

Refinement Checking for Libraries of Concurrents Objects

Ahmed Bouajjani

LIAFA, Univ Paris Diderot - Paris 7

Joint work with

Michael Emmi

IMDEA

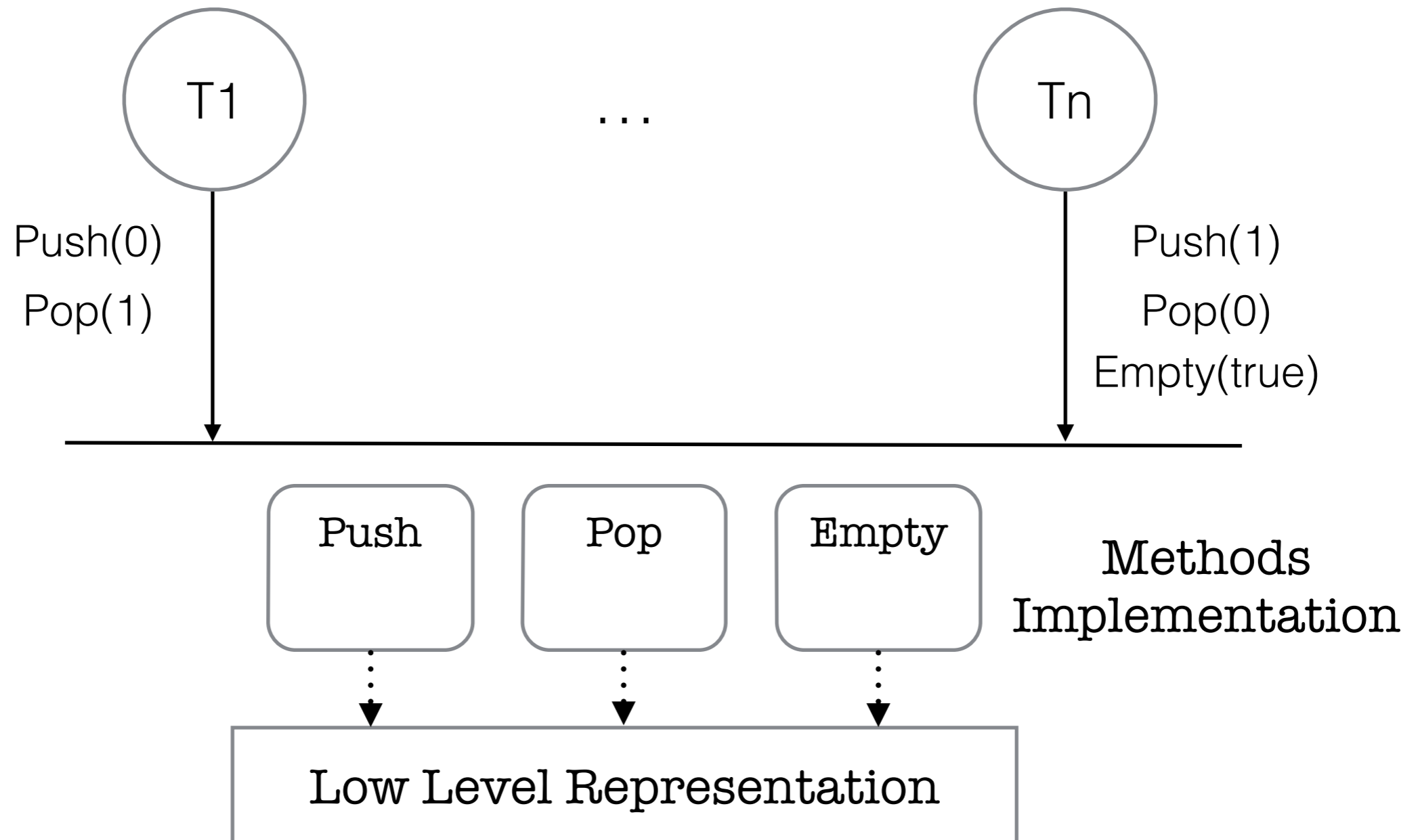
Constantin Enea

LIAFA, U Paris Diderot - P7

Jad Hamza

IFIP WG 2.2 meeting, Lucca, September 23, 2015

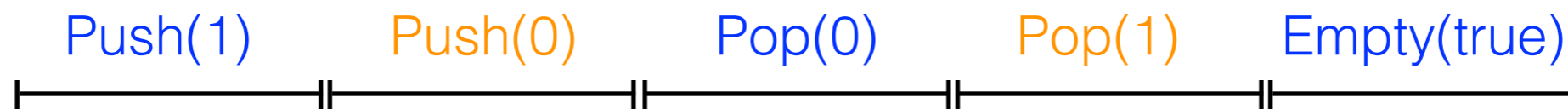
Concurrent Data Structures



Different atomicity levels

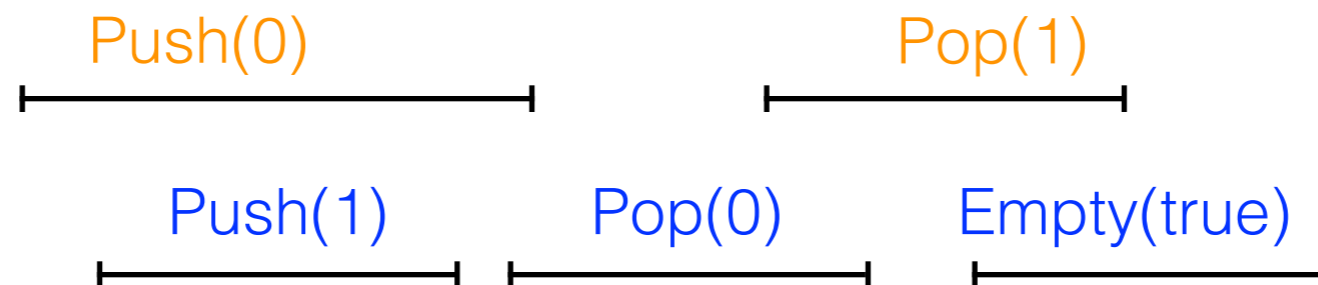
Client view:

