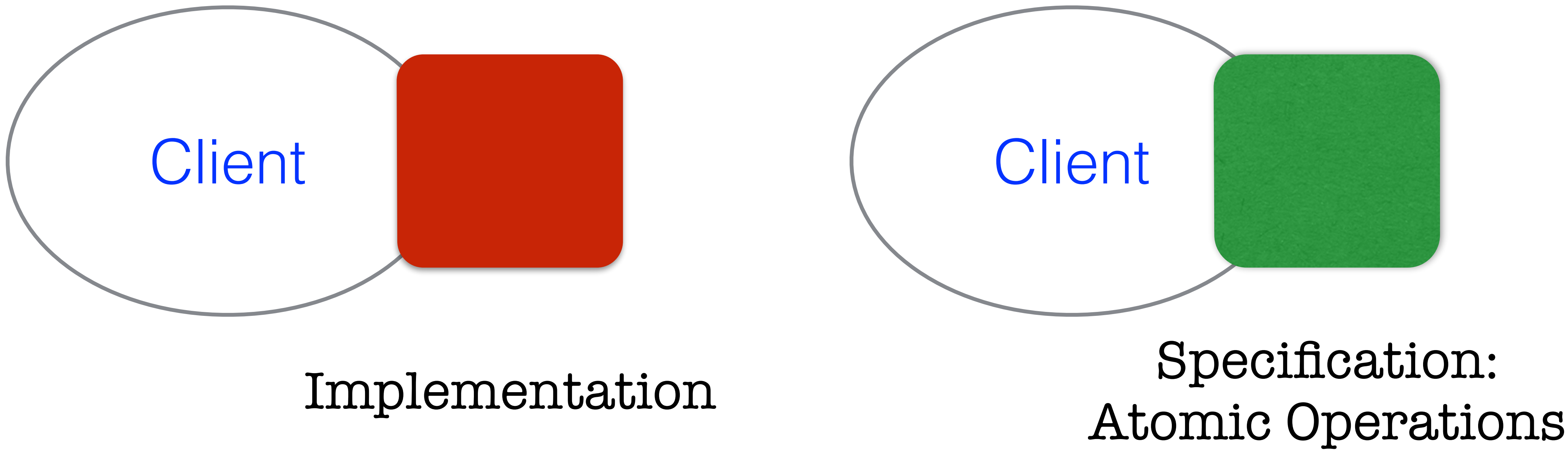


Implementation:

