Decidability and Complexity for Quiescent Consistency and its Variations

Brijesh Dongol, Robert M. Hierons

Department of Computer Science, Brunel University London, UK

Abstract

Quiescent consistency is a notion of correctness for a concurrent object that gives meaning to the object's behaviours in quiescent states, i.e., states in which none of the object's operations are being executed. The condition enables greater flexibility in object design by allowing more behaviours to be admitted, which in turn allows the algorithms implementing quiescent consistent objects to become more efficient (when executed in a multithreaded environment).

Quiescent consistency of an implementation object is defined in terms of a corresponding abstract specification. This gives rise to two important verification questions: *membership* (checking whether a behaviour of the implementation is allowed by the specification) and *correctness* (checking whether all behaviours of the implementation are allowed by the specification). In this paper, we show that the membership problem for quiescent consistency is NP-complete and that the correctness problem is decidable, but coNEXPTIME-complete. For both problems, we consider restricted versions of quiescent consistency by assuming an upper limit on the number of events between two quiescent points. Here, we show that the membership problem is in PTIME, whereas correctness is PSPACE-complete.

Quiescent consistency does not guarantee sequential consistency, i.e., it allows operation calls by the same process to be reordered when mapping to an abstract specification. Therefore, we also consider quiescent sequential consistency, which strengthens quiescent consistency with an additional sequential consistency condition. We show that the unrestricted versions of membership and correctness are NP-complete and undecidable, respectively. When placing a limit on the number of events between two quiescent points, membership is in PTIME, while correctness is PSPACE-complete. Finally, we consider a version of quiescent sequential consistency that places an upper limit on the number of processes for every run of the implementation, and show that the membership problem for quiescent sequential consistency with this restriction is in PTIME.

1. Introduction

Correctness conditions form a basis for designing a concurrent object that implements a sequential object; concurrent objects are necessary for efficient execution in a multi/many-core environment. Due to the possibility of parallel executions, correctness of an operation of a concurrent object cannot be stated in terms of pre/post conditions. Instead, correctness is expressed in terms of a history of operation invocation/response events, capturing the interaction between a concurrent object and its client. There are many notions of correctness for safety [1, 2] — relaxed notions are more permissive, and hence, allow greater flexibility in an object's design. Such flexibility is necessary in the presence of observations such as Amdahl's Law and Gustafson's Law [3], which show that sequential bottlenecks within an implementation must be reduced to improve performance [4].

This paper studies quiescent consistency [4, 5] a relaxed notion of correctness for concurrent objects, derived from a similar notion in replicated databases [6], that gives meaning to an object in its *quiescent states*, i.e., states in which none of its operations are currently executing. Here, correctness is defined by mapping a concurrent object's history (with potentially overlapping operation calls) to a sequential history of its corresponding specification object (with no overlapping operation calls).

1. A history of a concurrent object is considered to be correct with respect to a correctness condition C iff the history can be mapped to a valid (sequential) history of the object's specification and the mapping satisfies C.
2. A concurrent object satisfies C iff each of its histories is correct with respect to C.

These two issues give rise to two distinct verification problems: the former gives rise to a *membership* problem, and the latter a *correctness* problem. In this paper, we extend the existing approach of Alur et al. [7] and study the decidability and complexity of both membership and correctness for two correctness conditions: quiescent consistency and quiescent sequential consistency.

Informally speaking, quiescent consistency is defined as follows. A concurrent object is said to be in a quiescent state if none of its operations are being executed in that state. Quiescent consistency allows operations calls in a concurrent history between two consecutive quiescent states to be reordered when mapping to a history of the sequential specification, but disallows operations that are separated by a quiescent state from being reordered [2, 5, 4]. Compared to other conditions in the literature, quiescent consistency is more permissive. For example, unlike linearizability [8, 1, 2], it allows the effects of operation calls to be reordered even if they do not overlap in a concurrent history. Unlike sequential consistency [9, 1, 2], it allows the effects of operation calls by the same process to be reordered.

In the context of client-object systems, to guarantee observational refinement [10] of the client, it turns out that sequential consistency must be maintained [11]. Therefore, we additionally consider quiescent sequential consistency, a variation of quiescent consistency that adds a *sequential consistency* constraint [9] to quiescent consistency, i.e., we are not allowed to reorder the events of the same process.

In this paper, we make the following main contributions.

1. We describe how quiescent consistency can be expressed using independence from Mazurkiewicz Trace Theory [12] and encoded as finite automata.
2. Show that deciding membership for quiescent consistency is an NP-complete problem if the number of events between two quiescent states is unbounded, but deciding membership is polynomial (with respect to the size of the input run) if the number of events between two quiescent states has a fixed upper bound.
3. Show that correctness for quiescent consistency is decidable and coNEXPTIME-complete and is PSPACE-complete if the number of events between two quiescent states has a fixed upper bound.
4. Show that deciding membership for quiescent sequential consistency is an NP-complete problem but can be solved in polynomial time (with respect to the size of the input run) if the number of events between two quiescent states has a fixed upper limit, or if the number of processes can be predetermined.
5. Show that correctness for quiescent sequential consistency is undecidable but is PSPACE-complete if the number of events between two quiescent states has a fixed upper bound.

This paper extends our earlier paper on decidability and complexity for quiescent consistency [13] by additionally studying quiescent sequential consistency. A point of difference between these conditions (from the point-of-view of our study) is that they have different notions of concurrency. For quiescent consistency, (due to the absence of sequential consistency) two operations on the same thread may be treated as being concurrent if they occur between the same quiescent points. On the other hand, quiescent sequential consistency ensures that operation calls on the same thread are never considered to be concurrent.

This paper is organised as follows. In Section 2, we motivate the problem through an example, and describe the formal background of finite automata and independence used in the rest of the paper. Section 3 defines quiescent consistency, develops a finite automata encoding of quiescent consistency as well as the membership and correctness problems. Our results for the membership and correctness problems are given in Section 4 and Section 5, respectively. Section 6 describes quiescent sequential consistency, then Sections 7 and 8 present the results for membership and correctness for quiescent sequential consistency, respectively. A survey of related work and concluding remarks is given in Section 9.

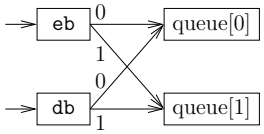

Figure 1: A 1-level diffracting queue with two queues

```
Init:  eb, db = 0

enqueue(el:T)                    dequeue
E1:  do lb := eb;                D1:  do lb := db;
E2:  until CAS(eb,lb,1-lb)       D2:  until CAS(db,lb,1-lb)
E3:  Enq(queue[lb],el)           D3:  return Deq(queue[lb])
```

Figure 2: Enqueue and dequeue operations on the diffraction queue

2. Background

This section motivates quiescent consistency with a queue example (Section 2.1), then gives a finite automata formalisation for studying the problem (Section 2.2). We will use a notion of independence from Mazurkiewicz Trace Theory, which we describe in Section 2.3.

2.1. A Quiescent Consistent Queue

We consider the quiescent consistent queue from [5] (see Figures 1 and 2). The queue is based on the architecture of *diffracting trees*, which uses the following principle (adapted from counting networks [14]). Elements called *balancers* are arranged in a binary tree, which may have arbitrary depth. Each balancer contains one bit, which determines the direction in which the tree is traversed; a balancer value of 0 causes a traversal up and a value 1 causes a traversal down. The leaves of the tree point to a concurrent data structure. Operations on the tree (and hence data structures) start at the root of the tree and traverse the tree based on the balancer values. Each traversal is coupled with a bit flip, so that the next traversal occurs along the other branch. Upon reaching a leaf, the process performs a corresponding operation on the data structure at the leaf.

Our example consists of two 1-level balancers `eb` and `db` used by enqueue and dequeue operations, respectively. Both operations share the two queues at the leaves (see Figure 1). Pseudocode for the queue is given in Figure 2. Both operations are implemented using a non-blocking atomic `CAS` (Compare-And-Swap) operation that compares the stored local value `old` with the shared variable `var` and updates `var` to a new value `new` if the values of `var` and `old` are still equal:

```
CAS(var,old,new) ==
     atomic{ if var = old
             then var := new; return true
             else return false}
```

Both operations read their corresponding bit and try to flip it using a `CAS`. If they succeed, they perform an *enqueue* `Enq` or *dequeue* `Deq` on the queue of their local bit. For simplicity, we assume that `Enq` and `Deq` are atomic operations (though they could be implemented by any linearizable operation). The queue only satisfies quiescent consistency if `Deq` is *blocking*, i.e., waits until an element is found in the queue. The diffracting queue is not quiescent consistent if `Deq` returns on empty (see [5] for details).

Example 1. *The following is a possible history for the blocking concurrent queue implementation:*

$$h_1 = D_1\ E_2(a)\ \widehat{E}_2\ E_3(b)\ \widehat{E}_3\ D_4\ \widehat{D}_4(b)\ D_5\ \widehat{D}_5(a)\ E_6(c)\ \widehat{E}_6\ \widehat{D}_1(c)$$

where D_1 denotes a `dequeue` *invocation by process* 1*, $\widehat{D}_1(c)$ denotes a* `dequeue` *by process* 1 *that returns c, $E_2(a)$ denotes an* `enqueue` *invocation by process* 2 *with input a, and $\widehat{E}_2$ denotes the corresponding return event. Operations D_1, E_2, D_5, and E_6 act on* `queue[0]`*, whereas E_3 and D_4 act on* `queue[1]`*. There is not much concurrency in h_1. Only the first dequeue is running concurrently with the rest of the operations. However, due to the first dequeue invocation, h_1 is only quiescent at the beginning and end.*

History h_1 is not linearizable [8] because the dequeues by processes 4 *and* 5 *violate the FIFO order of enqueues by processes* 2 *and* 3*, and linearizability does not allow non-overlapping operations to be reordered (see [5] for details). However, h_1 is quiescent consistent because quiescent consistency allows operations between two consecutive quiescent states to be reordered even if they do not overlap. This means that it may*

be matched with the following sequential history, which satisfies a specification of a sequential queue data structure.

$$h_2 = E_3(b)\ \widehat{E}_3\ E_2(a)\ \widehat{E}_2\ D_4\ \widehat{D}_4(b)\ D_5\ \widehat{D}_5(a)\ E_6(c)\ \widehat{E}_6\ D_1\ \widehat{D}_1(c)$$ □

Example 1 provides a concrete example in which the difference between quiescent consistency and quiescent sequential consistency can be seen with respect to the decidability and membership problems that we study, e.g., in history h_1 (Example 1) the operations by processes 3-6 could be executed by process 2 (bounding program-level concurrency to two processes), yet deciding quiescent consistency for this history is as hard as deciding quiescent consistency for h_1. In particular, Section 7.2 presents a process-based restriction on the quiescent sequential consistency membership problem that reduces its complexity. Such a restriction is not helpful for quiescent consistency.

2.2. Problem Representation

In this section, we present our formal framework. The behaviour of a system will be a sequence of events. Given a set A we will let A^* denote the set of finite sequences of elements of A and $\varepsilon \notin A$ denote the empty sequence. Like Alur et al. [7], the specification and implementation are both represented by finite automata, whose alphabet is a set of events recording the invocation/response of an operation.

Definition 1. *A* finite automaton *(FA) is a tuple $(M, m_0, \Sigma, t, M_\dagger)$ in which M is the finite set of states, $m_0 \in M$ is the initial state, Σ is the finite alphabet, $t \subseteq M \times \Sigma \times M$ is a ternary relation called the transition relation, and $M_\dagger \subseteq M$ is the set of final states.*

Given a finite automaton $\mathcal{M} = (M, m_0, \Sigma, t, M_\dagger)$, $(m, e, m') \in t$ is interpreted as "it is possible for $\mathcal{M}$ to move from state m to state m' via event e" and this defines the *transition* (m, e, m'). We will use natural numbers $\mathbb{N}$ to identify processes.

Example 2. *Suppose $P \subseteq \mathbb{N}$ is a finite set of process identifiers and T is a finite set of values. For the example in Figure 2, the finite automaton has state space*

$$Q \subseteq \mathbb{B} \times \mathbb{B} \times T^* \times T^* \times (P \to (\mathbb{B} \times T \times PC))$$

where the first two components are the values of `eb` *and* `db`*, respectively, and the second and third components correspond to the two queues (which we assume has bounded size). The final component corresponds to the local state, which records the value of* `lb`*, the value being enqueued/dequeued and the program counter for each process, where $PC = \{\mathtt{I}, \mathtt{E1}, \mathtt{E2}, \mathtt{E3}, \mathtt{D1}, \mathtt{D2}, \mathtt{D3}\}$ with the value* `I` *denoting idle. The initial state and set of events are respectively given by*

$$q_0 = (0, 0, \epsilon, \epsilon, (\lambda p : P.\ (0, \mathtt{I}, 0))$$
$$\Sigma = \{\tau\} \cup \left(\bigcup_{p:P} \left(\{\widehat{E}_p, D_p\} \cup \bigcup_{v:T} \{E_p(v), \widehat{D}_p(v)\}\right)\right)$$

where τ is a special label that distinguishes internal transitions. The label τ will generally be ignored in our analysis below. The set of transitions corresponds to each line of the program and modifies the state in the obvious way. We assume that when the upper bound of the queue's size is reached, no new enqueue operations may be invoked. A process may only invoke a new operation if its program counter value is `I`*. Similarly, the program counter value is (re)set to* `I` *when the operation it is executing completes. Thus, we assume each final state in $Q_\dagger$ matches $(\_, \_, \_, \_, (\lambda p.\ \_, \_, \mathtt{I}))$.* □

Example 3. *The specification of a queue has state space $S \subseteq T^* \times PC$, where the first component is the abstract queue the second component the program counter where $PC = \{\mathtt{I}, \mathtt{I}'\}$ with* `I` *denoting idle and* `I'` *denoting non-idle. The initial state and set of events are given by $s_0 = (\epsilon, \mathtt{I})$ and Σ (from Example 2), respectively. The set of transitions modifies the queue in the obvious way, transitioning to* `I'` *when an operation is invoked, and returning to* `I` *when an operation returns. As in Example 2, we assume a fixed upper bound on the size of the abstract queue. Note that there are no internal (i.e., τ) transitions in $\mathcal{S}$, but we use the same alphabets in the implementation and specification levels for simplicity. Each final state in $S_\dagger$ matches $(\_, \mathtt{I})$.* □

A *path* of $\mathcal{M}$ is a sequence $\rho = (m_1, e_1, m_2), (m_2, e_2, m_3), \ldots, (m_k, e_k, m_{k+1})$ of consecutive transitions. The path ρ has starting state $start(\rho) = m_1$, ending state $end(\rho) = m_{k+1}$, event string $es(\rho) = e_1 e_2 \ldots e_k$ and label $label(\rho) = obs_\Sigma(es(\rho))$, where $obs_\Sigma(\sigma)$ restricts σ to the observable (i.e., non-τ) events in Σ. We let $Paths(\mathcal{M})$ denote the set of paths of $\mathcal{M}$. The FA $\mathcal{M}$ defines the regular language $L(\mathcal{M})$ of labels of paths that start in m_0 and end in final states. More formally,

$$L(\mathcal{M}) = \left\{ label(\rho) \mid \rho \in Paths(\mathcal{M}) \wedge start(\rho) = m_0 \wedge end(\rho) \in M_\dagger \right\}$$

Given $\sigma \in L(\mathcal{M})$ we let $\mathcal{M}(\sigma)$ denote the set of states of $\mathcal{M}$ that are ending states of paths in $Paths(\mathcal{M})$ that have label σ. Note that because quiescent consistency is a safety property, considering only finite runs is adequate; this restriction is also made by Alur et al. for linearizability [7].