- Operations are **atomic**
- Thread executions are interleaved

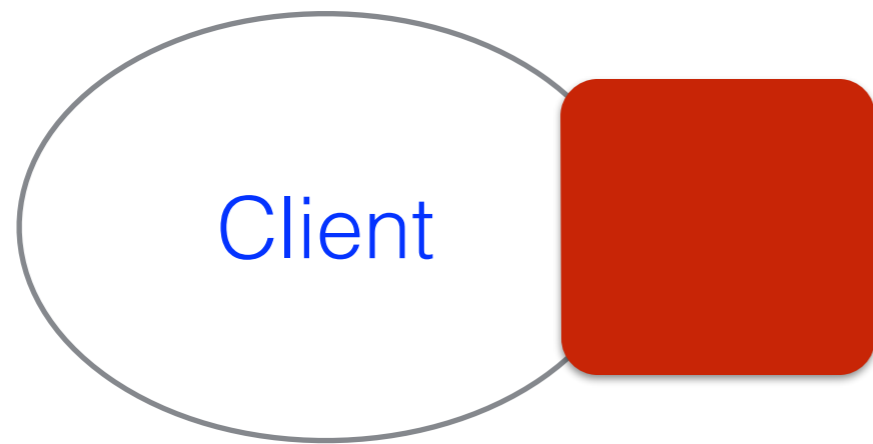


Implementation:

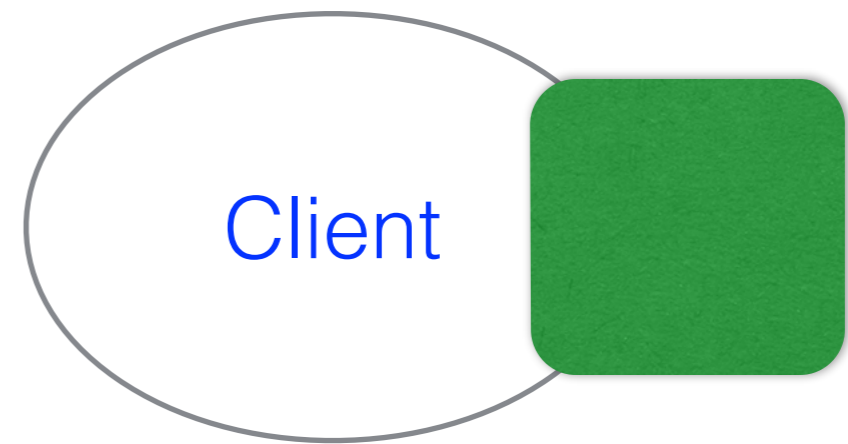
- **Naive solution:** Coarse-grain locking
- **Performances:** Avoid coarse-grain locking
- => **Execution intervals may overlap**



Observational Refinement



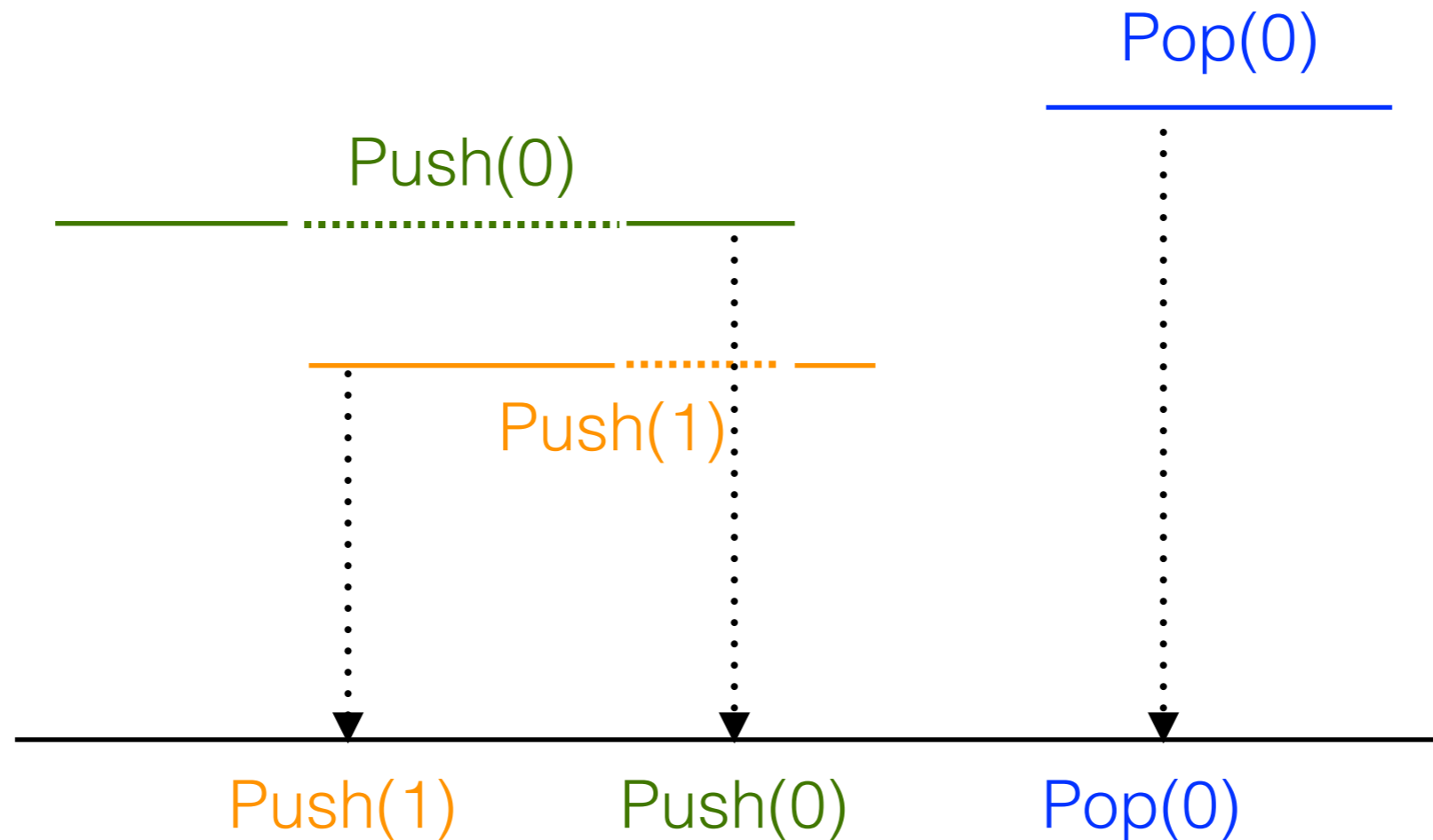
Implementation



Specification:
Atomic Operations

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 \rightarrow calls
- Find “linearization points” within execution time intervals
- \Rightarrow match a sequential execution

Linearizability *implies* Observational Refinement

[Filipovic, O’Hearn, Rinetzky, Yang, 2009]

Complexity

- Checking linearizability of a **single execution** is **NP-complete**
[Gibbons, Korach, 1997]
- For finite-state implementations and specifications:

	Linearizability	State Reachability
Fixed Nb Threads	EXPSPACE-complete (1)	PSPACE-complete
Unbounded Nb of Threads	Undecidable (2)	EXPSPACE-complete

(1) Upper Bound: Alur, McMillan, Peled 1996 — Lower Bound: Hamza 2015

(2) B., Emmi, Enea, Hamza, 2013

Tractable Reduction of Linearizability/OR to State Reachability?

