Reducing Linearizability to Classic Verification Problems
Checking Lin. using “bad patterns”

• Reduce linearizability checking to reachability (EXPSPACE-complete):
  • Define (sequential) data-structure $S$ using inductive rules
  • $S$ is data independent and closed under projection
  • Characterize sequential executions of $S$ using bad patterns
  • Characterize concurrent executions, linearizable w.r.t. $S$ using bad patterns (one per rule)
  • Define a regular automaton $A_i$ for each bad pattern
  • Reduce “L is linearizable w.r.t. $S$” to: for all $i$, $L \cap A_i = \emptyset$
Histories = Posets of events

Thread 1
- push(1)
- pop ⇒ 2

Thread 2
- pop ⇒ 1
- push(2)
- push(3)
- pop ⇒ 3

happens-before partial order
Concurrent Queues

Value v dequeued without being enqueued
\[
\text{deq} \Rightarrow v
\]

Value v dequeued before being enqueued
\[
\text{deq} \Rightarrow v \quad \text{enq}(v)
\]

Value v dequeued twice
\[
\text{deq} \Rightarrow v \quad \text{deq} \Rightarrow v
\]

Value \(v_1\) and \(v_2\) dequeued in the wrong order
\[
\text{enq}(v_1) \quad \text{enq}(v_2) \quad \text{deq} \Rightarrow v_2 \quad \text{deq} \Rightarrow v_1
\]

Dequeue wrongfully returns empty
\[
\text{deq} \Rightarrow \text{empty}
\]

\[
\text{enq}(v_1) \quad \text{deq} \Rightarrow v_1
\]

\[
\text{enq}(v_2) \quad \text{deq} \Rightarrow v_2
\]

\[
\text{enq}(v_n) \quad \text{deq} \Rightarrow v_{n-1}
\]

\[
\text{deq} \Rightarrow v_n
\]
Concurrent Stacks

Value v popped without being pushed
Value v popped twice
Value v popped before being pushed
Pop wrongfully returns empty

Pop doesn’t return the top of the stack

push(v) \rightarrow pop \Rightarrow v

push(v_1) \rightarrow pop \Rightarrow v_1

push(v_2) \rightarrow pop \Rightarrow v_2

\ldots \ldots \ldots

push(v_{n-1}) \rightarrow pop \Rightarrow v_{n-1}

push(v_n) \rightarrow pop \Rightarrow v_n
Checking Lin. using “bad patterns”

- Reduce linearizability checking to reachability (EXPSPACE-complete):
  - Define (sequential) data-structure S using inductive rules

- S is data independent and closed under projection

- Characterize sequential executions of S using bad patterns

- Characterize concurrent executions, linearizable w.r.t. S using bad patterns (one per rule)

- Define a regular automaton $A_i$ for each bad pattern

- Reduce “L is linearizable w.r.t. S” to: for all i, $L \cap A_i = \emptyset$
Inductive definition of the Register

\[ R_{wr} : \ u \in R \implies Write_x \cdot (Read_x)^* \cdot u \in R \]
- including the empty sequence

Examples
Inductive definition of the Queue

Two rules to build the sequences belonging to the Queue such as

\[ Enq_4 Enq_3 Deq_4 Deq_3 EMP Enq_2 Enq_1 Deq_2 Deq_1 \in Q \]

- \( R_{Enq} : \quad u \in Q \land u \in Enq^* \Rightarrow u \cdot Enq_x \in Q \)
- \( R_{EnqDeq} : \quad u \cdot v \in Q \land u \in Enq^* \Rightarrow Enq_x \cdot u \cdot Deq_x \cdot v \in Q \)
- \( R_{EMP} : \quad u \cdot v \in Q \land \text{no unmatched Enq in } u \Rightarrow u \cdot EMP \cdot v \in Q \)

