Machine-Verifying Concurrent Data Structures

Siddhartha Jayanti

Google Research

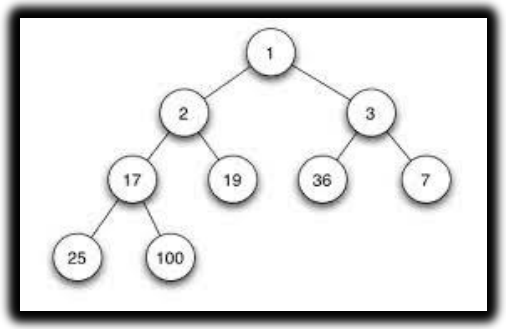
Dartmouth

Collaborators and Co-authors

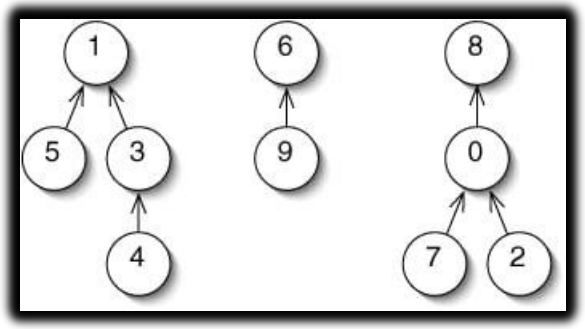
[POPL 2024] with Prasad Jayanti, Ugur Yavuz, Lizzie Hernandez
[DIST 2021 , PODC 2019, PODC 2016] with Robert Tarjan
[Google Research] with several, including Laxman Dhulipala, Guy Blelloch

