# Normality: A Consistency Condition for Concurrent Objects

Vijay K. GARG[*]
ECE Department
The University of Texas
Austin, TX 78712 (USA)
garg@ece.utexas.edu

Michel RAYNAL
IRISA
Campus de Beaulieu
35042 Rennes Cédex (France)
raynal@irisa.fr

**Abstract**

Linearizability is a consistency condition for concurrent objects (objects shared by concurrent processes) that exploits the semantics of abstract data types. It provides the illusion that each operation applied by concurrent processes takes effect instantaneously at some point between the beginning and the end of its execution. When compared with other consistency conditions (such as sequential consistency) Linearizability satisfies the Locality property (i.e, a system is linearizable if each object taken individually is linearizable) and the Non-Blocking property (i.e., termination of an invoked operation does not depend on other pending invocations). Those are noteworthy properties as they allow concurrent systems to be designed and constructed in a modular fashion.

This paper introduces a consistency condition called Normality that is less constraining than Linearizability (in the sense it does not refer to a global real-time order) and still satisfies Locality and Non-Blocking. As it does not refer to a global real-time, Normality is well-suited to objects supported by asynchronous distributed systems and can consequently be seen as an adaptation of Linearizability for these systems.

Categories and Subject Descriptors: B.3.2 [**Memory Structures**]: Design Styles-*shared memory*; C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures; D.1.3 [**Programming Techniques**]: Concurrent programming; D.2.1 [**Software engineering**]: Requirements; D.4.2 [**Operating Systems**]: Storage management-*distributed memories*; F.1.2 [**Computation by Abstract Devices**]: Modes of computation-*parallelism and concurrency*

General Term: Design, Theory

Key Words: Concurrent Objects, Consistency Condition, Linearizability, Locality Property, Non-Blocking Property.

## 1 Introduction

A set of sequential processes communicating through shared typed objects constitutes a concurrent system. Each shared object (or concurrent object) has a type that provides processes with a set of operations with which they can manipulate objects of this type. Each object type is defined by a sequential specification that describes the effect of each operation on an object of this type when executed alone. As in a concurrent system an object can be accessed

[*]supported in part by the NSF Grants ECS-9414780, CCR-9520540, and a General Motors Fellowship.

concurrently by several processes, it is necessary to define consistency conditions for concurrent objects. Sequential consistency [5] and register atomicity [6] are two such consistency conditions. Serializability is a consistency condition well-known in transactional systems [1].

In [3] Herlihy and Wing have introduced a consistency condition called Linearizability. This consistency condition generalizes the classical Atomicity consistency condition (designed for register objects) to objects whose set of operations is richer than the simple read and write operations. Intuitively, an execution of a concurrent system is linearizable (i.e., satisfies the Linearizability consistency condition) if it could appear to an external observer as a sequence composed of the operations invoked by processes that respects objects specifications and real-time precedence ordering on operations. So, Linearizability provides the illusion that each operation on shared objects issued by concurrent processes takes effect instantaneously at some point between the beginning and the end of its execution. This consistency condition has a great practical interest: it satisfies the Locality property (i.e., a concurrent system is linearizable if each of its objects taken individually is linearizable) and it satisfies the Non-Blocking property (i.e., termination of an invoked operation does not depend on other pending invocations). This means objects can be implemented and verified independently, so it allows modular design, interoperability and individual object-based scheduling policies.

This paper presents a consistency condition called Normality which is less constraining than Linearizability (in the sense it requires less constraints to be satisfied) while retaining Locality and Non-Blocking properties. Normality can be seen as a weakening of Linearizability that does not refer to real-time, so it is well-suited to asynchronous distributed systems where the concept of global real-time is impractical and awkward [2]. The paper consists of five sections. Section 2 presents the system model. Section 3 is a short introduction to the Linearizability theory. Section 4 introduces Normality and proves it has the Locality and Non-Blocking properties. Finally Section 5 compares Linearizability and Normality in a more general model where operations can span several objects. Section 6 concludes the paper.