Derivation:

\[ \epsilon \in Q \]
\[ \rightarrow \quad Enq_1 Deq_1 \in Q \]
\[ \rightarrow \quad Enq_2 Enq_1 Deq_2 Deq_1 \in Q \]
\[ \rightarrow \quad Enq_3 Deq_3 Enq_2 Enq_1 Deq_2 Deq_1 \in Q \]
\[ \rightarrow \quad Enq_4 Enq_3 Deq_4 Deq_3 Enq_2 Enq_1 Deq_2 Deq_1 \in Q \]
\[ \rightarrow \quad Enq_4 Enq_3 Deq_4 Deq_3 EMP Enq_2 Enq_1 Deq_2 Deq_1 \in Q \]
Inductive definition of the Stack

\[ R_{PushPop} : u \cdot v \in S \land \text{no unmatched } Push \text{ in } u, v \Rightarrow Push_x \cdot u \cdot Pop_x \cdot v \in S \]

\[ R_{Push} : u \cdot v \in S \land \text{no unmatched } Push \text{ in } u \Rightarrow u \cdot Push_x \cdot v \in S \]

\[ R_{EMP} : u \cdot v \in S \land \text{no unmatched } Push \text{ in } u \Rightarrow u \cdot EMP \cdot v \in S \]

Derivation for  \( Push_1 Push_2 Pop_2 Pop_1 EMP Push_3 Pop_3 \in S \)

\[ \epsilon \in S \]
\[ \rightarrow \text{ Push}_3 \text{ Pop}_3 \in S \]
\[ \rightarrow \text{ Push}_2 \text{ Pop}_2 \text{ Push}_3 \text{ Pop}_3 \in S \]
\[ \rightarrow \text{ Push}_1 \text{ Push}_2 \text{ Pop}_2 \text{ Pop}_1 \text{ Push}_3 \text{ Pop}_3 \in S \]
\[ \rightarrow \text{ Push}_1 \text{ Push}_2 \text{ Pop}_2 \text{ Pop}_1 \text{ EMP } \text{ Push}_3 \text{ Pop}_3 \in S \]
Data Independence

• Input methods = methods taking an argument
• A sequential execution $u$ is called *differentiated* if for all input methods $m$ and every $x$, $u$ contains at most one invocation $m(x)$

$S_\neq$ is the set of differentiated executions in $S$

A *renaming* $r$ is a function from $\mathbb{D}$ to $\mathbb{D}$. Given a sequential execution (resp., execution or history) $u$, we denote by $r(u)$ the sequential execution (resp., execution or history) obtained from $u$ by replacing every data value $x$ by $r(x)$.

**Definition 6.** The set of sequential executions (resp., executions or histories) $S$ is data independent if:

- for all $u \in S$, there exists $u' \in S_\neq$, and a renaming $r$ such that $u = r(u')$,
- for all $u \in S$ and for all renaming $r$, $r(u) \in S$.

**Theorem:** A data-independent implementation $I$ is linearizable w.r.t. a data-independent specification $S$ iff $I_\neq$ is linearizable w.r.t. $S_\neq$
Closure under projection

**Projection:** Subsequence consistent with the values

If

\[ Enq_4 Enq_3 Deq_4 Deq_3 Enq_2 Enq_1 Deq_2 Deq_1 \in Q \]

Then

\[ Enq_4 Deq_4 Enq_2 Enq_1 Deq_2 Deq_1 \in Q \]

---

**Lemma**

Any **data structure** defined in our framework is **closed under projection**

---

**Proof.**

The **predicates** used (\( u \in Enq^* \) and “no unmatched \( Enq \) in \( u \)” ) are closed under projection
Characterization of sequential executions

We assume that the rules defining a data-structure are well-formed, that is:

- for all \( u \in [S] \), there exists a unique rule, denoted by \( \text{last}(u) \), that can be used as the last step to derive \( u \), i.e., for every sequence of rules \( R_{i_1}, \ldots, R_{i_n} \) leading to \( u \), \( R_{i_n} = \text{last}(u) \). For \( u \notin [S] \), \( \text{last}(u) \) is also defined but can be arbitrary, as there is no derivation for \( u \).

- if \( \text{last}(u) = R_i \), then for every permutation \( u' \in [S] \) of a projection of \( u \), \( \text{last}(u') = R_j \) with \( j \leq i \). If \( u' \) is a permutation of \( u \), then \( \text{last}(u') = R_i \).

Example 6. For Queue, we define \( \text{last} \) for a sequential execution \( u \) as follows:

- if \( u \) contains a DeqEmpty operation, \( \text{last}(u) = R_{\text{DeqEmpty}} \),
- else if \( u \) contains a Deq operation, \( \text{last}(u) = R_{\text{EnqDeq}} \),
- else if \( u \) contains only Enq’s, \( \text{last}(u) = R_{\text{Enq}} \),
- else (if \( u \) is empty), \( \text{last}(u) = R_0 \).