Motivation: Fast Data Structures

Data Structure

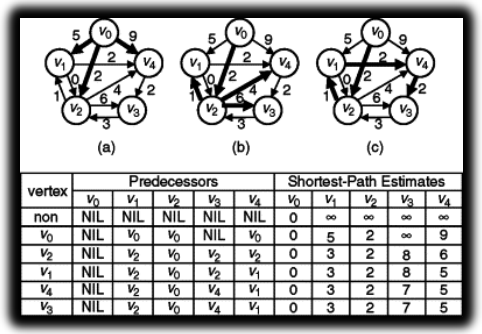


Priority Queue

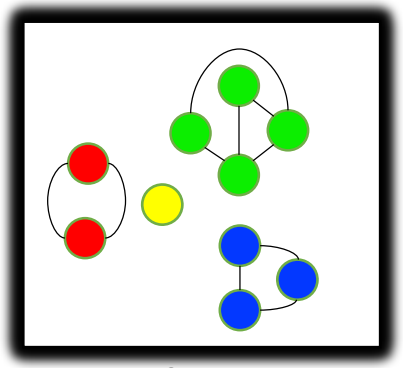


Union Find

Algorithm



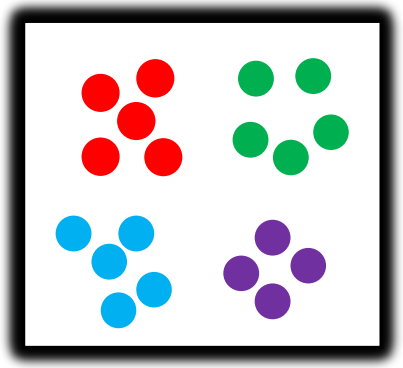
Shortest Paths



Connected Components

Problem

Navigation

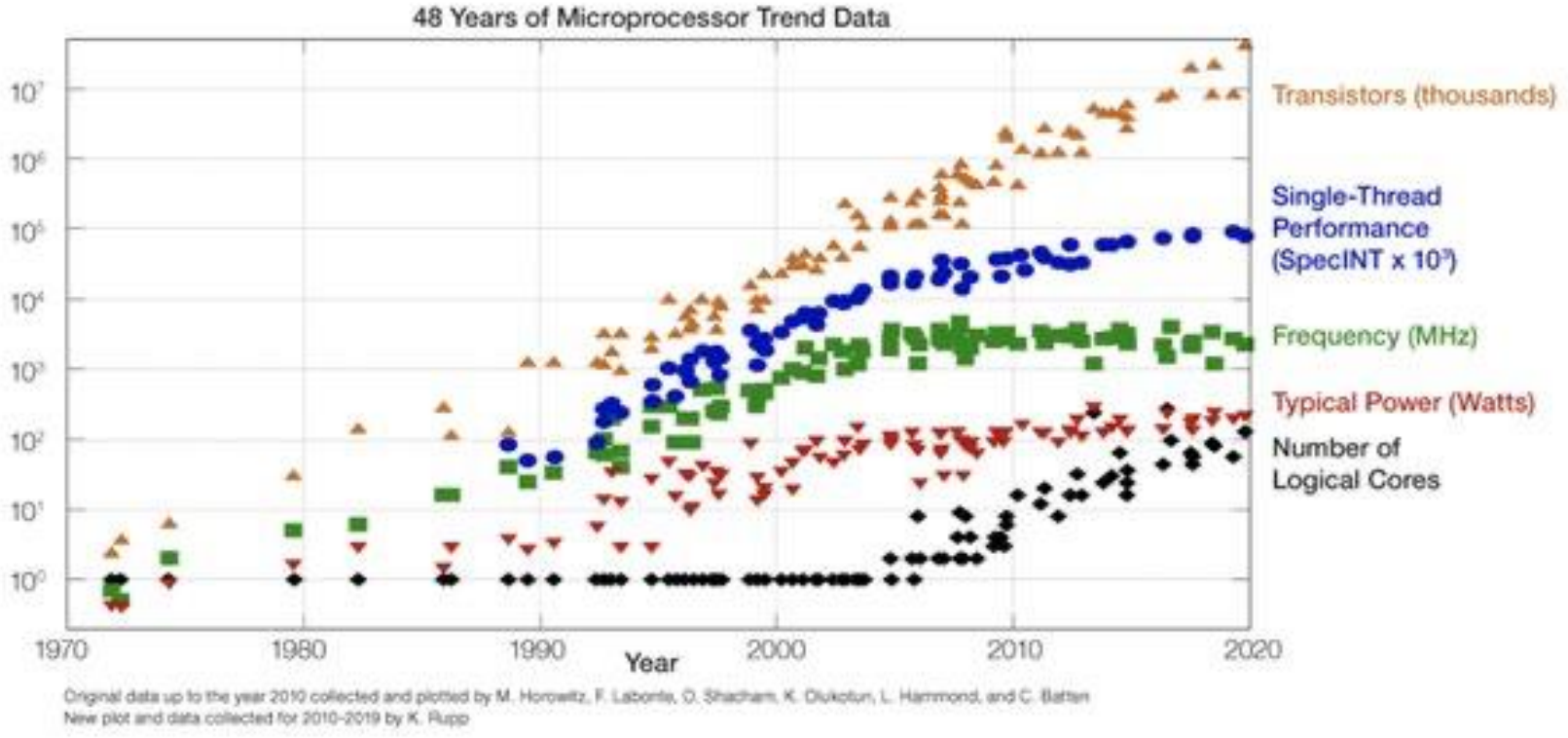


Clustering

Motivation: Fast **Concurrent** Data Structures

- Fast Data Structures → Fast Algorithms

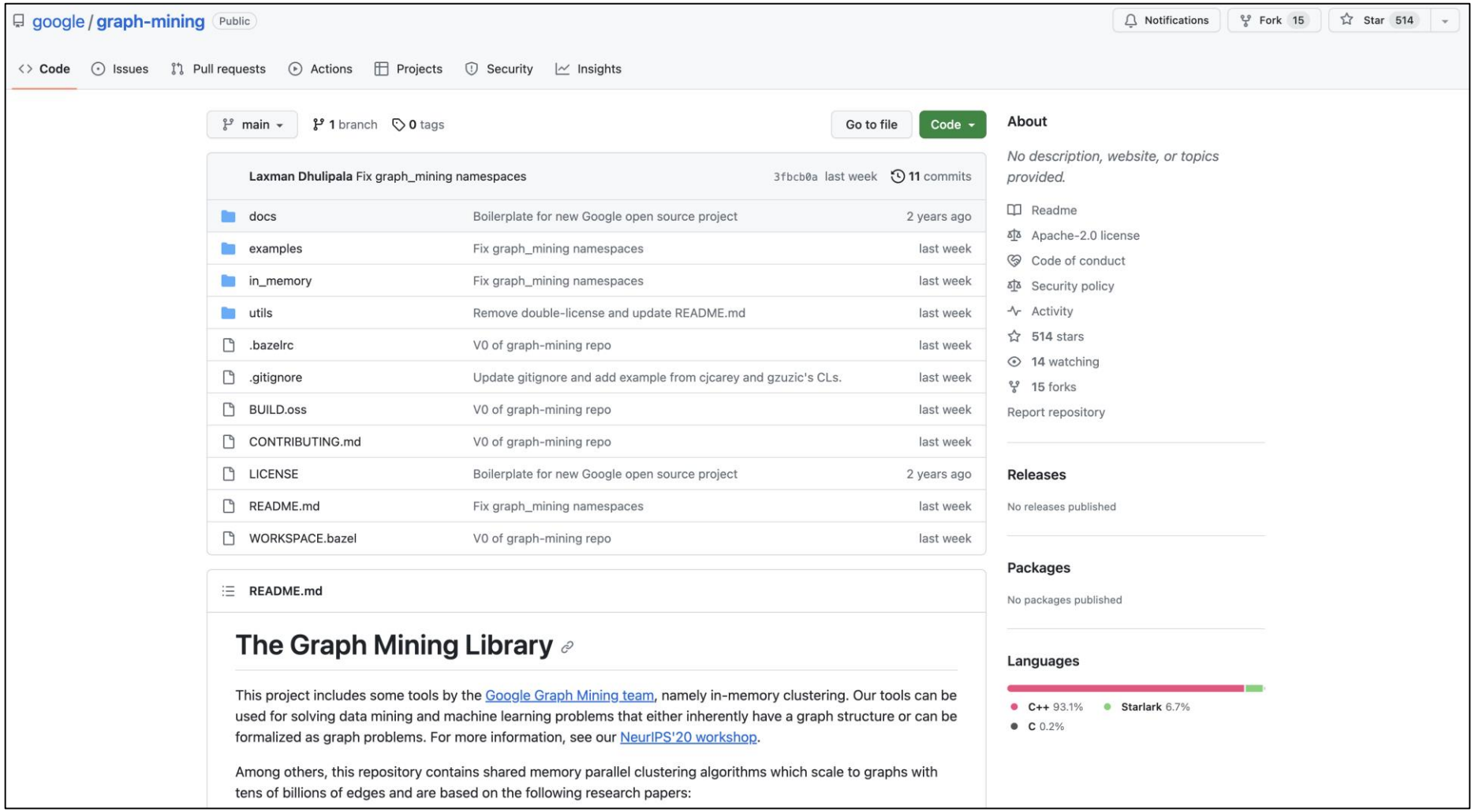
- **Fast Concurrent Data Structures → Fast Parallel Algorithms**



Goal

Design and Deploy **Fast and Scalable** Multicore (i.e., Concurrent) Data Structures

