भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Proving Correctness of Concurrent Objects by Validating Linearization Points

Nandini Singhal, Muktikanta Sa, Ajay Singh,
Archit Somani, Sathya Peri

Department of Computer Science Engineering, IIT Hyderabad

Outline

1. Introduction
   - Sequential Object and Sequential Specification
   - Sequential vs Concurrent History
2. Linearizability
3. Proposed Technique: Validating LPs
4. Conclusion
5. Future Work

Outline

1. Introduction
   - Sequential Object and Sequential Specification
   - Sequential vs Concurrent History
2. Linearizability
3. Proposed Technique: Validating LPs
4. Conclusion
5. Future Work

Introduction

Sequential Object and Sequential Specification

Sequential Object

- Each object has a *state*.
    - Ex: sequence of item in a queue.

Sequential Object

- Each object has a *state*.
    - Ex: sequence of item in a queue.
- Each object has a set of *methods* which can manipulate its state.
    - Ex: *enq* and *deq* methods.

Sequential Object

- Each object has a *state*.
  - Ex: sequence of item in a queue.
- Each object has a set of *methods* which can manipulate its state.
  - Ex: *enq* and *deq* methods.

Sequential Specification

- Set of correct histories which can be generated by single threaded execution.

Introduction

Sequential Object and Sequential Specification

Sequential Object

- Each object has a *state*.
    - Ex: sequence of item in a queue.
- Each object has a set of *methods* which can manipulate its state.
    - Ex: *enq* and *deq* methods.

Sequential Specification

- Set of correct histories which can be generated by single threaded execution.
- Pre-condition : state before you call the method.

Sequential Object

- Each object has a *state*.
    - Ex: sequence of item in a queue.
- Each object has a set of *methods* which can manipulate its state.
    - Ex: *enq* and *deq* methods.

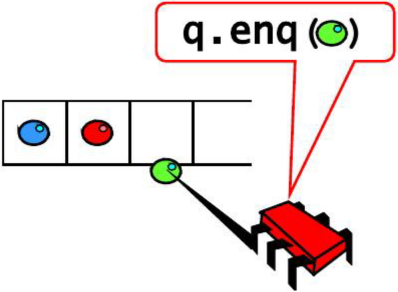
Sequential Specification

- Set of correct histories which can be generated by single threaded execution.
- Pre-condition : state before you call the method.
- Post-condition : other state after the method returns.

Introduction

Example of Sequential Specification

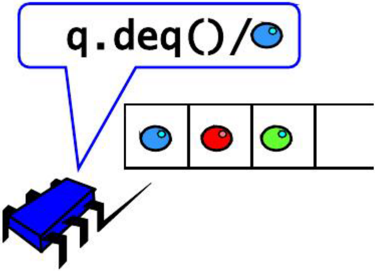
FIFO Queue: Enqueue Method



Introduction

Example of Sequential Specification

FIFO Queue: Dequeue Method



Outline

1. Introduction
   - Sequential Object and Sequential Specification
   - Sequential vs Concurrent History
2. Linearizability
3. Proposed Technique: Validating LPs
4. Conclusion
5. Future Work

Sequential vs Concurrent History

Sequential History

- Object methods are invoked one at a time by a single process.

Sequential vs Concurrent History

Sequential History

- Object methods are invoked one at a time by a single process.
- The meaning of methods can be given by pre- and post- conditions.

Sequential vs Concurrent History

Sequential History

- Object methods are invoked one at a time by a single process.
- The meaning of methods can be given by pre- and post- conditions.

