A Consistent Understanding of Consistency

Regular Submission

Subhajit Sidhanta, Affiliation: INESC-ID
Email: ssidhanta@gsd.inesc-id.pt
Address: INESC-ID, R. Alves Redol 9, 1000-029 Lisboa
Phone: +351 912413257

Ricardo J. Dias, Affiliation: NOVA LINCS, Universidade NOVA de Lisboa & SUSE Linux GmbH

Rodrigo Rodrigues, Affiliation: INESC-ID/IST (U. Lisboa)

Abstract: We propose a specification language named *ConSpec*, which enables the formalization of different consistency semantics that a storage system may provide, using a simple and uniform syntax, which is independent of the design and implementation of the target storage system. ConSpec addresses the recent profusion of consistency definitions, which are often accompanied by either informal definitions, or definitions that are tied to implementation-level details. To enable a simple and uniform description of various existing proposals, ConSpec builds on recent proposals that view weak consistency definitions as partial orderings between operations forming a visibility graph, which reduces the consistency definitions to a set of restrictions on those partial orders, described using Linear Temporal Logic (LTL). We use ConSpec to revisit several existing models in light of a common way to define and compare them. Furthermore, the use of LTL enabled us to leverage existing automatic checkers to build a tool for validating whether a given trace of the execution of a system meets a certain consistency semantics. Finally, using our generic definitions and the insights from analyzing different models, we restate the CAP theorem to precisely define the class of consistency definitions that can and cannot be implemented in a highly-available, partition-tolerant way, in contrast with the original definition which only considered linearizability.

A Consistent Understanding of Consistency

SUBHAJIT SIDHANTA*, INESC-ID
RICARDO J. DIAS, NOVA LINCS, Universidade NOVA de Lisboa & SUSE Linux GmbH
RODRIGO RODRIGUES, INESC-ID/IST (U. Lisboa)

We propose a specification language named *ConSpec*, which enables the formalization of different consistency semantics that a storage system may provide, using a simple and uniform syntax, which is independent of the design and implementation of the target storage system. ConSpec addresses the recent profusion of consistency definitions, which are often accompanied by either informal definitions, or definitions that are tied to implementation-level details. To enable a simple and uniform description of various existing proposals, ConSpec builds on recent proposals that view weak consistency definitions as partial orderings between operations forming a visibility graph, which reduces the consistency definitions to a set of restrictions on those partial orders, described using Linear Temporal Logic (LTL). We use ConSpec to revisit several existing models in light of a common way to define and compare them. Furthermore, the use of LTL enabled us to leverage existing automatic checkers to build a tool for validating whether a given trace of the execution of a system meets a certain consistency semantics. Finally, using our generic definitions and the insights from analyzing different models, we restate the CAP theorem to precisely define the class of consistency definitions that can and cannot be implemented in a highly-available, partition-tolerant way, in contrast with the original definition which only considered linearizability.

ACM Reference format:
Subhajit Sidhanta, Ricardo J. Dias, and Rodrigo Rodrigues. 2016. A Consistent Understanding of Consistency. 1, 1, Article 1 (January 2016), 17 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The development of modern Internet-based applications and services requires application developers to take into account the semantics of the underlying storage system, to ensure correctness and acceptable performance. In fact, there is evidence that the lack of a good comprehension of these semantics can lead programmers to unintendedly break the application semantics [6]. However, the task of understanding and comparing the semantics provided by storage systems, normally encapsulated in a consistency definition, is made difficult by several factors. First, the current set of possible consistency models is not only large but also expanding, as new storage systems often coin new terms for the consistency semantics that they

*The corresponding author

DOI: 10.1145/nnnnnnn.nnnnnnn

offer [11, 18, 23]. Second, existing consistency semantics have definitions that are often imprecise and/or are tied to implementation specifics (such as versioning [1] or the existence of replicas at different sites [21]). Third, even in the cases for which precise definitions exist, it is difficult to compare different definitions, since they are written using different formalisms and using community-specific terms or notations.

To address these issues, we propose a specification language called *ConSpec*, a generic definition for consistency models, which can be easily parameterized to obtain precise definitions of a variety of semantics. ConSpec builds on the observations made by recent proposals that it is possible to define weak consistency semantics in terms of partial orders over the set of operations that were executed in the system, which comprise a visibility graph [14, 20, 21]. This allows us to reduce the configurable part of the consistency definition to a set of restrictions, written in Linear Temporal Logic (LTL), to a generic partial order. Intuitively, this allows us to express a set of rules that specify the allowed order in which results of operations can be visible to a client application.

With this framework in place, we were able to express several existing consistency definitions in a common language, and compare them in a precise way, thereby defining a hierarchy of consistency models.

Another contribution of this paper is that we built a tool, which is available online, to check whether a given trace violates a certain consistency model. This tool leverages our LTL based syntax, which allowed us to reuse existing automatic checkers and adapt them to our constructs in a direct way.

Our final contribution is that we were able to restate in much broader terms the CAP theorem [8, 13], whose original proof used linearizability as synonym for the strong consistency captured by the "C" property. In our new formulation of this theorem, we are able to define necessary and sufficient conditions for a given consistency model to be bound by the impossibility of being implemented in a highly available, partition-tolerant way.

The remainder of this paper is organized as follows. Section 2 discusses relevant related work. Section 3 presents the assumed system model, and basic definitions and terminology used in the paper. Section 4 presents a general format of specifications based on ConSpec. Section 5 lists the specifications of various consistency models, expressed using ConSpec. Section 6 presents a reformulation of the CAP theorem, and formally proves the same. Section 7 analyzes the previously mentioned consistency models in terms of the restated CAP theorem. Finally, Section 8 presents the design of a prototype automated verification tool that can validate a session trace with respect to a ConSpec specification, and we conclude in Section 9 with a series of future work directions.

2 RELATED WORK

Consistency definitions restrict the set of valid traces for the execution of a given system with a storage-like interface. In broad terms, the gold standard of consistency definitions are so-called "strong" consistency levels, which have the characteristic of approximating the behavior that is obtained when interacting with a system whose implementation has a single, centralized server that executes operations one at a time. There are several examples of such consistency models [3, 5, 15, 19, 22, 26, 29], most of which include precise definitions of their intended semantics.

Strong consistency is often forfeited by the algorithms that implement these storage systems, in order to achieve better performance (e.g., when processors cache possibly stale data in a multiprocessor) and/or

better availability (e.g., when multiple replicas of the data exist and operations proceed while contacting only a subset of them) [12, 18, 25, 28, 30, 31].