Google open source graph-mining library



https://github.com/google/graph-mining

Challenge

Asynchrony makes concurrent algorithm design notoriously hard

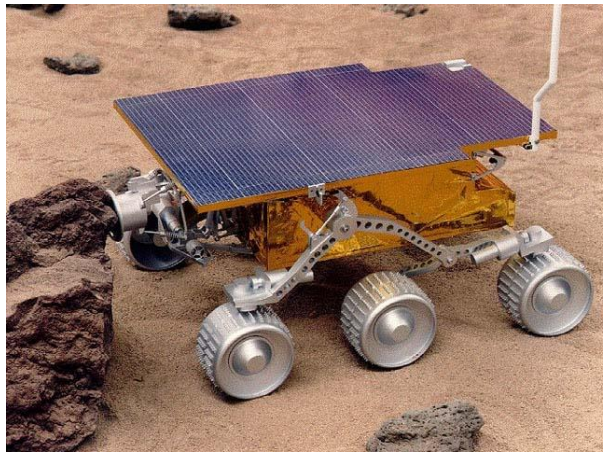
- p^t possible executions of length t
- Uncountably many infinite executions
- ALL possible executions must produce desired behavior

Subtle Race Conditions: some “hard to detect” executions goes wrong

Challenge

Asynchrony makes concurrent algorithms hard to design and prove correct

Subtle race conditions have plagued even mission-critical concurrent code



Mars Pathfinder Rover

Multi-million dollar space project jeopardized

Northeast Blackout of 2003

~45 Million people in the US affected by power cuts which lasted 8 hours

Therac-25

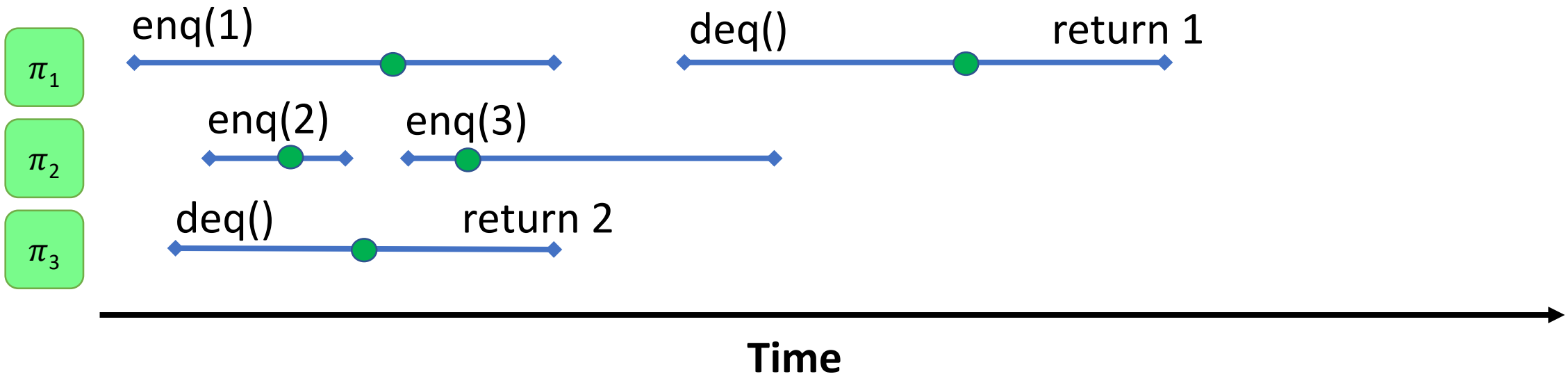
Patients given Radiation doses >100x desired amount; 3 deaths

Goal

Machine-verified proofs of correctness (i.e., **linearizability**) for concurrent data objects

Correctness = Linearizability

[Herlihy & Wing, 1990]