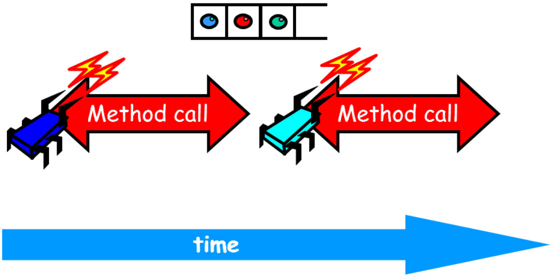


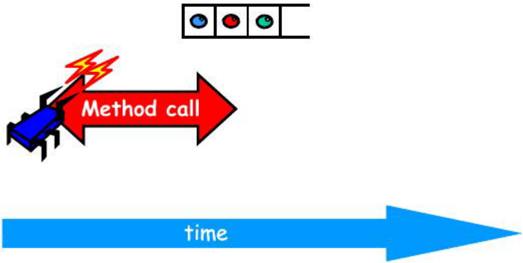
Figure: Sequential History

Sequential vs Concurrent History

Concurrent History

- Object methods can be invoked by concurrent processes.
- It is necessary to give a meaning to possible interleavings of operations invocation.

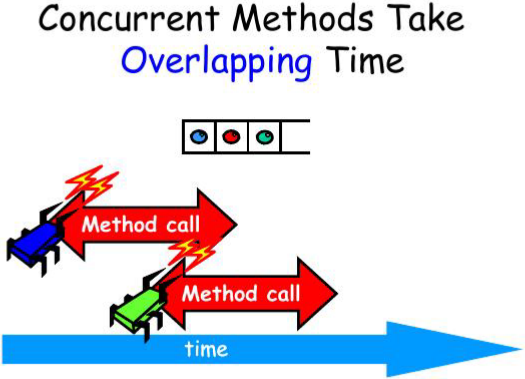
Concurrent Methods Take Overlapping Time



Sequential vs Concurrent History

Concurrent History

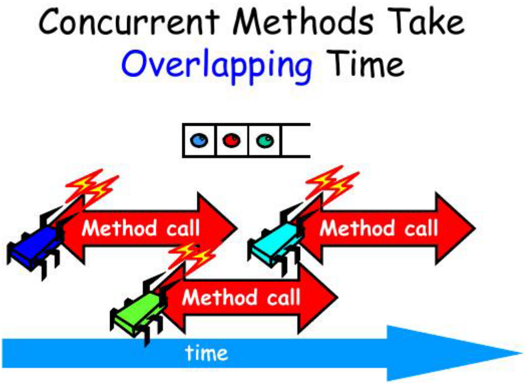
- Object methods can be invoked by concurrent processes.
- It is necessary to give a meaning to possible interleavings of operations invocation.



Sequential vs Concurrent History

Concurrent History

- Object methods can be invoked by concurrent processes.
- It is necessary to give a meaning to possible interleavings of operations invocation.



Panic!

- Because method calls overlap, must characterize *all* possible interactions with concurrent calls.

Panic!

- Because method calls overlap, must characterize *all* possible interactions with concurrent calls.
- Everything can potentially interact with everything else.

Panic!

- Because method calls overlap, must characterize *all* possible interactions with concurrent calls.
- Everything can potentially interact with everything else.

What does it mean for a concurrent object to be correct?

Panic!

- Because method calls overlap, must characterize *all* possible interactions with concurrent calls.
- Everything can potentially interact with everything else.

What does it mean for a concurrent object to be correct?

Correctness Criteria: **Linearizability**.

Outline

1. Introduction
   - Sequential Object and Sequential Specification
   - Sequential vs Concurrent History
2. Linearizability
3. Proposed Technique: Validating LPs
4. Conclusion
5. Future Work

Linearizability

Linearizability

- Each method should
  - Take effect
  - Instantaneously
  - Between invocation and response events.

Linearizability

Linearizability

- Each method should
  - Take effect
  - Instantaneously
  - Between invocation and response events.

- Object is correct if it adheres to it's sequential specification.

Linearizability

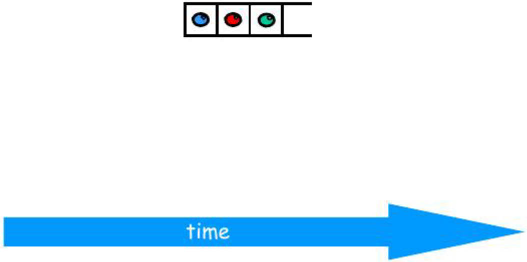
Linearizability

- Each method should
  - Take effect
  - Instantaneously
  - Between invocation and response events.