Given an FA $\mathcal{M}$ that represents either a specification or implementation, Σ (the alphabet of $\mathcal{M}$) is the set of events, and so, the language $L(\mathcal{M})$ denotes the possible sequences of events (called *runs*). In this setting, each $\sigma \in L(\mathcal{M})$ of an automaton representing an object is also a possible history of the object.

We use $\mathcal{S} = (S, s_0, \Sigma, t_S, S_\dagger)$ to denote the FA that represents the specification and $\mathcal{Q} = (Q, q_0, \Sigma, t_Q, Q_\dagger)$ to denote the FA that represents the implementation. We will typically use $s_1, \ldots$ for the names of states of $\mathcal{S}$ and $q_1, \ldots$ for the names of states of $\mathcal{Q}$. If $\mathcal{S}$ is the FA for a sequential queue object, it will generate runs like h_2 in Example 1, and if $\mathcal{Q}$ is the FA for the implementation in Figure 2, then it will generate runs such as h_1. In this paper we will be interested in two different problems.

1. Deciding whether a run $\sigma \in L(\mathcal{Q})$ of the implementation is allowed by the specification $\mathcal{S}$. (membership)
2. Deciding whether all runs of $\mathcal{Q}$ are allowed by the specification $\mathcal{S}$ and thus whether $\mathcal{Q}$ is a correct implementation of $\mathcal{S}$. (correctness)

To model concurrent operations, we assume that an operation has separate invoke and return events. We make the following assumption, which is a common restriction used in the literature.

Assumption 1. *The number of processes in the specification and implementation is bounded.*

This assumption is implicitly met by the fact that we use FA $\mathcal{S}$ and $\mathcal{Q}$.

Each event in Σ is associated with a process, an operation, and an input or output value. Like [7], our theory is data independent in the sense that the input and output values are ignored. We simply assume that the event sets of the specification and implementation are equal, and hence, every input/output that is possible for an event of the implementation is also possible for the specification. Given process p, $\Sigma(p)$ denotes the set of events associated with p. We write $e \rightarrowtriangle e'$ to denote that e *matches* e', i.e., e is an invoke event and e' the corresponding response, which holds whenever the process and operation corresponding to e and e' are the same. We let $\pi_p(\sigma)$ denote the run that restricts σ to events of process p, which is defined by

$$\pi_p(\varepsilon) = \varepsilon \qquad \pi_p(e\sigma) = \mathsf{if}\ e \in \Sigma(p)\ \mathsf{then}\ e\pi_p(\sigma)\ \mathsf{else}\ \pi_p(\sigma)$$

The empty run ε is *sequential*. A non-empty run $\sigma = e_0 \ldots e_k$ is *sequential* iff e_0 is an invoke event, for each even $i < k$, $e_i \rightarrowtriangle e_{i+1}$, and if k is even, e_k is an invoke event. σ is *legal* iff for each process p, $\pi_p(\sigma)$ is sequential. Legality ensures that each process calls at most one operation at a time. Furthermore, legality is *prefix closed*, i.e., if σ is legal, then all prefixes of σ are legal.

As is common in the literature, we make the following assumption on each specification object, which essentially means that its operations are atomic.

Assumption 2. *The specification $\mathcal{S}$ is sequential (and hence legal).*

Furthermore, as is common in the literature [7, 8, 1, 15], we ignore the behaviour of clients that use the concurrent object in question, but assume that each client process calls at most one operation of the object it uses at a time (although different client threads may call concurrent operations). This is captured by Assumption 3 below.

Assumption 3. *All runs of the implementation $\mathcal{Q}$ are legal.*

Example 4. *Consider the history h_1 from Example 1. We have that $D_1 \twoheadrightarrow \widehat{D}_1(c)$, $E_2(a) \twoheadrightarrow \widehat{E}_2$, etc. Furthermore, h_1 is legal because $\pi_p(h_1)$ is sequential for each process p.* □

2.3. Independence

In this paper, we study quiescent consistency and explore how it can be represented in terms of *independence* from Mazurkiewicz Trace Theory [12]. Here, a symmetric independence relation $I \subseteq \Sigma \times \Sigma$ is used to define equivalence classes of runs. If $(e, e') \in I$, then consecutive e and e' within a run can be swapped. The independence relation defines a partial commutation — some pairs of elements commute, but there may also be pairs that do not. This leads to an equivalence relation $\sim_I$, where $\sigma \sim_I \sigma'$ iff run σ can be transformed into run σ' via a sequence of rewrites of the form $\sigma_1 e\, e' \sigma_2 \to_I \sigma_1 e'\, e \sigma_2$ for each $(e, e') \in I$.

Example 5. *For h_1 and h_2 in Example 1, if $I = \Sigma \times \Sigma$ then $h_1 \sim_I h_2$.* □

Given a run σ we will let $[\sigma]_I = \{\sigma' \mid \sigma \to_I \sigma'\}$ denote the set of runs that can be produced from σ using zero or more applications of the rewrite rules defined by I. We will let $\mathcal{L}_I(\mathcal{M}) = \cup_{\sigma \in L(\mathcal{M})}[\sigma]_I$ denote the set of runs that can be formed from those in $\mathcal{M}$ using rewrites based on I. We can now state membership and correctness as stated in Section 2.2 more precisely as follows.

1. Deciding whether $\sigma \in \mathcal{L}_I(\mathcal{S})$ for a given $\sigma \in L(\mathcal{Q})$. (membership)
2. Deciding whether $L(\mathcal{Q}) \subseteq \mathcal{L}_I(\mathcal{S})$. (correctness)

N.B., the correctness problem is sometimes referred to as the *model checking* problem. In the next section we explore how problems associated with quiescent consistency can be expressed in this manner, and will see that this requires the FA that represent the specification and implementation to be slightly adapted.

3. Quiescent Consistency

In this section we define quiescent consistency and explore its properties. In Section 3.1, we define quiescent runs and state a number of properties that will be used in the rest of the paper, then in Section 3.2, we present an adaptation of the FA from the previous section to enable reasoning about membership and correctness for quiescent consistency. In Section 3.3, we define quiescent consistency and state the membership and correctness problems in terms of the adapted FA. Sections 4 and 5 then explore these problems.

3.1. Quiescent Runs

We first define quiescent runs and state some properties that we use in the rest of this paper. If $\sigma = \sigma_1 e \sigma_2$ and e is an invocation event, we say e is a *pending invocation* if for all $e' \in \sigma_2$, $e \not\twoheadrightarrow e'$. A run σ is *quiescent* if it does not contain any pending invocations. Thus, if a legal run is quiescent then there is a one-to-one correspondence between invoke and response events. A path $\rho = (q_0, e_1, q_1), (q_1, e_2, q_2), \ldots, (q_{k-1}, e_k, q_k)$ is *quiescent* if $\mathit{label}(\rho)$ is quiescent.

Example 6. *Run h_1 in Example 1 is quiescent, but the run $h_1\, E_1(x)\, D_3\, \widehat{E}_1$ is not because the invocation D_3 is pending. Note that quiescence does not guarantee legality, e.g., runs $\widehat{D}_2(\xi)$ and $D_1\, D_1\, \widehat{D}_2(\xi)\, \widehat{D}_2(\xi)$ are both quiescent, but neither is legal.* □

The following result links quiescence and legality.

Proposition 1. *Suppose $\sigma = \sigma_1 \sigma_2 \ldots \sigma_k$ is a legal and quiescent run, such that each σ_i (for $1 \leq i \leq k$) is a quiescent run. Then for all $1 \leq i \leq k$, σ_i is legal.*

Proof. Suppose σ is a legal quiescent run and $\sigma = \sigma_1 \sigma_2 \ldots \sigma_k$, where each σ_j is quiescent. If $k = 1$ then we are done so assume that $k > 1$. Let σ_i for $1 \leq i \leq k$ be the first subsequence that is not legal, i.e., there exists a process p such that $\pi_p(\sigma_i)$ is non-empty and not sequential. Because legality is prefix closed, $\sigma' = \sigma_1 \ldots \sigma_{i-1}$ must be legal. Moreover, for each process q, $\pi_q(\sigma')$ is either empty, or a non-empty sequential run ending with a return event. Thus for p, we have that $\pi_p(\sigma' \sigma_i)$ is not sequential, which contradicts the assumption that σ is legal. □

We say that a run σ is *end-to-end quiescent* iff it is quiescent and all non-empty proper prefixes of σ are not quiescent. We write $\xi(\sigma)$ to denote σ being end-to-end quiescent. For example, the run h_1 in Example 1 is end-to-end quiescent, and h_2 is quiescent but not end-to-end quiescent. The next result states that any legal quiescent run can be expressed as the concatenation of legal end-to-end quiescent runs.

Proposition 2. *Suppose σ is a legal quiescent run. Then σ can be written in the form $\sigma_1\sigma_2\ldots\sigma_k$ such that each σ_i is a legal end-to-end quiescent run.*

Proof. If σ is end-to-end quiescent, we are done. Otherwise, there must exist σ_1 and σ_2, where $\sigma = \sigma_1\sigma_2$, such that σ_1 is non-empty, legal and end-to-end quiescent, and σ_2 is legal and quiescent. Because σ_2 is quiescent, it is possible to inductively apply the construction above, which completes the proof. □

3.2. Distinguishing Quiescence

We now develop an extension to the FA in Section 2.2 to facilitate reasoning about quiescent consistency in an automata-theoretic setting. Quiescent consistency is defined in terms of quiescent runs and so we will consider the behaviour of the implementation/specification to be its quiescent runs. This assumption is stated formally in terms of FA as follows.

Assumption 4. *A path of $\mathcal{S}$ (and $\mathcal{Q}$) starting from the initial state of $\mathcal{S}$ (and $\mathcal{Q}$) is quiescent iff it ends in a final state of $\mathcal{S}$ (and $\mathcal{Q}$).*

Note that the FA in Examples 2 and 4 satisfy this assumption since the final states for both automata are precisely those in which there are no invoked operations.

By Assumption 2, $\mathcal{S}$ is sequential, and hence, distinguishing its quiescent states is straightforward. The following proposition gives a sufficient condition under which it is possible to partition the state set of $\mathcal{Q}$ into quiescent states and non-quiescent states (and so it is straightforward to ensure that Assumption 4 holds).

Proposition 3. *Suppose that every path of $\mathcal{Q}$ starting from q_0 is a prefix of a legal quiescent path of $\mathcal{Q}$. If ρ is a quiescent path of $\mathcal{Q}$ such that $start(\rho) = q_0$ and $end(\rho) = q$, then all paths of $\mathcal{Q}$ starting from q_0 and ending in q are quiescent.*

Proof. The proof is by contradiction. Assume that there exist ρ, ρ' and q such that paths ρ and ρ' end at q, ρ is quiescent and ρ' is not quiescent. Since ρ' can be completed to form a quiescent path, there must be a path ρ'' from q such that $\rho'\rho''$ is quiescent. Further, ρ'' must contain more responses than invokes. Thus, since ρ is quiescent we can conclude that the path $\rho\rho''$ from the initial state of $\mathcal{Q}$ has more responses than invokes. This provides a contradiction as required, since, by Assumption 3, all histories of $\mathcal{Q}$ are legal. □

Note that in the proof of Proposition 3, it might be necessary to invoke a new operation in order to complete the non-quiescent path ρ under consideration and reach a quiescent state. For example, consider our diffraction queue in Section 2.1, where the `dequeue` operation *blocks* when the queue is empty. Suppose we have a path ρ' such that

$$label(\rho') = D_1\ D_2\ E_3(x)\ \widehat{D}_2(x)\ \widehat{E}_3$$

It is not possible for ρ' to reach a quiescent state by only completing the pending invocations in $label(\rho')$ — the only pending invocation D_1 is blocked because the queue is empty. However, it is possible to reach a quiescent state by following a path where a new enqueue operation is invoked (by some process), and this new operation along with the pending D_1 in $label(\rho')$ is completed by adding matching returns. This observation does not invalidate our results, which only require that we identify the quiescent states.

We now work towards a definition of allowable behaviours for quiescent consistency (Section 3.3), stated in terms of an independence relation (Section 2.3). In particular, by using a special event $\delta \notin \Sigma$ that signifies quiescence, we aim to use the *universal independence relation*:

$$U = \Sigma \times \Sigma$$

which defines a partial commutation that allows *all* events different from δ in a run to commute. Thus, the alphabet of the FA we use is extended to $\Sigma_\delta = \Sigma \cup \{\delta\}$. Note that using U as the independence relation

means that matching invocations and responses of the specification may also be reordered when checking both membership and correctness. However, as is standard in the literature, we have assumed that all runs of the implementation are legal (Assumption 3), and hence, do not generate runs such that a response precedes an invocation, i.e., commutations of a response that is followed by a matching invocation will never be used.

We now consider how we should add δ events to the FA $\mathcal{S}$ (representing the specification) and $\mathcal{Q}$ (representing the implementation) by extending their transition relations, which results in automata $\mathcal{S}_\delta$ and $\mathcal{Q}_\delta$.