Often, the definitions of these consistency models is vague and/or underspecified. For instance, the strong consistency option of Cassandra, a widely used NoSQL storage system, is stated in terms of the size of the quorums that are used for read and write operations, leaving unspecified what happens as the system reconfigures and the set of replicas of a data item changes [17]. Even when the specification is more precise, it may suffer from being tied to implementation details that may not be widely applicable. For instance, some definitions assume the existence of a centralized server that keeps monotonically increasing version number associated with the data [1]; others explicitly define consistency in terms of the state maintained by different replicas of the data [21]. Finally, Terry et al. defined a set of session guarantees that strengthen the consistency offered by eventually consistent systems; but these were defined operationally in terms of the replicas that are accessed and the operations that these replica process and the respective order in which they are processed [33].

With the goal of obtaining more precise and less operational definitions, Chockler et al. [10] defined these session guarantees (and some other consistency levels) using first-order logic expressions. Even more broadly, Burckhardt et al. [9] present the definitions of a wide variety of consistency models using properties like visibility and happens before order as building blocks. Our definitions build on recent proposals for new consistency models that are weakly consistent by default but distinguish a subset of the operations, and enforce visibility restrictions among those operations [14, 20, 21]. In contrast, our goal is not to propose a new consistency model, but to gain a deeper understanding of existing ones.

Compared to these prior approaches, ConSpec provides a generic way to describe consistency specifications, where each consistency level corresponds to a different parameterization of our generic definition. Furthermore, we use our generic framework to generalize the CAP theorem as proved by Gilbert and Lynch [13]. Finally, we provide a software artifact to check whether traces meet a certain consistency level.

Other authors have explored the CAP theorem beyond its original formulation and first proof. Mahajan et al. [24] defined Real Time Causal Consistency, a stronger variant of causal consistency, and proved that it is the strongest consistency model that can be provided in a highly available and eventually consistent implementation. Recently, Attiya et al. [4] provided a formal specification of systems that implement causal consistency, and proved that Observable Causal Consistency, a stronger variant of causal consistency, is the strongest consistency model that can be provided in a highly available, partition tolerant manner. Their definitions are stated in terms of some implementation-level concepts, namely a set of replicas connected by a network. In contrast, we generalize CAP by defining necessary and sufficient conditions for consistency models to be implemented in an available and partition-tolerant way, which are applied to precisely characterize the CAP line in a hierarchy of consistency models.

3 BASIC DEFINITIONS

We assume a set of processes (or clients) that interface the storage system by invoking operations. An operation comprises an invocation and the respective response. A very large class of consistency definitions (namely those describing the behavior of loads and stores on computer hardware) assume that these operations are partitioned into two classes, namely read operations that do not affect the result of subsequent operations and write operations that do. Given that many of the consistency levels we describe

require this interface, we will also incorporate this division into our definitions. Note, however, that our definitions can be generalized in a straightforward way to other models. For example, databases make a similar distinction between queries and updates, and the state machine replication model distinguishes between read-only and read-write requests.

Similarly, many consistency definitions reason about an interface that exposes the existence of multiple objects (e.g., different memory addresses seen by a CPU or different keys in a key-value store). As such, whenever required by the consistency definition, we allow for the possibility that the interface allows the programmer to specify an object o associated with reads and writes.

Our execution models each process as a deterministic state machine, whose transitions can be triggered either by an external input (i.e., an operation invocation) or by receiving a message from the network, and where the transition can trigger sending messages and/or issuing outputs (i.e., an operation response). A *session trace st* is a sequence of operations $o \in O$ executed by the same client, ordered by the time when they were invoked. In this paper we assume that clients are well-formed, i.e., a client only invokes an operation after the preceding operation has returned its response value. As such, session traces can be modelled as sequences of pairs $o = \langle invocation, response \rangle$. Invocations (resp. responses) belong to a generic set of possible invocations $\mathcal{I}$ (resp. responses $\mathcal{R}$). We define a session invocation trace *sit* as the sequence of invocations that are obtained from transforming each element in a session trace using the projection operator to obtain only the invocations. We define a session invocation trace to be compatible with a session trace if the projection of the invocations in the session trace matches the session invocation trace (denoted $st \bowtie sit$).

In our notation, we denote an invocation of a write operation that writes a value v to an object x, as $w(x, v)$ (with an empty response). Conversely, a read operation on object x that outputs a value v' is denoted $r(x)v'$. In our formulas, when a quantifiers restricts an operation to a given type (read or write), we simply use the letters R and W, i.e., we write $\forall R \in O$ as a shorthand for $\forall r(x)v \in O$.

The *global session trace* $\mathcal{S}_t$ (resp. global session invocation trace $\mathcal{S}_{it}$) denotes the set of all session traces (resp. session invocation traces) in a given execution of the system.

Our notation defines special purpose LTL operators as shorthands to longer expressions. These correspond to scope operators that specify the context within which the expression is restricted. For example, we define a special-purpose operator F_{st} to restrict the LTL operator "eventually" to operations comprised in the session trace st. E.g., an expression $w\, F_{st}\, r$ denotes that the operation $w(x, v)$ is followed by the operation $r(x)v$ in a session trace st.

Finally, we assume the system has a sequential specification, which is a correctness condition, corresponding to the output of the operations in centralized system that executes operation in sequence, one at a time. In the case of the sequential specification of a system whose interface are read and write operations, the read operation to object x must output the value associated with the most recent write operation to the same object x. For other types of interfaces (e.g., in state machine replication), that specification may change, and in some cases is specific to the service interface (e.g., the state machine being replicated).

4 CONSPEC

In this section we present our generic ConSpec definition, which can be parameterized to obtain specifications for commonly used consistency models and isolation levels. Note that ConSpec focuses only on safety conditions. Defining liveness conditions is left as future work.

ConSpec builds on recent proposals for consistency definitions that treat a subset of the operations in a given trace differently (e.g., operations labeled as being "strongly consistent"), since they only enforce visibility among those specific operations [14, 20, 21]. We can similarly see different consistency definitions as enforcing different visibility relationships only among a subset of the system operations, often depending on their types (e.g., reads versus writes).

As such, the generic definition of ConSpec requires the existence of a partial order that intuitively forms a "visibility graph", i.e., the output of each operation must reflect the effects of the operations that precede it according to that partial order. This allows us to see different consistency models as imposing different restrictive conditions on this precedence. Such a restrictive condition is then expressed as an LTL expression E^s.

Definition 4.1. Generalized form of ConSpec: Given a global session trace $\mathcal{S}_t$, we say that $\mathcal{S}_t$ satisfies a consistency model C if there exists a partial order $(O_{St}, \preccurlyeq)$ over the set O_{St} comprising operations present in all session traces in $\mathcal{S}_t$, i.e., $O_{St} = \bigcup_{st \in St}\{o \mid o \in st\}$, such that 1) for every operation o in $\mathcal{S}_t$, its output is equal to the one obtain by executing the sequential specification of a linear extension of the operations preceding o in $\preccurlyeq$, and 2) $(O_{St}, \preccurlyeq)$ obeys E^s, which is an LTL expression restricting $(O_{St}, \preccurlyeq)$.