## 2 System Model

The system model is basically the same as the one introduced in [3] from where the following definitions are taken.

**Objects and Processes.**
A *concurrent system* consists of a finite set of *sequential processes* (named $p_1, p_2, ..., p_n$) that communicate through *shared objects* (or *concurrent objects*). Each object has a name and a type. The type defines a set of possible values and a set of primitives operations that provide the only means to manipulate objects of this type. Execution of an operation takes some time; this is modeled by two events, namely an *invocation* event and a *response* event. A process sequentially applies operations to objects; this is modeled as a sequence of alternating invocation (*inv*) and matching response (*resp*) events. Let *op(arg,res)* be an operation on object $X$ issued at $p_i$; *arg* and *res* denote *op*'s input and output parameters, respectively. Invocation and response events $inv(op(arg))$ $on\ X\ at\ p_i$ and $resp(op(res))$ $from\ X\ at\ p_i$ will be abbreviated as *inv(op)* and *resp(op)* when parameters, object name and process identity are not necessary.

**Histories.**
Execution of a concurrent system is modeled by a *history* $\mathcal{H}$ which is a finite sequence of operation invocation and response events. Let $\leadsto_{\mathcal{H}}$ be the total order relation defined by $\mathcal{H}$ on *inv* and *resp* events, i.e., if $ev_1$ and $ev_2$ are two events and if $ev_1$ precedes $ev_2$ in $\mathcal{H}$, then $ev_1 \leadsto_{\mathcal{H}} ev_2$.

A subhistory of $\mathcal{H}$ is a subsequence of the events of $\mathcal{H}$. A history is *complete* if for each $inv(op)$ event that belongs to $\mathcal{H}$, the matching $resp(op)$ event belongs also to $\mathcal{H}$.

A history $\mathcal{H}$ is *sequential* if (1) its first event is an invocation and (2) each invocation[1] (response) event is immediately followed (preceded) by the matching response (invocation). A history that is not sequential is *concurrent.* In a concurrent history some operations overlap in time; if operations $e$ and $f$ are such that $\neg(resp(e) \rightsquigarrow_{\mathcal{H}} inv(f))$ and $\neg(resp(f) \rightsquigarrow_{\mathcal{H}} inv(e))$, $e$ and $f$ are said to be *concurrent.*

A process subhistory $\mathcal{H}|p_i$ ($\mathcal{H}$ at $p_i$) of a history $\mathcal{H}$ is the subsequence of all events in $\mathcal{H}$ whose process names are $p_i$. An object subhistory is defined in a similar way for an object $X$; it denoted $\mathcal{H}|X$ ($\mathcal{H}$ at $X$). Two histories $\mathcal{H}$ and $\mathcal{H}'$ are *equivalent* if for every process $p_i$ we have $\mathcal{H}|p_i = \mathcal{H}'|p_i$.

A history $\mathcal{H}$ is *well-formed* if each process subhistory $\mathcal{H}|p_i$ is sequential. In the following we consider only well-formed histories. Such histories model sequential processes accessing concurrent objects. As some operations on a same object $X$ can be concurrent, it is important to note that object subhistories of well-formed histories are not necessarily sequential.

**Object Specification.**
We consider that each object operation is specified by using a pre- and a post-condition. The specification of an object $X$ is defined as the set of all the sequential histories $\mathcal{S}_X$ of events that include $X$ and in which the pre- and the post-condition of each operation is satisfied.

**Legality.**
A sequential history $\mathcal{H}$ is *legal* if the pre- and the post-condition of each operation of $\mathcal{H}$ are satisfied.

## 3 Linearizability

As indicated in the introduction, Linearizability is a consistency condition for concurrent objects that has been introduced by Herlihy and Wing [3] to exploit the semantics of abstract data types. It provides the illusion that each operation applied by concurrent processes takes effect instantaneously at some point between its invocation and its response. When restricted to objects providing only read and write operations (register objects), it is equivalent to *atomicity* as defined by Misra in [6]. So, Linearizability generalizes Misra's approach to objects with a richer set of operations. The two important requirements of Linearizability are: (1) each operation should appear to take effect instantaneously, and (2) the order of non concurrent operations should be preserved.