First, consider the specification $\mathcal{S}$. One option is to insist that a δ is included in a run *whenever* a quiescent state is reached. However, if we apply this approach to the specification, then the runs of $\mathcal{S}$ will all be of the form $\delta e_1 \widehat{e}_1 \delta e_2 \widehat{e}_2 \delta \ldots$, i.e., a run σ of the implementation can only be equivalent to a run σ' of $\mathcal{S}$ under the partial commutation defined by U if $\sigma = \sigma'$, which is not what is intended under quiescent consistency. This situation is a result of applying the restriction — that one can only reorder between instances of quiescence — to runs of the (sequential) specification; this restriction should only be applied to runs of the implementation. Thus, we should not require a δ to appear in a run of $\mathcal{S}$ whenever a quiescent state is reached. Instead, we rewrite $\mathcal{S}$ to form an FA $\mathcal{S}_\delta$ so that if s is a quiescent state of $\mathcal{S}$ (i.e., after each return event) then there is a self-loop transition (s, δ, s) in $\mathcal{S}_\delta$. These are the only transitions of $\mathcal{S}_\delta$ that have label δ. Overall, we construct $\mathcal{S}_\delta$ such that we *allow* the inclusion of δ whenever a run of $\mathcal{S}$ reaches a quiescent state. This notion is formalised as follows.

Definition 2. *Given a FA $\mathcal{S} = (S, s_0, \Sigma, t, S_\dagger)$ that models a specification, $\mathcal{S}_\delta$ is the FA $(S, s_0, \Sigma, t', S_\dagger)$ in which $t' = t \cup \{(s, \delta, s) \mid s \in S_\dagger\}$.*

Now consider the implementation $\mathcal{Q}$. Here, we must insist that there is a δ in a run of $\mathcal{Q}$ whenever a quiescent state is reached, therefore we rewrite $\mathcal{Q}$ to form an FA $\mathcal{Q}_\delta$ such that if q is a quiescent state of $\mathcal{Q}$ then all transitions that leave q in $\mathcal{Q}_\delta$ have label δ. These are the only transitions of $\mathcal{Q}_\delta$ that have label δ. In particular, for each quiescent state q of $\mathcal{Q}$ we simply add a new state q_δ, make q_δ the initial state of all transitions of $\mathcal{Q}$ that leave q, and add the transition (q, δ, q_δ). We must have that q is a final state of $\mathcal{Q}$, i.e., $q \in Q_\dagger$, and so we make q_δ a final state of $\mathcal{Q}_\delta$ instead of q. Overall, we construct $\mathcal{Q}_\delta$ such that we *require* the inclusion of δ when $\mathcal{Q}$ reaches a quiescent state. Formally, we have the following definition.

Definition 3. *Given FA $\mathcal{Q} = (Q, q_0, \Sigma, t, Q_\dagger)$ that models an implementation, $\mathcal{Q}_\delta$ is the FA $(Q', q_0, \Sigma, t', Q'_\dagger)$ in which*

- $Q' = Q \cup \{q_\delta \mid q \in Q_\dagger\}$
- $t' = t \cup \{(q, \delta, q_\delta) \mid q \in Q_\dagger\} \cup \{(q_\delta, a, q_2) \mid q \in Q_\dagger \wedge (q, a, q_2) \in t\} \setminus \{(q, a, q_2) \in t \mid q \in Q_\dagger\}$
- $Q'_\dagger = \{q_\delta \mid q \in Q_\dagger\}$

The inclusion of δ in runs of $\mathcal{S}$ allows us to compare runs of $\mathcal{S}$ and $\mathcal{Q}$ (once rewritten based on independence relation U).

Example 7. *Returning to runs h_1 and h_2 in Example 1, there are many possible δ extensions of h_2 (which is a run of the specification), for example:*

$$\begin{aligned}
h^\delta_{2.1} &= \delta\ E_3(b)\ \widehat{E}_3\ \delta\ E_2(a)\ \widehat{E}_2\ \delta\ D_4\ \widehat{D}_4(b)\ \delta\ D_5\ \widehat{D}_5(a)\ \delta\ E_6(c)\ \widehat{E}_6\ \delta\ D_1\ \widehat{D}_1(c)\ \delta \\
h^\delta_{2.2} &= \delta\ E_3(b)\ \widehat{E}_3\ E_2(a)\ \widehat{E}_2\ \delta\ D_4\ \widehat{D}_4(b)\ \delta\ D_5\ \widehat{D}_5(a)\ \delta\ E_6(c)\ \widehat{E}_6\ D_1\ \widehat{D}_1(c)\ \delta \\
h^\delta_{2.3} &= \delta\ E_3(b)\ \widehat{E}_3\ E_2(a)\ \widehat{E}_2\ D_4\ \widehat{D}_4(b)\ D_5\ \widehat{D}_5(a)\ E_6(c)\ \widehat{E}_6\ D_1\ \widehat{D}_1(c)\ \delta
\end{aligned}$$

In contrast, there is exactly one δ extension of h_1, namely $\delta\ h_1\ \delta$. If h_2 had been a run of the implementation, then the only δ extension of h_2 is $h^\delta_{2.1}$. □

In addition to adding δ to the runs of $\mathcal{S}$ and $\mathcal{Q}$, we must also reason about runs with δ removed. To this end, we define the following projection

$$\pi_\Sigma(\varepsilon) = \varepsilon \qquad \pi_\Sigma(e\sigma) = \mathsf{if}\ e \in \Sigma\ \mathsf{then}\ e\pi_\Sigma(\sigma)\ \mathsf{else}\ \pi_\Sigma(\sigma)$$

Thus, for example $\pi_\Sigma(\delta h_1 \delta) = h_1$ and $\pi_\Sigma(h^\delta_{2.1}) = h_2$.

3.3. Allowable Quiescent Consistent Behaviours

In this section we formalise what it means for a run of $\mathcal{Q}$ to be allowed by $\mathcal{S}$, and this is stated in terms of the extended automaton $\mathcal{Q}_\delta$. Under quiescent consistency, runs σ and σ' are equivalent if they have the same (multi-)sets of events between two consecutive occurrences of quiescence. As a result, all elements in Σ commute (we do not care about the relative order of these events) but nothing commutes with δ.

Under quiescent consistency a quiescent run σ is allowed by specification $\mathcal{S}$ if σ can be rewritten to form a run of $\mathcal{S}$ by permuting events between consecutive quiescent points. We thus obtain the following definition (note that, due to Proposition 2, it is always possible to write a legal quiescent σ in the form $\sigma = \sigma_1\sigma_2 \ldots \sigma_k$ in which each σ_i is legal and end-to-end quiescent).

Definition 4. *Suppose $\sigma = \sigma_1\sigma_2 \ldots \sigma_k$ is a legal quiescent run and each σ_i is legal and end-to-end quiescent. Then σ is* allowed by $\mathcal{S}$ under quiescent consistency *iff there exists a permutation $\sigma'_i \sim_U \sigma_i$ for each $1 \leq i \leq k$ such that $\sigma'_1\sigma'_2 \ldots \sigma'_k \in L(\mathcal{S})$.*

We now define what it means for a run $\sigma \in L(\mathcal{Q}_\delta)$ to be allowed by a specification $\mathcal{S}$ under quiescent consistency.

Definition 5. *Run $\sigma \in L(\mathcal{Q}_\delta)$ is* allowed *by $\mathcal{S}$ under quiescent consistency if $\pi_\Sigma(\sigma)$ is allowed by $\mathcal{S}$ under quiescent consistency.*

We say σ' is a *legal permutation* of a legal run σ iff $\sigma \sim_U \sigma'$ and σ' is legal.

Proposition 4. *If σ is legal and quiescent, then any legal permutation of σ is quiescent.*

We can now express the membership and correctness problems in terms of $\mathcal{Q}_\delta$ and $\mathcal{S}_\delta$, instead of between $\mathcal{Q}$ and $\mathcal{S}$ as done in Section 2.3.

Lemma 1 (Membership)**.** *Suppose $\sigma \in L(\mathcal{Q}_\delta)$. Then σ is allowed by $\mathcal{S}$ under quiescent consistency iff $\sigma \in \mathcal{L}_U(\mathcal{S}_\delta)$.*

Proof. Suppose $\sigma \in L(\mathcal{Q}_\delta)$. By Proposition 2 and the construction of $\mathcal{Q}_\delta$, we have $\sigma = \delta\sigma_1\delta \ldots \sigma_k\delta$ such that the σ_i do not include δ (i.e., each σ_i is end-to-end quiescent).

First assume that σ is allowed by $\mathcal{S}$ under quiescent consistency. By Definition 5, $\sigma_1\sigma_2 \ldots \sigma_k$ is allowed by $\mathcal{S}$, and hence, by Definition 4 we have that $\mathcal{S}$ has a run $\sigma'_1\sigma'_2 \ldots \sigma'_k$ such that σ'_i is a legal permutation of σ_i (for all $1 \leq i \leq k$). Furthermore, $\mathcal{S}$ is initially quiescent and by Proposition 4, each σ'_i is quiescent, therefore $\mathcal{S}_\delta$ has the run $\sigma' = \delta\sigma'_1\delta\sigma'_2\delta \ldots \delta\sigma'_k\delta$. By definition, $\sigma \in \mathcal{L}_U(\mathcal{S}_\delta)$ as required.

Now assume $\sigma \in \mathcal{L}_U(\mathcal{S}_\delta)$. Then, $L(\mathcal{S}_\delta)$ contains a run $\sigma' = \delta\sigma'_1\delta\sigma'_2\delta \ldots \delta\sigma'_k\delta$ for some $\sigma'_1, \ldots, \sigma'_k$ such that σ'_i is a permutation of σ_i (all $1 \leq i \leq k$). We therefore have that $L(S)$ contains a run $\sigma'_1 \ldots \sigma'_k$ such that σ'_i is a permutation of σ_i (all $1 \leq i \leq k$), and hence, have that σ is allowed by $\mathcal{S}$ as required. □

Lemma 2 (Correctness)**.** *Under quiescent consistency, $\mathcal{Q}$ is a correct implementation of $\mathcal{S}$ iff $L(\mathcal{Q}_\delta) \subseteq \mathcal{L}_U(\mathcal{S}_\delta)$.*

Proof. By Lemma 1 and the definition of quiescent consistency. □

4. The Membership Problem for Quiescent Consistency

In this section we explore the following problem: given a specification $\mathcal{S}$ and run $\sigma \in L(\mathcal{Q}_\delta)$, do we have that $\sigma \in \mathcal{L}_U(\mathcal{S}_\delta)$? We show that this question is in general NP-complete (Section 4.1), but by assuming an upper bound between occurrences of two quiescent states, the question can be solved in polynomial time (Section 4.2).

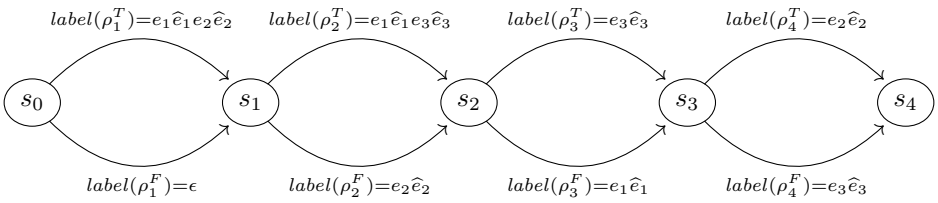

Figure 3: Finite automaton for Example 8

4.1. Unrestricted Quiescent Consistency

We first establish that the membership problem for quiescent consistency is indeed in NP.

Lemma 3. *The membership problem for quiescent consistency is in NP.*

Proof. Given a run $\sigma \in L(\mathcal{Q}_\delta)$ and a specification $\mathcal{S}$, a non-deterministic Turing machine can solve the membership problem of deciding whether $\sigma \in \mathcal{L}_U(\mathcal{S}_\delta)$ as follows. First, the Turing machine guesses a run σ' of $\mathcal{S}_\delta$ with the same length as σ. The Turing machine then guesses a permutation σ'' of σ that is consistent with the independence relation U. Finally, the Turing machine checks whether $\sigma' = \sigma''$. This process takes polynomial time and hence, since a non-deterministic Turing machine can solve the membership problem in polynomial time, the problem is in NP. □

We now prove that this problem is NP-hard by showing how instances of the one-in-three SAT problem can be reduced to it. An instance of the one-in-three SAT problem is defined by boolean variables $v_1, \ldots, v_k$ and clauses $C_1, \ldots, C_n$ where each clause is the disjunction of three literals (a literal is either a boolean variable or the negation of a boolean variable). The one-in-three SAT problem is to decide whether there is an assignment to the boolean variables such that each clause contains *exactly* one true literal and is known to be NP-complete [16].[1]

The construction in the proof of the result below takes an instance of the one-in-three SAT problem defined by boolean variables $v_1, \ldots, v_k$ and clauses $C_1, \ldots, C_n$, then constructs a specification $\mathcal{S}^{k,n}$ that has $k+1$ 'main' states $s_0, \ldots, s_k$. For boolean variable v_i it has two paths from s_{i-1} to s_i: one path ρ_i^T has a matching invocation/response pair $e_j, \widehat{e}_j$ for every clause C_j that contains literal v_i and the other path ρ_i^F has a matching invocation/response pair $e_j, \widehat{e}_j$ for every clause C_j that contains literal $\neg v_i$. The relative order of the pairs of events in ρ_i^F and ρ_i^T will not matter. A path from s_0 to s_k is of the form $\rho_1^{B_1}\rho_2^{B_2}\ldots\rho_k^{B_k}$ for some $B_1, \ldots, B_k \in \{T, F\}$. Furthermore, the number of times that the events e_j and $\widehat{e}_j$ appear in the label of the path is the number of literals in clause C_j that evaluate to true under this assignment of values to $v_1, \ldots, v_k$. As a result, such a path contains e_j and $\widehat{e}_j$ exactly once iff the assignment of B_i to v_i (all $1 \leq i \leq k$) leads to exactly one literal in C_j evaluating to true. Thus, there is a path from s_0 to s_k that contains each e_j and $\widehat{e}_j$ exactly once iff there is a solution to this instance of the one-in-three SAT problem.