Condition 1, when applied to a system whose interface consists only of reads and writes, translates to a requirement that every read operation in $\mathcal{S}_t$ must return the value of the most recent write according to $\preccurlyeq$. (In the case of two or more concurrent preceding writes, the system must arbitrate an order for them, e.g., use the "last writer wins" policy [34].) Condition 2 can be expressed as $E^s \models (O_{St}, \preccurlyeq)$, where E^s is the ConSpec parameterization for each consistency model C, and $\models$ is the satisfies operator.

5 SPECIFYING EXISTING MODELS

In this section, we present the ConSpec specifications for several common consistency models documented in the literature [9, 10, 32]. The derivations of ConSpec definitions from their original counterparts are made available in an extended version of the paper at https://github.com/ssidhanta/ConSpecTool/blob/master/ConSpecPODCFull.pdf. We start by specifying a series of session guarantees, which were originally proposed by Terry et al. in the context of a mobile computing storage system called Bayou [32]. These are four guarantees that apply to individual sessions, allowing applications to see a view of the storage system that is consistent with their previous operations. Their original definition is tied to some implementation-level concepts, since it is stated, for instance, in terms of the order in which writes are applied at various server replicas. Subsequently there were authors who wrote formal definitions for these session guarantees [9, 10].

We start with the Read Your Writes (RYW) session guarantee, which informally precludes a read operation r from reading a value for object x that precedes a value the same client previously wrote to the same object in the same session. Following the format of Definition 4.1, RYW can be defined by the following restrictions on $\preccurlyeq$.

$$\forall st \in \mathcal{S}_t \quad E^s = \forall\, W', R' \in st :\; W'\, F_{st}\, R' \Rightarrow W' \preccurlyeq_{st+w} R', \tag{1}$$

where $st \in \mathcal{S}_t$ is the session from the standpoint of which the session guarantees are being upheld, and $\preccurlyeq_{st+w}$ denotes the restriction of $\preccurlyeq$ to the elements of st and the write operations of all other clients.

The Read (or Session) Monotonic, also called Monotonic Reads guarantee (MR) specifies intuitively that read operations on a given object invoked from the same session must always return results in an increasing order of recency. MR is formally represented by the following expression constraining the partial order defined by ConSpec.

$$\forall st \in \mathcal{S}_t \quad E^s = \forall\, R', R'' \in st :\; R'\, F_{st}\, R'' \Rightarrow R' \preccurlyeq_{st+w} R'', \tag{2}$$

with the same meaning for st and $\preccurlyeq_{st+w}$ as in the previous definition.

Write Follows Read (WFR) specifies the following: a write operation that follows a read operation in the same session must be ordered after the write operation seen by that read. WFR is expressed in ConSpec as follows.

$$\forall st \in \mathcal{S}_t \quad E^s = \forall\, R', W' \in st :\; R'\, F_{st}\, W' \Rightarrow R' \preccurlyeq_{st+w} W', \tag{3}$$

with the same meaning for st and $\preccurlyeq_{st+w}$ as in RYW.

Finally, the last session guarantee is the Monotonic Writes (MW) model, which is specified as follows: successive write operations invoked from the same session must be applied in the order in which they appear in the session trace. MW is expressed in ConSpec as follows.

$$\forall st \in \mathcal{S}_t \quad E^s = \forall\, W', W'' \in st :\; W'\, F_{st}\, W'' \Rightarrow W' \preccurlyeq_{st+w} W'', \tag{4}$$

with the same meaning for st and $\preccurlyeq_{st+w}$ as in RYW.

Our generic formulation allows for an interesting perspective on the definition of causal consistency [2], which can be expressed as the union of the previous session guarantees for all sessions. As such, any pair of operations in the same session need to be constrained by the partial order of the ConSpec definition:

$$E^s = \forall\; st \in \mathcal{S}_t,\; Op', Op' \in st :\; Op'\, F_{st}\, Op'' \Rightarrow Op' \preccurlyeq Op'', \tag{5}$$

with the subtle difference that this is a global property, instead of being specific to a given session trace $st \in \mathcal{S}_t$ and the respective subset of the relation $\preccurlyeq$.

For Processor Consistency, we consider the more well known Goodman's definition of Processor Consistency (PC) over the alternate definition used implemented in the DASH system [2]. Goodman's PC specifies that write operations performed from each client application must be observed by all clients (i.e., must occur in all session traces) according to their invocation orders. Thus, PC can be expressed as follows.

$$\begin{aligned} E^s = &\forall st \in \mathcal{S}_t,\; R'(x), R''(y), W'(x), W''(y) \in st : \\ &W'(x)\, F_{st}\, W''(y) \Rightarrow W'(x) \preccurlyeq W''(y) \wedge R'(x)\, F_{st}\, R''(y) \Rightarrow R'(x) \preccurlyeq R''(y). \end{aligned} \tag{6}$$

where $\preccurlyeq$ is a partial order over the global session trace $\mathcal{S}_t$.

Session 1	Session 2
w(x,1)	w(y,1)
w(x,99)	w(y,99)
r(y)1	r(x)1

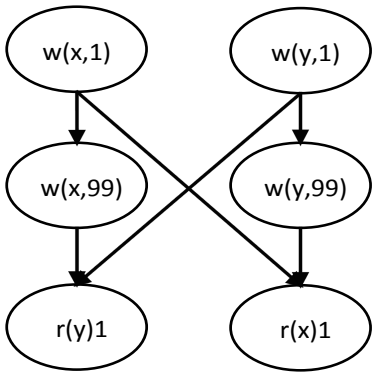

Fig. 1. Example of a global session trace and a partial order that is a superset of a partial order that can be used by each of the ConSpec definitions we studied (except for Sequential Consistency, whose constraints are not met). Some of the ordering constraints are not required by the each of the session guarantees, depending on the type of the pair of operations ordered. Furthermore, the set of operations being considered must be adapted to exclude the reads from the sessions other than the one from the standpoint of which the session guarantees are being upheld. Note that in the case of a read operation that is partially ordered after two concurrent writes the definition allows us to pick any of the linear extensions of the partial order.

Sequential consistency (SC) is a consistency model which requires that a global execution comprising operations executed from one or more clients must be equivalent to the result of executing the operations in a sequential order, such that the mutual order among operations from each client in the sequence respects the invocation order of the operations defined in the client. Using ConSpec, SC is expressed as follows.

$$\begin{aligned} E^s = \forall st \in \mathcal{S}_t,\ Op, Op' \in st,\ Op'', Op''' \in \mathcal{O}_{St} :\\ Op\, F_{st}\, Op' \Rightarrow Op \preccurlyeq Op' \quad \wedge \quad (Op'' \preccurlyeq Op''' \vee Op''' \preccurlyeq Op'') \end{aligned} \tag{7}$$

Figure 1 shows an example execution where the global session trace contains two sessions, each with three operations (two writes followed by a read). This execution meets all the consistency levels we defined with the exception of sequential consistency. The right hand side of the figure shows a partial order that is a superset of what can be used to support the execution in various definitions, and that meets the constraints specified by all the consistency levels except those specified by sequential consistency. For sequential consistency, the constraints would require the partial order to be a total order, which is impossible to achieve while also obeying the session order and explaining the results that are observed

according to the sequential specification of a read/write interface. This is because one would have to serialize both reads before the respective writes of value 99, but that would be impossible to achieve in a total order that respects the session orders.