Why?

- Reuse existing tools for SR
- Lower complexity
- Decidability

Two approaches:

- **Under approximations:** special classes of executions
 - Parametrized under-approximation schema?
 - Good coverage, low complexity, scalability
- **Special classes of objects:**
Stacks, queues, etc.

Under-approximate detection of OR violations ?

[B, Emmi, Enea, Hamza, POPL'15]

Characterizing OR as a History Inclusion Problem

- “Histories” are a special partial orders: Interval orders
- => Interval-length bounding

Efficient Reduction to Reachability

- Use counting representations
- => Correctness of a single history is Polynomial time
- => Scalable dynamic and static analysis techniques

Good coverage with small bounds

- ≤ 3
- Cut-off bounds for common data structures

Histories

History of an execution e :

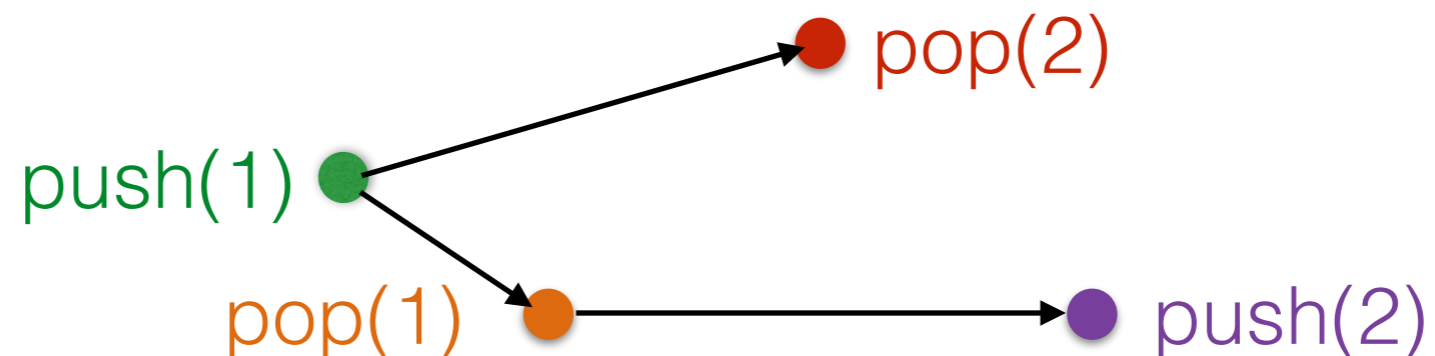
$$H(e) = (O, \text{label}, <)$$

where

- $O = \text{Operations}(e)$
- $\text{label}: O \rightarrow M \times V \times V$
- $<$ is a partial order s.t.

$O1 < O2$ iff $\text{Return}(O1)$ is *before* $\text{Call}(O2)$ in e

$c(\text{push}, 1)$ $r(\text{push}, \text{tt})$ $c(\text{pop}, -)$ $c(\text{pop}, -)$ $r(\text{pop}, 1)$ $c(\text{push}, 2)$ $r(\text{push}, \text{tt})$ $r(\text{pop}, 2)$



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

Lemma:

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

Approximation Schema for detecting OR violations

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

- $A_k(h) \leq h$
- $A_0(h) \leq A_1(h) \leq A_2(h) \leq \dots \leq h$
- 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.

- $A_k(h) \leq h$
- $A_0(h) \leq A_1(h) \leq A_2(h) \leq \dots \leq h$
- 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 $H(L_2)$?
- **Lemma** \Rightarrow If $A_k(h)$ not in $H(L_2)$, then h is not in $H(L_2)$

Histories are Interval Orders

Interval Orders = partial order $(O, <)$ such that

$(o_1 < o_1'$ and $o_2 < o_2')$ implies $(o_1 < o_2'$ or $o_2 < o_1')$



Prop: For every execution e , $H(e)$ is an interval order

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 $h = (O, <)$ 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

Bounded interval-length approximation

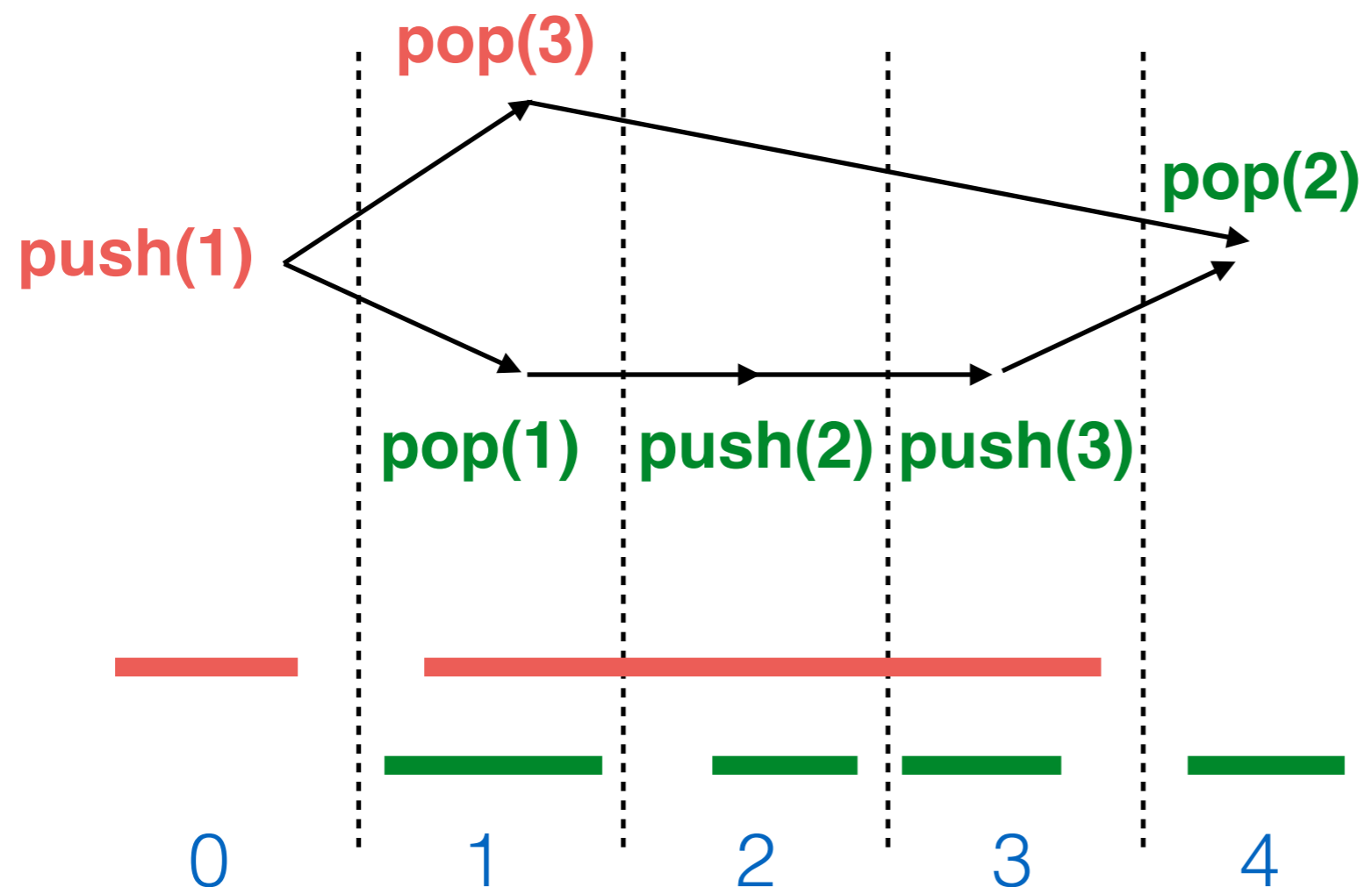
A_k maps each h to some $h' \leq h$ of length k