Example 8. *Suppose we have four boolean variables $v_1, \ldots, v_4$ and clauses $C_1 = v_1 \vee v_2 \vee \neg v_3$, $C_2 = v_1 \vee \neg v_2 \vee v_4$, and $C_3 = v_2 \vee v_3 \vee \neg v_4$. This leads to the FA shown in Figure 3. In this, for example, the label of ρ_1^T is $e_1\widehat{e}_1e_2\widehat{e}_2$ because C_1 and C_2 both have literal v_1, and the label of ρ_2^F is $e_2\widehat{e}_2$ since C_2 is the only clause that contains literal $\neg v_2$. Consider now the path $\rho_1^T\rho_2^F\rho_3^F\rho_4^F$, which has label $e_1\widehat{e}_1e_2\widehat{e}_2e_2\widehat{e}_2e_1\widehat{e}_1e_3\widehat{e}_3$. The label of this path tells us that if we assign true to v_1 and false to each of v_2, v_3, v_4 then clause C_1 contains two true literals (since e_1 appears twice), C_2 contains two true literals (since e_2 appears twice), and C_3 contains one true literal (since e_3 appears once). Thus, this assignment is not a solution to this instance of the one-in-three SAT problem because more than one clause of C_1 and C_2 evaluates to true.* □

Note that we are not asking whether the clauses can be satisfied, but whether they can be satisfied in a way that makes *exactly* one literal of each true. This is equivalent to asking whether we can make the

[1] Note that the one-in-three SAT problem differs slightly from the more well-known 3SAT problem.

clauses true if they are stated in terms of *isolating disjunction* $\dot{\vee}$, where

$$\dot{\vee}(v_1, v_2, \ldots, v_n) = \bigvee_{1 \leq i \leq n} (v_i \wedge \bigwedge_{j \neq i} \neg v_j)$$

Example 9. *Checking the assignment in Example 8, is equivalent to checking for a satisfying assignment to the clauses* $\dot{C}_1 = \dot{\vee}(v_1, v_2, \neg v_3)$, $\dot{C}_2 = \dot{\vee}(v_1, \neg v_2, v_4)$, *and* $\dot{C}_3 = \dot{\vee}(v_2, v_3, \neg v_4)$, *where, for example,*

$$\dot{C}_1 = (v_1 \wedge \neg v_2 \wedge v_3) \vee (\neg v_1 \wedge v_2 \wedge v_3) \vee (\neg v_1 \wedge \neg v_2 \wedge \neg v_3)$$

We now prove NP-hardness of the membership problem. The proof essentially uses the finite automaton described above and the run $\sigma = e_1 \widehat{e}_1 \ldots e_n \widehat{e}_n$. Additional events are included to allow these events to be reordered. In particular, we add an initial invocation e_0 and a final response $\widehat{e}_0$ in the run and so the implementation is only quiescent in its initial state and at the end of the run. This allows the events of the run to be reordered; without the initial invocation and final response we could only compare σ with runs of the specification in which the pairs $e_i, \widehat{e}_i$ are met in the order found in σ.

Lemma 4. *The membership problem for quiescent consistency is NP-hard.*

Proof. The specification $\mathcal{S}$ is formed from $\mathcal{S}_\delta^{n,k}$ by adding two states s, s', transitions (s_k, e_0, s), and $(s, \widehat{e}_0, s')$ and making s' a final state. Consider the run $\sigma = e_0 e_1 \widehat{e}_1 \ldots e_n \widehat{e}_n \widehat{e}_0$. We prove that σ is in $\mathcal{L}_U(\mathcal{S})$ iff there is a solution to the instance of the one-in-three SAT problem defined by $v_1, \ldots, v_k$ and $C_1, \ldots, C_n$. First note that if $\sigma \in \mathcal{L}_U(\mathcal{S})$ then the corresponding run of $\mathcal{S}$ must end at state s' since σ contains the events e_0 and $\widehat{e}_0$. In addition, σ is end-to-end quiescent and so we simply require that some permutation of σ is in $L(\mathcal{S})$. Thus, $\sigma \in \mathcal{L}_U(\mathcal{S})$ iff $\mathcal{S}$ has a path from s_0 to s_k whose label σ_1 contains each e_i and $\widehat{e}_i$ exactly once for $1 \leq i \leq n$. Furthermore, σ_1 must be the label of a path of $\mathcal{S}$ that is of the form $\rho = \rho_1^{B_1} \rho_2^{B_2} \ldots \rho_k^{B_k}$. Thus, $\sigma \in \mathcal{L}_U(\mathcal{S})$ iff there is an assignment $v_1 = B_1, \ldots, v_k = B_k$ such that each clause $C_1, \ldots, C_n$ contains exactly one true literal. This is the case iff there is a solution to this instance of the one-in-three SAT problem. The result now follows from the one-in-three SAT problem being NP-complete and the construction of $\mathcal{S}$ and σ taking polynomial time. □

The following brings together these results.

Theorem 1. *The membership problem for quiescent consistency is NP-complete.*

4.2. Upper Bound for Restricted Quiescent Consistency

We now consider a restricted version of quiescent consistency that assumes an upper limit on the number of events between two quiescent states. It turns out that the membership problem under this assumption is polynomial with respect to the size of the specification $\mathcal{S}$ and the length of σ. To prove this, we convert the membership problem into the problem of deciding whether two finite automata define a common word, which is a problem that can be solved in polynomial time. In particular, for a given run $\sigma \in L(\mathcal{Q})$, we construct a finite automaton $\mathcal{M}[\sigma]$ (see Definition 7) such that $\sigma \in \mathcal{L}_U(\mathcal{S}_\delta)$ iff $L(\mathcal{M}[\sigma]) \cap L(\mathcal{S})$ is non-empty.

For any run σ, we define a finite automaton $\mathcal{M}_U[\sigma]$ that accepts any permutation of the events in σ. The states of $\mathcal{M}_U[\sigma]$ are multi-sets of events from σ and so in the following $\{$ and $\}$ are used for multi-sets. We use $\uplus$ for multi-set union and $\mathcal{P}(\Sigma)$ for the set of subsets of multi-set Σ.

Definition 6. *Given run* $\sigma = e_1 \ldots e_k$, *we let* $\mathcal{M}_U[\sigma]$ *be the FA* $(\mathcal{P}(\Sigma), \emptyset, \Sigma, t, \{\Sigma\})$ *where* $\Sigma = \{e_1, \ldots, e_k\}$ *and for all* $T, T' \in \mathcal{P}(\Sigma)$, *we have* $(T, e, T') \in t$ *iff* $T' = T \uplus \{e\}$.

Note that the construction $\mathcal{M}_U[\sigma]$ is generic, but we only use it in situations where σ is legal and end-to-end quiescent.

Next, we define $\mathcal{M}[\sigma]$ for runs $\sigma \in L(\mathcal{Q}_\delta)$. We use $L_1 \cdot L_2$ to denote the *language product* of languages L_1 and L_2 and for FA $\mathcal{A}$ and $\mathcal{B}$, we let $\mathcal{A} \cdot \mathcal{B}$ be the FA such that $L(\mathcal{A} \cdot \mathcal{B}) = L(\mathcal{A}) \cdot L(\mathcal{B})$. In what follows, $\mathcal{A}$ only has one final state ($\mathcal{A}$ is $\mathcal{M}_U[\sigma]$ for some σ), and hence, we can construct $\mathcal{A} \cdot \mathcal{B}$ by adding an empty transition from the final state of $\mathcal{A}$ to the initial state of $\mathcal{B}$. Recall that $\xi(\sigma_i)$ denotes σ being end-to-end quiescent.

Definition 7. *For run $\sigma = \delta\sigma_1\delta\sigma_2\delta\ldots\delta\sigma_k\delta$ such that $\xi(\sigma_i)$ for each $1 \leq i \leq k$, we let $\mathcal{M}[\sigma] = \mathcal{M}_U[\sigma_1] \cdot \mathcal{M}_U[\sigma_2] \cdot \ldots \cdot \mathcal{M}_U[\sigma_k]$.*

The next result uses the automata construction in Definition 7 to convert the membership problem into a problem of deciding whether two automata accept a common word. Its proof is clear from the definitions.

Proposition 5. *For any $\sigma \in L(\mathcal{Q}_\delta)$, we have $\sigma \in \mathcal{L}_U(\mathcal{S}_\delta)$ iff $L(\mathcal{M}[\sigma]) \cap L(S) \neq \emptyset$.*

We now arrive at our main result for this section.

Theorem 2. *Suppose that there exists an upper limit $b \in \mathbb{N}$, such that for each $\sigma \in L(\mathcal{Q}_\delta)$ there are at most b events between two occurrences of δ in σ. Then the membership problem for quiescent consistency is in PTIME.*

Proof. By Assumption 4, σ is quiescent, and by Proposition 2 and the definition of $\mathcal{Q}_\delta$, σ can be written as $\sigma = \delta\sigma_1\delta\sigma_2\delta\ldots\delta\sigma_k\delta$, where each σ_i is legal and end-to-end quiescent.

For each σ_i, the size of $\mathcal{M}_U[\sigma_i]$ is exponential in terms of the length of σ_i. If we place an upper limit b on the number of events between two occurrences of quiescence then the size of $\mathcal{M}_U[\sigma_i]$ is polynomial (it is exponential in terms of b). Therefore, $\mathcal{M}[\sigma]$ is of polynomial size (the sum of the sizes of the $\mathcal{M}_U[\sigma_i]$) and the result follows from it being possible to decide whether $L(\mathcal{M}[\sigma]) \cap L(S) \neq \emptyset$ in time that is polynomial in terms of the sizes of $\mathcal{S}$ and $\mathcal{M}[\sigma]$. □

5. The Correctness Problem for Quiescent Consistency

For the correctness problem, we might directly compare $L(\mathcal{Q})$ and $L(\mathcal{S})$, i.e., require that $L(\mathcal{Q}) \subseteq L(\mathcal{S})$. However, this limits the potential for concurrency — $\mathcal{Q}$ would essentially be sequential. The effect of using relaxed notions of correctness (such as quiescent consistency) is that it allows $L(\mathcal{Q})$ to be compared with $L(\mathcal{S})$ using some notion of *observational equivalence*. Therefore, for quiescent consistency, we explore the following problem: given an implementation $\mathcal{Q}$ and specification $\mathcal{S}$, do we have that $L(\mathcal{Q}_\delta) \subseteq \mathcal{L}_U(\mathcal{S}_\delta)$? We show that this question is decidable and coNEXPTIME-complete.

A language is a *rational trace language* if it is defined by a finite automaton and a symmetric independence relation. Decidability of the correctness problem is proved by using the following result from trace theory [17].

Lemma 5. *Suppose $\mathcal{A}$ and $\mathcal{B}$ are FA with set of events Σ and $I \subseteq \Sigma \times \Sigma$ is a symmetric independence relation. Then, the inclusion $\mathcal{L}_I(\mathcal{A}) \subseteq \mathcal{L}_I(\mathcal{B})$ is decidable iff I is transitive.*

The following is an immediate consequence.

Theorem 3. *$L(\mathcal{Q}_\delta) \subseteq \mathcal{L}_U(\mathcal{S}_\delta)$ is decidable.*

Proof. The independence relation $U = \Sigma \times \Sigma$ is transitive. This result thus follows from Lemma 5 and the fact that $L(\mathcal{Q}_\delta) \subseteq \mathcal{L}_U(\mathcal{S}_\delta)$ iff $\mathcal{L}_U(\mathcal{Q}_\delta) \subseteq \mathcal{L}_U(\mathcal{S}_\delta)$. □

We now explore the complexity of the correctness problem, which is equivalent to the complexity of deciding whether the inclusion $\mathcal{L}_U(\mathcal{Q}_\delta) \subseteq \mathcal{L}_U(\mathcal{S}_\delta)$ holds. We show that this problem is coNEXPTIME-hard by considering the problem of deciding inclusion of the set of Parikh images of regular languages. For the rest of this section we assume that $\mathcal{A}$ and $\mathcal{B}$ are FA.

5.1. Lower Bound for Unrestricted Quiescent Consistency

Given alphabet $\Sigma = \{e_1, \ldots, e_k\}$ and $\sigma \in \Sigma^*$, the *Parikh image* of σ is the tuple $(n_1, \ldots, n_k)$ such that σ contains exactly n_i instances of e_i (all $1 \leq i \leq k$). We use $PI(\mathcal{A})$ to denote the set of Parikh images of the runs in $L(\mathcal{A})$ and the inclusion problem for Parikh images is to decide whether $PI(\mathcal{A}) \subseteq PI(\mathcal{B})$. Deciding inclusion for the Parikh images of regular languages is known to be coNEXPTIME-complete [18].

To use the coNEXPTIME-hardness result for Parikh images, we construct FA $\mathcal{A}'$ and $\mathcal{B}'$ from $\mathcal{A}$ and $\mathcal{B}$ such that $PI(\mathcal{A}) \subseteq PI(\mathcal{B})$ iff $\mathcal{L}_U(\mathcal{A}'_\delta) \subseteq \mathcal{L}_U(\mathcal{B}'_\delta)$, where $\mathcal{A}'_\delta$ (and $\mathcal{B}'_\delta$) extends $\mathcal{A}'$ (resp. $\mathcal{B}'$) with δ events and transitions as defined in Section 3.2. Suppose Σ is the alphabet of both $\mathcal{A}$ and $\mathcal{B}$. For each $x \in \Sigma$ we define an invoke event e_x and corresponding response event $\widehat{e}_x$. We also include an additional invoke event e and corresponding response $\widehat{e}$ that do not correspond to any $x \in \Sigma$ and hence, the resulting event set is:

$$\Gamma = \{e, \widehat{e}\} \cup \{e_x \mid x \in \Sigma\} \cup \{\widehat{e}_x \mid x \in \Sigma\}$$

To construct FA $\mathcal{A}'$, we initialise the state set of $\mathcal{A}'$ to the state set A of $\mathcal{A}$ and the event set of $\mathcal{A}'$ to Γ. We then construct the initial state, transitions, and final states of $\mathcal{A}'$ as follows.

1. For the initial state q_0 of $\mathcal{A}$, add a new state $q'_0 \notin A$ to A', make q'_0 the initial state of $\mathcal{A}'$, and add the transition (q'_0, e, q_0) to $\mathcal{A}'$.
2. For each transition $t = (q, x, q')$ in $\mathcal{A}$, add transitions (q, e_x, q_t) and $(q_t, \widehat{e}_x, q')$ in $\mathcal{A}'$, where $q_t \notin A$, then add q_t to A'.
3. Add a state $q_F \notin A$ to A', make this the only final state, and from every final state q of $\mathcal{A}$, add the transition $(q, \widehat{e}, q_F)$.