6 REWRITING THE CAP THEOREM IN TERMS OF CONSPEC

The CAP conjecture was initially stated informally as the impossibility of simultaneously achieving strong Consistency, Availability, and Partition tolerance in a replicated system [8]. When this theorem was subsequently proven by Gilbert and Lynch [13], these three properties were stated precisely, and, in this context, strong consistency was defined as atomicity (or linearizability [15]).

The fact that the original proof of CAP is restricted to linearizability raises the question of whether CAP holds using other definitions from the wide array of consistency models supported by modern storage systems. In this section, we rewrite the CAP theorem in terms of ConSpec to precisely define the class of consistency models that can and cannot be implemented in a highly-available, partition-tolerant way.

To begin with, we need a helper definition to enumerate all admissible partial orders for a given restriction condition E^s and set of operations in a global session invocation trace $\mathcal{S}_{it}$.

Definition 6.1 (Partial order enumeration). Given a global session invocation trace $\mathcal{S}_{it}$ and a restrictive condition E^s for a ConSpec definition, we define the partial order enumeration of this session invocation trace and condition, $\Pi(\mathcal{S}_{it}, E^s)$ as the set of partial orders over the elements of any compatible session trace $\mathcal{S}_{it}$ that are valid under E^s, i.e.:

$\Pi(\mathcal{S}_{it}, E^s) \equiv \{\preccurlyeq: \exists \mathcal{S}_t \, (E^s \vdash (\mathcal{S}_t, \preccurlyeq) \wedge \mathcal{S}_t \bowtie \mathcal{S}_{it})\}$

This allows us to define the following necessary and sufficient condition for a consistency model to have an available and partition tolerant implementation.

THEOREM 6.2 (EXTENDED CAP THEOREM). *In an asynchronous system, it is possible to implement a consistency model E^s while simultaneously providing availability and partition tolerance if and only if for any global session invocation trace $\mathcal{S}_{it}$ and all of its partial orderings that are allowed by E^s, when you consider the set of maxima of each partial order, it is always possible to make them depend only on the previous operation in the same session and still obtain a valid partial order, i.e.:*

$\forall \mathcal{S}_{it} \; \forall \preccurlyeq \in \Pi(\mathcal{S}_{it}, E^s) \; \forall o \in max(\mathcal{S}_{it}, \preccurlyeq)$ (RemoveAllExceptSession$(\preccurlyeq, o) \in \Pi(\mathcal{S}_{it}, E^s)$)

where we define REMOVEALLEXCEPTSESSION *as a partial order where the maximum o is only directly ordered after prior operations in the same session, i.e.:*

REMOVEALLEXCEPTSESSION$(\preccurlyeq, o) \equiv \preccurlyeq \setminus \{\langle o', o\rangle$*, where $\langle o', o\rangle$ belongs to the transitive reduction of $\preccurlyeq$, and o' does not belong to the same session as o*$\}$.

Proof: We start by proving the implication in the direction ($\Rightarrow$). Following the proof style of Gilbert et al. [13], we prove this by contradiction as follows. Let us assume, by contradiction, that consistency model E^s is implemented by an algorithm that is highly available during partitions but does not meet the condition at the end of the Theorem. Let us consider initially that there are only two clients, c_1 and c_2, with sessions $s1$ and $s2$, respectively. The fact that E^s does not meet this condition means precisely that there must exist a global session trace $\mathcal{S}_t$ with a valid partial order $\preccurlyeq$ that has a maximum element o_{s1} such that $o_{s2} \preccurlyeq o_{s1}$ (where o_{s1}, o_{s2} belong to $s1$ and $s2$), and where it is not admissible to have a partial order where $o_{s2} \not\preccurlyeq o_{s1}$.

Now let us construct the following execution. First, we run the system under the exact same conditions that produced $\mathcal{S}_t$ until client c_1 is about to execute o_{s1} and client c_2 is about to execute o_{s2}. At this point, a partition occurs that separates c_1 and c_2, which persists until the end of the execution. By the availability and partition-tolerance properties, the operations o_{s1} and o_{s2} will eventually complete and, by our initial assumption in the previous paragraph, the former operation must see the effects of the latter, i.e., the partial order that supports that execution must be such that $o_{s2} \preccurlyeq o_{s1}$. Now run the exact same execution, but where the client c_2 crashes right before invoking o_{s2}. This execution is indistinguishable from the previous one from the standpoint of c_1. Thus c_1 will follow the same sequence of states and produce the same outputs as in the previous execution. This would mean that the algorithm would not meet its ConSpec specification, since o_{s1} would reflect the execution of an operation that was not part of the global session trace $\mathcal{S}_t$, namely o_{s2}. This contradicts the fact that the algorithm that was used meets that specification and the CAP properties.

The assumption about there being only two clients does not lose generality because, with more clients, a pair of clients c_1, c_2 under the conditions above must also exist. Then the proof generalizes beyond two clients by the partitioning the clients into two sets, one containing c_1 and another containing c_2, and crashing all the clients in the same side of the partition as c_2.

Next we focus on the implication in the direction ($\Leftarrow$). Here, we need to prove that if a consistency model E^s meets the condition:

$\forall \mathcal{S}_{it}\ \forall \preccurlyeq \in \Pi(\mathcal{S}_{it}, E^s)\ \forall o \in max(\mathcal{S}_{it}, \preccurlyeq)$ (RemoveAllExceptSession$(\preccurlyeq, o) \in \Pi(\mathcal{S}_{it}, E^s)$)

then it has an available and partition-tolerant implementation.

Given any global session invocation trace $\mathcal{S}_{it}$, we prove this by induction on the length of the execution that produced $\mathcal{S}_{it}$. The base case with an empty execution is vacuously true, since an empty trace meets any consistency condition (no safety properties are ever violated by an empty trace). For the induction step, we need to prove that, given an execution for which an available and partition-tolerant implementation produced a trace that conforms to E^s, it is possible for a client to invoke a new operation and produce an output that is also consistent. This is true because, even in the case that the client that invokes the new operation is partitioned from the remaining of the clients, it is always legal to produce an operation that depends on a prior operation from the same session and all the operations that transitively precede it according to $\preccurlyeq$. Furthermore, the valid output of this operation can be determined by using only information that is local to the session, by running the sequential specification of the system on the graph of preceding operations.