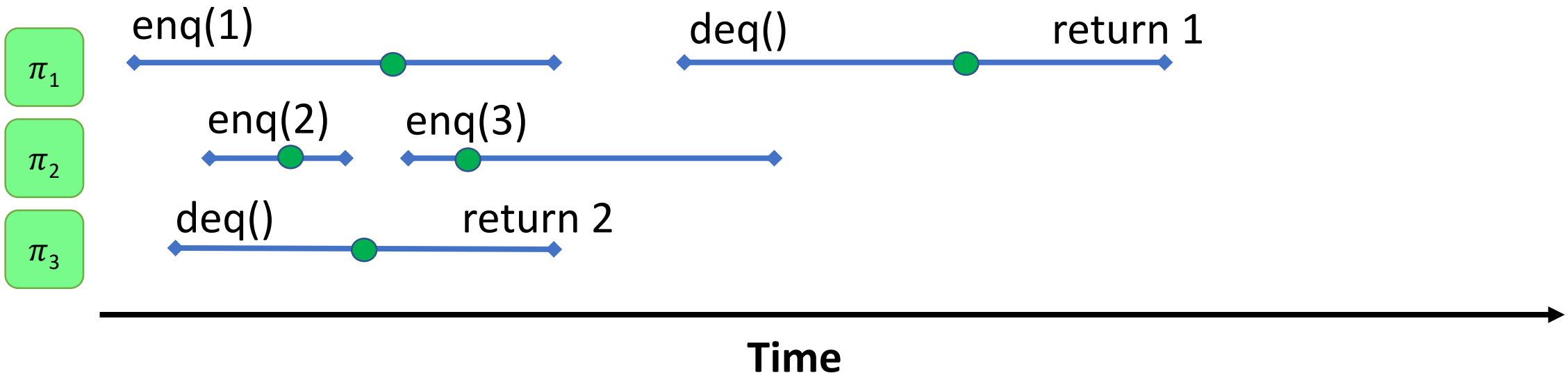


*Each operation **must appear to take effect atomically**, i.e.,*

instantaneously at some point between its invocation and completion

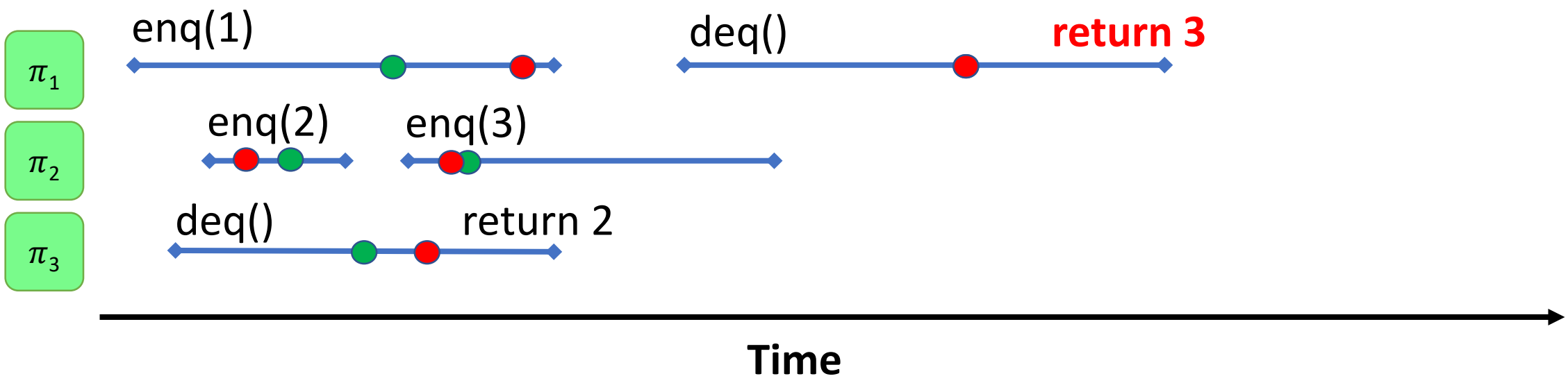
Forward Simulation: A Simple Proof Technique

[Jonsson, 1991]



- Maintain an atomic reference object
- Induct over arbitrary run
- Identify Linearization Points as they occur
- Show that return values of implemented object match those of the atomic object if operations taking place at linearization points

But what if…



- Is this linearizable?
- Actually, **it is linearizable**: just with different linearization points
- **Lessons:**
  - We can't always determine linearization points as they occur
  - Our choice of linearization points of past operations may depend on **future** outcomes

Future-Dependence

Some implementations are inherently future-dependent
(e.g., the Herlihy-Wing queue)

So, standard forward simulation is simple but not complete
(forward simulation can't be used to prove all data structures)

Is there a ***simple*** proof technique for linearizability, that is **complete** and can be used to obtain **machine-verified proofs**?

Previous Work

• Linearizability Introduced	Herlihy and Wing 1990
• Forward Simulation	Jonsson 1991
• Prophecy Variables	Vafeiadis 2008
• Aspect-Oriented Proofs	Henzinger et al. 2013
• Backward Simulation	Schellhorn et al. 2014
• Commitment Points	Bouajjani et al. 2017
• Partial-Order Maintenance	Khyzha et al. 2017
• Prophecy Variables in Separation Logic	Jung et al. 2019
• Category Theory Based	Olivera-Vale et al. 2023

Color key: Incomplete, Not Universal, Unmechanized, Not Simple

Our Contributions: Proof Technique

Meta-Configurations Tracking proof technique for Linearizability

- Universal: Objects of **any type** can be handled
- Complete: **Any algorithm** that is linearizable can be proved so

First such **forward reasoning** technique

Proofs can be Machine-Certified

Our Contributions: Strong Linearizability

We also introduce a related proof technique, called **Singleton Tracking**, which is a universal, sound, and complete forward-reasoning technique to prove Strong Linearizability (a close-cousin of linearizability).

Our Contributions: Machine-verified Proofs

- **Herlihy-Wing Queue**
  - Intricate future-dependent linearization structure
- **Jayanti Snapshot (single-writer, single-scanner)**
  - Efficient algorithm with future-dependent linearization
- **Jayanti-Tarjan Union-Find Objects**
  - Widely-used across domains & in **Google's graph-mining library**

Computation	Citation	#cores	Benchmark	**speed-up**
Spatial Clustering	[Wang et al]	36	*Sequential*	**5 – 33×**
Structural Clustering	[Tseng et al]	48	*Sequential*	**5 – 32×**
Model Checking	[Bloemen]	64	*Sequential*	**14.16 – 24.35 ×**
Connected Components	[Dhulipala et al]	72	Multicore	**1.5 – 4.02×**

