Decidability and Complexity for Quiescent Consistency

Brijesh Dongol Robert M. Hierons

Department of Computer Science,
Brunel University London, UK
firstname.lastname@brunel.ac.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 be 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 consider the membership and correctness conditions for quiescent consistency, as well as a restricted form that assumes an upper limit on the number of events between two quiescent states. We show that the membership problem for unrestricted quiescent consistency is NP-complete and that the correctness problem is decidable, coNEXPTIME-hard, and in EXPSPACE. For the restricted form, we show that membership is in PTIME, while correctness is PSPACE-complete.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification; D.3.1 [*Programming Languages*]: Formal Definitions and Theory—Semantics; F.1.2 [*Computation by Abstract Devices*]: Modes of Computation—Parallelism and concurrency; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs—Logics of programs; H.2.4 [*Systems*]: Concurrency

General Terms Algorithms, Theory, Verification

Keywords Quiescent consistency, concurrent objects, decidability, Mazurkiewicz Trace Theory

1. Introduction

Due to the possibility of interference, correctness of a concurrent object cannot be stated in terms of pre/post conditions of its operations. Instead, correctness is expressed in terms of a *history* of operation invocation/response events, capturing the interaction between a concurrent object and its clients. In particular, a concurrent object's history (with potentially overlapping operation calls) is mapped to a sequential history of its specification (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 notions 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 study the decidability and complexity of both membership and correctness for systems in which the condition C above is *quiescent consistency* [11, 35].

Quiescent consistency is derived from a similar notion in replicated databases [16], that gives meaning to an object in its *quiescent states*, i.e., states in which none of the object's operations are executing. Quiescent consistency allows events of a concurrent object's history *between* two consecutive quiescent states to be reordered (when mapping to a sequential history) but disallows reorderings for events *separated* by a quiescent state [11, 15, 35]. For both membership and correctness, we consider two versions of the problem: an unrestricted version with no limits between two quiescent states and a restricted version that assumes a fixed upper bound on the number of events between two quiescent states.

This paper's main contributions are as follows. *(1)* We describe how quiescent consistency can be expressed using independence from Mazurkiewicz Trace Theory [32] and encoded as finite automata. This extends the methodology developed by Alur et al. [4], and demonstrates the generality of their approach. *(2)* Prove that deciding membership for quiescent consistency is *(i)* NP-complete if the number of events between two quiescent states is unrestricted, and *(ii)* polynomial (with respect to the size of the input run) if the number of events between two quiescent states has a fixed upper limit. *(3)* Prove that correctness for quiescent consistency is *(i)* decidable, coNEXPTIME-hard, and in EXPSPACE in the unrestricted case, and *(ii)* PSPACE-complete in the restricted case.

The restricted version of quiescent consistency has previously not been studied. Our complexity results for it provide motivation for giving the condition greater consideration, e.g., an implementation could use a combination of scheduling algorithms, "try-once" designs and exponential backoff schemes [3] to guarantee that quiescence will eventually occur within a fixed finite number of steps; the benefit being that the membership and correctness problems are both simpler than the unrestricted case.

Sequential bottlenecks within a concurrent implementation must be reduced to improve performance [19, 35], and using a relaxed notion of correctness has been shown to lead to greater performance [6, 7, 37] because it allows greater flexibility in an

CONF 'yy, Month d–d, 20yy, City, ST, Country.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

object's design. Shavit has argued for quiescent consistency as the condition for the multi-core age [35] as it allows reorderings that are not allowed by other conditions in the literature. For example, unlike linearizability [15, 23, 24], it allows the effects of operation calls to be reordered even if the calls do not overlap in a concurrent history; unlike sequential consistency [15, 23, 31], it allows the effects of operation calls by the same process to be reordered. When necessary, an algorithm designer is able to choose from a range of other conditions to provide other types of guarantees, e.g., by taking quantitative aspects into account [2, 28], altering the specification of the object at hand [22], or taking buffers into account in the presence of relaxed memory [12, 15, 36]. Consideration of decidability and complexity questions for these other known correctness conditions lies outside the scope of this paper.

This paper is organised as follows. In Section 2, we motivate the problem using a diffracting queue example, and describe the formal background of finite automata and independence used in the rest of the paper. Section 3 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 Sections 4 and 5, respectively.

2. Preliminaries

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 (see Section 2.3).