More formally let us consider a history $\mathcal{H}$ and the "real-time" precedence ordering on operations denoted $<_{\mathcal{H}}$ and defined in the following way ($e$ and $f$ are two operations):

$$e <_{\mathcal{H}} f \quad \text{if} \quad resp(e) \rightsquigarrow_{\mathcal{H}} inv(f)$$

So, operations concurrent in $\mathcal{H}$ are not related by $<_{\mathcal{H}}$. If $\mathcal{H}$ is sequential then $<_{\mathcal{H}}$ is a total order.

**Linearizability (Definition)** A history $\mathcal{H}$ is *linearizable* if there exists an equivalent sequential history $\mathcal{S}$ that is legal and preserves $<_{\mathcal{H}}$ (i.e., if $e <_{\mathcal{H}} f$ then $resp(e) \rightsquigarrow_{\mathcal{S}} inv(f)$).

In a history $\mathcal{H}$, an object $X$ is linearizable if its concurrent history $\mathcal{H}|X$ is linearizable (i.e.,

[1]Except possibly the last one if the history is not complete.

belongs to its specification). Herlihy and Wing have proved the following important theorem (see [3], pages 470-471): *A history $\mathcal{H}$ is linearizable iff, for each object $X$, $\mathcal{H}|X$ is linearizable.*

A property P of a concurrent system is *local* if the system as a whole satisfies P whenever each individual object satisfies P. So, Linearizability is a local property. Locality is very important. It enhances modularity and concurrency: objects can be implemented and verified independently. As noted in [3], sequential consistency [5] and serializability [1] are not local properties; so, protocols implementing these consistency conditions must rely on global conventions to ensure all concurrency control mechanisms are mutually compatible. Such a compatibility is not necessary when all objects are linearizable.

It is also show in [3] that Linearizability satisfies the Non-Blocking property, i.e., a pending invocation of a totally defined operation is never required to wait for another pending invocation to complete.

## 4 Normality

Let us consider a history $\mathcal{H}$ and the following *happens before* relation defined on operations of $\mathcal{H}$. Let $e$ and $f$ be two operations of $\mathcal{H}$. Let *object(op)* and *proc(op)* be the set of objects and the process associated with the operation *op* respectively. $e \rightarrow_{\mathcal{H}} f$ ($e$ happens before $f$ in $\mathcal{H}$)[2] if one of the three following conditions holds:

`Process Order`: $(proc(e) = proc(f)) \;\; \wedge \;\; (resp(e) \rightsquigarrow_{\mathcal{H}} inv(f))$ (i.e., $e$ and $f$ are invoked by the same process with $e$ first).

`Object Order`[3]: $(object(e) \cap object(f) \neq \emptyset) \;\; \wedge \;\; (resp(e) \rightsquigarrow_{\mathcal{H}} inv(f))$.

`Transitivity`: $\exists g \;\;$ such that $\;\; (e \rightarrow_{\mathcal{H}} g) \;\; \wedge \;\; g \rightarrow_{\mathcal{H}} f)$.

Note that $\rightarrow_{\mathcal{H}} \subseteq <_{\mathcal{H}}$. Consequently, (1) operations on the same object that are concurrent are not related by $\rightarrow_{\mathcal{H}}$; (2) $e \rightarrow_{\mathcal{H}} f$ implies $resp(e) \rightsquigarrow_{\mathcal{H}} inv(f)$ in $\mathcal{H}$ but the converse does not necessarily hold: if $e$ and $f$ are issued by two distinct processes and are on distinct objects we can have $resp(e) \rightsquigarrow_{\mathcal{H}} inv(f)$ and $\neg(e \rightarrow_{\mathcal{H}} f)$.