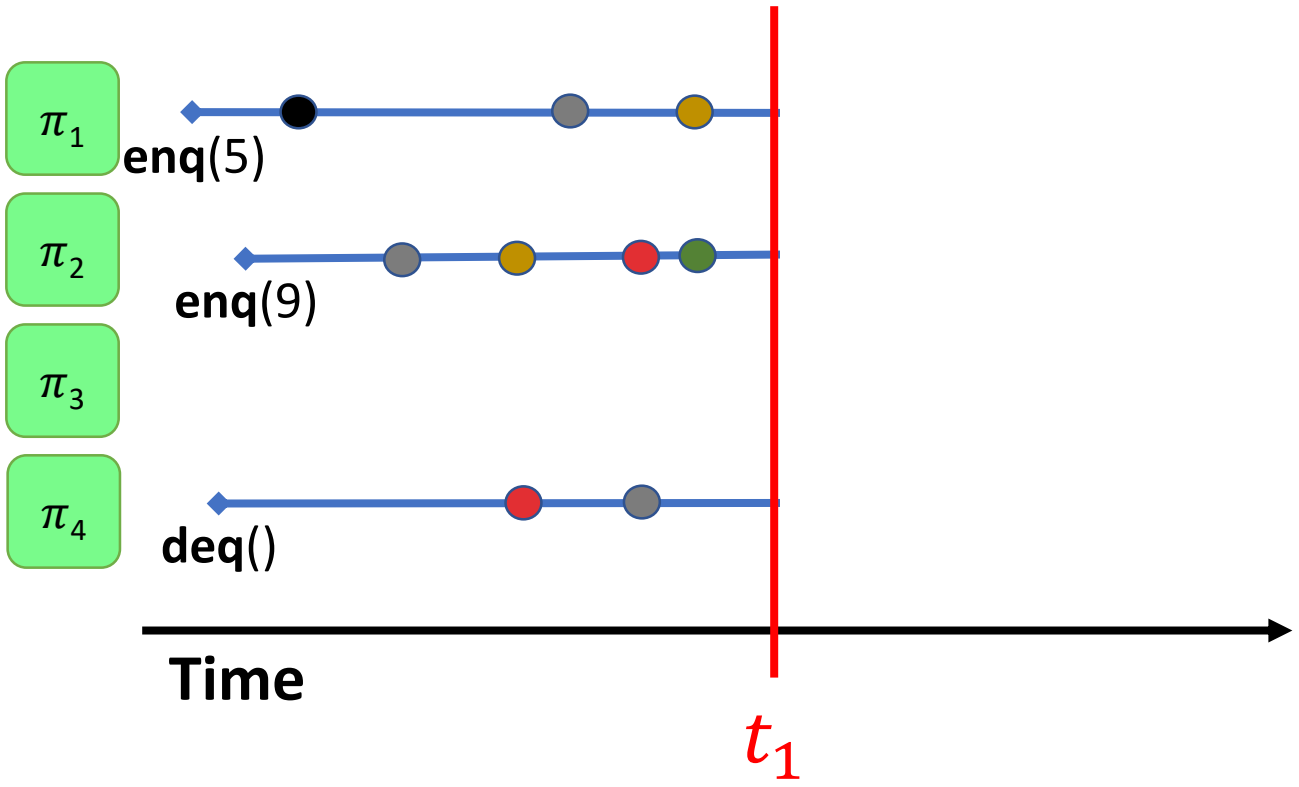
Our Idea

- Given original algorithm $\mathbf{A}$:
  - Design Augmentation $\mathbf{A}^*$ that *tracks* the set M of *all possible* atomic reference objects, corresponding to all possible linearizations

THEOREM

$\mathbf{A}$ is linearizable ***iff*** $M \neq \emptyset$ is invariant of $\mathbf{A}^*$

Meta-Configurations



State: ()
π_1: (enq, 5, –)
π_2: (enq, 9, –)
π_3: (–, –, –)
π_4: (deq, –, –)

State: (9, 5)
π_1: (enq, 5, ack)
π_2: (enq, 9, ack)
π_3: (–, –, –)
π_4: (deq, –, –)

State: (5)
π_1: (enq, 5, ack)
π_2: (enq, 9, –)
π_3: (–, –, –)
π_4: (deq, –, –)

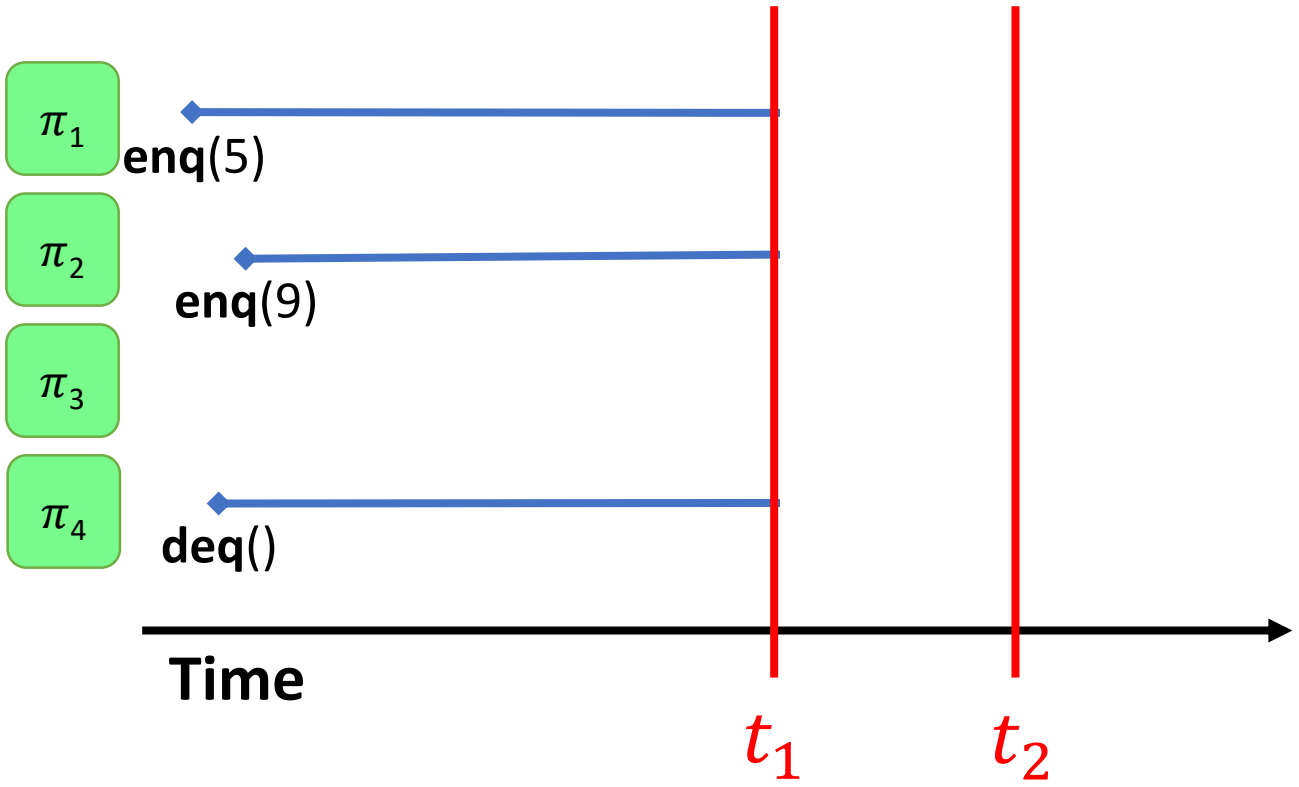
State: (5)
π_1: (enq, 5, ack)
π_2: (enq, 9, ack)
π_3: (–, –, –)
π_4: (deq, –, 9)