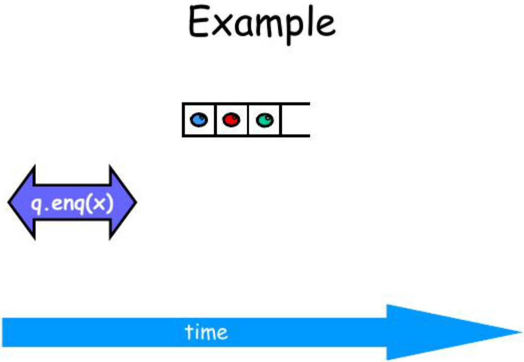
- Object is correct if it adheres to it's sequential specification.
- Any such concurrent object is *Linearizable*.

Linearizability

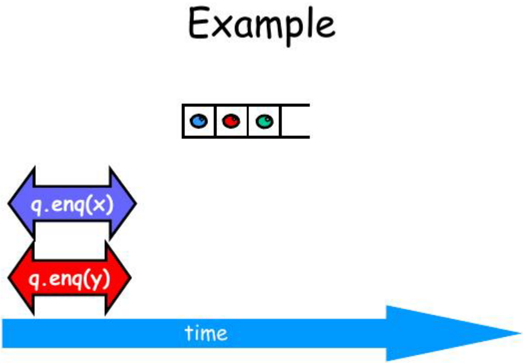
Example



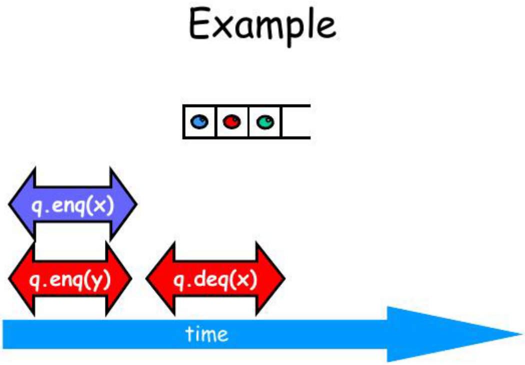
Linearizability



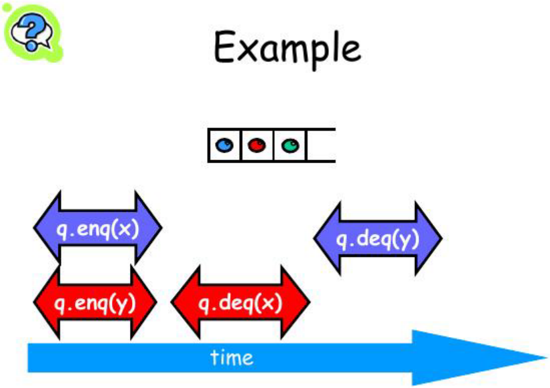
Linearizability



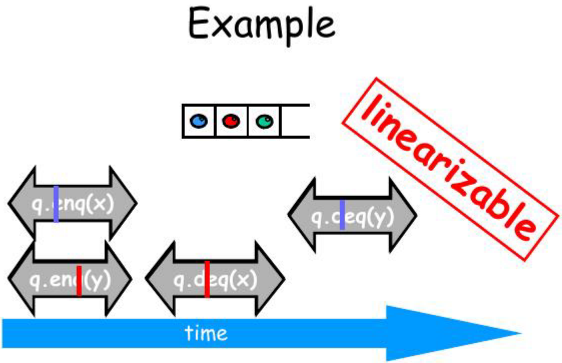
Linearizability



Linearizability

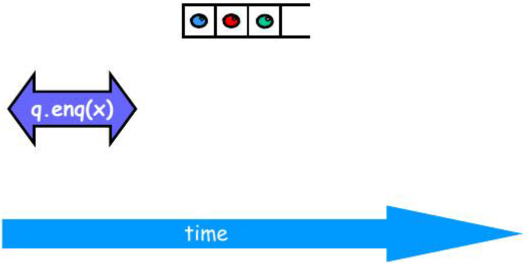