$\Rightarrow A_k$ 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

$$i = |\{\text{past}(o') : o' < o\}| \text{ and} \\ j = |\{\text{past}(o') : \text{not } (o < o')\}| - 1$$



$$\begin{aligned} I(\text{push}(1)) &= [0, 0] \\ I(\text{pop}(1)) &= [1, 1] \\ I(\text{push}(2)) &= [2, 2] \\ I(\text{push}(3)) &= [3, 3] \\ I(\text{pop}(3)) &= [1, 3] \\ I(\text{pop}(2)) &= [4, 4] \end{aligned}$$

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, let $\Pi(h)$ be the multi-set $\{ [\text{label}(o), I(o)] : o \text{ in } O \}$
- Use a counter per type of operation and interval

Prop: $H(e)$ 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
- \Rightarrow Dynamic and static analysis

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
- \Rightarrow Dynamic and static analysis

How to get $\Pi(H(L))$?

Operation counting formulas

- Logic for expressing operation counting constraints

Presburger arithmetics with $\#(a, i, j)$ predicates
where

$$[\#(a, i, j)](h) = |\{o : l(o) = [i, j] \text{ and } \text{label}(o) = a\}|$$

Number of occurrences of a in the interval $[i, j]$

- $h \models F$ is polynomial time (for a fixed quantifier count)

Building operation counting formulas

- For any k , a counting formula for $\Pi(H(L))$ can be provided for common data structures (**stack, queue, set,...**)
- Assume **data independence** \Rightarrow push's have \neq values

Violation of FIFO property (FV):

$$\exists x_1, x_2, y_1, y_2.$$

$$\text{Match}(x_1, y_1) \ \& \ \text{Match}(x_2, y_2)$$

$$\text{Before}_k(x_1, x_2) \ \& \ \text{Before}_k(y_2, y_1)$$

where

$$\text{Match}(x, y) = \text{IsPush}(x) \ \& \ \text{IsPop}(y) \ \& \ \text{SameVal}(x, y)$$

$$\text{Before}_k(x, y) = \bigvee_{i \leq k} (\text{count}(x, 0, i) > 0) \ \& \ \text{count}(y, 0, i) = 0 \ \& \ \text{count}(x, i+1, k) = 0$$

where

$$\text{count}(x, i, j) = \sum_{i \leq i' \leq j' \leq j} \#(x, i', j')$$

Building operation counting formulas

FIFO queue data structure:

Violation of FIFO property (FV):

$$\exists x_1, x_2, y_1, y_2. \text{Match}(x_1, y_1) \ \& \ \text{Match}(x_2, y_2) \ \& \ \text{Before}_\kappa(x_1, x_2) \ \& \ \text{Before}_\kappa(y_2, y_1)$$

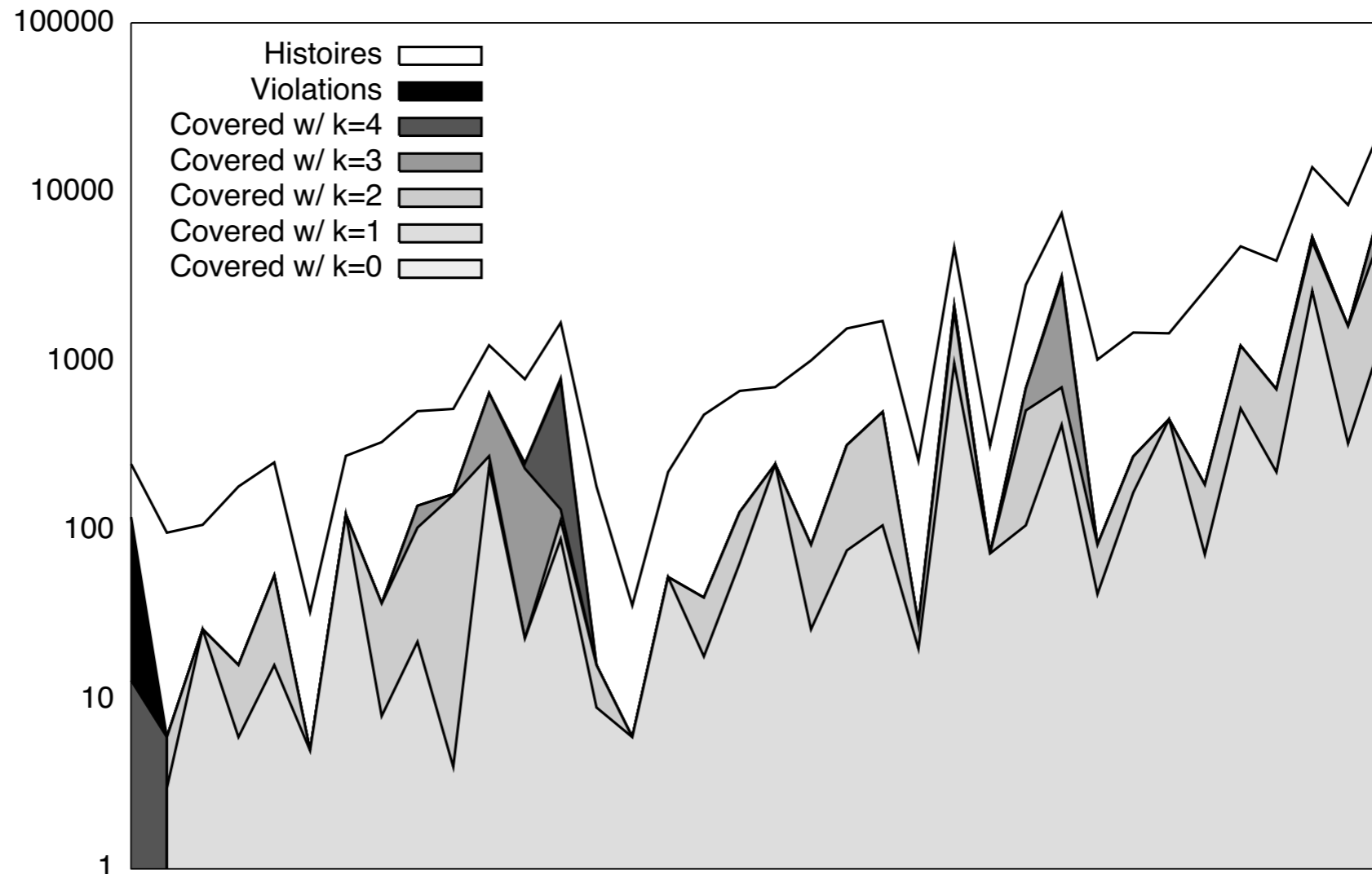
Violation of Remove property (RV):