2.1 A quiescent consistent queue

We consider the quiescent consistent queue from [11] (see Figs. 1 and 2). The queue is based on the architecture of *diffracting trees*, which uses the following principle (adapted from counting networks [5]). 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 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 Fig. 1). Pseudocode for the queue is given in Fig. 2. Both operations are implemented using a non-blocking atomic `CAS` (Compare-And-Swap) operation that compares the stored local value `e` with the shared variable `v` and updates `v` to a new value `n` if the values of `v` and `e` are still equal:

```
CAS(v,e,n) ==
     atomic{ if v = e
             then v := n; 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 [11] for details).

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

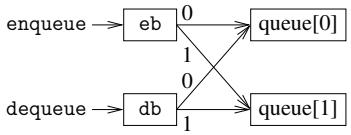

Figure 1. A 1-level diffracting queue with two queues

```
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 of the diffraction queue

$$h_1 = D_1\ \mathbf{E}_2(a)\ \mathbf{E}_3(b)\ \mathbf{D}_4(b)\ \mathbf{D}_5(a)\ \mathbf{E}_6(c)\ \widehat{D}_1(c)$$

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. We also assume $\mathbf{E}_i(j) = E_i(j)\widehat{E}_i$ and $\mathbf{D}_i(j) = D_i\ \widehat{D}_i(j)$.*

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 [24] 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 [11] 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.

$$h_2 = \mathbf{E}_3(b)\ \mathbf{E}_2(a)\ \mathbf{D}_4(b)\ \mathbf{D}_5(a)\ \mathbf{E}_6(c)\ \mathbf{D}_1(c) \qquad \square$$

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. [4], 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, $t : M \times \Sigma \leftrightarrow M$ is the transition relation and $M_\dagger \subseteq M$ is the set of final states.*

Given a finite automaton $\mathcal{M} = (M, m_0, \Sigma, t, M_\dagger)$, $m' \in t(m, e)$ 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'). 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 $end(\rho) = m_{k+1}$ and label $label(\rho) = e_1 e_2 \ldots e_k$. 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,

$$L(\mathcal{M}) = \left\{ label(\rho) \;\middle|\; \begin{array}{l} \rho \in Paths(\mathcal{M}) \wedge \\ start(\rho) = m_0 \wedge end(\rho) \in M_\dagger \end{array} \right\}$$

Given run $\sigma \in L(\mathcal{M})$ 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.

If $\mathcal{M}$ 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 will 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 generates runs such as h_2 in Example 1, and if $\mathcal{Q}$ is the FA for the implementation in Fig. 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)

Note that using FA forces examples such as Section 2.1 to be statically bounded. However, this is not different from other treatments in the literature (e.g., for linearizability [4]). Moreover, the lower bounds still hold for unbounded data structures and it is possible that our results can be extended to context-free languages.

To model concurrent operations, we assume that an operation has separate invoke and return events. We will use natural numbers $\mathbb{N}$ to identify processes and 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 $\mathcal{Q}$. Others have considered infinite-state systems in the context of linearizability [8]. Here, dropping Assumption 1 causes the correctness problem for linearizability to become undecidable, whereas linearizability with a bound on the number of processes is decidable [4]. 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 [8] (see [13, 23] for examples of such implementations).

Each event in Σ is associated with a process, an operation, and an input or output value. Like [4], 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 $e \twoheadrightarrow e'$ 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 $i < k$, $e_i \twoheadrightarrow e_{i+1}$, 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 [4, 23, 24, 28], 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 (different client threads may call concurrent operations). This is captured by Assumption 3.

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

Example 2. *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 $\pi_p(\tau)$ is sequential for each process p.* □

2.3 Independence

In this paper, we study quiescent consistency by using the concept of *independence* from Mazurkiewicz Trace Theory [32]. 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 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 σ' via a sequence of rewrites of the form $\sigma_1 e\, e' \sigma_2 \to_I \sigma_1 e' e \sigma_2$ for $(e, e') \in I$.

Example 3. *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 if $\sigma \in \mathcal{L}_I(\mathcal{S})$ for a given $\sigma \in L(\mathcal{Q})$. (membership)
2. Deciding if $L(\mathcal{Q}) \subseteq \mathcal{L}_I(\mathcal{S})$. (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 $label(\rho)$ is quiescent.

Example 4. *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 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, 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 a 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 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. 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.

Proposition 3. *Suppose that every path of $\mathcal{Q}$ starting from q_0 is a prefix of some 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. □

Due to Proposition 3, it is straightforward to make the following assumption on $\mathcal{Q}$.

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 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 that *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). We introduce a new event $\delta \notin \Sigma$ that signifies quiescence and 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 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 $\mathcal{S}_\delta$ with label δ. Thus, $\mathcal{S}_\delta$ *allows* the inclusion of δ whenever a run of $\mathcal{S}$ reaches a quiescent state.