Linearizability



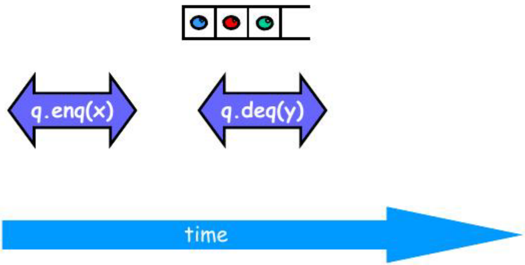
Linearizability

Example



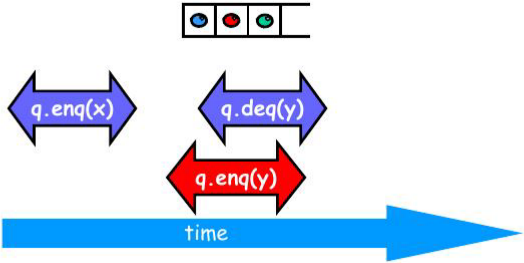
Linearizability

Example

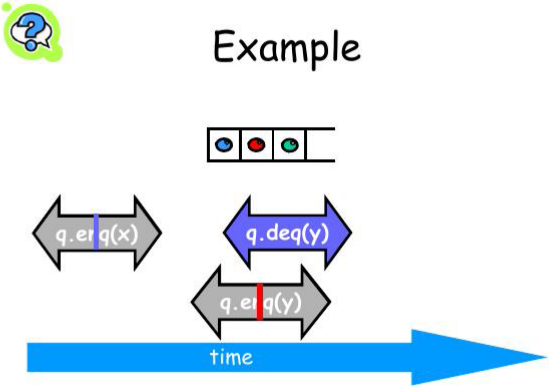


Linearizability

Example

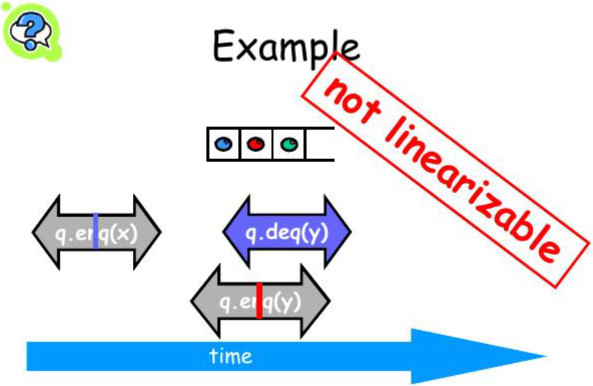


Linearizability



Example

Linearizability



Proving Linearizability Manually

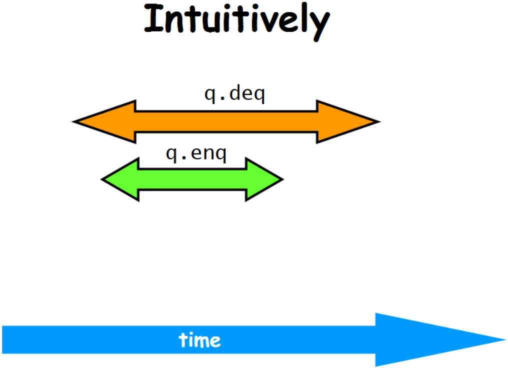
- Challenging even for simple data structures.