State: (9)
π_1: (enq, 5,–)
π_2: (enq, 9, ack)
π_3: (–, –, –)
π_4: (deq, –, –)

State: ()
π_1: (enq, 5, –)
π_2: (enq, 9, ack)
π_3: (–, –, –)
π_4: (deq, –, 9)

$$M = \{ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \bullet\bullet\bullet \}$$

Meta-Configurations



$$M = \{ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \blacksquare\ \bullet\bullet\bullet \}$$

State: ()
π_1: (enq, 5, –)
π_2: (enq, 9, –)
π_3: (–, –, –)
π_4: (deq, –, –)

State: (9, 5)
π_1: (enq, 5, ack)
π_2: (enq, 9, ack)
π_3: (–, –, –)
π_4: (deq, –, –)

State: (5)
π_1: (enq, 5, ack)
π_2: (enq, 9, –)
π_3: (–, –, –)
π_4: (deq, –, –)

State: (5)
π_1: (enq, 5, ack)
π_2: (enq, 9, ack)
π_3: (–, –, –)
π_4: (–, –, –)

State: (9)
π_1: (enq, 5,–)
π_2: (enq, 9, ack)
π_3: (–, –, –)
π_4: (deq, –, –)

State: ()
π_1: (enq, 5, –)
π_2: (enq, 9, ack)
π_3: (–, –, –)
π_4: (–, –, –)

How we track all meta-configurations

Initially M := {m_0}, where m_0 has the initial state and all processes are idle

Three Update Rules:

1. When π invokes op(arg)

   For each m ∈ M: update m.π from (–, –, –) to (op, arg, –)

2. When π returns ***r*** for op(arg)

   Filter out each m ∈ M if m.π's third field is not ***r***

   otherwise, update m.π to (–, –, –) since π is now idle

3. When π executes a step

   Evolve each m ∈ M to reflect that any set of pending operations can linearize in any order.

Meta-Configurations Tracking

Initialize

M := {m_0}, where m_0.state is initial state, $\forall \pi \in \Pi$, $m_0.\pi$ = (–, –, –)

Update Rules:

- When <u>π invokes op(arg)</u>

  M := {m′ | ∃m∈ M:m′.π = (op, arg, –) and otherwise m′ = m}

- When <u>π executes a step</u>

  M := {m′ | ∃m ∈ M, ∃S ⊆ *pending*(m), ∃α ∈ *perm*(S): m′ = α(m)}

- When <u>π returns ***r*** for op(arg)</u>

  M := {m′ | ∃m∈ M: m.π.res = ***r***, m′.π = (–, –, –), and o/w m′ = m}

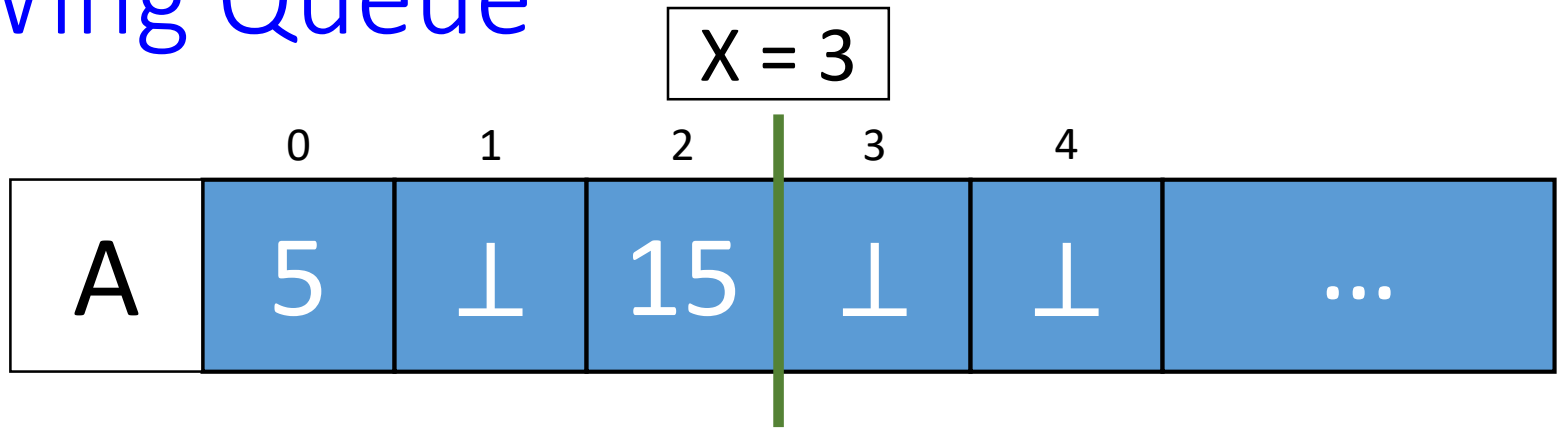
Meta-Configurations Tracking: Main Theorem

- For an algorithm $\mathbf{A}$, let $\mathbf{A}^*$ be the same algorithm augmented with a variable M that tracks ***all*** meta-configurations.