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). If q is a final state of $\mathcal{Q}$, i.e., $q \in Q_\dagger$, we will 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.

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 5. *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:*

$$h^{\delta}_{2.1} = \delta\ \mathbf{E}_3(b)\ \delta\ \mathbf{E}_2(a)\ \delta\ \mathbf{D}_4(b)\ \delta\ \mathbf{D}_5(a)\ \delta\ \mathbf{E}_6(c)\ \delta\ \mathbf{D}_1(c)\ \delta$$

$$h^{\delta}_{2.2} = \delta\ \mathbf{E}_3(b)\ \mathbf{E}_2(a)\ \delta\ \mathbf{D}_4(b)\ \delta\ \mathbf{D}_5(a)\ \delta\ \mathbf{E}_6(c)\ \mathbf{D}_1(c)\ \delta$$

$$h^{\delta}_{2.3} = \delta\ \mathbf{E}_3(b)\ \mathbf{E}_2(a)\ \mathbf{D}_4(b)\ \mathbf{D}_5(a)\ \mathbf{E}_6(c)\ \mathbf{D}_1(c)\ \delta$$

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 $\mathcal{S}$, stating this in terms of $\mathcal{Q}_\delta$. 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.

Definition 2. *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 (N.B., by Proposition 2, it is always possible to write σ in such a form). 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 3. *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 3, $\sigma_1\sigma_2 \ldots \sigma_k$ is allowed by $\mathcal{S}$, and hence, by Definition 2, $\mathcal{S}$ 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

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).

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 [34].[1]

The construction in the proof of the result below takes an instance of the one-in-three SAT problem and constructs a specification $\mathcal{S}$ that has $k+1$ 'main' states $s_0, \ldots, s_k$ and for boolean variable v_i it has two paths from s_{i-1} to s_i: one path ρ^T_i 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^F_i 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^F_i and ρ^T_i will not matter. A path from s_0 to s_{k+1} is of the form $\rho^{B_1}_1\rho^{B_2}_2 \ldots \rho^{B_k}_k$ 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 each 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 q_0 to q_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 6. *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^T_1 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^F_2 is $e_2\widehat{e}_2$ since C_2 is the only clause that contains literal $\neg v_2$. Consider now the path*

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

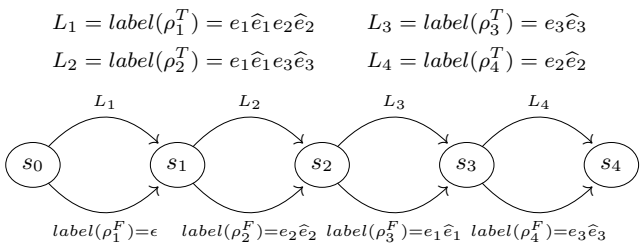

Figure 3. Finite automaton for Example 6

$\rho_1^T\rho_2^F\rho_3^F\rho_4^F$, *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 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 7. *Checking the assignment in Example 6, 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 run $\sigma = e_1\widehat{e}_1 \ldots e_n\widehat{e}_n$ and the FA described above. 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. Assume that we are given an instance of the one-in-three SAT problem defined by boolean variables $v_1, \ldots, v_k$ and clauses $C_1, \ldots, C_n$. We define a specification with invocation events $e_0, e_1, \ldots, e_n, e$ and corresponding return events $\widehat{e}_0, \widehat{e}_1, \ldots, \widehat{e}_n, \widehat{e}$. Define a finite automaton specification $\mathcal{S}$ as follows. The state set of $\mathcal{S}_\delta$ includes states $s_0, s_1, \ldots, s_k$ and s, s' with s being the initial state. For all $1 \leq i \leq k$ there are two paths from s_{i-1} to s_i: path ρ_i^T has invocation/response pair $e_j, \widehat{e}_j$ for every clause C_j that contains literal v_i; and path ρ_i^F has invocation/response pair $e_j, \widehat{e}_j$ for every clause C_j that contains literal $\neg v_i$. Thus, 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 $B_1, \ldots, B_k \in \{T, F\}$. From the initial state s the path to s_0 has run $e_0\widehat{e}_0$ and from s_k the path to the final state s' has run $e\,\widehat{e}$.