$$\exists x, y. \text{Match}(x, y) \ \& \ \text{Before}_\kappa(y, x)$$

Violation of Empty property (EV):

$$\exists x, y, z. \text{Match}(x, z) \ \& \ \text{EmptyPop}(y) \ \& \ \text{Before}_\kappa(y, z) \ \& \ \text{Before}_\kappa(y, z)$$

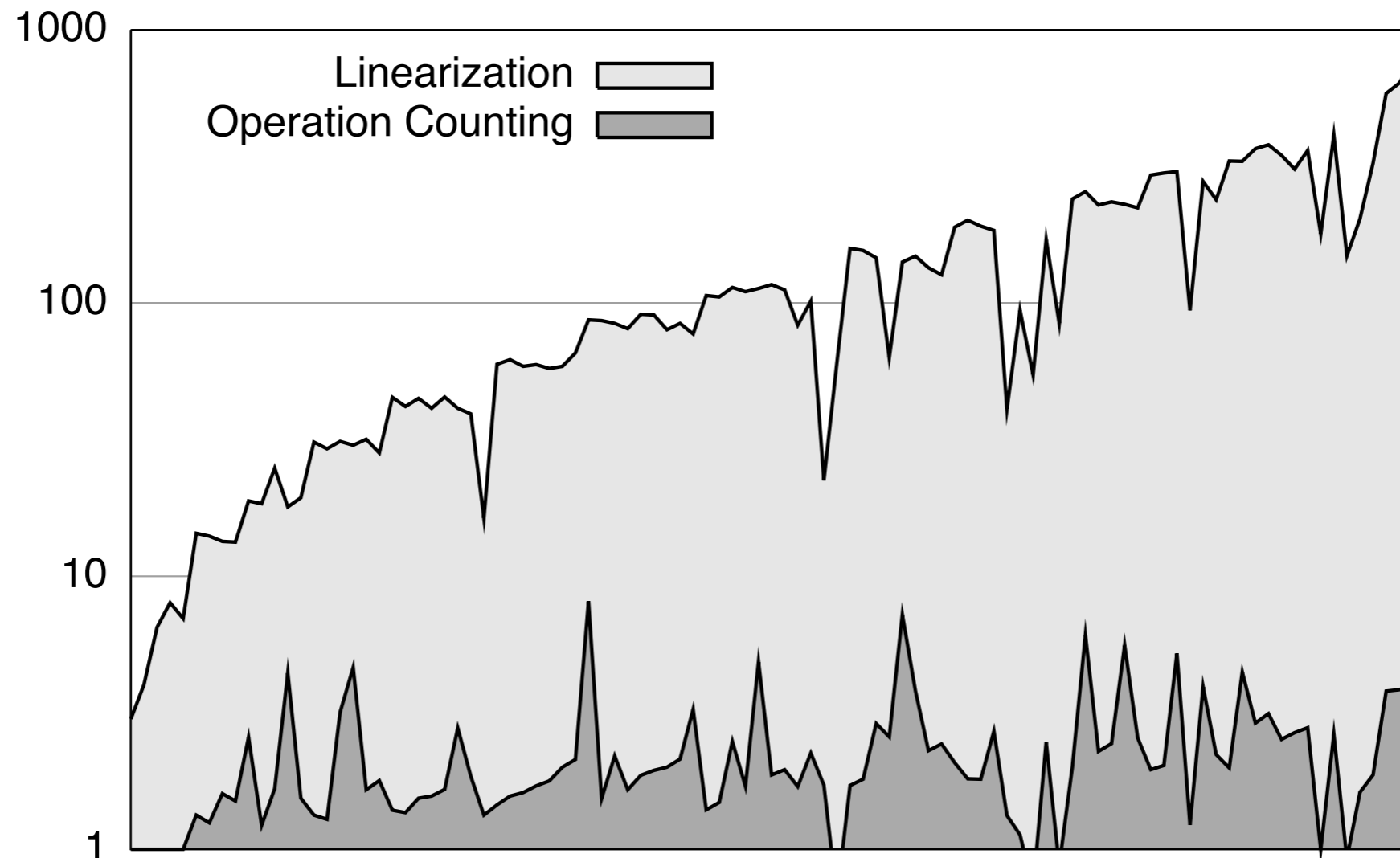
Experimental Results: Coverage



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 ≤ 4 enqueue and ≤ 4 dequeue
- x-axis: increasing number of executions (1023-2359292)
- White: total number of unique histories over a given set of executions
- Black: violations detected by traditional linearizability checker (e.g., Line-up)

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

Focusing on Special Classes of Objects

[B., Emmi, Enea, Hamza, ICALP 2015]

- Inductive definition of sequential objects (restricted language based on constrained rewrite rules)
- Characterizing concurrent violations using a finite number of “bad patterns”
- Defining finite-state automata recognising each of the “bad patterns” (using data independence assumption)
- Reducing linearizability to checking the emptiness of the intersection with these automata.