Proving Linearizability Manually

- Challenging even for simple data structures.
- Several techniques have been proposed
  - Linearization Points [HerlihyWing90]
  - Rely Guarantee [Vafeiadis, et al. 06]
  - Hindsight Lemma [O'Hearn10]
  - Base Point Analysis [KfirKeidar15]

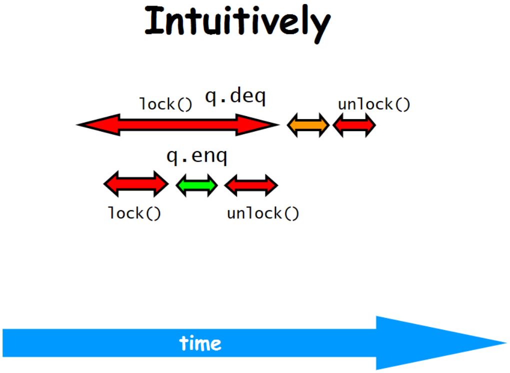
Linearization Points [HerlihyWing90]

- Every operation "appears to happen" at some individual instruction between invocation and response.
- In coarse locks, LP could be anywhere in the critical section.



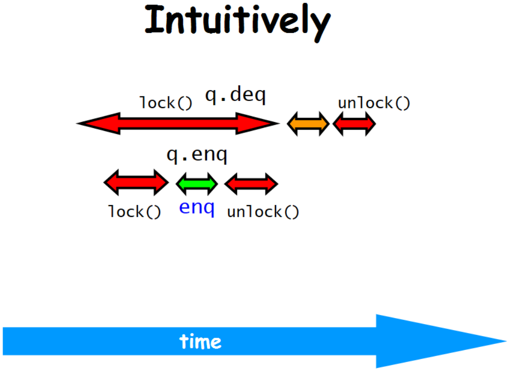
Linearization Points [HerlihyWing90]

- Every operation "appears to happen" at some individual instruction between invocation and response.
- In coarse locks, LP could be anywhere in the critical section.



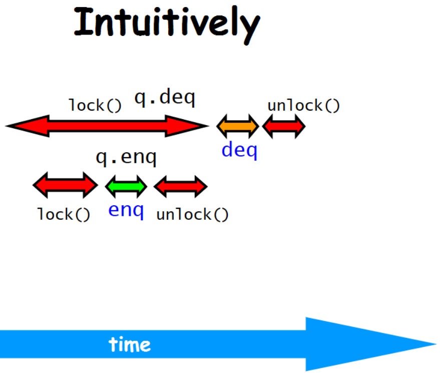
Linearization Points [HerlihyWing90]

- Every operation "appears to happen" at some individual instruction between invocation and response.
- In coarse locks, LP could be anywhere in the critical section.



Linearization Points [HerlihyWing90]

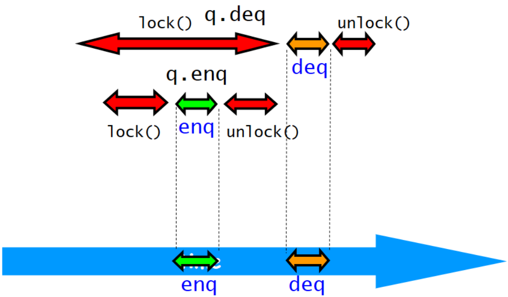
- Every operation "appears to happen" at some individual instruction between invocation and response.
- In coarse locks, LP could be anywhere in the critical section.



Linearization Points [HerlihyWing90]

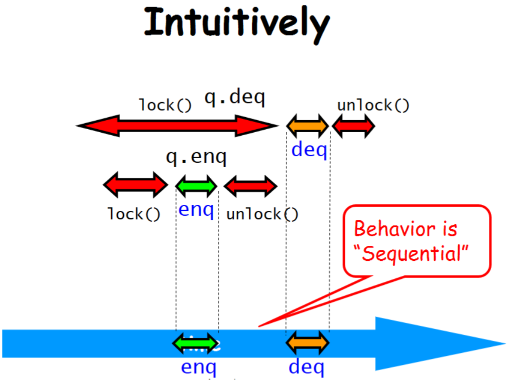
- Every operation "appears to happen" at some individual instruction between invocation and response.
- In coarse locks, LP could be anywhere in the critical section.

Intuitively



Linearization Points [HerlihyWing90]

- Every operation "appears to happen" at some individual instruction between invocation and response.
- In coarse locks, LP could be anywhere in the critical section.