Consider the run $\sigma = e_0e_1\widehat{e}_1 \ldots e_n\widehat{e}_ne\,\widehat{e}\,\widehat{e}_0$. We prove that σ is in $\mathcal{L}_U(\mathcal{S}_\delta)$ 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 $\sigma \in \mathcal{L}_U(\mathcal{S}_\delta)$ then the corresponding run of $\mathcal{S}_\delta$ must end at state s' since σ contains the events e and $\widehat{e}$. In addition, σ is end-to-end quiescent and so we simply require that some permutation of σ is in $L(\mathcal{S}_\delta)$. Thus, $\sigma \in \mathcal{L}_U(\mathcal{S}_\delta)$ iff $\mathcal{S}_\delta$ 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 $\mathcal{S}_\delta$ that is of the form $\rho = \rho_1^{B_1}\rho_2^{B_2}\ldots\rho_k^{B_k}$. Thus, $\sigma \in \mathcal{L}_U(\mathcal{S}_\delta)$ 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 $\mathcal{S}_\delta$ 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 5) 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 4. *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 $\mathcal{B}$.

Definition 5. *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 Definition 5 to convert the membership problem into a problem of deciding whether two FA 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 assume 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 the correctness problem, we might directly compare $L(\mathcal{Q})$ and $L(\mathcal{S})$, i.e., requires that $L(\mathcal{Q}) \subseteq L(\mathcal{S})$ holds. However, this limits the potential for concurrency — $\mathcal{Q}$ would essentially be sequential. The effect of using a 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, coNEXPTIME-hard and in EXPSPACE.

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 [1].

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-hard [20]

To use the coNEXPTIME-hard 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 A' to the state set of 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.
$x_1 x_2 \ldots x_k \in L(\mathcal{A})$ *iff* $e\, e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \ldots e_{x_k} \widehat{e}_{x_k} \widehat{e} \in L(\mathcal{A}')$.

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 B' to the state set of 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.
$x_1 x_2 \ldots x_k \in L(\mathcal{B})$ *iff* $e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \ldots e_{x_k} \widehat{e}_{x_k} e\, \widehat{e} \in L(\mathcal{B}')$.

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 $x_1 x_2 \ldots x_k \in L(\mathcal{A})$ such that $\sigma \sim_U \delta\sigma'\delta$, where $\sigma' = e\, e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \ldots e_{x_k} \widehat{e}_{x_k} \widehat{e}$. Since $PI(\mathcal{A}) \subseteq PI(\mathcal{B})$ we have that $L(\mathcal{B})$ contains a permutation $y_1 \ldots y_k$ of $x_1 \ldots x_k$. By Proposition 7, $e_{y_1} \widehat{e}_{y_1} e_{y_2} \widehat{e}_{y_2} \ldots e_{y_k} \widehat{e}_{y_k} e\, \widehat{e} \in L(\mathcal{B}')$ and so we also have that $\delta\sigma''\delta \in \mathcal{L}_U(\mathcal{B}'_\delta)$ where $\sigma'' = e_{y_1} \widehat{e}_{y_1} e_{y_2} \widehat{e}_{y_2} \ldots e_{y_k} \widehat{e}_{y_k} e\, \widehat{e}$. As $y_1 \ldots y_k$ is a permutation of $x_1 \ldots x_k$, $\sigma'' \sim_U \sigma'$. 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 $\sigma' = x_1 \ldots x_k$ in $L(\mathcal{A})$ with Parikh Image γ. By Proposition 6, $e\, e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \ldots e_{x_k} \widehat{e}_{x_k} \widehat{e} \in L(\mathcal{A}')$. Thus, $\delta\, e\, e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \ldots e_{x_k} \widehat{e}_{x_k} \widehat{e}\, \delta \in \mathcal{L}_U(\mathcal{A}'_\delta)$. Since $\mathcal{L}_U(\mathcal{A}'_\delta) \subseteq \mathcal{L}_U(\mathcal{B}'_\delta)$, $\delta\, e\, e_{x_1} \widehat{e}_{x_1} e_{x_2} \widehat{e}_{x_2} \ldots e_{x_k} \widehat{e}_{x_k} \widehat{e}\, \delta \in \mathcal{L}_U(\mathcal{B}'_\delta)$. By construction, this implies that $e_{y_1} \widehat{e}_{y_1} e_{y_2} \widehat{e}_{y_2} \ldots e_{y_k} \widehat{e}_{y_k} e\, \widehat{e} \in L(\mathcal{B}')$ for some permutation $y_1 \ldots y_k$ of $x_1 \ldots x_k$. By Proposition 7 we therefore know that $y_1 \ldots y_k \in L(\mathcal{B})$. Finally, since $y_1 \ldots y_k$ and $x_1 \ldots x_k$ 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, which holds by Lemma 6 and inclusion of Parikh images being coNEXPTIME-hard.