We have the following relationship between $L(\mathcal{A})$ and $L(\mathcal{A}')$.

Proposition 6. *$x_1 x_2 \dots x_k \in L(\mathcal{A})$ iff $e\, e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \dots e_{x_k} \widehat{e}_{x_k} \widehat{e} \in L(\mathcal{A}')$.*

One important property of $\mathcal{A}'$ is that every $\sigma \in L(\mathcal{A}')$ is end-to-end quiescent. Thus, under quiescent consistency, σ is allowed by the specification of $\mathcal{A}'$ iff some permutation of σ is in the language defined by the specification.

FA $\mathcal{B}'$ is constructed as follows. Initialise the state set of $\mathcal{B}'$ to the state set of $\mathcal{B}$ and the event set of $\mathcal{B}'$ to Γ, then set the initial state of $\mathcal{B}$ as the initial state of $\mathcal{B}'$. Then perform the following.

1. For each transition $t = (q, x, q')$ in $\mathcal{B}$, add transitions (q, e_x, q_t) and $(q_t, \widehat{e}_x, q')$ to $\mathcal{B}'$ for a new state $q_t \notin B'$, then add q_t to B'.
2. Add new states q'' and q_F to B', then for every final state q of $\mathcal{B}$ add transitions (q, e, q'') and $(q'', \widehat{e}, q_F)$ to $\mathcal{B}'$. Finally, make q_F the only final state of $\mathcal{B}'$.

We have the following relationship between $L(\mathcal{B})$ and $L(\mathcal{B}')$.

Proposition 7. *$x_1 x_2 \dots x_k \in L(\mathcal{B})$ iff $e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \dots e_{x_k} \widehat{e}_{x_k} e\, \widehat{e} \in L(\mathcal{B}')$.*

The next lemma links inclusion of Parikh images for $\mathcal{A}$ and $\mathcal{B}$ to inclusion of the languages of $\mathcal{A}'_\delta$ and $\mathcal{B}'_\delta$ under independence relation U.

Lemma 6. *$PI(\mathcal{A}) \subseteq PI(\mathcal{B})$ iff $\mathcal{L}_U(\mathcal{A}'_\delta) \subseteq \mathcal{L}_U(\mathcal{B}'_\delta)$.*

Proof. First assume $PI(\mathcal{A}) \subseteq PI(\mathcal{B})$. Suppose that $\sigma \in \mathcal{L}_U(\mathcal{A}'_\delta)$; it is sufficient to prove that $\sigma \in \mathcal{L}_U(\mathcal{B}'_\delta)$. By Proposition 6 there is some $x_1 x_2 \dots x_k \in L(\mathcal{A})$ such that $\sigma \sim_U \delta\sigma'\delta$, where $\sigma' = e\, e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \dots e_{x_k} \widehat{e}_{x_k}$ $\widehat{e}$. Since $PI(\mathcal{A}) \subseteq PI(\mathcal{B})$ we have that $L(\mathcal{B})$ contains a permutation $y_1 \dots y_k$ of $x_1 \dots x_k$. By Proposition 7, $e_{y_1} \widehat{e}_{y_1} e_{y_2} \widehat{e}_{y_2} \dots e_{y_k} \widehat{e}_{y_k} e\, \widehat{e} \in L(\mathcal{B}')$ and so we also have that $\delta\sigma''\delta \in \mathcal{L}_U(\mathcal{B}'_\delta)$ where $\sigma'' = e_{y_1} \widehat{e}_{y_1} e_{y_2} \widehat{e}_{y_2} \dots e_{y_k} \widehat{e}_{y_k}$ $e\, \widehat{e}$. As $y_1 \dots y_k$ is a permutation of $x_1 \dots x_k$, $\sigma'' \sim_U \sigma'$. Since $\delta\sigma''\delta \in \mathcal{L}_U(\mathcal{B}'_\delta)$ and $\sigma'' \sim_U \sigma'$ we have that $\delta\sigma'\delta \in \mathcal{L}_U(\mathcal{B}'_\delta)$. Thus, since $\sigma = \delta\sigma'\delta$, we have that $\sigma \in \mathcal{L}_U(\mathcal{B}'_\delta)$ as required.

Now assume $\mathcal{L}_U(\mathcal{A}'_\delta) \subseteq \mathcal{L}_U(\mathcal{B}'_\delta)$. Suppose that $\gamma \in PI(\mathcal{A})$ and so there is some $\sigma' = x_1 \dots x_k$ in $L(\mathcal{A})$ with Parikh Image γ. By Proposition 6, $e\, e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \dots e_{x_k} \widehat{e}_{x_k} \widehat{e} \in L(\mathcal{A}')$. Thus, $\delta\, e\, e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \dots e_{x_k} \widehat{e}_{x_k}$ $\widehat{e}\,\delta \in \mathcal{L}_U(\mathcal{A}'_\delta)$. Since $\mathcal{L}_U(\mathcal{A}'_\delta) \subseteq \mathcal{L}_U(\mathcal{B}'_\delta)$, $\delta\, e\, e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \dots e_{x_k} \widehat{e}_{x_k} \widehat{e}\,\delta \in \mathcal{L}_U(\mathcal{B}'_\delta)$. By construction, this implies that $e_{y_1} \widehat{e}_{y_1} e_{y_2} \widehat{e}_{y_2} \dots e_{y_k} \widehat{e}_{y_k} e\, \widehat{e} \in L(\mathcal{B}')$ for some permutation $y_1 \dots y_k$ of $x_1 \dots x_k$. By Proposition 7 we therefore know that $y_1 \dots y_k \in L(\mathcal{B})$. Finally, since $y_1 \dots y_k$ and $x_1 \dots x_k$ are permutations of one another they have the same Parikh Image and so $\gamma \in PI(\mathcal{B})$ as required. □

We therefore have the following result.

Theorem 4. *The correctness problem for Quiescent Consistency is coNEXPTIME-hard.*

Proof. By Lemma 6 and inclusion of Parikh images being coNEXPTIME-hard. □

5.2. Upper Bound for Unrestricted Quiescent Consistency

We now investigate upper bounds on the complexity of deciding correctness of quiescent consistency and show that the problem is in coNEXPTIME. This proof is much more involved than the lower bound result as it is necessary to first derive an algorithm for checking correctness of quiescent consistency (see Algorithm 1) and derive an upper bound on its running time.

We start by introducing some new notation. For $m \in M$ and FA $\mathcal{M} = (M, m_0, \Sigma, t, M_\dagger)$, we let $m \lhd \mathcal{M}$ denote the FA $(M, m, \Sigma, t, M_\dagger)$ formed by replacing the initial state of $\mathcal{M}$ by m. Furthermore, for $M' \subseteq M$ (recalling that $\xi(\sigma)$ denotes that σ is end-to-end quiescent), we define:

$$\mathsf{Z}_{\mathcal{M}}(m) = \{\sigma \in \mathcal{L}_U(m \lhd \mathcal{M}) \mid \xi(\sigma)\} \qquad \mathsf{Z}_{\mathcal{M}}(M') = \bigcup_{m \in M'} \mathsf{Z}_{\mathcal{M}}(m)$$

Thus, $\mathsf{Z}_{\mathcal{M}}(m)$ is the set of end-to-end quiescent runs that start in state m of $\mathcal{M}$. The following is immediate from this definition.

Proposition 8. *If $\mathcal{Q}$ is a correct implementation of $\mathcal{S}$ with respect to quiescent consistency and q_0 and s_0 are the initial states of $\mathcal{Q}$ and $\mathcal{S}$ respectively then $\mathsf{Z}_{\mathcal{Q}}(q_0) \subseteq \mathsf{Z}_{\mathcal{S}}(s_0)$.*

We will use an implicit powerset construction when reasoning about quiescent consistency. Given states $m, m' \in M$ of $\mathcal{M}$, sets of states $M_1, M_2 \subseteq M$ and a run σ, we define some further notation:

$$m \xrightarrow{\sigma}_{\mathcal{M}} m' \text{ iff } \exists \rho \in \mathit{Paths}(\mathcal{M}) \,.\, \mathit{start}(\rho) = m \wedge \mathit{end}(\rho) = m' \wedge \mathit{label}(\rho) \in [\sigma]_U$$
$$M_1 \xRightarrow{\sigma}_{\mathcal{M}} M_2 \text{ iff } \forall \rho \in \mathit{Paths}(\mathcal{M}) \,.\, \mathit{start}(\rho) \in M_1 \wedge \mathit{label}(\rho) \in [\sigma]_U \Rightarrow \mathit{end}(\rho) \in M_2$$

Thus, $m \xrightarrow{\sigma}_{\mathcal{M}} m'$ holds iff there is some path in $\mathcal{M}$ with label in $[\sigma]_U$ from state m to state m'. Furthermore, $M_1 \xRightarrow{\sigma}_{\mathcal{M}} M_2$ holds iff every path of $\mathcal{M}$ starting from a state in M_1 with label in $[\sigma]_U$ ends in a state of M_2.

If $\mathcal{Q}$ is not a correct implementation of $\mathcal{S}$ with respect to quiescent consistency then there must be a quiescent run σ that demonstrates this. We will use the following result, which shows that if there is a counterexample to quiescent consistency then there is one of the form $\sigma = \sigma_1 \dots \sigma_{k+1}$ (where $\xi(\sigma_i)$) such that σ_{k+1} is the portion of σ that is in $\mathcal{Q}$ but not in $\mathcal{S}$ (under independence relation U) and k is bounded by $|Q| \cdot 2^{|S|}$.

Proposition 9. *$\mathcal{Q}$ is not a correct implementation of $\mathcal{S}$ under quiescent consistency iff there exists some run $\sigma = \sigma_1 \dots \sigma_k$ for end-to-end quiescent $\sigma_1, \dots, \sigma_k$ and corresponding pairs $(q_0, S_0), (q_1, S_1), \dots, (q_k, S_k) \in Q \times 2^S$ such that $S_0 = \{s_0\}$, $q_{i-1} \xrightarrow{\sigma_i}_{\mathcal{Q}} q_i$ and $S_{i-1} \xRightarrow{\sigma_i}_{\mathcal{S}} S_i$ (for all $1 \leq i \leq k$), $S_k = \emptyset$, and $k < |Q| \cdot 2^{|S|}$.*

Proof. The existence of such a σ demonstrates that $\mathcal{Q}$ is not a correct implementation of $\mathcal{S}$ under quiescent consistency since the condition $S_k = \emptyset$ tells us that $\sigma \notin \mathcal{L}_U(\mathcal{S})$. As a result, it is sufficient to prove the left-to-right direction. We therefore assume that $\mathcal{Q}$ is not a correct implementation of $\mathcal{S}$ under quiescent consistency. Thus, there exists a quiescent run σ that is in $L(\mathcal{Q})$ but not in $\mathcal{L}_U(\mathcal{S})$. Assume that we have a shortest such run σ, $\sigma = \sigma_1 \dots \sigma_k$ for end-to-end quiescent $\sigma_1, \dots, \sigma_k$. Since σ is in $L(\mathcal{Q})$ but not in $\mathcal{L}_U(\mathcal{S})$, by the minimality of σ we must have that $\sigma_k \in \mathsf{Z}_{\mathcal{Q}}(q_{k-1})$ and $\sigma_k \notin \mathsf{Z}_{\mathcal{S}}(S_{k-1})$ and so $S_k = \emptyset$. Further, by the minimality of σ we must have that $(q_i, S_i) \neq (q_j, S_j)$, all $0 \leq i < j \leq k$; otherwise we can remove $\sigma_i \dots \sigma_{j-1}$ from σ to obtain a shorter run that is in $\mathcal{L}_U(\mathcal{Q})$ but not in $\mathcal{L}_U(\mathcal{S})$. But, there are $|Q| \cdot 2^{|S|}$ possible pairs and so the condition, $k < |Q| \cdot 2^{|S|}$, must hold. □

Using Proposition 9, we develop Algorithm 1, which defines a non-deterministic Turing Machine that solves the problem of deciding whether Q is not correct. At each iteration, the non-deterministic Turing Machine increments the counter c and guesses a next pair (q_c, S_c). It then checks that there is some σ_c such that $q_{c-1} \xrightarrow{\sigma_c}_{\mathcal{Q}} q_c$ and $S_{c-1} \xRightarrow{\sigma_c}_{\mathcal{S}} S_c$. If there is such a σ_c then the process can continue, otherwise the result is inconclusive. If the process continues and $S_c = \emptyset$ then the conditions of Proposition 9 have been satisfied so we have shown that $\mathcal{Q}$ is not a correct implementation of $\mathcal{S}$ with respect to quiescent consistency. The

Algorithm 1 Deciding incorrectness for quiescent consistency

$c = 0$, $S_0 = \{s_0\}$, $Q_0 = \{q_0\}$
while $c \leq |Q| \cdot 2^{|S|}$ **do**
  $c = c + 1$
  Choose some $(q_c, S_c) \in Q \times 2^S$
  if $\nexists \sigma_c$ such that $q_{c-1} \xrightarrow{\sigma_c}_{\mathcal{Q}} q_c$ and $S_{c-1} \overset{\sigma_c}{\Longrightarrow}_{\mathcal{S}} S_c$ **then**
    Terminate without having shown incorrectness
  end if
  if $S_{c-1} = \emptyset$ **then**
    Return Fail
  end if
end while
Terminate without having shown incorrectness

bound on c ensures that the algorithm terminates as long as we can decide the condition contained in the first **if** statement (we explore this below).

If a non-deterministic Turing Machine operates as above then it will return Fail if there is some sequence of choices that shows incorrectness. The following is thus immediate from Proposition 9.

Proposition 10. *If a non-deterministic Turing Machine applies Algorithm 1 to $\mathcal{Q}$ and $\mathcal{S}$ then it returns Fail iff $\mathcal{Q}$ is not a correct implementation of $\mathcal{S}$ with respect to quiescent consistency.*

We now consider the problem encoded in the first **if** statement of Algorithm 1: the problem of deciding, for states q_{c-1}, q_c of $\mathcal{Q}$ and sets S_{c-1}, S_c of states of $\mathcal{S}$, whether there exists some σ_c that can

(i) take $\mathcal{Q}$ from q_{c-1} to q_c and
(ii) take $\mathcal{S}$ from the set S_{c-1} of states to (a subset of) the set S_c of states.