Since the conditions we use to define \( \text{last} \) are closed under permutations, we get that for any permutation \( u_2 \) of \( u \), \( \text{last}(u) = \text{last}(u_2) \), and \( \text{last} \) can be extended to histories. Therefore, the rules \( R_0, R_{\text{EnqDeq}}, R_{\text{DeqEmpty}} \) are well-formed.
Characterization of sequential executions

- \( \text{MS}(R) = \) the set of sequences “matching” a rule \( R \)

**Lemma 3.** Let \( S = R_1, \ldots, R_n \) be a data-structure and \( u \) be a differentiated sequential execution. Then,

\[
 u \in S \iff \text{proj}(u) \subseteq \bigcup_{i \in \{1, \ldots, n\}} \text{MS}(R_i)
\]

**Lemma (Characterization of Queue Sequential Executions)**

\( w \in Q \) iff every projection \( w' \) of \( w \) is either of the form

- \( \text{Enq}_x \cdot u \cdot \text{Deq}_x \cdot v \) (with \( u \in \text{Enq}^* \)) or
- \( u \cdot \text{EMP} \cdot v \) (with no unmatched \( \text{Enq} \) in \( u \))
Characterization of concurrent executions

**Definition 7.** A data-structure \( S = R_1, \ldots, R_n \) is said to be step-by-step linearizable if for any differentiated execution \( e \), any \( i \in \{1, \ldots, n\} \) and \( x \in \mathbb{D} \), if \( e \) is linearizable with respect to \( \text{MS}(R_i) \) with witness \( x \), we have:

\[
e \setminus x \subseteq [R_1, \ldots, R_i] \implies e \subseteq [R_1, \ldots, R_i]
\]

- the history linearizable \( \text{MS}(R_{\text{EnqDeq}}) \) with witness \( d_1 \)
  - \( \text{Enq}(d_1) \) is minimal among all operations and \( \text{Deq}(d_1) \) minimal among all dequeue

- Excluding the operations on \( d_1 \), the history is linearizable w.r.t. \( [R_{\text{Enq}}, R_{\text{EnqDeq}}] \), i.e., \( \text{Enq}(d_2) \) \( \text{Enq}(d_3) \) \( \text{Deq}(d_2) \) \( \text{Deq}(d_3) \)

- The notion of step-by-step linearizable ensures that the history is linearizable w.r.t. Queue
Step-by-Step Lin. of Register

Lemma 9. Register is step-by-step linearizable.

Proof. Let $h$ be a differentiated history, and $u$ a sequential execution such that $h \subseteq u$ and such that $u$ matches the rule $R_{WR}$ with witness $x$. Let $a$ and $b_1, \ldots, b_s$ be respectively the Write and Read’s operations of $h$ corresponding to the witness.

Let $h' = h \setminus x$ and assume $h' \subseteq [R_0,R_{WR}]$. Let $u' \in [R_0,R_{WR}]$ such that $h' \subseteq u'$. Let $u_2 = a \cdot b_1 \cdot b_2 \cdots b_s \cdot u'$. By using rule $R_{WR}$ on $u'$, we have $u_2 \in [R_0,R_{WR}]$. Moreover, we prove that $h \subseteq u_2$ by contradiction. Assume that the total order imposed by $u_2$ doesn’t respect the happens-before relation of $h$. All three cases are not possible:

- the violation is between two $u'$ operations, contradicting $h' \subseteq u'$,
- the violation is between $a$ and another operation, i.e. there is an operation $o$ which happens before $a$ in $h$, contradicting $h \subseteq u$,
- the violation is between some $b_i$ and a $u'$ operation, i.e. there is an operation $o$ which happens before $b_i$ in $h$, contradicting $h \subseteq u$.

Thus, we have $h \subseteq u_2$ and $h \subseteq [R_0,R_{WR}]$, which ends the proof. □
Characterization of concurrent executions

Lemma 4. Let \( S \) be a data-structure with rules \( R_1, \ldots, R_n \). Let \( e \) be a differentiated execution. If \( S \) is step-by-step linearizable, we have (for any \( j \)):

\[
e \subseteq [R_1, \ldots, R_j] \iff \text{proj}(e) \subseteq \bigcup_{i \leq j} \text{MS}(R_i)
\]

Proof \((\iff)\) By induction on the size of \( e \). We know \( e \in \text{proj}(e) \) so it can be linearized with respect to a sequential execution \( u \) matching some rule \( R_k \) \((k \leq j)\) with some witness \( x \). Let \( e' = e \setminus x \).