□

7 ANALYSIS OF CONSISTENCY MODELS WITH RESPECT TO CAP

The previous section defined necessary and sufficient conditions for a consistency model C to have an available and partition-tolerant implementation. Now, we analyze the E^s-expressions for the consistency models that we studied in Section 5, to determine how they fare with respect to Theorem 6.

We can see that the session gurantees (MR, MW, RYW, WFR) and both the causal and processor consistency definitions are only forcing constraints on the partial ordering across operations from the same session. This implies that these constraints are compatible with the conditions on the right hand side of the equivalence of Theorem 6. In particular, it is the case that it is always legal to remove orderings between operations across sessions, since these are never constrained by the implications in the various

different E^s expressions. Therefore, we conclude that Causal Consistency, Processor Consistency, and all four session guarantees are not affected by CAP, i.e., can have highly available and partition-tolerant implementations.

In contrast, SC requires that the visibility order $\preccurlyeq$ among operations from all the clients in the system forms a total order. This implies that if an operation is related by the transitive reduction of $\preccurlyeq$ to a previous operation from another session, it is not possible to remove this element of the partial order and still obtain a valid partial order, since it would violate the condition in the definition of SC that any two operations need to be ordered with respect to each other. Thus, this does not meet the necessary and sufficient condition for a partition-tolerant, highly available implementation.

8 IMPLEMENTATION

We provide an open source automated verification tool built using Spin [16], an open source software verification framework. The source code of the ConSpec tool and instructions for running it is made publicly available in a github repository https://github.com/ssidhanta/ConSpecTool. A given session trace is supplied to the tool as an input. The expression E^s for a particular consistency model C is specified as a Spin LTL formula. E^s acts as a safety property, and the system behaviour is modelled from a given session trace using the PROMELA meta language. The PROMELA source file provides an abstract model of interaction of the system with clients. The tool systematically attempts to verify whether a given session trace is valid under a given consistency model C, specified in terms of E^s. Internally, Spin translates the PROMELA source file into C code. The Spin driver then runs the built-in model checker to check for counter-examples for the above generated C code against the Spin formula extracted from E^s. The Spin model checker validates the generated C code with the Spin formula as the invariant.

9 CONCLUSIONS

In this paper, we presented a generic framework called ConSpec for defining consistency. ConSpec enables definitions that are precise, follow a generic structure, and are independent of implementation details. We used ConSpec to derive several concrete definitions of existing consistency levels. Furthermore, ConSpec also enabled a generic version of the CAP theorem, where the "C" property is not longer tied to a specific strong consistency definition. Instead, we define necessary and sufficient conditions for a consistency level to be within the scope of CAP, i.e., for the existence or not of a partition tolerant and available implementation. Furthermore, we developed an automated tool for verifying whether a given session trace satisfies a consistency model.

ConSpec opens several interesting avenues for future work. First, we intend to apply ConSpec to a wider range of consistency models. Second, we intend to extend it to support isolation levels of transactional systems, where the visibility of individual operations within a transaction must be constrained. Finally, we intend to further develop our automatic verification tools and promote their adoption by the developer community.

REFERENCES

[1] ADYA, A., LISKOV, B., AND O'NEIL, P. E. Generalized isolation level definitions. In *ICDE* (2000), pp. 67–78.

[2] AHAMAD, M., BAZZI, R. A., JOHN, R., KOHLI, P., AND NEIGER, G. The power of processor consistency. In *Proceedings of the Fifth Annual ACM Symposium on Parallel Algorithms and Architectures* (New York, NY, USA, 1993), SPAA '93, ACM, pp. 251–260.

[3] AHAMAD, M., NEIGER, G., BURNS, J. E., KOHLI, P., AND HUTTO, P. Causal memory: Definitions, implementation and programming, 1994.
[4] ATTIYA, H., ELLEN, F., AND MORRISON, A. Limitations of highly-available eventually-consistent data stores. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 2015), PODC '15, ACM, pp. 385–394.
[5] ATTIYA, H., AND WELCH, J. L. Sequential consistency versus linearizability. *ACM Trans. Comput. Syst. 12*, 2 (May 1994), 91–122.
[6] BAILIS, P., FEKETE, A., FRANKLIN, M. J., GHODSI, A., HELLERSTEIN, J. M., AND STOICA, I. Feral concurrency control: An empirical investigation of modern application integrity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, ACM, pp. 1327–1342.
[7] BAILIS, P., GHODSI, A., HELLERSTEIN, J. M., AND STOICA, I. Bolt-on causal consistency. In *SIGMOD '13*.
[8] BREWER, E. A. Towards robust distributed systems (Invited Talk). In *Proc. of the 19th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (2000).
[9] BURCKHARDT, S. Principles of eventual consistency. *Found. Trends Program. Lang. 1*, 1-2 (Oct. 2014), 1–150.
[10] CHOCKLER, G., FRIEDMAN, R., AND VITENBERG, R. *Consistency Conditions for a CORBA Caching Service*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 374–388.
[11] COOPER, B. F., RAMAKRISHNAN, R., SRIVASTAVA, U., SILBERSTEIN, A., BOHANNON, P., JACOBSEN, H.-A., PUZ, N., WEAVER, D., AND YERNENI, R. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow. 1*, 2 (Aug. 2008), 1277–1288.
[12] DECANDIA, G., HASTORUN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., AND VOGELS, W. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev. 41*, 6 (Oct. 2007), 205–220.
[13] GILBERT, S., AND LYNCH, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News 33*, 2 (June 2002), 51–59.
[14] GOTSMAN, A., YANG, H., FERREIRA, C., NAJAFZADEH, M., AND SHAPIRO, M. 'cause i'm strong enough: Reasoning about consistency choices in distributed systems. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (New York, NY, USA, 2016), POPL '16, ACM, pp. 371–384.
[15] HERLIHY, M. P., AND WING, J. M. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst. 12*, 3 (July 1990), 463–492.
[16] HOLZMANN, G. *Spin Model Checker, the: Primer and Reference Manual*, first ed. Addison-Wesley Professional, 2003.
[17] INC., D. Configuring data consistency. http://docs.datastax.com/en/archived/cassandra/2.0/cassandra/dml/dml_config_consistency_c.html, 2017.
[18] LAKSHMAN, A., AND MALIK, P. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev. 44*, 2 (Apr. 2010), 35–40.
[19] LAMPORT, L. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Comput. 28*, 9 (Sept. 1979), 690–691.
[20] LI, C., LEITÃO, J. A., CLEMENT, A., PREGUIÇA, N., AND RODRIGUES, R. Minimizing coordination in replicated systems. In *Proceedings of the First Workshop on Principles and Practice of Consistency for Distributed Data* (2015), PaPoC '15, ACM, pp. 8:1–8:4.
[21] LI, C., PORTO, D., CLEMENT, A., GEHRKE, J., PREGUIÇA, N., AND RODRIGUES, R. Making geo-replicated systems fast as possible, consistent when necessary. In *Proc. of the 10th USENIX conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2012), OSDI'12, USENIX Association, pp. 265–278.
[22] LIPTON, R., AND SANDBERG, J. S. PRAM : a scalable shared memory. Tech. Rep. CS-TR-180-88, Princeton University (NJ US), 1988.
[23] LLOYD, W., FREEDMAN, M. J., KAMINSKY, M., AND ANDERSEN, D. G. Don't settle for eventual: Scalable causal consistency for wide-area storage with cops. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2011), SOSP '11, ACM, pp. 401–416.
[24] MAHAJAN, P., ALVISI, L., AND DAHLIN, M. Consistency, availability, convergence. Tech. Rep. TR-11-22, Computer Science Department, University of Texas at Austin, May 2011.
[25] MEIKLEJOHN, C. Riak PG: Distributed process groups on dynamo-style distributed storage. In *Erlang '13*.