Linearization Points [HerlihyWing90]

- Every operation "appears to happen" at some individual instruction between invocation and response.
- In coarse locks, LP could be anywhere in the critical section.
- LPs depend on the execution and return value of each operation.

Linearization Points [HerlihyWing90]

- Every operation "appears to happen" at some individual instruction between invocation and response.
- In coarse locks, LP could be anywhere in the critical section.
- LPs depend on the execution and return value of each operation.

Problem!

How do you know if you have identified the correct LPs indeed?

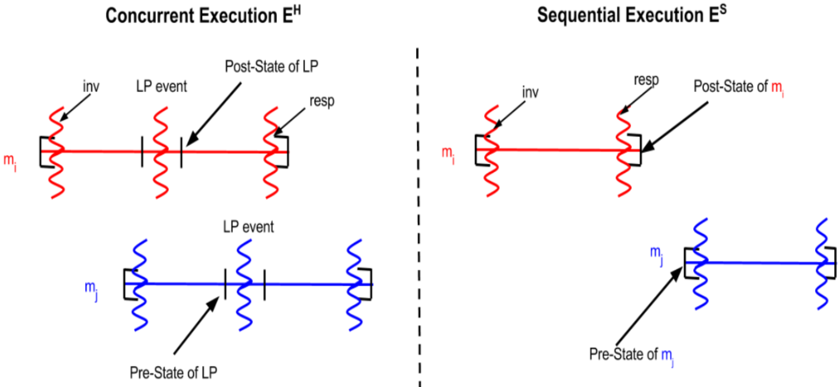
Outline

1. Introduction
   - Sequential Object and Sequential Specification
   - Sequential vs Concurrent History
2. Linearizability
3. Proposed Technique: Validating LPs
4. Conclusion
5. Future Work

Proposed Technique: Validating LPs

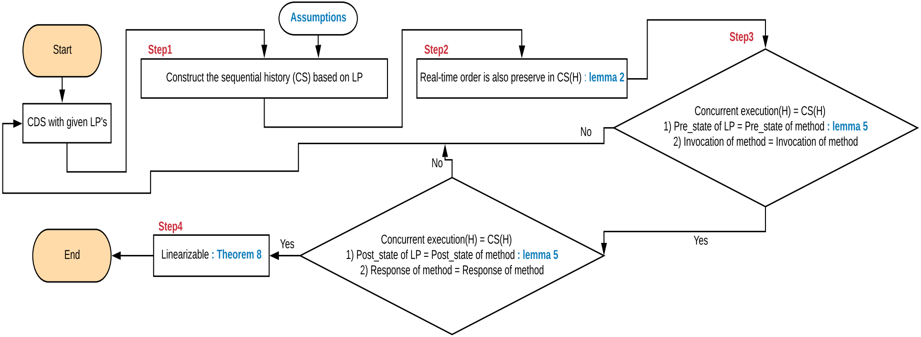
Hand-crafted generic technique for validating LPs

- Terminology:
  - Events, Methods
  - State
  - History, Execution, Complete History
  - Abstract data structure (AbDS)



Proposed Technique: Validating LPs

Hand-crafted generic technique for validating LPs



Start
CDS with given LP's
Step1
Construct the sequential history (CS) based on LP
Assumptions
Step2
Real-time order is also preserve in CS(H) : lemma 2
Step3
Concurrent execution(H) = CS(H)
1) Pre_state of LP = Pre_state of method : lemma 5
2) Invocation of method = Invocation of method
No
No
Yes
Concurrent execution(H) = CS(H)
1) Post_state of LP = Post_state of method : lemma 5
2) Response of method = Response of method
Yes
Step4
Linearizable : Theorem 8
End

Figure: Iterative steps to prove linearizability of a CDS with given LP's.

Proposed Technique: Validating LPs

Hand-crafted generic technique for validating LPs

- Assumptions:
    - Every sequential history S generated by the concurrent data structure(CDS) is *legal*.