THEOREM

$\mathbf{A}$ is linearizable ***iff*** $M \neq \emptyset$ is invariant of $\mathbf{A}^*$

Herlihy-Wing Queue



Initially A[0,1,2,...] = [⊥, ⊥, ⊥,...] and X = 0

Enqueue(v):

// take next empty slot & place v into it.

s := X.F&Inc()

A[s] := v

Dequeue():

// read X; scan A[0,...,X-1] and return first found element; try again, if no element is found.

while (true)

    len := X

    for (s := 0; s < len; s++)

        r := A[s].swap(⊥)

        if r != ⊥ then return r

Herlihy-Wing Queue: Proof

Want to prove: $M \neq \{\}$ is an invariant of A*

- **Main Intellectual Work:** Identify structure of M.
- **Strengthen to Inductive Invariant**
- **Proof by Induction:** over the length of A*
- **Encode the proof in TLAPS** and let the machine verify**!**

Herlihy-Wing Queue: Proof

$$\mathcal{I} \equiv \mathcal{I}_L \wedge \mathcal{I}_X \wedge \mathcal{I}_Q \wedge \mathcal{I}_v \wedge \mathcal{I}_i \wedge \mathcal{I}_l \wedge \mathcal{I}_j \wedge \mathcal{I}_x \wedge \mathcal{I}_{pc} \wedge \mathcal{I}_{\mathcal{M}} \wedge \mathcal{I}_{1,5} \wedge \mathcal{I}_2 \wedge \mathcal{I}_3 \wedge \mathcal{I}_4 \wedge \mathcal{I}_6 \wedge \mathcal{I}_7 \wedge \mathcal{I}_8 \wedge \mathcal{I}_9 \wedge \mathcal{I}_a \wedge \mathcal{I}_b$$

In the above expression, the various conjuncts on the right hand side are defined below.

- $\mathcal{I}_L \equiv \mathcal{M} \neq \emptyset$
- $\mathcal{I}_X \equiv X \in \mathbb{N}^+$, $\mathcal{I}_Q \equiv \forall k \in \mathbb{N}^+ : Q[k] \in \mathbb{N}^+ \cup \{\perp\}$, $\mathcal{I}_v \equiv \forall \pi \in \Pi : v_\pi \in \mathbb{N}^+$, $\mathcal{I}_i \equiv \forall \pi \in \Pi : i_\pi \in \mathbb{N}^+$
  $\mathcal{I}_l \equiv \forall \pi \in \Pi : l_\pi \in \mathbb{N}^+$, $\mathcal{I}_j \equiv \forall \pi \in \Pi : j_\pi \in \mathbb{N}^+$, $\mathcal{I}_x \equiv \forall \pi \in \Pi : x_\pi \in \mathbb{N}^+ \cup \{\perp\}$, $\mathcal{I}_{pc} \equiv \forall \pi \in \Pi : pc_\pi \in [9]$
- $\mathcal{I}_{\mathcal{M}} \equiv \mathcal{M} \subseteq \{(\sigma, f) : \sigma \in \bigcup_{n \in N} (\mathbb{N}^+)^n, f \in (\{\textsc{Enqueue}, \textsc{Dequeue}\} \times (\mathbb{N}^+ \cup \{\perp\}) \times (\mathbb{N}^+ \cup \{ack, \perp\}))^\Pi\}$
- $\mathcal{I}_{1,5} \equiv \forall \pi \in \Pi : \forall (\sigma, f) \in \mathcal{M} : pc_\pi \in \{1, 5\} \implies f(\pi) = (\perp, \perp, \perp)$
- $\mathcal{I}_2 \equiv \forall \pi \in \Pi : \forall (\sigma, f) \in \mathcal{M} : pc_\pi = 2 \implies f(\pi) = (\textsc{Enqueue}, v_\pi, \perp)$
- $\mathcal{I}_3 \equiv \forall \pi \in \Pi : \forall (\sigma, f) \in \mathcal{M} : pc_\pi = 3 \implies f(\pi) \in \{\textsc{Enqueue}\} \times \{v_\pi\} \times \{ack, \perp\} \wedge (1 \leq i_\pi < X) \wedge (Q[i_\pi] = \perp) \wedge (\forall \pi' \in \Pi - \{\pi\} : pc_{\pi'} \in \{3, 4\} \implies i_{\pi'} \neq i_\pi)$
- $\mathcal{I}_4 \equiv \forall \pi \in \Pi : \forall (\sigma, f) \in \mathcal{M} : pc_\pi = 4 \implies f(\pi) \in \{\textsc{Enqueue}\} \times \{v_\pi\} \times \{ack, \perp\} \wedge (1 \leq i_\pi < X) \wedge (\forall \pi' \in \Pi - \{\pi\} : pc_{\pi'} \in \{3, 4\} \implies i_{\pi'} \neq i_\pi)$
- $\mathcal{I}_6 \equiv \forall \pi \in \Pi : \forall (\sigma, f) \in \mathcal{M} : pc_\pi = 6 \implies f(\pi) = (\textsc{Dequeue}, \perp, \perp)$
- $\mathcal{I}_7 \equiv \forall \pi \in \Pi : \forall (\sigma, f) \in \mathcal{M} : pc_\pi = 7 \implies f(\pi) = (\textsc{Dequeue}, \perp, \perp) \wedge 1 \leq l_\pi \leq X$
- $\mathcal{I}_8 \equiv \forall \pi \in \Pi : \forall (\sigma, f) \in \mathcal{M} : pc_\pi = 8 \implies f(\pi) = (\textsc{Dequeue}, \perp, \perp) \wedge 1 \leq j_\pi < l_\pi \leq X$
- $\mathcal{I}_9 \equiv \forall \pi \in \Pi : \forall (\sigma, f) \in \mathcal{M} : pc_\pi = 9 \implies f(\pi) \in \{\textsc{Dequeue}\} \times \{\perp\} \times (\mathbb{N}^+ \cup \{\perp\})$
- $\mathcal{I}_a \equiv \forall k \in \mathbb{N}^+ : k > X - 1 \implies Q[k] = \perp$
- $\mathcal{I}_b \equiv \forall A \subseteq [X - 1] : GoodEnqSet(A) \implies (\forall s \in Perm(A) : (JInvSeq(s) \implies (\exists C \in \mathcal{M} : GoodRes(A, C) \wedge ValuesMatchInds(s, C))))$