[26] MIZUNO, M., RAYNAL, M., AND ZHOU, J. Z. *Sequential consistency in distributed systems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, pp. 224–241.
[27] OWENS, S., SARKAR, S., AND SEWELL, P. A better x86 memory model: X86-tso. In *Proceedings of the 22Nd International Conference on Theorem Proving in Higher Order Logics* (Berlin, Heidelberg, 2009), TPHOLs '09, Springer-Verlag, pp. 391–407.
[28] PLUGGE, E., HAWKINS, T., AND MEMBREY, P. *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing*, 1st ed. Apress, Berkely, CA, USA, 2010.
[29] RAYNAL, M., AND SCHIPER, A. *From causal consistency to sequential consistency in shared memory systems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, pp. 180–194.
[30] SCHÜTT, T., SCHINTKE, F., AND REINEFELD, A. Scalaris: Reliable transactional P2P key/value store. In *Proceedings of the 7th ACM SIGPLAN Workshop on ERLANG* (New York, NY, USA, 2008), ERLANG '08, ACM, pp. 41–48.
[31] SUMBALY, R., KREPS, J., GAO, L., FEINBERG, A., SOMAN, C., AND SHAH, S. Serving large-scale batch computed data with project Voldemort. In *Proc. of the 10th USENIX Conference on File and Storage Technologies (FAST)* (2012).
[32] TERRY, D. B., DEMERS, A. J., PETERSEN, K., SPREITZER, M., THEIMER, M., AND WELCH, B. W. Session guarantees for weakly consistent replicated data. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems* (Washington, DC, USA, 1994), PDIS '94, IEEE Computer Society, pp. 140–149.
[33] TERRY, D. B., THEIMER, M. M., PETERSEN, K., DEMERS, A. J., SPREITZER, M. J., AND HAUSER, C. H. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *SOSP '95*.
[34] THOMAS, R. H. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst. 4*, 2 (June 1979), 180–209.

10 APPENDIX

10.1 Deriving ConSpec Specifications From Original Definitions

RYW, MR, WFR, MW, Causal and Sequential consistency have been defined by Chockler et al. [10], whereas PC have been defined in [2, 27]. In this section, we illustrate that the ConSpec specifications of consistency models [2, 10, 27] and original definitions are equivalent. We show that it is possible to formally derive ConSpec specifications from the original definitions. We begin by describing the notations used in the original consistency definitions given by Chockler et al., and how they can be translated to the syntax of ConSpec. According to Chockler et al., a system comprises a group of processes that communicate with each other by invoking read or write operations on a group of objects. Chockler et al. denotes a pair of read or write operations on an object x, invoked from the i^{th} process p_i in the system, as o^1 and o^2, respectively. In this paper, we refer to a process as a client, and use the notation Cl_i in place of p_i for the i^{th} client in the system, without any loss of information. The notation σ_i is used by Chockler et al. to denote a local execution composed of a sequence of read and write operations performed by a client Cl_i. Each operation is comprised of an invocation event and a response event, such that the response for each operation follows its corresponding invocation. The notation σ denotes a global execution comprising all such local executions performed by all clients in the system. Chockler et al. uses the symbol $\rightarrow$ to denote a precedence relationship [7] between two operations. $\xrightarrow{\sigma_i}$ is a specialised form of the precedence operator, where the superscript σ_i is used to restrict the precedence relationship $\rightarrow$ to operations comprised in a particular execution σ_i. According to the above system model, invocation event of an operation can not occur unless response event of preceding operation has occurred in an execution. Hence, the expression $o^1 \xrightarrow{\sigma_i} o^2$ implies that both the invocation and response for operation o^1 precedes (i.e., happens before) the invocation and response of operation o^2 in an execution sequence σ_i. We define a session trace st as a sequence of operations performed by a client Cl_i, ordered according

to invocation time of operations. For means and purposes of this paper, an execution sequence σ_i of Chockler et al. is equivalent to a session trace *st* in ConSpec. Hence, without any loss of information, we replace each reference to σ_i in the the definitions of Chockler et al. with the notation *st*. Additionally, Chockler et al. uses the notation $\sigma|i + w$ to denote a partial execution, where $i + w$ implies the restriction of a global execution σ to an execution comprising all operations performed by a given client Cl_i (or process p_i) plus all writes invoked by other clients. A legal serialization is a linear sequence of invocation of operations such that each read operation in the sequence returns the result of the last preceding write. S_p denotes an equivalent *legal serialization* for a partial execution $\sigma|i + w$. A special-purpose operator $\xrightarrow{S_p}$ is used to denote precedence relation among operations comprised in the legal serialization S_p.

Using the above notations, Chockler et al. states the RYW consistency model as follows. RYW is given in the form