Proposed Technique: Validating LPs

Hand-crafted generic technique for validating LPs

- Assumptions:
  - Every sequential history S generated by the concurrent data structure(CDS) is *legal*.
  - Each method has a unique atomic LP event within its invocation and response.

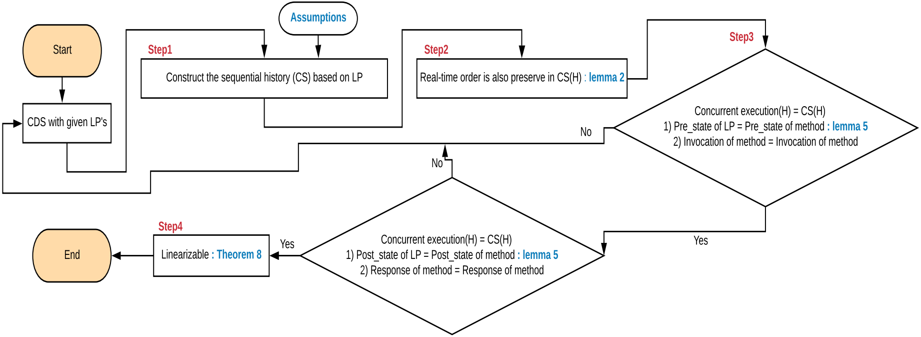
Proposed Technique: Validating LPs

Hand-crafted generic technique for validating LPs

- Assumptions:
  - Every sequential history S generated by the concurrent data structure(CDS) is *legal*.
  - Each method has a unique atomic LP event within its invocation and response.
  - Only the LP events of a method can change AbDS of CDS.

Proposed Technique: Validating LPs

Hand-crafted generic technique for validating LPs



Start

CDS with given LP's

Assumptions

Step1

Construct the sequential history (CS) based on LP

Step2

Real-time order is also preserve in CS(H) : lemma 2

Step3

Concurrent execution(H) = CS(H)
1) Pre_state of LP = Pre_state of method : lemma 5
2) Invocation of method = Invocation of method

No

Yes

No

Concurrent execution(H) = CS(H)
1) Post_state of LP = Post_state of method : lemma 5
2) Response of method = Response of method

Yes

Step4

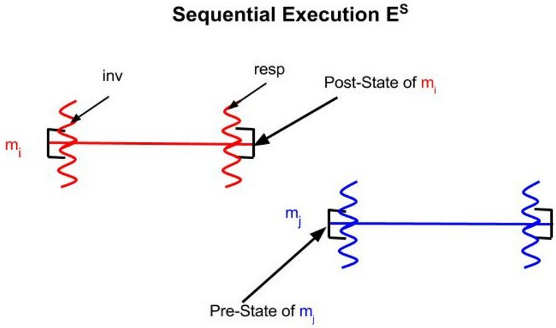
Linearizable : Theorem 8

End

Figure: Iterative steps to prove linearizability of a CDS with given LP's.

Proposed Technique: Validating LPs

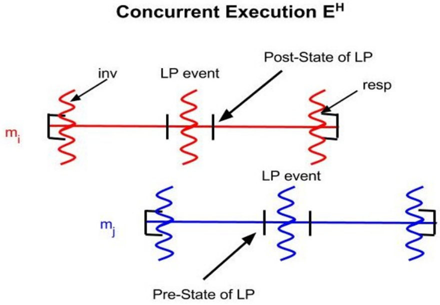
Hand-crafted generic technique for validating LPs



E^S : Post-state of m_i = Pre-state of m_j

Proposed Technique: Validating LPs

Hand-crafted generic technique for validating LPs



E^H : Post-state of $m_i.LP$ = Pre-state of $m_j.LP$

Proposed Technique: Validating LPs

Hand-crafted generic technique for validating LPs

$\forall m : \langle$ Pre-state of $E^H.m_i.LP =$ Pre-state of $E^S.m_i\rangle \wedge\langle E^H.m_i.inv = E^S.m_i.inv \rangle \rightarrow \langle$ Post-state of $E^H.m_i.LP =$ Post-state of $E^S.m_i\rangle \wedge\langle E^H.m_i.resp = E^S.m_i.resp \rangle$