While Linearizability requires real-time order (as defined by $\rightsquigarrow_{\mathcal{H}}$) be respected, Normality is weaker in the sense it requires only that the order $\rightarrow_{\mathcal{H}}$ be respected. More formally we have:

**Normality (Definition)** A history $\mathcal{H}$ is *normal* if there exists an equivalent sequential history $\mathcal{S}$ that is legal and preserves $\rightarrow_{\mathcal{H}}$ (i.e., if $e \rightarrow_{\mathcal{H}} f$ then $resp(e) \rightsquigarrow_{\mathcal{S}} inv(f)$).

The following Lemma shows that a history $\mathcal{H}$ is normal if and only if it is linearizable. Locality and Non-Blocking properties of normal histories will follow from this Lemma. Given any history $\mathcal{H}$, let $\Sigma_N(\mathcal{H})$ represent the set of equivalent legal sequential histories that preserve the order $\rightarrow_{\mathcal{H}}$. Similarly, let $\Sigma_L(\mathcal{H})$ denote the set of equivalent legal sequential histories that preserve the order $<_{\mathcal{H}}$.

**Lemma.** For any history $\mathcal{H}$: $\quad ( \Sigma_L(\mathcal{H}) \neq \emptyset ) \;\Leftrightarrow\; ( \Sigma_N(\mathcal{H}) \neq \emptyset )$.

**Proof:**

[2]Lamport's *happens before* relation for message-passing distributed systems [4] is a special case of the Normality *happens before* relation. In a distributed system the only shared objects are channels between processes.

[3]Here, as *object(op)* contains exactly one object, $(object(e) \cap object(f) \neq \emptyset)$ is equivalent to $(object(e) = object(f))$. This will no longer be true in the more general model introduced in Section 5 where *object(op)* can contain several objects.

$\Rightarrow$ : Since $\rightarrow_{\mathcal{H}}$ is contained in $<_{\mathcal{H}}$ it follows that $\Sigma_L(\mathcal{H}) \subseteq \Sigma_N(\mathcal{H})$. This clearly implies that if a history $\mathcal{H}$ is linearizable (i.e., $\Sigma_L(\mathcal{H})$ is non-empty), then it is also normal (i.e., $\Sigma_N(\mathcal{H})$ is also non-empty).

$\Leftarrow$ : Let $\mathcal{S} \in \Sigma_N(\mathcal{H})$. Thus, $\mathcal{S}$ is a legal sequential history equivalent to $\mathcal{H}$ which preserves $\rightarrow_{\mathcal{H}}$ (i.e., process order and object order). We construct another legal sequential history $\mathcal{S}'$ from $\mathcal{S}$ that preserves $<_{\mathcal{H}}$ (i.e., real-time order defined by $\mathcal{H}$) and that is equivalent to $\mathcal{H}$. If $\mathcal{S}$ also preserves $<_{\mathcal{H}}$ (real-time order), then $\mathcal{S}' = \mathcal{S}$ and we are done. Otherwise, there exist operations in $\mathcal{S}$ that violate real-time order $<_{\mathcal{H}}$. Let $e$ and $f$ be two such operations in $\mathcal{S}$ such that (Note that, as $\mathcal{S}$ is sequential, $<_S$ is a total order):

(P1) Operations $e$ and $f$ violate the real-time order. That is:

$$f <_{\mathcal{H}} e \quad \wedge \quad e <_S f$$

(P2) $e$ and $f$ is a pair of closest operations that satisfies P1 in $\mathcal{S}$; more explicitly, all the operations $g$ in $\mathcal{S}$ between $e$ and $f$ do not violate the real-time order ($<_{\mathcal{H}}$) with either $e$ or $f$ (so, such $g$ are concurrent with $e$ and $f$):

$$(e <_S g \ \wedge \ g <_S f) \quad \Rightarrow \quad (\neg(g <_{\mathcal{H}} e) \ \wedge \ \neg(f <_{\mathcal{H}} g))$$

We show existence of $\mathcal{S}' \in \Sigma_N(\mathcal{H})$ such that $f <_{S'} e$. The proof is by induction on the number of operations between $e$ and $f$ in $\mathcal{S}$. Formally, induction is on

$$k = |\{g \ \mid \ e <_S g \ \wedge \ g <_S f\}|$$

*Base case* ($k = 0$): Since $\mathcal{S}$ preserves process order and $f <_{\mathcal{H}} e$, it follows that $e$ and $f$ are on different processes. Similarly, since $\mathcal{S}$ preserves object order and $f <_{\mathcal{H}} e$, it follows that $e$ and $f$ are on different objects. This implies that $e$ and $f$ can be commuted without violating legality.