Specifying queues and stacks

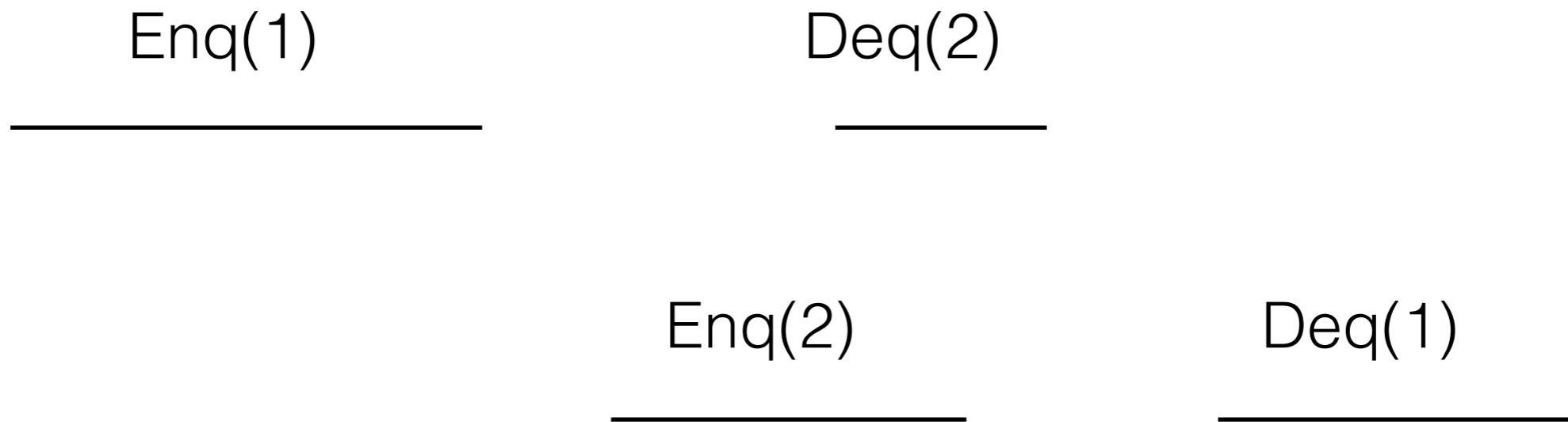
Queue

- $u . v : Q \ \& \ u : \text{ENQ}^* \longrightarrow \mathbf{Enq(x)} . u . \mathbf{Deq(x)} . v : Q$
- $u . v : Q \ \& \ \text{no unmatched } \textit{Enq} \text{ in } u \longrightarrow u . \mathbf{Emp} . v : Q$

Stack

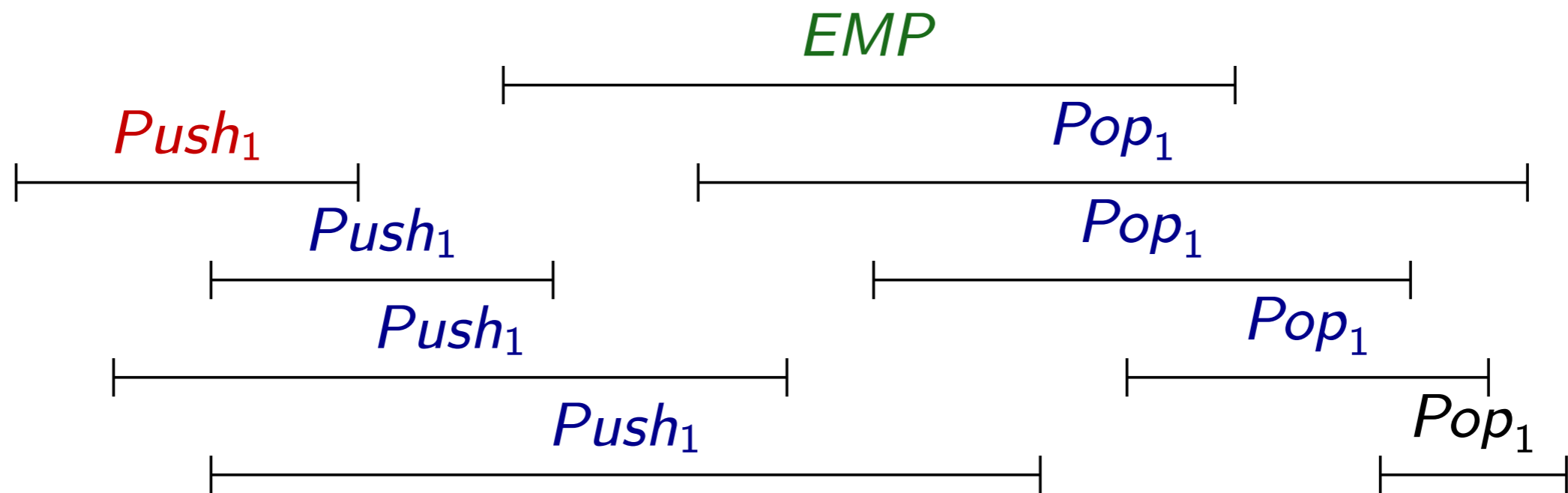
- $u . v : S \ \& \ \text{no unmatched } \textit{Push} \text{ in } u \longrightarrow \mathbf{Push(x)} . u . \mathbf{Pop(x)} . v : S$
- $u . v : S \ \& \ \text{no unmatched } \textit{Push} \text{ in } u \longrightarrow u . \mathbf{Emp} . v : S$