Theorem 4. *The correctness problem for quiescent consistency is coNEXPTIME-hard.*

5.2 Upper bound for unrestricted quiescent consistency

We now investigate the upper bounds on the complexity of deciding correctness of quiescent consistency and show that the problem is in EXPSPACE. This proof is much more involved than the lower bound result as it is necessary to first derive an algorithm for checking correctness 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:

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

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 run σ, we define some further notation:

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

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

Thus, $m \xrightarrow{\sigma}_{\mathcal{M}} m'$ holds iff there is some path in $\mathcal{M}$ with labels 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 there is some $\sigma = \sigma_1 \ldots \sigma_{k+1}$ (where $\xi(\sigma_i)$) that is a counter-example for correctness of quiescent consistency 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 $\sigma = \sigma_1 \ldots \sigma_{k+1}$ for end-to-end quiescent $\sigma_1, \ldots, \sigma_{k+1}$ and corresponding pairs $(q_0, S_0), (q_1, S_1), \ldots, (q_k, S_k) \in Q \times 2^S$ such that $S_0 = \{s_0\}$, $q_{i-1} \xrightarrow{\sigma_i}_{\mathcal{Q}} q_i$ and $S_{i-1} \xRightarrow{\sigma_i}_{\mathcal{S}} S_i$ (all $1 \leq i \leq k$) such that:*

1. *$\sigma_{k+1} \in \mathsf{Z}_{\mathcal{Q}}(q_k)$ and $\sigma_{k+1} \notin \mathsf{Z}_{\mathcal{S}}(S_k)$, and*
2. *$k \leq |Q| \cdot 2^{|S|}$.*

Proof. The existence of such a σ demonstrates that $\mathcal{Q}$ is not a correct implementation of $\mathcal{S}$ under quiescent consistency and so 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, $\sigma = \sigma_1 \ldots \sigma_{k+1}$ for end-to-end quiescent $\sigma_1, \ldots, \sigma_{k+1}$. Since σ is in $L(\mathcal{Q})$ but not in $\mathcal{L}_U(\mathcal{S})$, by the minimality of σ we must have that $\sigma_{k+1} \in \mathsf{Z}_{\mathcal{Q}}(q_k)$ and $\sigma_{k+1} \notin \mathsf{Z}_{\mathcal{S}}(S_k)$ and so the first condition holds. 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 $\sigma_i \ldots \sigma_{j-1}$ 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 second 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 correctness. At each iteration, the non-deterministic Turing Machine first checks whether $\mathsf{Z}_{\mathcal{Q}}(q_c) \not\subseteq \mathsf{Z}_{\mathcal{S}}(S_c)$; if not, it has demonstrated that $\mathcal{Q}$ is not a correct implementation of $\mathcal{S}$ (the first condition of Proposition 9). If this condition holds then 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. The bound on c ensures that the algorithm terminates as long as we can decide the conditions contained in the **if** statements (we explore this below).

Algorithm 1 Deciding correctness for quiescent consistency

```
c = 0, S_0 = {s_0}, Q_0 = {q_0}
while c ≤ |Q| · 2^|S| do
  if Z_Q(q_c) ⊈ Z_S(S_c) then
    Return Fail
  end if
  c = c+1
  Choose some (q_c, S_c) ∈ Q × 2^S
  if ∄σ_c such that q_{c-1} -σ_c->_Q q_c and S_{c-1} =σ_c=>_S S_c then
    Return Ok
  end if
end while
```

If a non-deterministic Turing Machine operates as above then it will return Fail if there is some sequence of choices that leads to Fail being returned. 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 two problems encoded in the conditions of Algorithm 1: deciding whether $\mathsf{Z}_{\mathcal{Q}}(q_c) \not\subseteq \mathsf{Z}_{\mathcal{S}}(S_c)$; and deciding whether there exists a run σ_c such that $q_{c-1} \xrightarrow{\sigma_c}_{\mathcal{Q}} q_c$ and $S_{c-1} \xRightarrow{\sigma_c}_{\mathcal{S}} S_c$.

We start with problem of deciding whether $\mathsf{Z}_{\mathcal{Q}}(q_c) \not\subseteq \mathsf{Z}_{\mathcal{S}}(S_c)$. This 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) [25].

Proposition 11. *It is possible to decide whether $\mathsf{Z}_{\mathcal{Q}}(q_c) \not\subseteq \mathsf{Z}_{\mathcal{S}}(S_c)$ in NEXPTIME.*

The remaining problem we need to decide, for states q_{c-1}, q_c of $\mathcal{Q}$ and sets S_{c-1}, S_c of states of $\mathcal{S}$, is 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 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).