$$Condition1 \Rightarrow Condition2, \tag{8}$$

where $\Rightarrow$ is the implies operator; Condition 1 and Condition 2 are given as follows. Condition 1 is given in the form $o^1 \xrightarrow{\sigma_i} o^2$, where o^1 and o^2 are read and write operations comprised in a given execution σ_i of a process p_i. Condition 2 is given as $o^1 \xrightarrow{S_p} o^2$, where S_p is an equivalent legal serialization for a partial execution $\sigma|i + w$ comprising operations invoked by p_i plus writes invoked by other clients. Since the clients are well-formed (refer to Section 3), the precedence relation $o^1 \xrightarrow{\sigma_i} o^2$ in Condition 1 implies that both the invocation and response for o^1 occurs before the invocation and response for o^2 can occur. Let inv_1 and inv_2 denote the invocations, and $resp_1$ and $resp_2$ denote the responses of o^1 and o^2, respectively. Thus, the precedence relation $o^1 \xrightarrow{\sigma_i} o^2$ in Condition 1 implies $inv_1 \xrightarrow{\sigma_i} inv_2$ and $resp_1 \xrightarrow{\sigma_i} resp_2$. By definition, the session trace *st* is a linear sequence that is composed of results of operations comprised in the execution σ_i obtained by executing the client Cl_i. Since *st* is ordered with respect to the invocation times of operations comprised in σ_i, the precedence relation $o^1 \xrightarrow{\sigma_i} o^2$ implies that the invocation of o^1 occurred before the invocation of o^2 in the temporal ordering of invocation times in *st*. Thus, $o^1 \xrightarrow{\sigma_i} o^2$ implies $inv^1 F_{st} inv^2$, where F_{st} is a the special-purpose LTL operator(defined in Section 3). Further, since clients in ConSpec are well-formed, invocation and response for a given operation precedes the invocation and response of the next operation in *st*. Thus, $inv^1 F_{st} inv^2$ implies $resp_1 F_{st} resp_2$, which, in turn, implies $o^1 F_{st} o^2$. Hence, we can rewrite a precedence relation $o^1 \xrightarrow{\sigma_i} o^2$ found in Chockler et al.'s definitions as $o^1 F_{st} o^2$. Let Op^1 and Op^2 be propositional logic variables that indicate whether invocations and responses of operations o^1 and o^2 have executed (if Op^1 and Op^2 is TRUE) or not. Then, Condition 1 reduces to $Op^1 F_{st} Op^2$ [1]. In the postcondition (Condition 2), Chockler et al. specify a precedence relation $o^1 \xrightarrow{S_p} o^2$, which restricts a precedence relation among o^1 and o^2 in the equivalent legal serialization S_p for the given $\sigma|i + w$. By definition of S_p in Section 3, both the invocation and response for an operation o^2 in S_p must appear after the invocation and response of a preceding operation o^1 in S_p, i.e., $o^1 \xrightarrow{S_p} o^2$ implies $inv^1 \xrightarrow{S_p} inv^2$ and $resp^1 \xrightarrow{S_p} resp^2$. Hence, all components of operation o^2 follow all components

[1]The above LTL expression implies: if the propositional variable Op^1 (denoting the event of execution of operation o^1) is true at one instant, Op^2 (denoting the event of execution of o^2) is eventually true in some later instant in a session trace *st*, i.e., the operation o^2 eventually follows the operation o^1 in the given session trace *st*.

of o^1 in S_p. Thus, we can rewrite the above precedence relation $o^1 \xrightarrow{S_p} o^2$, in terms of Linear Temporal Logic (i.e., LTL), as $Op^1 F_{S_p} Op^2$. Since RYW considers only those execution sequences where a write operation is followed by a read, Op^1 and Op^2 can be replaced by new propositional variables $W^{'}$ and $R^{''}$, without any loss of information. Thus, Condition 1 and Condition 2 can be expressed as $W^{'} F_{st} R^{''}$ and $W^{'} F_{S_p} R^{''}$, respectively. Further, we can replace S_p in the postcondition with the notation $\preccurlyeq_{st+w}$ (defined in Definition 4.1 and Equation 5) because of the following reasons. According to Chockler et al., a legal serialization S_p is a sequence of operations that satisfies the following properties: Property 1) it is a linear sequence that comprises all operations from a given client Cl_i plus writes from all other clients, and Property 2) each read in the sequence S_p returns the result of the preceding write in S_p. First, by definition (refer to definition of $\preccurlyeq$ in Definition 4.1), $\preccurlyeq_{st+w}$ is a partial order comprising all operations in a session trace st plus writes from other operations, thus $\preccurlyeq_{st+w}$ satisfies Property 1 for S_p. Second, following directly from Condition 1 in Definition 4.1, output of each operation in $\preccurlyeq_{st+w}$ is equivalent to that obtained by executing a linear sequence of the operations preceding that operation, thus $\preccurlyeq_{st+w}$ satisfies Property 2 for S_p. Hence, we can rewrite the postcondition $W^{'} F_{S_p} R^{''}$ for Condition 2 as $W^{'} \preccurlyeq_{st+w} R^{''}$. Combining the above conditions, Chockler's definition of RYW reduces into the ConSpec specification in Equation 5.

According to Chockler et al., MR is expressed as Equation 8, where both o^1 and o^2 are read operations. Again, following the same logic as in RYW, the above precedence relationships among operations o^1 and o^2 in Condition 1 can be directly expressed in terms of an LTL expression $R^{'} F_{st} R^{''}$. Similarly, the expression $o^1 \xrightarrow{S_p} o^2$ in Condition 2 can be expressed in terms of LTL as $R^{'} F_{S_p} R^{''}$. Further, similar to the derivation of RYW, we can rewrite the expression $R^{'} F_{S_p} R^{''}$ in Condition 2 as $R^{'} \preccurlyeq_{st+w} R^{''}$, thus reducing the above specification into Equation 2.

Chockler et al. expresses WFR consistency as Equation 8, where of a read operation o^1 is followed by a write operation o^2 in an execution σ_i. As in the case of of our derivations for sRYW and MR, the expressions $o^1 \xrightarrow{\sigma_i} o^2$, and $o^1 \xrightarrow{S_p} o^2$ for Conditions 1 and 2 can be rewritten as LTL expressions $R^{'} F_{st} W^{''}$ and $R^{'} F_{S_p} W^{''}$, respectively. Following from the reasoning of or derivations of RYW and MR, the expression $R^{'} F_{S_p} W^{''}$ for Condition 2 can be rewritten in terms of ConSpec as $R^{'} \preccurlyeq_{st+w} W^{''}$, thus reducing the WFR definition to Equation 3.

Chockler et al. expresses the MW consistency model as Equation 8, where o^1 and o^2 are write operations in an execution σ_i. Following the same line of reasoning as that of our derivations for RYW, MR, and WFR, the expressions $o^1 \xrightarrow{\sigma_i} o^2$, and $o^1 \xrightarrow{S_p} o^2$ can be rewritten as LTL expressions $W^{'} F_{st} W^{''}$ and $W^{'} F_{S_p} W^{''}$, respectively. Exactly like our previous derivations, the expression $W^{'} F_{S_p} W^{''}$ for Condition 2 can be rewritten as $W^{'} \preccurlyeq_{st+w} W^{''}$, thus reducing the definition into Equation 4.