Since \( S \) is well-formed, we know that no projection of \( e \) can be linearized to a matching set \( \text{MS}(R_i) \) with \( i > k \), and in particular no projection of \( e' \). Thus, we deduce that \( \text{proj}(e') \subseteq \bigcup_{i \leq k} \text{MS}(R_i) \), and conclude by induction that \( e' \subseteq [R_1, \ldots, R_k] \).

We finally use the fact that \( S \) is step-by-step linearizable to deduce that \( e \subseteq [R_1, \ldots, R_k] \) and \( e \subseteq [R_1, \ldots, R_j] \) because \( k \leq j \).

Lemma

\( E \) is linearizable to \( Q \) iff every projection \( E' \) of \( E \) is linearizable to the form \( \text{Enq}_x \cdot u \cdot \text{Deq}_x \cdot \nu \) (with \( u \in \text{Enq}^* \)) or to the form \( u \cdot \text{EMP} \cdot \nu \) (with no unmatched \( \text{Enq} \) in \( u \)).
Characterization of concurrent executions

Lemma 5. Let $S$ be a data-structure with rules $R_1, \ldots, R_n$. Let $e$ be a differentiated execution. If $S$ is step-by-step linearizable, we have:

$$e \in S \iff \forall e' \in \text{proj}(e). e' \in \text{MS}(R) \text{ where } R = \text{last}(e')$$

$$e \notin S \iff \exists e' \in \text{proj}(e). e' \notin \text{MS}(R) \text{ (where } R = \text{last}(e'))$$

**E is non-linearizable wrt Queue iff it has a projection $E'$ of the form bad pattern 1, or bad pattern 2.**

Bad Pattern 1 (rule $R_{\text{EnqDeq}}$):

- $\text{Enq}_1 \prec \text{Enq}_2$
- $\text{Deq}_2 \prec \text{Deq}_1$

or $\text{Deq}_1$ before $\text{Enq}_1$
Characterization of concurrent executions

**Lemma 5.** Let $S$ be a data-structure with rules $R_1, \ldots, R_n$. Let $e$ be a differentiated execution. If $S$ is step-by-step linearizable, we have:

$$e \in S \iff \forall e' \in \text{proj}(e). e' \in \text{MS}(R) \text{ where } R = \text{last}(e')$$

$$e \notin S \iff \exists e' \in \text{proj}(e). e' \notin \text{MS}(R) \text{ (where } R = \text{last}(e'))$$

*E is non-linearizable wrt Queue iff it has a projection $E'$ of the form bad pattern 1, or bad pattern 2.*

Bad Pattern 2: (rule $R_{EMP}$)

```
  Enq₁  EMP  Deq₁
  /     /     /
Enq₁  Enq₁  Deq₁
    /      /
  Enq₁  Deq₁
```
Characterization of concurrent executions

- define for each R, a finite state automaton A which recognizes (a subset of) the executions e which have a projection not linearizable w.r.t. MS(R)

**Definition 8.** A rule R is said to be co-regular if we can build an automaton A such that, for any data-independent implementation I, we have:

\[ \mathcal{I} \cap A \neq \emptyset \iff \exists e \in \mathcal{I}_+, e' \in \text{proj}(e). \text{last}(e') = R \land e' \notin \text{MS}(R) \]

\[ R_{\text{EnqDeq}} \]

\[ R_{\text{EnqDeq}} \]

\[ R_{\text{EnqDeq}} \]
Characterization of concurrent executions

- define for each $R$, a finite state automaton $A$ which recognizes (a subset of) the executions $e$ which have a projection not linearizable w.r.t. $MS(R)$

**Definition 8.** A rule $R$ is said to be co-regular if we can build an automaton $A$ such that, for any data-independent implementation $I$, we have:

$$I \cap A \neq \emptyset \iff \exists e \in I, e' \in \text{proj}(e). \text{last}(e') = R \wedge e' \notin MS(R)$$

**REM**

we assume that all actions call Enq(1) occur at the beginning