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

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 can take $\mathcal{S}$ to all states in S_c (and so that $\sigma_c \in \bigcap_{s \in S_c} \mathsf{Z}_{\mathcal{S}}(S_{c-1}, s)$) and 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 negation of the overall condition thus reduces to the following.

$$\nexists \sigma_c \in (A \setminus B) \cap C \tag{1}$$

where $A = \bigcap_{s \in S_c} \mathsf{Z}_{\mathcal{S}}(S_{c-1}, s)$, $B = \bigcup_{s \in (S \setminus S_c)} \mathsf{Z}_{\mathcal{S}}(S_{c-1}, s)$, and $C = \mathsf{Z}_{\mathcal{Q}}(q_{c-1}, q_c)$.

Using some straightforward set manipulation, (1) is equal to $A \cap C \subseteq B$. Thus, the problem is reduced to deciding whether the intersection of a set of Parikh Images of regular languages is contained within the Parikh Image of another regular language. We also note that if we use $\overline{L}$ to represent the complement of a language L then $A \cap C \subseteq B$ if and only if $C \subseteq B \cup \overline{A}$, and by de Morgan's Law $\overline{\bigcap_i A_i}$ is equivalent to $\bigcup_i \overline{A_i}$. Condition (1) therefore becomes

$$\mathsf{Z}_{\mathcal{Q}}(q_{c-1}, q_c) \subseteq \left(\bigcup_{s \in (S \setminus S_c)} \mathsf{Z}_{\mathcal{S}}(S_{c-1}, s)\right) \cup \left(\bigcup_{s \in S_c} \overline{\mathsf{Z}_{\mathcal{S}}(S_{c-1}, s)}\right)$$

The Parikh Image of a regular language can be represented by a semi-linear set that contains exponentially many terms [30]. In addition, the complement of a semi-linear set can be represented by polynomially many terms [27]. Thus, all of $\mathsf{Z}_{\mathcal{Q}}(q_{c-1}, q_c)$, $\bigcup_{s \in (S \setminus S_c)} \mathsf{Z}_{\mathcal{S}}(S_{c-1}, s)$, and $\bigcup_{s \in S_c} \overline{\mathsf{Z}_{\mathcal{S}}(S_{c-1}, s)}$ can be represented using exponentially many terms (linear sets). Further, the problem of deciding whether one semi-linear set is contained in another is in Σ_2^p [26][2] and so is in PSPACE. The overall problem is thus in EXPSPACE (since there are exponentially many terms).

Proposition 12. *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 EXPSPACE.*

We can now bring these results together.

Theorem 5. *The correctness problem for quiescent consistency is in EXPSPACE.*

Proof. We know that a non-deterministic Turing Machine can use Algorithm 1 to solve the problem. By Propositions 11 and 12 the conditions of the **if** statements can be solved in NEXPTIME and EXPSPACE. Observe also that the storage required for the algorithm, beyond determining the conditions in the **if** statements, is polynomial since the algorithm only has to store the current values of q_c, S_c and c, the latter taking $\log(|Q|2^{|S|})$ space. Since NEXPTIME is contained in EXPSPACE, we therefore have that a non-deterministic Turing Machine can solve the problem in non-deterministic EXPSPACE (NEXPSPACE). The result now follows from Savitch's theorem [33], which implies that NEXPSPACE = EXPSPACE. □

5.3 Complexity 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 13. *If there is a bound on the length of end-to-end quiescent runs in $\mathcal{Q}$ and $\mathcal{S}$, then is possible to decide whether $\mathsf{Z}_{\mathcal{Q}}(q_c) \subseteq \mathsf{Z}_{\mathcal{S}}(S_c)$ in PSPACE.*

Proof. A nondeterministic Turing Machine can solve this problem in PSPACE as follows. First, it guesses a run σ whose length is at most the upper bound. It then checks that σ is end-to-end quiescent. It then checks whether $\sigma \in \mathsf{Z}_{\mathcal{Q}}(q_c)$ and whether $\sigma \in \mathsf{Z}_{\mathcal{S}}(S_c)$; we know that these checks can be performed in polynomial time since this is an instance of the restricted membership problem. Finally, it returns failure if and only if $\sigma \in \mathsf{Z}_{\mathcal{Q}}(q_c)$ and $\sigma \notin \mathsf{Z}_{\mathcal{S}}(S_c)$. □

Proposition 14. *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 PSPACE.*

Proof. A nondeterministic Turing Machine can solve this problem in PSPACE 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

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

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

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

Proof. From Propositions 13 and 14 we know that the two conditions in Algorithm 1 can be decided in PSPACE. 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 $\mathcal{S}$. 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 [29]. □

6. Conclusions

This paper studied complexity questions for membership and correctness of quiescent consistency, which is a correctness condition for concurrent objects. Like Alur et al.'s results for linearizability [4], the study is based on notions of independence from trace theory. Our main results are that the membership problem for the unbounded case of quiescent consistency is NP-complete, but that the bounded case has polynomial complexity. Correctness, on the other hand, is in EXPSPACE and is coNEXPTIME-hard and for the bounded case it is PSPACE-complete.