In their definition of Causal consistency, Chockler uses the notion of a *direct precedence relation* between operations o and $o^{'}$ in an execution order σ_i, denoted as $\overset{\sigma_i}{\Rightarrow}$. The expression $o \overset{\sigma_i}{\Rightarrow} o^{'}$ implies that either of the following properties must hold: Property 1) $o^{'}$ is a read operation which returns the values written by a write operation o, or Property 2) the precedence relation $o \xrightarrow{\sigma_i} o^{'}$ holds for a given execution σ_i. Causal consistency is expressed as Equation 8. Condition 1 specifies that a transitive closure $\overset{\star}{\Rightarrow}$ exists over a direct precedence relation $o \overset{\sigma_i}{\Rightarrow} o^{'}$ among a given pair of operations o and $o^{'}$ in

σ_i. Condition 2 in Equation 8 can be expressed in terms of LTL as $\forall st.\exists S_p \left(Op^{'} F_{S_p} Op^{''}\right)$. Following the same line of reasoning as that used in our derivation for RYW, Condition 2 can be restated as: there must exist a partial order $\preccurlyeq$ which respects the order specified among the operations performed by each client, i.e., with respect to each observed session trace st, hence, with respect to the global session trace $\mathcal{S}_t$. Thus, Condition 2 reduces to the form $Op^{'} \preccurlyeq Op^{''}$, where $\preccurlyeq$ is a partial order with respect to operations in the global session trace. As in previous cases, the expression $o \overset{\sigma_i}{\Rightarrow} o^{'}$ in Condition 1 can be expressed in the form $Op^{'} F_{st} Op^{''}$. However, Condition 2 implies that read operation $o(x)v^{'}$, corresponding to the propositional variable $Op^{'}$, reads the value written by the write $o(x,v)^{''}$, corresponding to the propositional variable $Op^{''}$. Thus, the precondition, comprising a logical disjunction over Condition 1 and 2, can be expressed as $Op^{'} F Op^{''} \vee \left(\left(Op^{'} = W^{'}\right) \wedge \left(Op^{''} = R^{''}\right) \wedge \left(v_i = v_j\right)\right)$, where $R^{''}$ and $W^{'}$ are shortcut notations for $o(x)v^{'}$ and $o(x,v)^{''}$, respectively. For a given $\mathcal{S}_t$ to satisfy causal consistency, a transitive closure must exist over the above condition. However, it directly follows from the Condition 1 in Definition 4.1 that if Condition 2 holds, i.e., if a valid $\preccurlyeq$ comprising o and $o^{'}$ exists, every operation in $\preccurlyeq$ must reflect a result which is equivalent to that of executing the prior operations in $\preccurlyeq$ according to a linear sequence. Hence, $Op^{'} F_{st} Op^{''}$ implies that the transitivity condition holds over the expression $Op^{'} F_{st} Op^{''}$ in Condition 1. Hence, the precondition for Causality can simply be expressed as $Op^{'} F_{st} Op^{''}$. Thus, Chockler's definition of Causal Consistency reduces into the specification given in Equation 5.

Following the same approach as before, the ConSpec specification for Processor consistency (PC) can be directly derived from the definition of Goodman's PC provided by Ahamad et al. [2]. According to the above definition, there must exist a valid legal serialization for a partial execution $\sigma|i + w$ comprising operations performed by a processor p_i, which we denote as S_p, which must satisfy the following conditions. Condition 1 states: every pair of write operations $w(x,v)$ and $w^{'}(y,v^{'})$ in a system must occur in S_p in the same precedence order as the invocation order of $w(x,v)$ and $w^{'}(y,v^{'})$ in a processor p_i which invokes $w(x,v)$ and $w^{'}(y,v^{'})$. Ahamad et al. expresses Condition 1 as: $o^1 \xrightarrow{\sigma_i} o^2$ implies $o^1 \xrightarrow{S_p} o^2$. Condition 2 states: all writes to a given object in a system must follow an identical precedence order in valid legal serializations for all processors in the system. Following the same line of reasoning as in the previous derivations, we can rewrite the statement for Condition 1 in terms of ConSpec as follows. Using propositional variables $W^{'}(x)$, and $W^{''}(y)$ to denote the events of execution of write operations $w(x,v)$ and $w^{'}(y,v^{'})$ by processor p_i, Condition 1 can be expressed as $W^{'}(x) F_{st} W^{''}(y) \Rightarrow W^{'}(x) F_{S_p} W^{''}(y)$. Further, we can express a valid legal serialization S_p for a processor p_i in terms of a partial order $\preccurlyeq_{st+w}$. Thus, the above expression reduces to $W^{'}(x) F_{st} W^{''}(y) \Rightarrow W^{'}(x) \preccurlyeq_{st+w} W^{''}(y)$. Let us denote $\preccurlyeq$ as a partial order over all operations in a global session trace $\mathcal{S}_t$ observed for a global execution comprising all processors executing in the system. Then, Condition 2 can be restated as: there must exists a partial order $\preccurlyeq$ for $\mathcal{S}_t$ such that $W^{'}(x) \preccurlyeq W^{''}(y)$ holds, for a pair of writes performed by any clients. Applying Condition 2, the above expression for Condition 1 can be reduced to the form $W^{'}(x) F_{st} W^{''}(y) \Rightarrow W^{'}(x) \preccurlyeq W^{''}(y)$. Further, Condition 2 implies that the effect of successive writes must be observed in an identical precedence order in legal serializations for all processors in the system, i.e., the precedence order among successive reads by a process p_i must be preserved in all legal serializations for all processors. In other words, a partial order $\preccurlyeq$ for $\mathcal{S}_t$ must apply a pair of read operations $r(x)v$ and $r(x)v^{'}$ according to their precedence order in the session trace st, i.e., $R'(x)\, F_{st}\, R''(y) \Rightarrow R'(x) \preccurlyeq R''(y)$. The above expressions can be combined into the ConSpec specification given by Equation 6.

Chockler et al. states Sequential Consistency as: the precedence order among operations comprised in a local execution of a given process must match the precedence order among the operations in an equivalent legal serialization for the global execution comprising all processes executing on the system, i.e., $o^1 \xrightarrow{\sigma_i} o^2 \Rightarrow o^1 \xrightarrow{S} o^2$, where S in an equivalent legal serialization for the global session execution σ. Following the same approach as in previous derivation, the precedence relation $o^1 \xrightarrow{\sigma_i} o^2$ in the LHS of the above expression can be restated as $Op^{'} F_{st} Op^{''}$. Similarly, the RHS can be expressed as $\exists S \left(Op^{'} F_S Op^{''} \right)$. As in our previous derivations, the above postcondition can be expressed as $Op^{'} \preccurlyeq Op^{''}$, where $\preccurlyeq$ is a partial order comprising all operations in $\mathcal{S}_t$, hence all operations in the global execution σ. Further, since the condition $Op^{'} F_S Op^{''}$ implies a total order $\prec$ among o^1 and o^2, we can replace the partial order symbol $\preccurlyeq$ with $\prec$. This does not cause any loss of information since a total order is a special case of a partial order, i.e., $Op^{'} \prec Op^{''}$ implies $\left(Op^{'} \preccurlyeq Op^{''} \right) \vee \left(Op^{''} \preccurlyeq Op^{'} \right)$. Hence, we can rewrite the above postcondition as $Op^{'} \prec Op^{''}$. Thus, Chockler's definition of SC reduces into the specification given in Equation 7.