Order Violation

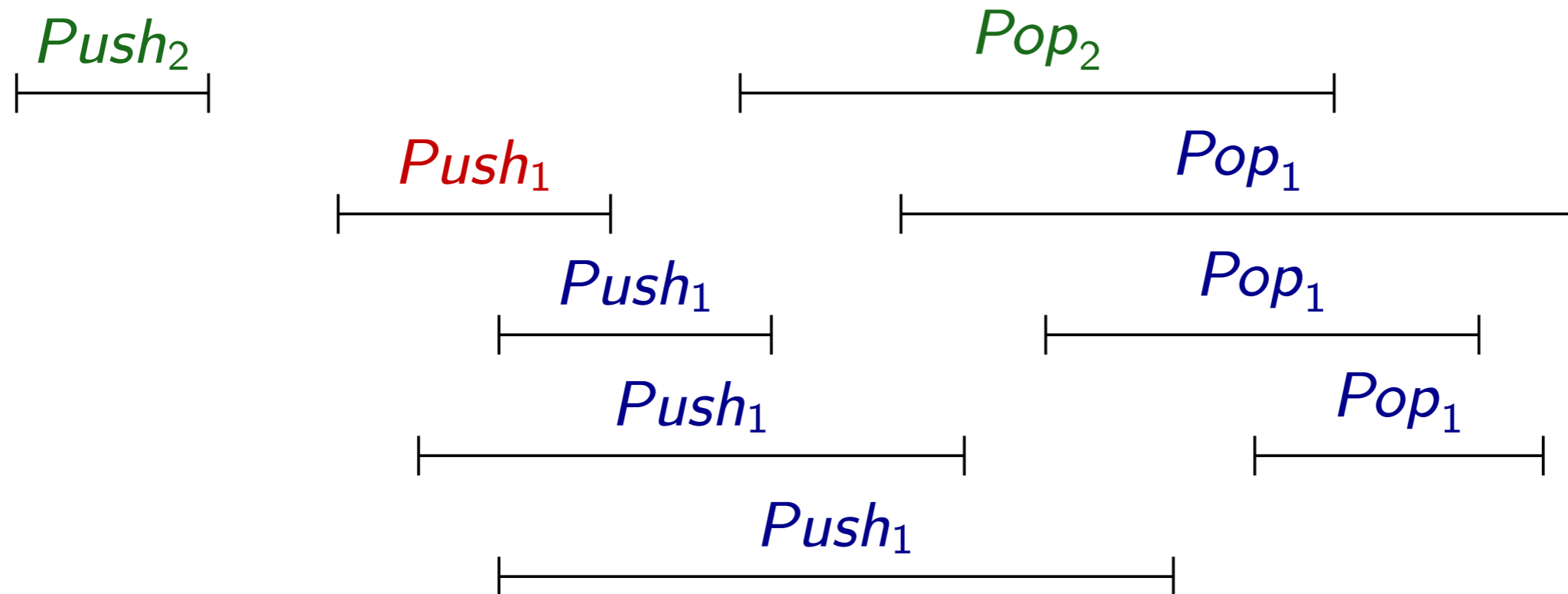


$\text{Enq}(1) < \text{Enq}(2) \ \& \ \text{Deq}(2) < \text{Deq}(1)$

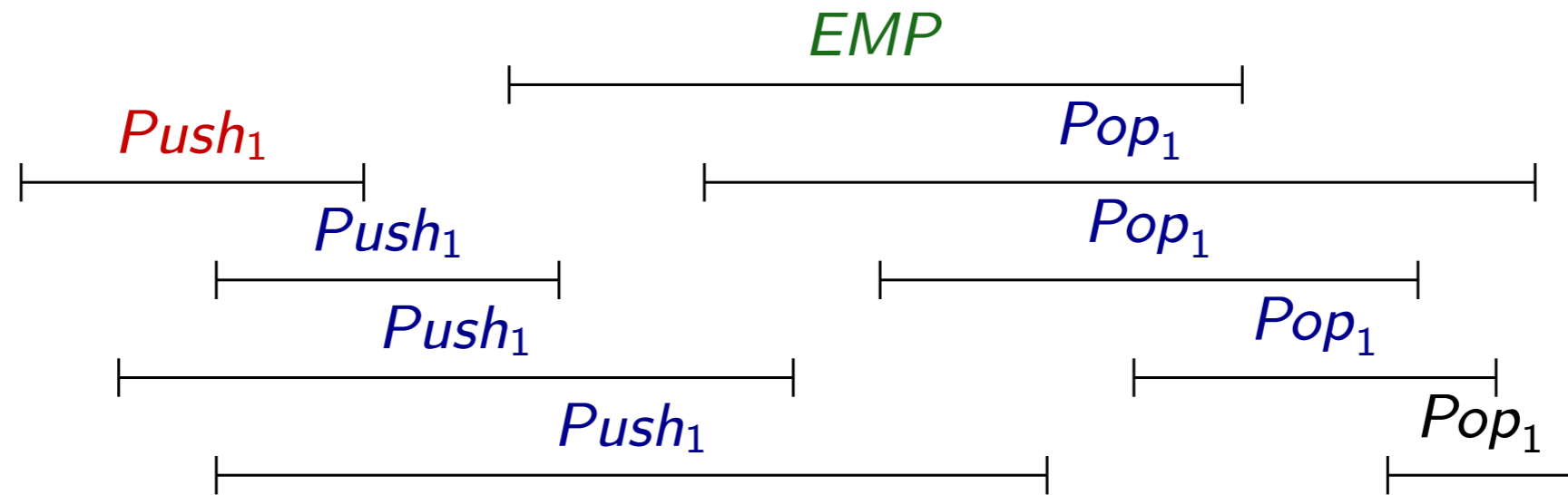
Empty Violation



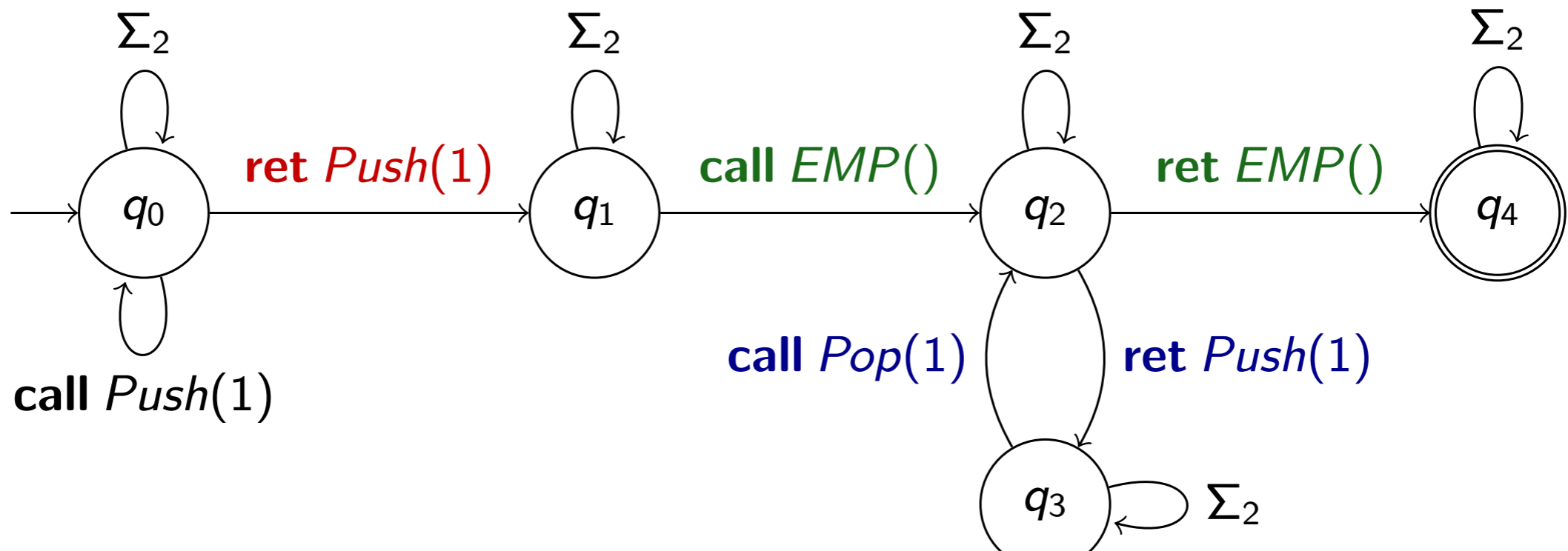
Order Violation cont.