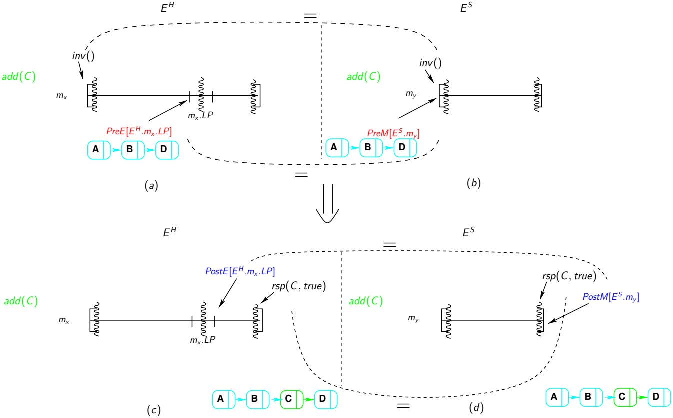


Figure: CDS Specific lemma

Outline

1. Introduction
   - Sequential Object and Sequential Specification
   - Sequential vs Concurrent History
2. Linearizability
3. Proposed Technique: Validating LPs
4. **Conclusion**
5. Future Work

Conclusion

- In spite of various approaches for proving linearizability, LPs seem most intuitive & constructive; but are difficult to identify.

Conclusion

- In spite of various approaches for proving linearizability, LPs seem most intuitive & constructive; but are difficult to identify.
- We have developed a hand-crafted technique of proving correctness of the CDSs by validating it LPs.

Conclusion

- In spite of various approaches for proving linearizability, LPs seem most intuitive & constructive; but are difficult to identify.
- We have developed a hand-crafted technique of proving correctness of the CDSs by validating it LPs.
- This technique will also offer the programmer some insight to develop more efficient variants of the CDS.

Conclusion

- In spite of various approaches for proving linearizability, LPs seem most intuitive & constructive; but are difficult to identify.
- We have developed a hand-crafted technique of proving correctness of the CDSs by validating it LPs.
- This technique will also offer the programmer some insight to develop more efficient variants of the CDS.
- We have shown the correctness of **lazy-list** and **hand-over-hand** locking list in technical report.

Outline

1. Introduction
   - Sequential Object and Sequential Specification
   - Sequential vs Concurrent History
2. Linearizability
3. Proposed Technique: Validating LPs
4. Conclusion
5. Future Work

Future Work

- We will extend it to the concept of **Linearization Blocks**.

Future Work

- We will extend it to the concept of **Linearization Blocks**.
- We will try to develope the automatic tool for validating LPs.

References

1. John Derrick, Gerhard Schellhorn, and Heike Wehrheim. Verifying linearisability with potential linearisation points. In Proceedings of the 17th International Conference on Formal Methods, FM'11, pages 323–337, Berlin, Heidelberg, 2011. Springer-Verlag.
2. Maurice P. Herlihy and Jeannette M. Wing. Linearizability: a correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst., 12(3):463–492, 1990.
3. Kfir Lev-Ari, Gregory V. Chockler, and Idit Keidar. On correctness of data structures under reads-write concurrency. In Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings, pages 273–287, 2014.
4. Kfir Lev-Ari, Gregory V. Chockler, and Idit Keidar. A constructive approach for proving data structures linearizability. In Distributed Computing - 29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings, pages 356–370, 2015.
5. Peter W. O'Hearn, Noam Rinetzky, Martin T. Vechev, Eran Yahav, and Greta Yorsh. Verifying linearizability with hindsight. In Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC '10, pages 85–94, New York, NY, USA, 2010. ACM.
6. Viktor Vafeiadis, Maurice Herlihy, Tony Hoare, and Marc Shapiro. Proving correctness of highly-concurrent linearisable objects. In Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2006, New York, New York, USA, March 29-31, 2006, pages 129–136, 2006.

Thank you for your attention!

Any Questions or Comments?

11/11