- $GoodEnqSet(A) \triangleq \forall k \in [X - 1] : (Q[k] \neq \perp \implies k \in A) \wedge ((Q[k] = \perp \wedge k \in A) \implies \exists \pi' \in \Pi : pc_{\pi'} = 3 \wedge i_{\pi'} = k)$
- $JInvSeq(s) \triangleq \forall m, n \in [|s|] : (n < m \wedge s_m < s_n \wedge Q[s_m] \neq \perp) \implies (\exists \pi' \in \Pi : pc_{\pi'} = 8 \wedge s_n < l_{\pi'} \wedge s_m < j_{\pi'})$
- $GoodRes(A, C) \triangleq \forall \pi' \in \Pi : C.f(\pi').res = \begin{cases} ack & \text{if } (pc_{\pi'} = 3 \wedge i_{\pi'} \in A) \vee pc_{\pi'} = 4 \\ x_{\pi'} & \text{if } pc_{\pi'} = 9 \\ \perp & \text{otherwise.} \end{cases}$
- $ValuesMatchInds(s, C) \triangleq C.\sigma = (\alpha_1, \alpha_2, \ldots, \alpha_{|s|})$ where $\forall k \in [|s|]$

$$\alpha_k = \begin{cases} Q[s_k] & \text{if } Q[s_k] \neq \perp \\ v_{\pi'} & \text{where } \pi' \in \Pi : pc_{\pi'} = 3 \wedge i_{\pi'} = s_k. \end{cases}$$

Fig. 5. Invariant $\mathcal{I}$ of $\mathcal{A}(\overline{O})$, where $\overline{O}$ is the implementation of the queue tracker in Figure 4.

Herlihy-Wing Queue Machine Verification in TLAPS

```
\* Proof of full inductive invariant
THEOREM InductiveInvariant == Spec => []Inv
  BY InductiveInvariantNL, LinearizabilityFromInvNL, PTL DEF Inv, InvNL

\* Proof of linearizability
THEOREM Linearizability == Spec => [](M # {})
  BY InductiveInvariant, PTL DEF Inv, Linearizable
```

```
THEOREM InductiveInvariantNL == Spec => []InvNL
  <1> USE AckBotDef DEF InvNL, PosInts
  <1> SUFFICES /\ (Init => InvNL)
               /\ (InvNL /\ [Next]_vars => InvNL')
    BY PTL DEF Spec
  <1>1. Init => InvNL
    <2> SUFFICES ASSUME Init
                 PROVE  InvNL
      OBVIOUS
    <2> USE DEF Init
    <2>1. TypeOK
      BY InitTypeSafety
    <2>2. Inv_E2
      BY DEF Inv_E2
    <2>3. Inv_E3
      BY DEF Inv_E3
    <2>4. Inv_D2
      BY DEF Inv_D2
    <2>5. Inv_D3
      BY DEF Inv_D3
    <2>6. Inv_Q
      BY DEF Inv_Q
    <2>7. Inv_Main
      <3> SUFFICES ASSUME NEW A \in SUBSET (1..(X-1)),
                          GoodEnqSet(A),
                          NEW seq \in Perm(A),
                          JInvSeq(seq)
                   PROVE  \E c \in M : /\ ValuesMatchInds(seq, c.sigma)
                                       /\ GoodRes(A, c.fres)
        BY Zenon DEF Inv_Main
      <3>1. A = {}
        BY DEF GoodEnqSet
      <3>2. Cardinality(A) = 0
        BY <3>1, Zenon DEF Cardinality
      <3>3. seq = << >>
        BY <3>1, <3>2 DEF Perm, JInvSeq
      <3> QED
        BY <3>1, <3>3 DEF ValuesMatchInds, GoodRes
    <2> QED
      BY <2>1, <2>2, <2>3, <2>4, <2>5, <2>6, <2>7 DEF InvNL
  <1>2. InvNL /\ [Next]_vars => InvNL'
    <2> SUFFICES ASSUME InvNL /\ [Next]_vars
```

Conclusion

- Machine-verification of concurrent data structures can help avoid errors in mission critical deployments.

- We introduced **Meta-Configurations Tracking**: the first universal, sound, and complete, forward-reasoning proof technique for linearizability
  - Easy to express proofs of even complex future-dependent algorithms (e.g., Herlihy-Wing queue, Jayanti's Snapshot)
  - We have verified widely used concurrent objects (e.g., Jayanti-Tarjan Union-Find objects)
  - Proved using TLA+ Proof System

- Ongoing work: We are expanding the technique to variants of Linearizability

- We would love to see these techniques used to machine-certify more important and interesting concurrent algorithms

- We'd love to see support for Meta-Configurations Tracking in verification tools

Links

github.com/google/graph-mining

[POPL 24] Jayanti, Jayanti, Yavuz, Hernandez

Meta-Configuration Notation