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

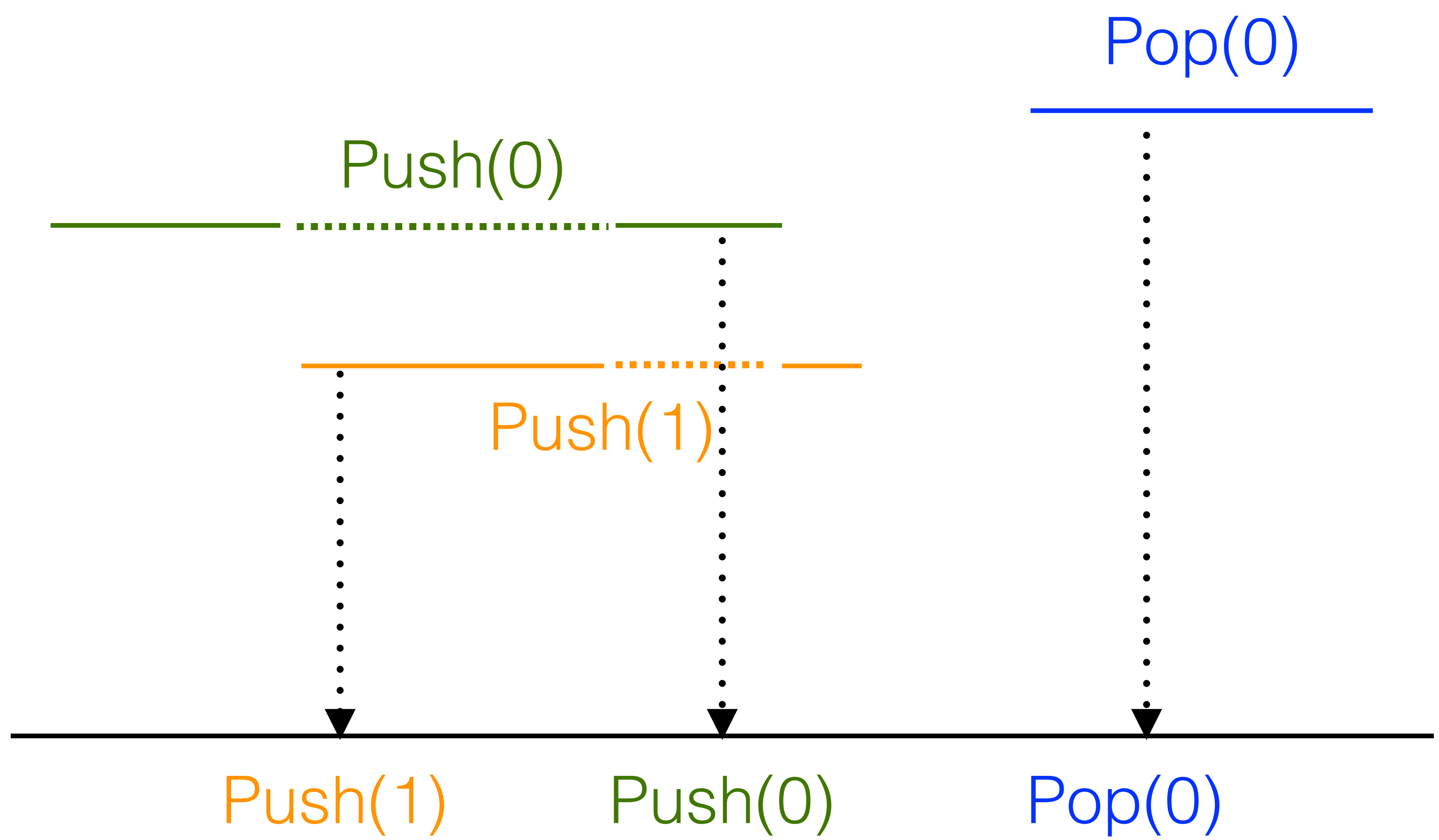


Observational Refinement



For every Client,
Client x Impl included in Client x Spec

Linearizability [Herlihy, Wing, 1990]



Valid sequence in the sequential specification

- reorder call/return events, while preserving returns —> calls
- find “linearization points” within execution time intervals
- => match a sequential execution

Linearizability *implies* Observational Refinement
[Filipovic et al. 2010]

Complexity

Finite number of threads, regular specification

- Checking linearizability is in EXPSPACE [Alur et al. 1996]
- Lower bound: EXSPACE-hardness [Jad Hamza ’14]
- Checking linearizability of a single execution is 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]

- Parametrized 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:

L1 refines L2

iff

$(P \times L1)|_A$ is a subset of $(P \times L2)|_A$, for every P

Histories

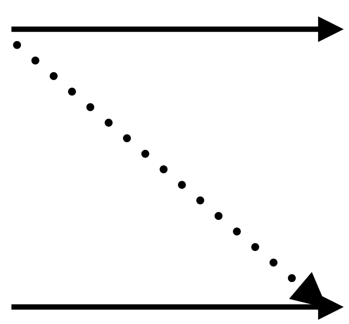
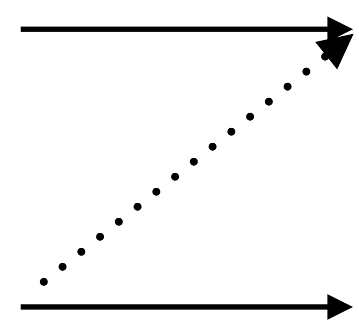
History of an execution e :