We introduce some further notation. For $m \in M$ and $M' \subseteq M$, we let $m \lhd \mathcal{M} \rhd M'$ denote the FA (M, m, Σ, t, M') formed by making m the initial state of $\mathcal{M}$ and M' the final states. We introduce the following (assuming all states in M' and M'' are quiescent).

$$\mathsf{Z}_{\mathcal{M}}(m, M') = \{\sigma \in \mathcal{L}_U(m \lhd \mathcal{M} \rhd M') \mid \xi(\sigma)\} \qquad \mathsf{Z}_{\mathcal{M}}(M', M'') = \bigcup_{m \in M'} \mathsf{Z}_{\mathcal{M}}(m, M'')$$

That is, $\mathsf{Z}_{\mathcal{M}}(m, M'')$ is the set of end-to-end quiescent runs of $\mathcal{M}$ that start in state m and end at a state in M''. We use shorthand $\mathsf{Z}_{\mathcal{M}}(m, m')$ for $\mathsf{Z}_{\mathcal{M}}(m, \{m'\})$ (similarly $\mathsf{Z}_{\mathcal{M}}(M', m')$).

Using this notation, condition (i) above may be formalised as the predicate $\sigma_c \in \mathsf{Z}_{\mathcal{Q}}(q_{c-1}, q_c)$. Condition (ii) above requires that σ_c cannot take $\mathcal{S}$ from S_{c-1} to any state outside of S_c (and so that $\sigma_c \notin \bigcup_{s \in (S \setminus S_c)} \mathsf{Z}_{\mathcal{S}}(S_{c-1}, s)$). The overall condition thus reduces to the following.

$$\mathsf{Z}_{\mathcal{Q}}(q_{c-1}, q_c) \not\subseteq \left(\bigcup_{s \in (S \setminus S_c)} \mathsf{Z}_{\mathcal{S}}(S_{c-1}, s)\right)$$

The problem, that represents the main **if** statement of the algorithm, thus involves checking whether the Parikh Image of one regular language is (not) contained in the Parikh Image of another regular language. It is known that this problem can be solved in non-deterministic exponential time (NEXPTIME) [19].

Proposition 11. *It is possible to decide whether there exists run σ_c such that $q_{c-1} \xrightarrow{\sigma_c}_{\mathcal{Q}} q_c$ and $S_{c-1} \overset{\sigma_c}{\Longrightarrow}_{\mathcal{S}} S_c$ in NEXPTIME.*

We can now determine the complexity of the correctness problem for quiescent consistency.

Theorem 5. *The correctness problem for quiescent consistency is coNEXPTIME-complete.*

Proof. From Theorem 4 we know that the problem is coNEXPTIME-hard and so we focus on showing that it is in coNEXPTIME.

Proposition 10 tells us that a non-deterministic Turing Machine can use Algorithm 1 to check whether correctness does not hold. Further, by Proposition 11, the condition of the first **if** statement can be decided in NEXPTIME. In addition, the number of iterations is at worst exponential and so a non-deterministic Turing Machine can apply Algorithm 1 in exponential time. As a result, the problem of checking that quiescent consistency does not hold is in NEXPTIME and so correctness for quiescent consistency is in coNEXPTIME. □

5.3. Upper Bound for Restricted Quiescent Consistency

We now consider the case where there is a limit b on the lengths of subsequences of runs of $\mathcal{Q}$ between two occurrences of quiescence.

Proposition 12. *Let us suppose that there is a bound on the length of end-to-end quiescent runs in $\mathcal{Q}$ and $\mathcal{S}$. It is possible to decide whether there exists run σ_c such that $q_{c-1} \xrightarrow{\sigma_c}_{\mathcal{Q}} q_c$ and $S_{c-1} \overset{\sigma_c}{\Longrightarrow}_{\mathcal{S}} S_c$ in NP.*

Proof. A nondeterministic Turing Machine can solve this problem in polynomial time as follows. First, it guesses a run σ whose length is at most the upper bound and checks that σ is end-to-end quiescent. It then checks whether

$$\sigma \in \mathsf{Z}_{\mathcal{Q}}(q_{c-1}, q_c) \quad \text{and} \quad \sigma \notin \mathsf{Z}_{\mathcal{S}}(S_{c-1}, S \setminus S_c).$$

We know that the first check (solving the membership problem for bounded quiescent consistency) can be performed in polynomial time. The second check, of deciding whether $\sigma \notin \mathsf{Z}_{\mathcal{S}}(S_{c-1}, S \setminus S_c)$ can again be performed in polynomial time. The nondeterministic Turing Machine returns True iff it finds that $\sigma \in \mathsf{Z}_{\mathcal{Q}}(q_{c-1}, q_c)$ and $\sigma \notin \mathsf{Z}_{\mathcal{S}}(S_{c-1}, S \setminus S_c)$. □

We now prove that correctness is PSPACE-complete for bounded quiescent consistency. Note that although the above propositions demonstrate that the steps of Algorithm 1 are in NP, this does not imply that the overall problem is in NP since the steps may have to be applied exponentially many times.

Theorem 6. *The correctness problem for bounded quiescent consistency is PSPACE-complete.*

Proof. From Proposition 12 we know that the condition in the first **if** statement in Algorithm 1 can be decided in NP. Thus, a nondeterministic Turing Machine can apply Algorithm 1 using polynomial space. We therefore have that the problem is in PSPACE.

We now show that the problem is PSPACE-hard. If the implementation is sequential then correctness corresponds exactly to the inclusion of the regular languages recognised by $\mathcal{Q}$ and $\mathcal{S}$. In addition, there is a bound of 2 on the lengths of subsequences of runs of $\mathcal{Q}$ between two occurrences of quiescence. The result thus follows from it being possible to represent any instance of regular language inclusion in this way and regular language inclusion being PSPACE-hard [20]. □

6. Quiescent Sequential Consistency

In this section, we consider quiescent sequential consistency, which adds a *sequential consistency* constraint [9] to quiescent consistency, i.e., we are not allowed to reorder the events of the same process. For concurrent objects, this means that the order of effects of operation calls by the same process will take place in program order: if operation calls identified by events e, $\widehat{e}$ and e', $\widehat{e}'$ all have the same process, and a concrete implementation has a run where $\widehat{e}$ occurs before e', then such a trace cannot be justified by a sequential run where e' occurs before $\widehat{e}$.

In the context of client-object systems, sequential consistency has been shown to be equivalent to *observational refinement* [10] provided that the client threads are independent (i.e., do not share data) [11]. Observational refinement provides the conditions necessary for replacing a specification object within a client program by an implementation. The sorts of guarantees that quiescent consistency provides a client

is still a subject of further study; as Shavit says, exploiting concurrency in the multiprocessor age requires a rethinking of traditional notions of correctness [4].

We now present some background for quiescent sequential consistency in preparation for the membership and correctness problems. In order to formally define quiescent sequential consistency, we define a projection function that also preserves δ states in the projection.

Definition 8. *Given run $\sigma \in \Sigma_\delta^*$, event $e \in \Sigma_\delta$ and process p, $\pi_p^\delta(\sigma)$ is defined by the following:*

$$\pi_p^\delta(\varepsilon) = \varepsilon \qquad \pi_p^\delta(e\sigma) = \mathsf{if}\ e \in \Sigma(p) \cup \{\delta\}\ \mathsf{then}\ e\pi_p^\delta(\sigma)\ \mathsf{else}\ \pi_p^\delta(\sigma)$$

For $\sigma, \sigma' \in \Sigma_\delta^*$, we write $\sigma \approx \sigma'$ iff $\pi_p^\delta(\sigma) = \pi_p^\delta(\sigma')$ for every process p.

We can now define quiescent sequential consistency in a similar manner to quiescent consistency, except that we include the constraint that events on a process are ordered.

Definition 9. *Suppose $\sigma = \sigma_1\sigma_2 \ldots \sigma_m \in \Sigma^*$ is a quiescent run and each σ_i is end-to-end quiescent. Then σ is* allowed by specification $\mathcal{S}$ under quiescent sequential consistency *iff there exists a permutation σ_i' for each $1 \leq i \leq m$ such that $\sigma_i \approx \sigma_i'$ and $\sigma_1'\sigma_2' \ldots \sigma_m'$ is a run of $\mathcal{S}$.*

Since we cannot reorder events on a process we obtain the following independence relation.

$$R = \{(a, b) \mid \exists p, p'.p \neq p' \wedge a \in \Sigma(p) \wedge b \in \Sigma(p')\}$$

The essential idea is that quiescence (δ) does not commute with anything, as with quiescent consistency, and that two events from Σ are independent if and only if they are on different processes.

Given an FA M with alphabet Σ_δ, we will use $\mathcal{L}_R(M)$ to denote $\mathcal{L}_I(M)$ in which the independence relation I is R.

We will use the FA $\mathcal{Q}_\delta$ and $\mathcal{S}_\delta$, which are derived from the implementation $\mathcal{Q}$ and specification $\mathcal{S}$, respectively, via the construction described in Section 3.2. If we consider a quiescent run σ of $\mathcal{Q}_\delta$ we have that δ is included whenever σ is quiescent. We can define what it means for a run that includes δ to be allowed by $\mathcal{S}$.

Definition 10. *Run σ of $\mathcal{Q}_\delta$ is* allowed *by $\mathcal{S}$ under quiescent sequential consistency if the run $\pi_\Sigma(\sigma)$ formed from σ by removing all instances of δ is allowed by $\mathcal{S}$ under quiescent sequential consistency.*

Recall also that all processes observe quiescence. As a result, we have the following property.

Lemma 7. *Suppose $\sigma = \sigma_1\sigma_2 \ldots \sigma_m$ is a quiescent run and each σ_i is end-to-end quiescent. Given run $\sigma' \in \Sigma_\delta^*$ we have that $\sigma' \approx \sigma$ iff $\sigma' = \sigma_1'\sigma_2' \ldots \sigma_m'$ for some $\sigma_1', \ldots, \sigma_m'$ with $\sigma_j \approx \sigma_j'$ (all $1 \leq j \leq m$).*

Based on Definition 10, this leads directly to the following simplified ways of expressing when a run is allowed under quiescent sequential consistency.

Proposition 13. *Suppose $\sigma \in L(\mathcal{Q}_\delta)$ is a quiescent run. Then the following statements are equivalent:*

1. *σ is allowed by $\mathcal{S}$ under quiescent sequential consistency.*
2. *There exists a $\sigma' \in L(\mathcal{S}_\delta)$ such that $\sigma' \approx \sigma$.*
3. *$\sigma \in \mathcal{L}_R(\mathcal{S}_\delta)$.*

The following lemma links quiescent consistency and quiescent sequential consistency.

Lemma 8. *If $\sigma \in L(\mathcal{Q}_\delta)$ is allowed by $\mathcal{S}$ under quiescent sequential consistency then σ is allowed by $\mathcal{S}$ under quiescent consistency, but not vice-versa.*

Proof. The first part follows from the independence relation R for quiescent sequential consistency being a subset of the independence relation U for quiescent consistency. To prove the second part it is sufficient to obtain a run σ and specification $\mathcal{S}$ such that σ is allowed by $\mathcal{S}$ for quiescent consistency but not for quiescent sequential consistency. Suppose $\mathcal{Q}$ allows run $\sigma = e\, e_1 e_2 \widehat{e}_1 \widehat{e}_2 \widehat{e}$ where events e_1, e_2, $\widehat{e}_1$ and $\widehat{e}_2$ are on the same process p and $\mathcal{S}$ allows the run $\sigma' = e\, \widehat{e}\, e_1 \widehat{e}_1 e_2 \widehat{e}_2$ but no other permutation of σ. Then σ' is allowable under quiescent consistency, but not under quiescent sequential consistency. □

7. Membership for Quiescent Sequential Consistency

In this section we consider the membership problem for quiescent sequential consistency. Some of the results are similar to those for quiescent consistency, and hence, the proofs for these results are elided. The structure of this section is similar to Section 4 — we first present the unrestricted case (Section 7.1), then present the upper bounds for the restricted cases (Section 7.2).

7.1. Unrestricted Quiescent Sequential Consistency

The unrestricted version of quiescent sequential consistency is NP-complete. First, we show that the problem is in NP, then show that the problem is NP-hard.

Lemma 9. *The membership problem for quiescent sequential consistency is in NP.*

Proof. Given run σ and specification $\mathcal{S}$, a non-deterministic Turing machine can solve the membership problem, of deciding whether $\sigma \in \mathcal{L}_R(\mathcal{S}_\delta)$, as follows. First, the Turing machine guesses a run σ' of $\mathcal{S}_\delta$ with the same length as σ. The Turing machine then guesses a permutation σ'' of σ that is consistent with the independence relation R. Finally, the Turing machine checks whether $\sigma'' = \sigma'$. This process takes polynomial time and so, since a non-deterministic Turing machine can solve the membership problem in polynomial time, the problem is in NP. □

We can adapt the proof, that the membership problem for quiescent consistency is NP-hard (Lemma 4), by simply having a separate process for each invoke/response pair. We therefore have the following.

Lemma 10. *The membership problem for quiescent sequential consistency is NP-hard.*

Theorem 7. *The membership problem for quiescent sequential consistency is NP-complete.*

7.2. Restricted Quiescent Sequential Consistency

We now adapt the approach developed for quiescent consistency, to show that the membership problem can be solved in polynomial time if we either have an upper limit on the number of events between any two instances of quiescence, or on the number of processes in the system.

Upper limit on number of events between quiescence. Similar to Definition 6, we construct a finite automaton that accepts any permutation of run σ that preserves the order of events within a single process. We also assume that no event in σ is repeated since it is helpful to distinguish between the different occurrences of an event; an alternative would be to use multi-sets but this would lead to slightly more complex definitions. This requirement, that events are distinct, is not a restriction since one can simply label events, removing these labels once an FA has been formed.

For a run $\sigma = e_1 \dots e_k$, $1 \leq i \leq k$ and process p, we let

$$pre_p(\sigma, i) = \{e_j \mid 1 \leq j < i \wedge e_j \in \Sigma(p)\}$$

be the set of elements of σ with index smaller than i that are part of process p.