The notion of quiescent consistency we have considered is based on the definition by Derrick et al. [11], which is a formalisation of Shavit's definition [35]. This definition allows operation calls by the same process to be reordered, i.e., *sequential consistency* [31] is not necessarily preserved. The absence of sequential consistency in quiescent consistency makes the notion of "concurrent operations" imprecise. Two operations on the same thread may be treated as being concurrent if they occur between the same quiescent points, 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.

Variations of quiescent consistency have also been developed. We have studied membership and correctness for a stronger version of quiescent consistency that preserves sequential consistency [14]. Here, bounding the number of processes changes the complexity of the problem. Others have considered quantitative strengthenings of quiescent consistency [28], which includes quantitative relaxations of linearizability [2, 22]. It is straightforward 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 [15, 36], where the notion of a quiescent state incorporates pending write operations stored in local buffers. Consideration of these different variations is a task for future work.

The problem of deciding membership and correctness for several other correctness conditions have been studied. Correctness for sequentially consistency is undecidable, even when the number of concurrent processes is bounded [4]. Serializability (a consistency

[2] Cited in [25].

condition used in databases) is in PSPACE and linearizability is in EXPSPACE when the number of concurrent processes is bounded [17]. For an unbounded number of processes, Bouajjani et al. [8] have shown that serializability is EXPSPACE-complete, while decidability for linearizability varies depending on the type of linearization points of the operations. In particular, for operations with fixed linearization points deciding correctness of linearizability is EXPSPACE-complete, but in the general case linearizability is undecidable.

Bouajjani et al. have developed characterisations of algorithm designs that enable reduction of the linearizability problem to a (simpler) state reachability problem [9]. Other work [10] has considered (under) approximations of history inclusion with the aim of solving the *observational refinement* problem for concurrent objects [18, 21] directly. Linking quiescent consistency to the state reachability problem and under approximations for observational refinement are both topics for future work.

Acknowledgements. We are deeply indebted to our reviewers whose comments led to significant improvements, in particular, one reviewer whose comments helped us establish coNEXPTIME-hardness (unrestricted case) and PSPACE-hardness (restricted case). This work was funded by a Brunel University SEED grant and EPSRC grant EP/N016661/1.

References

[1] I. J. Aalbersberg and H. J. Hoogeboom. Characterizations of the decidability of some problems for regular trace languages. *Mathematical Systems Theory*, 22, 1989.

[2] Y. Afek, G. Korland, and E. Yanovsky. Quasi-linearizability: Relaxed consistency for improved concurrency. In *OPODIS*, volume 6490 of *LNCS*, pages 395–410. Springer, 2010.

[3] A. Agarwal and M. Cherian. Adaptive backoff synchronization techniques. In J. Syre, editor, *ISCA*, pages 396–406. ACM, 1989.

[4] R. Alur, K. L. McMillan, and D. Peled. Model-checking of correctness conditions for concurrent objects. *Information and Computation*, 160 (1-2):167–188, 2000.

[5] J. Aspnes, M. Herlihy, and N. Shavit. Counting networks. *J. ACM*, 41 (5):1020–1048, Sept. 1994.

[6] H. Attiya and J. L. Welch. Sequential consistency versus linearizability. *ACM Trans. Comput. Syst.*, 12(2):91–122, 1994.

[7] H. Attiya, R. Guerraoui, D. Hendler, P. Kuznetsov, M. M. Michael, and M. T. Vechev. Laws of order: expensive synchronization in concurrent algorithms cannot be eliminated. In T. Ball and M. Sagiv, editors, *POPL*, pages 487–498. ACM, 2011.

[8] A. Bouajjani, M. Emmi, C. Enea, and J. Hamza. Verifying concurrent programs against sequential specifications. In *ESOP*, volume 7792 of *LNCS*, pages 290–309. Springer, 2013.

[9] A. Bouajjani, M. Emmi, C. Enea, and J. Hamza. On reducing linearizability to state reachability. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *ICALP*, volume 9135 of *LNCS*, pages 95–107. Springer, 2015.

[10] A. Bouajjani, M. Emmi, C. Enea, and J. Hamza. Tractable refinement checking for concurrent objects. In S. K. Rajamani and D. Walker, editors, *POPL*, pages 651–662. ACM, 2015.