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

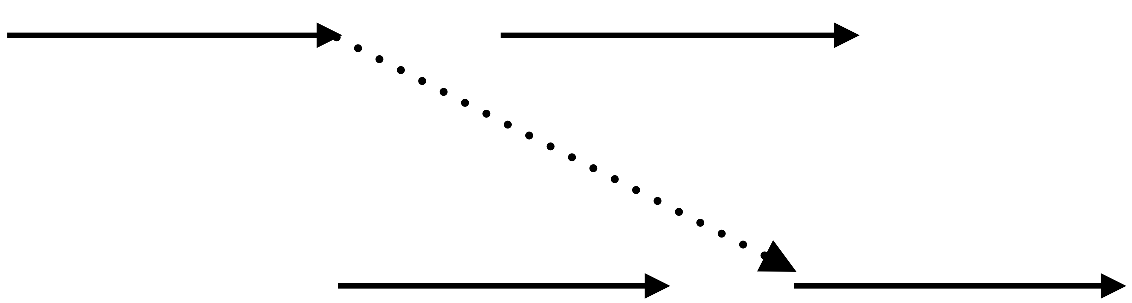
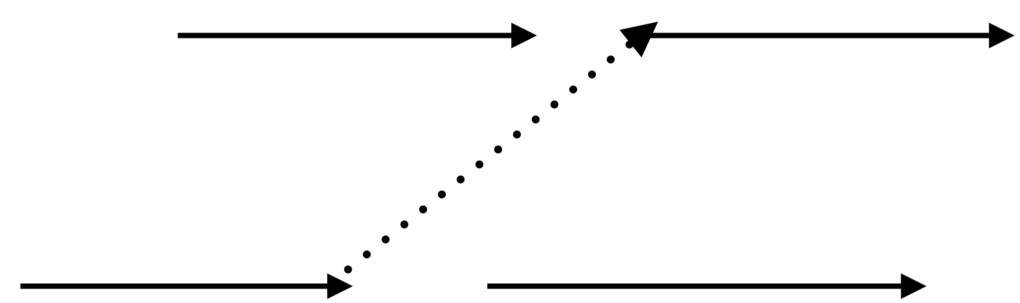
$O1 < O2$ iff Return($O1$) is before Call($O2$) in e

Histories as Interval Orders

- Interval Orders = partial order $(O, <)$ such that
  $(o1 < o1' \text{ and } o2 < o2') \text{ implies } (o1 < o2' \text{ or } o2 < o1')$

- For every execution *e*, H(*e*) is an interval order

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

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

$h_1 \leq h_2$ (h_1 is weaker than h_2)

iff

h_1 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.

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

Approximating History Inclusion

- Choose a parameter $k \geq 0$
- Is there an h in $H(L_1)$ s.t. $A_k(h)$ is not in $H(L_2)$?
- $\leq$-DC => 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: past(o) = {o' : o' < o}
- Lemma [Rabinovitch'78]:

  The set {past(o) : o in O} is linearly ordered

- The *length* of the order = number of pasts - 1

A Bounding Concept for Histories

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

Notion of length:

- Past of an operation: past(o) = {o' : o' < o}
- Lemma [Rabinovitch'78]:

  The set {past(o) : o in O} is linearly ordered

- The *length* of the order = number of pasts - 1

Bounded interval-length approximation

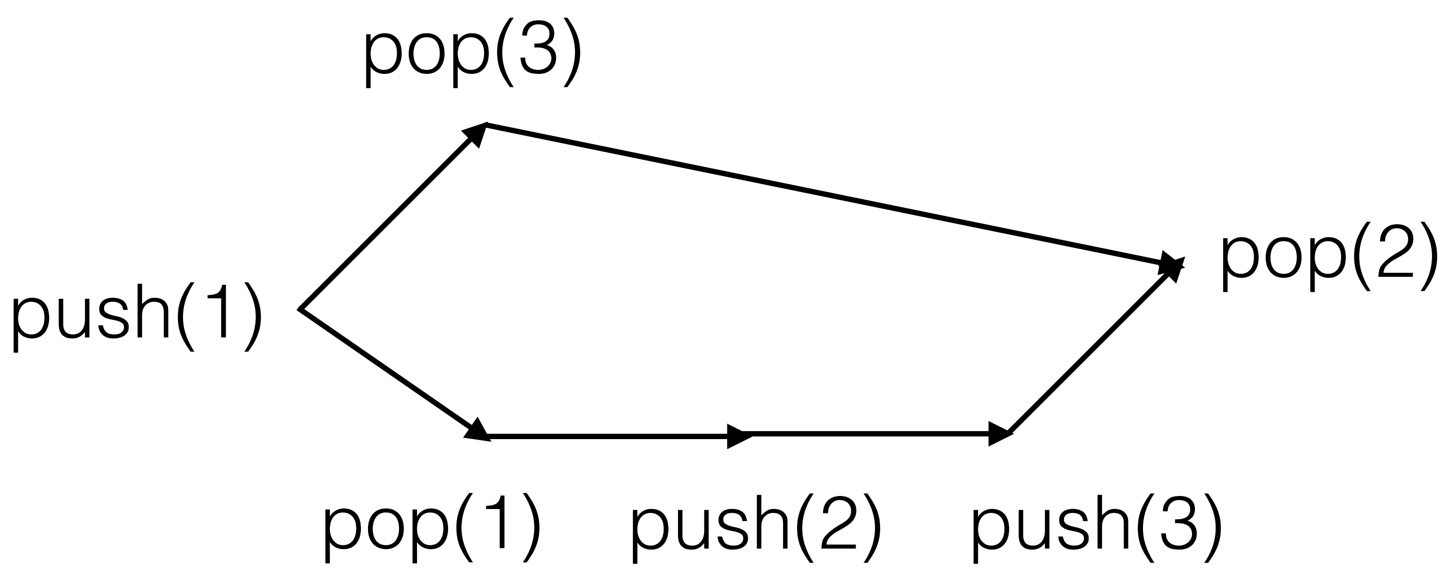
A_k maps each h to some h' ≤ h of length k

=> A_k keeps precise the information (bounds) about the k last intervals

Canonical Representation of Interval Orders

- Mapping I : O —> $[n]^2$ where n = length(h) [Greenough '76]
- I(o) = [i, j], with i, j ≤ n, such that

i = |{past(o') : o' < o}| and
j = |{past(o') : not (o < o') & past(o') ≠ past(o)}|



I(push(1)) = [0, 0]
I(pop(1)) = [1, 1]
I(push(2)) = [2, 2]

I(push(3) = [3, 3]
I(pop(3)) = [1, 3]
I(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 $I : O \longrightarrow [n]^2$
- Then, $\Pi(h)$ is the multi-set $\{\{ (o, I(o)) : o \text{ in } O \}\}$
- and $\Pi(L) = \{\Pi(h) : h \text{ in } L\}$