Definition 11. *Given run $\sigma_i = e_1 \dots e_k$, in which each e_i is distinct, we let $\mathcal{M}_R[\sigma_i]$ be the finite automaton $(2^\Sigma, \emptyset, \Sigma, t, \{\Sigma\})$ such that:*

- *$\Sigma = \{e_1, \dots, e_k\}$ and,*
- *for all $T, T' \in 2^\Sigma$ and $e_i \in \Sigma(p)$, we have $(T, e_i, T') \in t$ iff $e_i \notin T$, $T' = T \cup \{e_i\}$ and $pre_p(\sigma, i) \subseteq T$.*

Using this definition, we obtain a new FA $\mathcal{M}'[\sigma]$.

Definition 12. *Given run $\sigma = \sigma_1\sigma_2 \dots \sigma_k \in \Sigma^*$ such that each σ_i is end-to-end quiescent,*

$$\mathcal{M}'[\sigma] = \mathcal{M}_R[\sigma_1] \cdot \mathcal{M}_R[\sigma_2] \cdot \ldots \cdot \mathcal{M}_R[\sigma_k].$$

The following is clear from the definition and from Proposition 13.

Lemma 11. *Given run σ and specification $\mathcal{S}$, $\sigma \in \mathcal{L}_R(\mathcal{S}_\delta)$ iff $L(\mathcal{M}'[\sigma]) \cap L(S) \neq \emptyset$.*

As before, we have the following result.

Theorem 8. *If b is an upper limit on the number of events between two occurrences of quiescence in each run of $\mathcal{Q}$, then the membership problem for quiescent sequential consistency is in PTIME.*

Upper limit on number of processes. We now consider the membership problem for the case in which there is a fixed upper limit on the number of processes. Note that this notion is not covered by Assumption 1, which states that the number of processes for any particular implementation or specification is bounded. The results here state that if we place an upper bound on the number of processes, with that bound being applied to all specifications and implementations being considered, then the set of membership problems that satisfy this bound can be solved in polynomial time.

As before, we start by defining an FA whose language is $[\sigma]_R$. Given some σ_i, the basic idea is that the state of the FA will be a tuple that, for each process p, records the most recent event on p. Thus, a state q will be represented by a tuple of events (the most recent events observed on each process) and an event a on process p will only be possible in state q if the event that immediately precedes a on p is in this tuple. Again, we assume that events are distinct and note that this can be ensured by adding labels to events.

Definition 13. *Suppose that run $\sigma_i = e_1 \dots e_k$, in which each e_i is distinct, has projection $\pi_p(\sigma_i) = e^p_1 \dots e^p_{k_p}$ on process p ($1 \leq p \leq n$). We let $\mathcal{M}_V[\sigma_i]$ be the FA (T, q_0, Σ, t, F) such that*

- $\Sigma = \{e_1, \dots, e_k\}$,
- $T = \{e^1_0, e^1_1, \dots, e^1_{k_1}, \varepsilon\} \times \{e^2_0, e^2_1, \dots, e^2_{k_2}, \varepsilon\} \times \dots \times \{e^n_0, e^n_1, \dots, e^n_{k_n}, \varepsilon\}$,
- $q_0 = (e^1_0, \dots, e^n_0)$,
- $F = \{(e^1_{k_1}, e^2_{k_2}, \dots, e^n_{k_n})\}$, *and*
- $(T, a, T') \in t$ *for* $a \in \Sigma(p)$ *if and only if the following hold:* $T = (e^1_{j_1}, e^2_{j_2}, \dots, e^n_{j_n})$, $j_p < k_p$, $a = e^p_{j_p+1}$, *and* $T' = (e^1_{j_1}, e^2_{j_2}, \dots, e^p_{j_p+1}, \dots, e^n_{j_n})$.

The following defines $\mathcal{M}''[\sigma]$.

Definition 14. *Given run $\sigma = \sigma_1\sigma_2 \dots \sigma_k$ such that each σ_i is end-to-end quiescent,*

$$\mathcal{M}''[\sigma] = \mathcal{M}_V[\sigma_1] \cdot \mathcal{M}_V[\sigma_2] \cdot \dots \cdot \mathcal{M}_V[\sigma_k].$$

Lemma 12. *Given run σ and specification $\mathcal{S}$, $\sigma \in \mathcal{L}_R(\mathcal{S}_\delta)$ iff $L(\mathcal{M}''[\sigma]) \cap L(S) \neq \emptyset$.*

The important point now is that the state set of $\mathcal{M}_V[\sigma_i]$ has size that is exponential in terms of the number of processes but if the number of processes is bounded then the size is polynomial in terms of the length of σ. In particular, if there is an upper bound b on the number of processes then $\mathcal{M}_V[\sigma_i]$ has at most $|\sigma_i|^b$ states. We therefore obtain the following result.

Theorem 9. *If there is an upper limit on the number of processes for each run of $\mathcal{Q}$ then the membership problem for quiescent sequential consistency is in PTIME.*

8. Correctness for Quiescent Sequential Consistency

This section now presents decidability and complexity results for quiescent sequential consistency. Following the pattern of the previous sections, we present unrestricted quiescent sequential consistency (Section 8.1), and then restricted quiescent sequential consistency (Section 8.2).

8.1. Unrestricted Quiescent Sequential Consistency

If we consider a system with two processes 1 and 2 such that $e_1, e_2 \in \Sigma(1)$ and $e_3 \in \Sigma(2)$ then we have that: $(e_1, e_3) \in R$, $(e_2, e_3) \in R$ but $(e_1, e_2) \notin R$. Therefore, the independence relation is not transitive and so we expect correctness to be undecidable (Lemma 5). It could, however, be argued that we might have changed the nature of the problem by placing restrictions on the structure of the specification. In this section we therefore prove that, as expected, the correctness problem is undecidable for quiescent sequential consistency. The proof will be based on showing how an instance of Post's Correspondence Problem can be reduced to an instance of the correctness problem for quiescent sequential consistency.

Definition 15. *Given alphabet* Γ *and sequences* $\alpha_1, \ldots, \alpha_n \in \Gamma^*$ *and* $\beta_1, \ldots, \beta_n \in \Gamma^*$*, Post's Correspondence Problem (PCP) is to decide whether there is a non-empty sequence* $i_1 \ldots i_k \in [1, n]$ *of indices such that* $\alpha_{i_1} \ldots \alpha_{i_k} = \beta_{i_1} \ldots \beta_{i_k}$.

Post's Correspondence Problem is known to be undecidable [21].

First we explain how the proof operates. Given an instance of the PCP defined by sequences $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$, we will construct FA $\mathcal{Q}[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n]$ that will act as the implementation. The quiescent runs of this FA will have a particular form: a quiescent run σ will be defined by a sequence $i_1 \ldots i_k \in [1, n]$ of indices, the projection of σ on process 1 will correspond to $\alpha_{i_1} \ldots \alpha_{i_k}$ and the projection of σ on process 2 will correspond to $\beta_{i_1} \ldots \beta_{i_k}$. Thus, there is a solution to this instance of the PCP if and only if $\mathcal{Q}[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n]$ has a non-empty quiescent trace σ such that the projections on the two processes define the same sequences in Γ^*. We then define a specification $\mathcal{S}$ that allows the set of runs in which the projections on the two processes differ (plus the empty sequence). When brought together, $\mathcal{Q}[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n]$ is not a correct implementation of $\mathcal{S}$ under quiescent sequential consistency if and only if $\mathcal{Q}[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n]$ has a quiescent run whose projections on the two processes define the same sequences in Γ^* and this is the case if and only if there is a solution to this instance of the PCP. As a result, correctness under quiescent sequential consistency being undecidable follows from the PCP being undecidable.

We now construct the FA $\mathcal{Q}[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n]$. There will be two processes and for each $p \in \{1, 2\}$ and letter a in the alphabet Γ used, we will create an invoke event e_a^p and a response event $\widehat{e}_a^{\,p}$. We will add two additional events: a matching invoke e and response $\widehat{e}$ for process 1. (This choice of process 1 for e and $\widehat{e}$ is is arbitrary, i.e., we could have chosen e and $\widehat{e}$ to be events of process 2 as well). Thus, we have that:

$$\Sigma = \{e_a^p \mid a \in \Gamma \wedge p \in \{1, 2\}\} \cup \{\widehat{e}_a^{\,p} \mid a \in \Gamma \wedge p \in \{1, 2\}\} \cup \{e, \widehat{e}\}$$

We define a mapping from a sequence in Γ^* and process number p to a sequence in Σ^* as follows (in which $a \in \Gamma$ and $\gamma \in \Gamma^*$).

$$to_\Sigma(\varepsilon, p) = \varepsilon \qquad to_\Sigma(a\gamma, p) = e_a^p \; \widehat{e}_a^{\,p} \; to_\Sigma(\gamma, p)$$

Then, we define an equivalence relation on sequences in Σ^* that essentially 'ignores' the process number.

$$\begin{array}{rcl} \varepsilon & \equiv & \varepsilon \\ e_a^p \, \sigma_1 & \equiv & e_b^q \, \sigma_2 \text{ if and only if } a = b \text{ and } \sigma_1 \equiv \sigma_2 \\ \widehat{e}_a^{\,p} \, \sigma_1 & \equiv & \widehat{e}_b^{\,q} \, \sigma_2 \text{ if and only if } a = b \text{ and } \sigma_1 \equiv \sigma_2 \end{array}$$

From the initial state q_0 of $\mathcal{Q}[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n]$ there is a transition with label e to state q. For all $1 \leq i \leq n$ there is a path from q to state q' with the following label:

$$to_\Sigma(\alpha_i, 1) \, to_\Sigma(\beta_i, 2)$$

Similarly, for all $1 \leq i \leq n$ there is a path from q' to state q' with label $to_\Sigma(\alpha_i, 1) \, to_\Sigma(\beta_i, 2)$. Finally, there is a transition from q' with label $\widehat{e}$ to the unique final state q_F.

Now consider a path from q or q' that ends in q'. Such a path must have a corresponding sequence $i_1 \ldots i_k \in [1, n]$ of indices and the projections of the label of this path on processes 1 and 2 are $\alpha_{i_1} \ldots \alpha_{i_k}$ and $\beta_{i_1} \ldots \beta_{i_k}$ respectively. We therefore have the following key property.

Lemma 13. *$L(\mathcal{Q}[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n])$ contains a run $e\,\sigma\,\widehat{e}$ such that $\pi_1(\sigma) \equiv \pi_2(\sigma)$ iff there is a solution to the instance of the PCP defined by $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$.*

Another important property is that runs in $L(\mathcal{Q}[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n])$ are end-to-end quiescent. If we define a FA $\mathcal{S}$ such that $\mathcal{L}_R(\mathcal{S})$ is the language of all runs of the form $\sigma\, e\, \widehat{e}$ such that run σ does not have the same projections at processes 1 and 2 then we will have that $\mathcal{L}_R(\mathcal{Q}[\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n]) \subseteq \mathcal{L}_R(S)$ if and only if there is no solution to this instance of the PCP. The following shows how we can construct such a specification $\mathcal{S}$.

$\mathcal{S}$ has initial state s_0 and for all $a \in \Gamma$ there is a cycle that has label $e^1_a\, \widehat{e}^1_a\, e^2_a\, \widehat{e}^2_a$. This cycle models the case where the projections are identical. We add the following to $\mathcal{S}$ to represent the ways in which a first difference in projections, on processes 1 and 2, can occur.

1. For all $a, b \in \Gamma$ with $a \neq b$ there is a path from s_0 to state s_1 with label $e^1_a\, \widehat{e}^1_a\, e^2_b\, \widehat{e}^2_b$. In s_1 there are separate cycles with labels $e^1_a\, \widehat{e}^1_a$ and $e^2_a\, \widehat{e}^2_a$ for all $a \in \Gamma$.
2. For all $a \in \Gamma$ there is a path from s_0 to state s_2 with label $e^1_a\, \widehat{e}^1_a$. In s_2 there is a cycle with label $e^1_a \widehat{e}^1_a$ for all $a \in \Gamma$.
3. For all $a \in \Gamma$ there is a path from s_0 to state s_3 with label $e^2_a\, \widehat{e}^2_a$. In s_3 there is a cycle with label $e^2_a\, \widehat{e}^2_a$ for all $a \in \Gamma$.

State s_1 represents the case where after some common prefix we have $e^1_a\, \widehat{e}^1_a$ at process 1 and $e^2_b\, \widehat{e}^2_b$ at process 2 for some $a \neq b$. States s_2 and s_3 model the cases where one projection is a proper prefix of the other (s_2 models the case where the projection on process 1 is longer and s_3 models the case where the projection on process 2 is longer). We add a final state s_F and paths from s_1, s_2, s_3 to s_F with label $e\,\widehat{e}$. The following is immediate from the construction.

Lemma 14. *Run $\sigma \in \Sigma^*$ is in $L(\mathcal{S})$ if and only if $\sigma = \sigma'\, e\, \widehat{e}$ for some σ' such that $\pi_1(\sigma') \not\equiv \pi_2(\sigma')$.*

We therefore obtain the following result.

Theorem 10. *The correctness problem for quiescent sequential consistency is undecidable.*

Proof. This follows from Lemmas 13 and 14 and the PCP being undecidable. □

8.2. Restricted Quiescent Sequential Consistency

Our results for restricted quiescent sequential consistency extends the constructions used in Section 5.2. Given FA $\mathcal{M} = (M, m_0, \Sigma, t, M_\dagger)$ and state $m \in M$, we let $\mathsf{Y}_{\mathcal{M}}(q)$ denote the set of runs that are equivalent to end-to-end quiescent runs that label paths that start at m, i.e.,

$$\mathsf{Y}_{\mathcal{M}}(q) = \{\sigma \in \mathcal{L}_R(m \lhd \mathcal{M}) \mid \xi(\sigma)\}$$

We now consider the case where there is a fixed bound b on the number of events that can occur in an end-to-end quiescent trace. For the unbounded case we could not use Algorithm 1 since this would require us to decide whether $\mathsf{Y}_{\mathcal{Q}}(q_c) \subseteq \mathsf{Y}_{\mathcal{S}}(S_c)$ and we know that this is undecidable. However, it is straightforward to see that in the presence of bound b the conditions controlling the loop and If statement of Algorithm 1 involve reasoning about finite languages and so are decidable. We can therefore apply Algorithm 1 and will now show that correctness is in PSPACE.