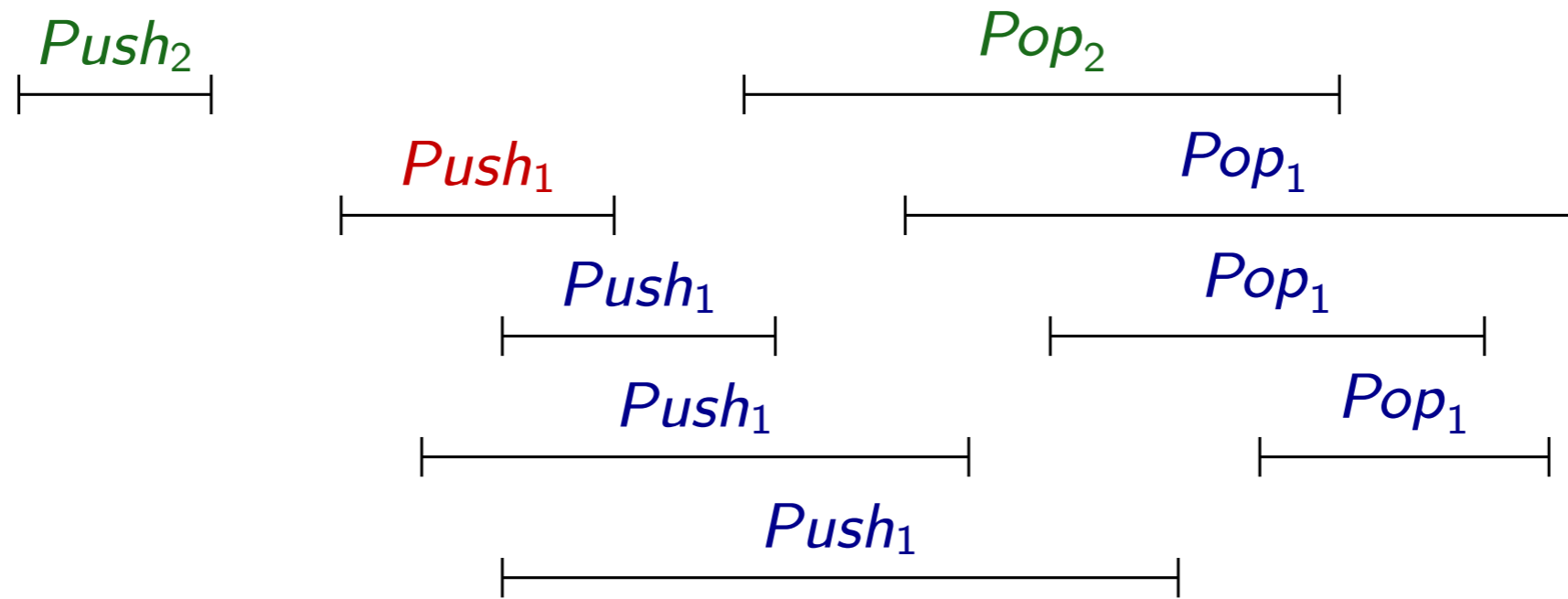
Automaton for Empty Violation



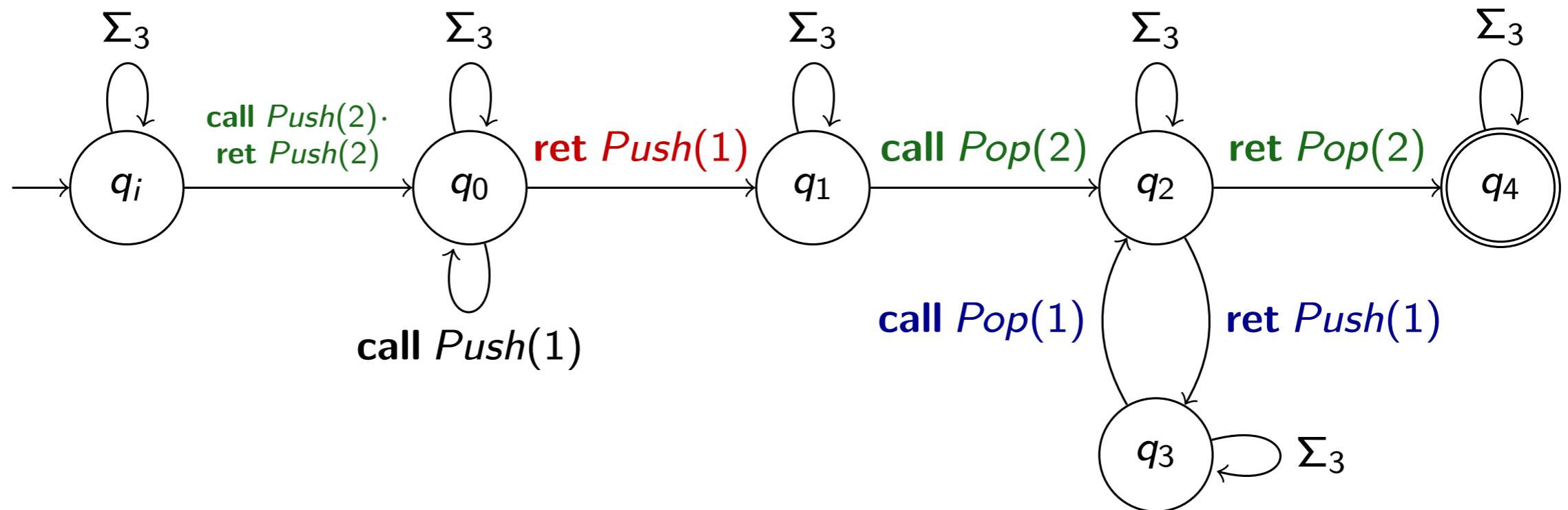
Recognized by:



Automaton for Push-Pop Order Violation



Recognized by:



Linearizability to State Reachability

Thm:

For each S in {Stack, Queue, Mutex, Register}, there is an automaton $A(S)$ s.t. for every data independent concurrent object C , C is linearisable wrt S iff the intersection of $H(C)$ wrt $H(A(S))$ is empty.

Same complexity as state reachability

Conclusion/Future work

- Bounding concept based on the notion of interval-length
- OR checking \rightarrow Reachability problem, using counting
- Suitable bounding concept: low complexity, small bounds
- Application in Dynamic and Static Analysis
- \Rightarrow Automatic synthesis of “specification”

- Reduction to SR for a common concurrent objects
- Reuse of existing verification technology
- \Rightarrow Extension to other kind of objects, e.g., sets
- \Rightarrow Abstractions / under-approximate analysis