$H(e)$ is in $H(L)$ iff $\Pi(H(e))$ is in $\Pi(H(L))$

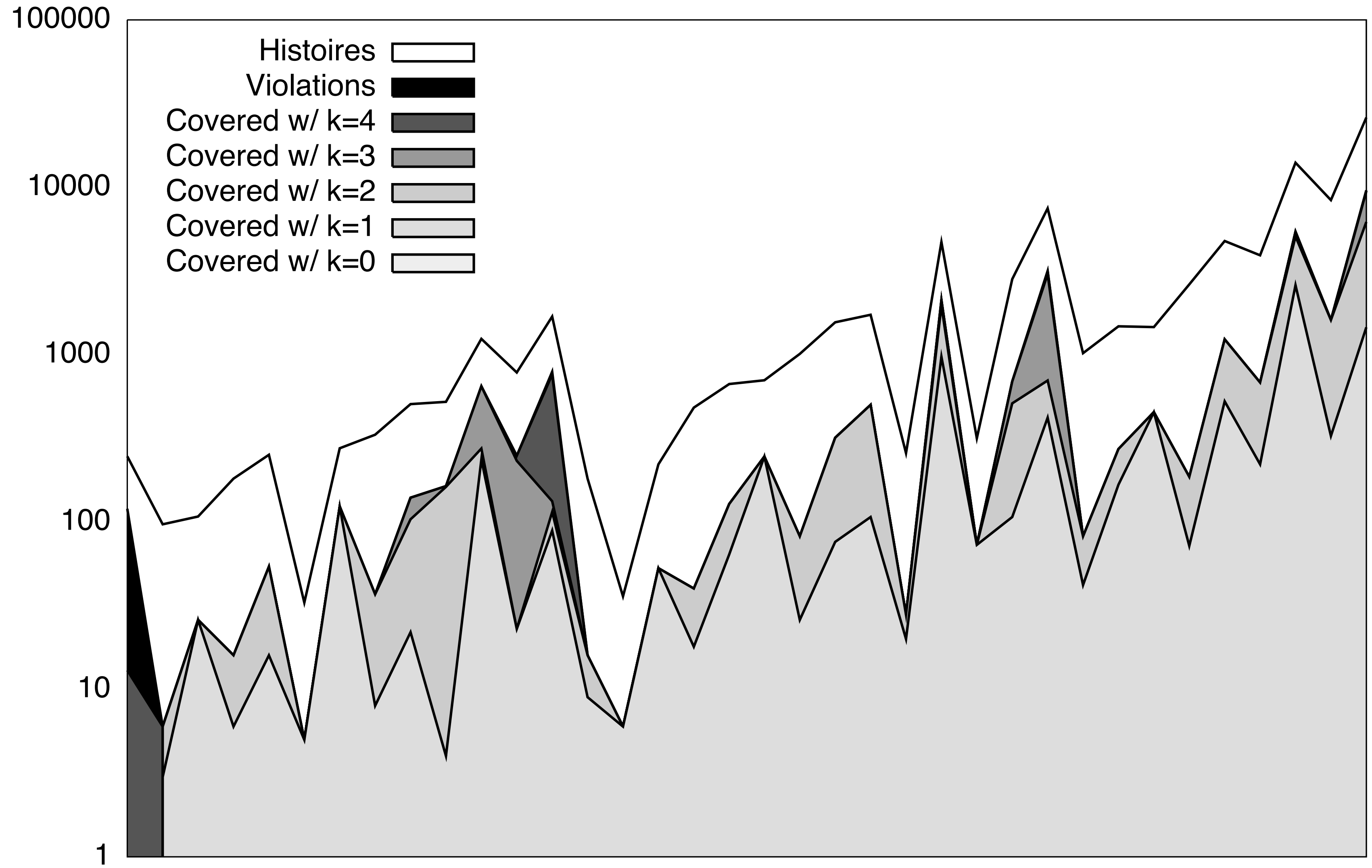
Reduction to Reachability with Counters

$H(L_1)$ subset of $H(L_2)$
iff
$\Pi(H(L_1))$ 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.)
- => dynamic and static analysis