*Induction case* ($k > 0$): In $\mathcal{S}$, starting at $e$, let $h$ be the first operation after $e$ and before $f$ which is not on *object(e)* or be the last operation before $f$ and after $e$ which is not on *object(f)*. Without loss of generality assume that $h$ satisfies the former condition (The reasoning when $h$ satisfies the later condition is analogous).
Let $e_0, e_1, e_2, ..., e_m (m \geq 0)$ be the sequence of operations from $e$ to $h$ with $e_0 = e$. We show that $h$ can be moved before $e$ without violating legality. Since all $e_i$'s for any $0 \leq i \leq m$ are on *object(e)*, which is different from *object(h)*, it is sufficient to show that for any $i$ we have $proc(e_i) \neq proc(h)$. Suppose $proc(e_i) = proc(h)$. This implies that

(I1) $\qquad resp(e_i) \rightsquigarrow_{\mathcal{H}} inv(h)$

because processes are well-formed. From (P2), by substituting $e_i$ for $g$ we get $\neg(e_i <_{\mathcal{H}} e)$, i.e., $\neg(resp(e_i) \rightsquigarrow_{\mathcal{H}} inv(e))$, i.e.,

(I2) $\qquad inv(e) \rightsquigarrow_{\mathcal{H}} resp(e_i)$ .

Combining inequalities I1 and I2, we get

(I3) $\qquad inv(e) \rightsquigarrow_{\mathcal{H}} inv(h)$.

However, from (P1) $resp(f) \rightsquigarrow_{\mathcal{H}} inv(e)$ which combined with inequality (I3) gives $resp(f) \rightsquigarrow_{\mathcal{H}} inv(h)$, i.e., $f <_{\mathcal{H}} h$. This contradicts (P2). Thus, $proc(e_i) \neq proc(h)$ for any $i$. Hence $h$ can be moved before $e$, reducing the number of operations between $e$ and $f$.

This proves the Lemma.

∎

In the system model considered, primitive operations on objects are on one object at a time, i.e., operations are *unary*. This is used in the proof of the Lemma as, for each operation $e$, we consider $object(e)$ is composed of exactly one object. So, albeit Normality is less constraining than Linearizability ($\rightarrow_{\mathcal{H}} \subseteq <_{\mathcal{H}}$), as a direct consequence of the Lemma we get the following theorem.

**Theorem 1** *If all operations on objects are unary, then Linearizability and Normality are equivalent consistency conditions (i.e., a history $\mathcal{H}$ is linearizable iff it is normal).*

The next two theorems follow from the previous theorem 1, the definition of $\mathcal{H}|X$ and the theorem of [3] (namely, a history $\mathcal{H}$ is linearizable iff, for each object $X$, $\mathcal{H}|X$ is linearizable).

**Theorem 2** (Locality) *A history $\mathcal{H}$ is normal if and only if, for each object $X$, $\mathcal{H}|X$ is normal.*
**Theorem 3** (Non-Blocking) *A normal history $\mathcal{H}$ satisfies the Non-Blocking property.*

## 5 A More General System Model

The system model introduced in Section 2 assumes each operation is on exactly one object. We consider here a more general model where primitive operations can be on several objects[4]; so, $object(e)$ denotes now the set of objects associated with $e$. As an example consider a system with two register objects $A$ and $B$ provided with the traditional *read*, *write* operations plus the binary primitive operation *sum* (when invoked by a process $p$ with $A$ and $B$ as input parameters, *sum* returns the sum of the values of $A$ and $B$).