[11] J. Derrick, B. Dongol, G. Schellhorn, B. Tofan, O. Travkin, and H. Wehrheim. Quiescent consistency: Defining and verifying relaxed linearizability. In *FM*, volume 8442, pages 200–214. Springer, 2014.

[12] J. Derrick, G. Smith, and B. Dongol. Verifying linearizability on TSO architectures. In E. Albert and E. Sekerinski, editors, *iFM*, volume 8739 of *LNCS*, pages 341–356. Springer, 2014.

[13] B. Dongol and J. Derrick. Verifying linearisability: A comparative survey. *ACM Comput. Surv.*, 48(2):19:1–19:43, Sept. 2015.

[14] B. Dongol and R. M. Hierons. Decidability and Complexity for Quiescent Consistency and its Variations. *ArXiv e-prints*, abs/1511.08447, 2015. URL `http://arxiv.org/abs/1511.08447`.

[15] B. Dongol, J. Derrick, L. Groves, and G. Smith. Defining correctness conditions for concurrent objects in multicore architectures. In J. T. Boyland, editor, *ECOOP*, volume 37 of *LIPIcs*, pages 470–494. Schloss Dagstuhl, 2015.

[16] C. A. Ellis. Consistency and correctness of duplicate database systems. In S. Rosen and P. J. Denning, editors, *SOSP*, pages 67–84. ACM, 1977.

[17] A. Farzan and P. Madhusudan. Monitoring atomicity in concurrent programs. In A. Gupta and S. Malik, editors, *CAV*, volume 5123 of *LNCS*, pages 52–65. Springer, 2008.

[18] I. Filipovic, P. W. O'Hearn, N. Rinetzky, and H. Yang. Abstraction for concurrent objects. *Theoretical Computer Science*, 411(51-52):4379 – 4398, 2010.

[19] J. L. Gustafson. Reevaluating Amdahl's law. *Commun. ACM*, 31(5): 532–533, 1988.

[20] C. Haase and P. Hofman. Tightening the complexity of equivalence problems for commutative grammars. In *STACS*, volume 47 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[21] J. He, C. A. R. Hoare, and J. W. Sanders. Data refinement refined. In B. Robinet and R. Wilhelm, editors, *ESOP*, volume 213 of *LNCS*, pages 187–196. Springer, 1986.

[22] T. A. Henzinger, C. M. Kirsch, H. Payer, A. Sezgin, and A. Sokolova. Quantitative relaxation of concurrent data structures. In *POPL*, pages 317–328. ACM, 2013.

[23] M. Herlihy and N. Shavit. *The art of multiprocessor programming*. Morgan Kaufmann, 2008. ISBN 978-0-12-370591-4.

[24] M. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM TOPLAS*, 12(3):463–492, 1990.

[25] D. T. Huynh. The complexity of equivalence problems for commutative grammars. *Information and Control*, 66(1/2):103–121, 1985.

[26] D. T. Huynh. A simple proof for the $\sum_2^p$ upper bound of the inequivalence problem for semilinear sets. *Elektronische Informationsverarbeitung und Kybernetik*, 22(4):147–156, 1986.

[27] T. Huynh. The complexity of semilinear sets. In J. W. de Bakker and J. van Leeuwen, editors, *ICALP*, volume 85 of *LNCS*, pages 324–337. Springer, 1980.

[28] R. Jagadeesan and J. Riely. Between linearizability and quiescent consistency - quantitative quiescent consistency. In *ICALP*, volume 8573 of *LNCS*, pages 220–231. Springer, 2014.

[29] S. C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, 1956.

[30] E. Kopczynski and A. W. To. Parikh images of grammars: Complexity and applications. In *LICS*, pages 80–89. IEEE Comp. Soc., 2010.

[31] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Comp.*, 28(9):690–691, 1979.

[32] A. W. Mazurkiewicz. Traces, histories, graphs: Instances of a process monoid. In M. Chytil and V. Koubek, editors, *MFCS*, volume 176 of *LNCS*, pages 115–133. Springer, 1984.

[33] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4 (2):177 – 192, 1970.

[34] T. J. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978.

[35] N. Shavit. Data structures in the multicore age. *Commun. ACM*, 54 (3):76–84, 2011.

[36] G. Smith, J. Derrick, and B. Dongol. Admit your weakness: Verifying correctness on TSO architectures. In I. Lanese and E. Madelaine, editors, *FACS*, volume 8997 of *LNCS*, pages 364–383. Springer, 2014.

[37] E. Talmage and J. L. Welch. Improving average performance by relaxing distributed data structures. In F. Kuhn, editor, *DISC*, volume 8784 of *LNCS*, pages 421–438. Springer, 2014.