Proposition 14. *Let us suppose that there is a limit b on the length of end-to-end quiescent runs in $\mathcal{Q}$ and $\mathcal{S}$. Given states q_{c-1} and q_c of $\mathcal{Q}$ and sets S_{c-1} and S_c of states of $\mathcal{S}$, it is possible to decide whether there exists run σ_c such that $q_{c-1} \xrightarrow{\sigma_c}_{\mathcal{Q}} q_c$ and $S_{c-1} \overset{\sigma_c}{\Longrightarrow}_{\mathcal{S}} S_c$ in NP.*

Proof. A non-deterministic Turing Machine can solve this as follows. It guesses a run σ whose length is at most b, requiring constant time, and checks that σ is end-to-end quiescent. It then determines whether $\sigma \in \mathsf{Y}_{\mathcal{Q}}(q_c)$ and whether $\sigma \in \mathsf{Y}_{\mathcal{S}}(S \setminus S_c)$. The non-deterministic Turing Machine returns True if and only if it finds that $\sigma \in \mathsf{Y}_{\mathcal{Q}}(q_c)$ and $\sigma \notin \mathsf{Y}_{\mathcal{S}}(S \setminus S_c)$. From Theorem 8 we know that these steps can be performed in time that is polynomial in the sizes of $\mathcal{Q}$ and $\mathcal{S}$ and so in polynomial time. Thus, a non-deterministic Turing Machine can solve this problem in polynomial time and so the problem is in NP. □

Theorem 11. *The correctness problem for restricted quiescent sequential consistency is PSPACE-complete.*

Proof. From Proposition 14 we know that the first condition in Algorithm 1 can be decided in NP. Thus, a non-deterministic Turing Machine can apply Algorithm 1 using polynomial space and so the problem is in PSPACE.

To see that the problem is PSPACE-hard it is sufficient to again observe that the (PSPACE-hard) FA language inclusion problem can be represented as a correctness problem in which the implementation is sequential and we have a bound $b = 2$. □

9. Conclusions

Concurrent objects (such as the queue example in Section 2.1) form an important class of objects, managing thread synchronisation on behalf of a programmer. The safety properties that a concurrent object satisfies can be understood in terms of the correctness conditions such as sequential consistency, linearizability and quiescent consistency [1].

Decidability and complexity for checking membership and correctness for these conditions have been widely studied. These generally extend Alur et al.'s methods [7], which in turn is based on the notions of independence from Mazurkiewicz Trace Theory. A summary of results from the literature is given in Table 1. The bounded and unbounded versions of linearizability that have been studied refer to the number of processes that a system is assumed to have — the unbounded version does not assume an upper limit on the number of processes. Our results on quiescent consistency and quiescent sequential consistency adds to this existing body of work.

	Membership		**Correctness**	
Correctness condition	Unrestricted	Restricted	Unrestricted	Restricted
Sequential consistency	NP-complete [22]	—	Undecidable [7]	—
Linearizability	NP-complete$^{(a)}$[22]	PTIME$^{(b)}$[22]	EXPSPACE-complete$^{(b)}$ [7, 23], Undecidable$^{(c)}$ [24]	EXPSPACE-complete$^{(d)}$ [24]
Serializability	—	—	PSPACE [7]	—
Conflict serializability	—	—	EXPSPACE-complete [24]	—
Quiescent consistency ⋆	NP-complete	PTIME $^{(b)}$	coNEXPTIME-complete	PSPACE-complete $^{(b)}$
Quiescent sequential consistency ⋆	NP-complete	PTIME $^{(b)}$ or $^{(e)}$	Undecidable	PSPACE $^{(b)}$

Table 1: Summary of decidability and complexity results

$^{(a)}$ Finite number of processes
$^{(b)}$ Predetermined upper limit on number of processes
$^{(c)}$ Potentially infinite number of processes
$^{(d)}$ Implementations with fixed linearization points, but potentially infinite number of processes
$^{(e)}$ Upper limit on number of events between two quiescent states
⋆ Our results

The notion of quiescent consistency we have considered is based on the definition by Derrick et al. [5], which is a formalisation of the definition by Shavit [4]. This definition allows operation calls by the same process to be reordered, i.e., sequential consistency is not necessarily preserved. However, for concurrent objects, sequential consistency is known to be necessary for observational refinement [11], which in turn guarantees substitutability on behalf of a programmer. Therefore, we also study a stronger version quiescent sequential consistency that disallows commutations of events corresponding to the same process.

We have made a number of assumptions on the implementation and specificaton FA under consideration. How does the problem change when these assumptions are modified?

- Assumption 1 introduces an upper bound on the number of processes. Others have considered infinite-state systems in the context of linearizability [24]. Here, dropping Assumption 1 causes the correctness problem for linearizability to become undecidable, whereas correctness for linearizability with a bound on the number of processes is decidable [7]. To recover decidability in the infinite case, one must place restrictions on the algorithms under consideration, in particular, linearizability is EXPSPACE-complete for implementations with "fixed" linearization points [24] (see [1, 25] for examples of such implementations). We believe that the undecidability property holds for quiescent consistency too, but it is not known whether quiescent consistent algorithms can analogously be categorised.

- Assumption 2, which ensures that the specification is sequential, can be relaxed to concurrent (but legal) specifications without affecting our complexity results. However, using such a specification would be at odds with the existing literature on concurrent objects, which assumes sequential specifications [1].

- Assumption 3 can also be relaxed (to allow a single process to call more than one operation at a time) and others have already considered such client behaviours in the context of linearizability [26]. However, it is currently unclear exactly what this means for quiescent consistency, so we leave it for future work.

- Assumption 4 is equivalent to the assumption that the runs under consideration are complete. Dropping this assumption is possible, but it complicates the analysis since each run would have to be completed by adding matching responses to each pending invocation (cf [8]). On the other hand, for quiescent sequential consistency, dropping Assumption 4 means that we immediately obtain an undecidability result for the correctness problem by leveraging Alur et al.'s existing undecidability result for sequential consistency [7], or the undecidability result for language inclusion for a *shuffle operator* [33, 32]. In particular, one could consider executions that contain a single pending invocation at the beginning of each run, where the only quiesent state is the initial state and the only remaining restriction on the reordering is sequential consistency. Our result shows that even for languages restricted to complete executions, quiescent sequential consistency is undecidable. In addition, we use a different reduction (Post's Correspondence Problem) from Alur et al [7] (who use counter machines).

There are further variations of quiescent consistency in the literature. Jagadeesan and Riely have developed a quantitative version of quiescent consistency [15] that only allows reordering if there is adequate contention in the system; here adequate contention is judged in terms of the number of open method calls in the system. It is straightforward to extend the approach used in this paper to show that the membership problem for this quantitative version is NP-complete, but decidability of correctness is not yet known. Versions of quiescent consistency suited to relaxed-memory architectures have also been developed [27, 2], where the notion of a quiescent state incorporates pending write operations stored in local buffers. Consideration of decidability and complexity for membership and correctness of these different variations is a task for future work. In particular, Jagadeesan and Riely's condition forms a class of quantitative correctness conditions, which includes quantitative relaxations of linearizability [28, 29] and sequential consistency [30].

Furbach et al. have studied complexity of the so called *testing problem* for a hierarchy of relaxed-memory models [31]. Here, an implementation generates a sequence of memory operations (e.g., reads/writes) for each thread, and the tester must check whether there is an interleaving of these sequences that is accepted by the memory model in consideration (which forms the specification). They establish that this problem is NP-hard for all but one of the weak memory models they consider. Recent studies have also considered decidability and complexity for *distributed* data structures [32, 33], where the objects in question are spread across multiple replicated sites. Data structures in such systems rely on causal notions of correctness. They show that the membership problem is NP-hard whereas the correctness problem is undecidable [33].

Bouajjani et al. have developed characterisations of algorithm designs that enable reduction of the linearizability problem to a (simpler) state reachability problem [34]. Other work [35] has considered (under)

approximations of history inclusion with the aim of solving the *observational refinement* problem for concurrent objects [10, 11] directly. Linking quiescent consistency and quiescent sequential consistency to the state reachability problem and under approximations for observational refinement are both topics for future work.

Acknowledgements. We thank Ahmed Bouajjani, Constatin Emea and Gustavo Petri for helpful discussions on this work. We are also indebted to the reviewers of this paper and our LICS 2016 paper; these two sets of comments helped us to strengthen the results and simplify some proofs. This work was funded by a Brunel University SEED grant. The first author also acknowledges funding from EPSRC grant EP/N016661/1.

References

[1] M. Herlihy, N. Shavit, The art of multiprocessor programming, Morgan Kaufmann, 2008.
[2] B. Dongol, J. Derrick, L. Groves, G. Smith, Defining correctness conditions for concurrent objects in multicore architectures, in: J. T. Boyland (Ed.), ECOOP, Vol. 37 of LIPIcs, Schloss Dagstuhl, 2015, pp. 470–494.
[3] J. L. Gustafson, Reevaluating Amdahl's law, Commun. ACM 31 (5) (1988) 532–533.
[4] N. Shavit, Data structures in the multicore age, Commun. ACM 54 (3) (2011) 76–84.
[5] J. Derrick, B. Dongol, G. Schellhorn, B. Tofan, O. Travkin, H. Wehrheim, Quiescent consistency: Defining and verifying relaxed linearizability, in: FM, Vol. 8442, Springer, 2014, pp. 200–214.
[6] C. A. Ellis, Consistency and correctness of duplicate database systems, in: S. Rosen, P. J. Denning (Eds.), SOSP, ACM, 1977, pp. 67–84.
[7] R. Alur, K. L. McMillan, D. Peled, Model-checking of correctness conditions for concurrent objects, Information and Computation 160 (1-2) (2000) 167–188.
[8] M. Herlihy, J. M. Wing, Linearizability: A correctness condition for concurrent objects, ACM TOPLAS 12 (3) (1990) 463–492.
[9] L. Lamport, How to make a multiprocessor computer that correctly executes multiprocess programs, IEEE Trans. Comp. 28 (9) (1979) 690–691.
[10] J. He, C. A. R. Hoare, J. W. Sanders, Data refinement refined, in: B. Robinet, R. Wilhelm (Eds.), ESOP, Vol. 213 of LNCS, Springer, 1986, pp. 187–196.
[11] I. Filipovic, P. W. O'Hearn, N. Rinetzky, H. Yang, Abstraction for concurrent objects, Theoretical Computer Science 411 (51-52) (2010) 4379 – 4398.
[12] A. W. Mazurkiewicz, Traces, histories, graphs: Instances of a process monoid, in: M. Chytil, V. Koubek (Eds.), Mathematical Foundations of Computer Science, Vol. 176 of LNCS, Springer, 1984, pp. 115–133.
[13] B. Dongol, R. M. Hierons, Decidability and complexity for quiescent consistency, in: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '16), ACM, 2016, pp. 116–125.
[14] J. Aspnes, M. Herlihy, N. Shavit, Counting networks, J. ACM 41 (5) (1994) 1020–1048.
[15] R. Jagadeesan, J. Riely, Between linearizability and quiescent consistency - quantitative quiescent consistency, in: ICALP, Vol. 8573 of LNCS, Springer, 2014, pp. 220–231.
[16] T. J. Schaefer, The complexity of satisfiability problems, in: STOC, 1978, pp. 216–226.
[17] I. J. Aalbersberg, H. J. Hoogeboom, Characterizations of the decidability of some problems for regular trace languages, Mathematical Systems Theory 22.
[18] C. Haase, P. Hofman, Tightening the complexity of equivalence problems for commutative grammars, in: STACS, Vol. 47 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 41:1–41:14.
[19] D. T. Huynh, The complexity of equivalence problems for commutative grammars, Information and Control 66 (1/2) (1985) 103–121.
[20] S. C. Kleene, Representation of events in nerve nets and finite automata, Automata Studies.
[21] E. L. Post, A variant of a recursively unsolvable problem, Bulletin of the American Mathematical Society 52 (1946) 264–268.
[22] P. B. Gibbons, E. Korach, The complexity of sequential consistency, in: SPDP, IEEE Computer Society, 1992, pp. 317–325.
[23] J. Hamza, On the complexity of linearizability, in: NETYS, Vol. 9466 of LNCS, Springer, 2015, pp. 308–321.
[24] A. Bouajjani, M. Emmi, C. Enea, J. Hamza, Verifying concurrent programs against sequential specifications, in: M. Felleisen, P. Gardner (Eds.), ESOP, Vol. 7792 of LNCS, Springer, 2013, pp. 290–309.
[25] B. Dongol, J. Derrick, Verifying linearisability: A comparative survey, ACM Comput. Surv. 48 (2) (2015) 19:1–19:43.
[26] A. Cerone, A. Gotsman, H. Yang, Parameterised linearisability, in: ICALP (2), Vol. 8573 of LNCS, Springer, 2014, pp. 98–109.
[27] G. Smith, J. Derrick, B. Dongol, Admit your weakness: Verifying correctness on TSO architectures, in: I. Lanese, E. Madelaine (Eds.), FACS, Vol. 8997 of LNCS, Springer, 2014, pp. 364–383.
[28] Y. Afek, G. Korland, E. Yanovsky, Quasi-linearizability: Relaxed consistency for improved concurrency, in: OPODIS, Vol. 6490 of LNCS, Springer, 2010, pp. 395–410.
[29] T. A. Henzinger, C. M. Kirsch, H. Payer, A. Sezgin, A. Sokolova, Quantitative relaxation of concurrent data structures, in: POPL, ACM, 2013, pp. 317–328.
[30] A. Sezgin, Sequential consistency and concurrent data structures, CoRR abs/1506.04910.
URL `http://arxiv.org/abs/1506.04910`

[31] F. Furbach, R. Meyer, K. Schneider, M. Senftleben, Memory model-aware testing - A unified complexity analysis, in: ACSD, IEEE Computer Society, 2014, pp. 92–101.
[32] J. Hamza, Algorithmic verification of concurrent and distributed data structures, Ph.D. thesis, CNRS et Université Paris Diderot (2015).
[33] A. Bouajjani, C. Enea, R. Guerraoui, J. Hamza, On verifying causal consistency, in: POPL, ACM, 2017, pp. 626–638.
[34] A. Bouajjani, M. Emmi, C. Enea, J. Hamza, On reducing linearizability to state reachability, in: M. M. Halldórsson, K. Iwama, N. Kobayashi, B. Speckmann (Eds.), ICALP, Vol. 9135 of LNCS, Springer, 2015, pp. 95–107.
[35] A. Bouajjani, M. Emmi, C. Enea, J. Hamza, Tractable refinement checking for concurrent objects, in: S. K. Rajamani, D. Walker (Eds.), POPL, ACM, 2015, pp. 651–662.