Definitions for a history $\mathcal{H}$, relations $\rightsquigarrow_{\mathcal{H}}$, $<_{\mathcal{H}}$ and $\rightarrow_{\mathcal{H}}$ and for Linearizability and Normality remain unchanged. The following example shows that the previous Lemma does not hold in this more general model (i.e., a history $\mathcal{H}$ can be normal but not linearizable). Let $\mathcal{H} = ev_1\ ev_2\ ...\ ev_6$ with ($A$ and $B$ are initialized to 0):

$$\begin{array}{lllll}
ev_1 = & inv(write(1)) & on & A & at\ \ p_1 \\
ev_2 = & inv(sum()) & on & A, B & at\ \ p_2 \\
ev_3 = & resp(write()) & from & A & at\ \ p_1 \\
ev_4 = & inv(write(2)) & on & B & at\ \ p_3 \\
ev_5 = & resp(write()) & from & B & at\ \ p_3 \\
ev_6 = & resp(sum(2)) & from & A, B & at\ \ p_2
\end{array}$$

This history $\mathcal{H}$ is not linearizable as $ev_3 \rightsquigarrow_{\mathcal{H}} ev_4$ ($resp(write())\ from\ A\ at\ p_1$ occurred before $inv(write(2))\ on\ B\ at\ p_3$) and *sum* returns a value indicating that the operation $write(B, 2)$ has happened but the operation $write(A, 1)$ has not. However $\mathcal{H}$ is normal as there is a sequential history $\mathcal{S}$, equivalent to $\mathcal{H}$, that is legal and respects $\rightarrow_{\mathcal{H}}$:

$$\mathcal{S} = write(B, 2)\ at\ p_3;\quad sum((A, B), 2)\ at\ p_2;\quad write(A, 1)\ at\ p_1$$

It follows that:

[4]Such operations are called *multi-methods* in object-oriented terminology. As unary operations, they can be specified by pre- and post-conditions.

- In a system model where object operations are unary: Linearizability and Normality (albeit its formulation is less constraining) (1) are equivalent consistency conditions (satisfying both Locality and Non-Blocking properties) and (2) are stronger than sequential consistency [5] (i.e., a history $\mathcal{H}$ can be linearizable/normal but not sequentially consistent).
- In a system model where primitive object operations can span several objects: Linearizability and Normality are no more equivalent. Linearizability is a consistency condition stronger than Normality which itself is a consistency condition stronger than sequential consistency.

## 6 Conclusion

Normality is particularly attractive in asynchronous distributed systems as, in this context, only process order and object order can be observed. In these systems there is no global time frame and objects (managed by specialized servers) are accessed through RPC-like protocols. Actually, in a model where each operation is on one object, we have: Normality = (Linearizability − global time). Seen that way, this paper showed that the Locality and Non-Blocking properties can be attained in distributed systems as soon as we consider Normality as the consistency condition for shared objects. The paper has also shown that Linearizability and Normality are not equivalent in models where primitive operations can span several objects.

The following problem remains open and deserves further study: "Among all consistency conditions for shared objects (with unary operations) equivalent to Linearizability is Normality the "optimal" one (i.e., the one that requires the least constraints to be satisfied by a history)?".

## Acknowledgements

We thank Craig Chase for some fruitful discussions on the topic.

## References

[1] P. Bernstein, V. Hadzilacos and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, MA, 370 pages, 1987.

[2] V.K. Garg and A. Tomlinson. Causality versus Time: How to Specify and Verify Distributed Programs. In *Proc. 6th IEEE Symposium on Parallel and Distributed Processing*, Dallas, TX, pp.249-256, 1994.

[3] M. Herlihy and J. Wing. Linearizability: a Correctness Condition for Concurrent Objects. *ACM Trans. on Prog. Lang. and Systems*, 12(3):463-492, 1990.

[4] L. Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558-565, 1978.

[5] L. Lamport. How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs. *IEEE Trans. on Computers*, C28(9):690-691, 1979.

[6] J. Misra. Axioms for Memory Access in Asynchronous Hardware Systems. *ACM Trans. on Prog. Lang. and Systems*, 8(1):142-153, 1986.