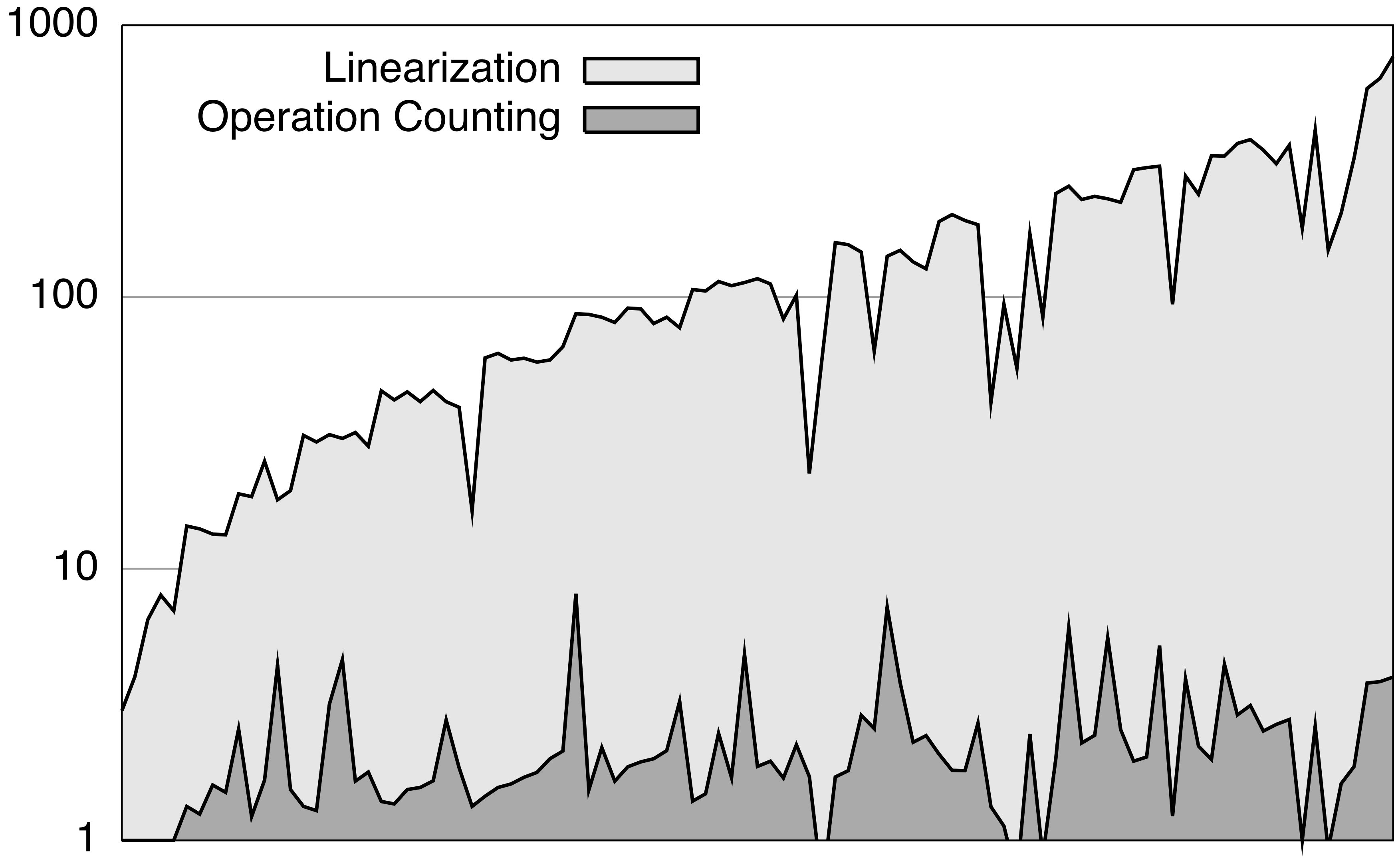
Experimental Results: Coverage



Comparison of violations covered with k ≤ 4

- Data point: Counts in logarithmic scale over all executions (up to 5 preemptions) on Scal's nonblocking bounded-reordering queue with ≤4 enqueue and ≤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

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

- 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.
- A. B., M. Emmi, C. Enea, J. Hamza: Tractable Refinement Checking for Concurrent Objects. POPL 2015.

Eventual Consistency

- A. B., C. Enea, J. Hamza: Verifying Eventual Consistency for Optimistic Replication Systems. POPL 2014.

Weak Memory Models:

- M.F. Atig, A. B., S. Burckhardt, M. Musuvathi: On the Verification Problem of Weak Memory Models. POPL 2010.
- M.F. Atig, A. B., S. Burckhardt, M. Musuvathi: What's Decidable about Weak Memory Models. ESOP 2012.
- A. B., E. Derevenetc, R. Meyer: Checking and Enforcing Robustness against TSO. ESOP 2013